Algorithms for Computing Triangular Decompositions of Polynomial Systems

Changbo Chen
ORCCA, University of Western Ontario (UWO)
London, Ontario, Canada
cchen252@csd.uwo.ca

Marc Moreno Maza
ORCCA, University of Western Ontario (UWO)
London, Ontario, Canada
moreno@csd.uwo.ca

ABSTRACT

We propose new algorithms for computing triangular decompositions of polynomial systems incrementally. With respect to previous works, our improvements are based on a *weakened* notion of a polynomial GCD modulo a regular chain, which permits to greatly simplify and optimize the sub-algorithms. Extracting common work from similar expensive computations is also a key feature of our algorithms. In our experimental results the implementation of our new algorithms, realized with the `RegularChains` library in MAPLE, outperforms solvers with similar specifications by several orders of magnitude on sufficiently difficult problems.

Categories and Subject Descriptors

I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms—*Algebraic algorithms*

General Terms

Algorithms, Experimentation, Theory

Keywords

regular chain, triangular decomposition, incremental algorithm, subresultant, polynomial system, regular GCD

1. INTRODUCTION

The Characteristic Set Method [22] of Wu has freed Ritt's decomposition from polynomial factorization, opening the door to a variety of discoveries in polynomial system solving. In the past two decades the work of Wu has been extended to more powerful decomposition algorithms and applied to different types of polynomial systems or decompositions: differential systems [2, 11], difference systems [10], real parametric systems [23], primary decomposition [18], cylindrical algebraic decomposition [5]. Today, triangular decomposition algorithms provide back-engines for computer algebra system front-end solvers, such as MAPLE's `solve` command.

Algorithms computing triangular decompositions of polynomial systems can be classified in several ways. One can first consider the relation between the input system S and the output triangular systems $S_1, \ldots, S_e$. From that perspective, two types of decomposition are essentially different: those for which $S_1, \ldots, S_e$ encode all the points of the zero set S (over the algebraic closure of the coefficient field of S) and those for which $S_1, \ldots, S_e$ represent only the "generic zeros" of the irreducible components of S.

One can also classify triangular decomposition algorithms by the algorithmic principles on which they rely. From this other angle, two types of algorithms are essentially different: those which proceed *by variable elimination*, that is, by reducing the solving of a system in n unknowns to that of a system in $n-1$ unknowns and those which proceed *incrementally*, that is, by reducing the solving of a system in m equations to that of a system in $m-1$ equations.

The Characteristic Set Method and the algorithm in [21] belong to the first type in each classification. Kalkbrener's algorithm [12], which is an elimination method solving in the sense of the "generic zeros", has brought efficient techniques, based on the concept of a *regular chain*. Other works [13, 17] on triangular decomposition algorithms focus on incremental solving. This principle is quite attractive, since it allows to control the properties and size of the intermediate computed objects. It is used in other areas of polynomial system solving such as the probabilistic algorithm of Lecerf [14] based on lifting fibers and the numerical method of Sommese, Verschelde, Wample [19] based on diagonal homotopy.

Incremental algorithms for triangular decomposition rely on a procedure for computing the intersection of an hypersurface and the quasi-component of a regular chain. Thus, the input of this operation can be regarded as well-behaved geometrical objects. However, known algorithms, namely the one of Lazard [13] and the one of the second author [17] are quite involved and difficult to analyze and optimize.

In this paper, we revisit this intersection operation. Let $R = \mathbf{k}[x_1, \ldots, x_n]$ be the ring of multivariate polynomials with coefficients in $\mathbf{k}$ and ordered variables $\mathbf{x} = x_1 < \cdots < x_n$. Given a polynomial $p \in R$ and a regular chain $T \subset \mathbf{k}[x_1, \ldots, x_n]$, the function call $\mathsf{Intersect}(p, T, R)$ returns regular chains $T_1, \ldots, T_e \subset \mathbf{k}[x_1, \ldots, x_n]$ such that we have:

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

(See Section 2 for the notion of a regular chain and related concepts and notations.) We exhibit an algorithm for computing $\mathsf{Intersect}(p, T, R)$ which is conceptually simpler and practically much more efficient than those of [13, 17]. Our improvements result mainly from two new ideas.

ISSAC'11, June 8–11, 2011, San Jose, California, USA.

Weakened notion of polynomial GCDs modulo regular chain. Modern algorithms for triangular decomposition rely implicitly or explicitly on a notion of GCD for univariate polynomials over an arbitrary commutative ring. A formal definition was proposed in [17] (see Definition 1) and applied to residue class rings of the form $\mathbb{A} = \mathbf{k}[\mathbf{x}]/\mathrm{sat}(T)$ where $\mathrm{sat}(T)$ is the saturated ideal of the regular chain T. A modular algorithm for computing these GCDs appears in [15]: if $\mathrm{sat}(T)$ is known to be radical, the performance (both in theory and practice) of this algorithm are very satisfactory whereas if $\mathrm{sat}(T)$ is not radical, the complexity of the algorithm increases substantially w.r.t. the radical case. In this paper, the ring $\mathbb{A}$ will be of the form $\mathbf{k}[\mathbf{x}]/\sqrt{\mathrm{sat}(T)}$ while our algorithms will not need to compute a basis nor a characteristic set of $\sqrt{\mathrm{sat}(T)}$. For the purpose of polynomial system solving (when retaining the multiplicities of zeros is not required) this weaker notion of a polynomial GCD is clearly sufficient. In addition, this yields a very simple procedure for computing such GCDs, see Theorem 1. To this end, we rely on the *specialization property of subresultants*. The technical report [4] reviews this property and provides corner cases for which we could not find a reference in the literature.

Extracting common work from similar computations. Up to technical details, if T consists of a single polynomial t whose main variable is the same as p, say v, computing $\mathsf{Intersect}(p, T, R)$ can be achieved by successively computing

(s_1) the resultant r of p and t w.r.t. v,

(s_2) a regular GCD of p and t modulo the squarefree part of r.

Observe that Steps (s_1) and (s_2) reduce essentially to computing the subresultant chain of p and t w.r.t. v. The algorithms of Section 4 extend this simple observation for computing $\mathsf{Intersect}(p, T, R)$ with an arbitrary regular chain. In broad terms, the intermediate polynomials computed during the "elimination phasis" of $\mathsf{Intersect}(p, T, R)$ are recycled for performing the "extension phasis" at essentially no cost.

The techniques developed for $\mathsf{Intersect}(p, T, R)$ are applied to other key sub-algorithms, such as the regularity test of a polynomial modulo the saturated of a regular chain, see Section 4. The primary application of the operation $\mathsf{Intersect}$ is to obtain triangular decomposition encoding all the points of the zero set of the input system. However, we also derive from it in Section 6 an algorithm computing triangular decompositions in the sense of Kalkbrener.

Experimental results. We have implemented the algorithms presented in this paper within the `RegularChains` library in MAPLE, leading to a new implementation of the Triangularize command. In Section 7, we report on various benchmarks. This new version of Triangularize outperforms the previous ones (based on [17]) by several orders of magnitude on sufficiently difficult problems. Other MAPLE commands or packages for solving polynomial systems (the `WSolve` package, the `Groebner:-Solve` command and the `Groebner:-Basis` command for a lexicographical term order) are also outperformed by the implementation of the algorithms presented in this paper both in terms of running time and, in the case of engines based on Gröbner bases, in terms of output size.

2. REGULAR CHAINS

We review hereafter the notion of a regular chain and its related concepts. Then we state basic properties (Propositions 1, 2, 3, 4, and Corollaries 1, 2) of regular chains, which are at the core of the proofs of the algorithms of Section 4.

Throughout this paper, $\mathbf{k}$ is a field, $\mathbf{K}$ is the algebraic closure of $\mathbf{k}$ and $\mathbf{k}[\mathbf{x}]$ denotes the ring of polynomials over $\mathbf{k}$, with ordered variables $\mathbf{x} = x_1 < \cdots < x_n$. Let $p \in \mathbf{k}[\mathbf{x}]$.

Notations for polynomials. If p is not constant, then the greatest variable appearing in p is called the *main variable* of p, denoted by $\mathrm{mvar}(p)$. Furthermore, the leading coefficient, the degree, the leading monomial, the leading term and the reductum of p, regarded as a univariate polynomial in $\mathrm{mvar}(p)$, are called respectively the *initial*, the *main degree*, the *rank*, the *head* and the *tail* of p; they are denoted by $\mathrm{init}(p)$, $\mathrm{mdeg}(p)$, $\mathrm{rank}(p)$, $\mathrm{head}(p)$ and $\mathrm{tail}(p)$ respectively. Let q be another polynomial of $\mathbf{k}[\mathbf{x}]$. If q is not constant, then we denote by $\mathrm{prem}(p, q)$ and $\mathrm{pquo}(p, q)$ the pseudo-remainder and the pseudo-quotient of p by q as univariate polynomials in $\mathrm{mvar}(q)$. We say that p is less than q and write $p \prec q$ if either $p \in \mathbf{k}$ and $q \notin \mathbf{k}$ or both are non-constant polynomials such that $\mathrm{mvar}(p) < \mathrm{mvar}(q)$ holds, or $\mathrm{mvar}(p) = \mathrm{mvar}(q)$ and $\mathrm{mdeg}(p) < \mathrm{mdeg}(q)$ both hold. We write $p \sim q$ if neither $p \prec q$ nor $q \prec p$ hold.

Notations for polynomial sets. Let $F \subset \mathbf{k}[\mathbf{x}]$. We denote by $\langle F \rangle$ the ideal generated by F in $\mathbf{k}[\mathbf{x}]$. For an ideal $\mathcal{I} \subset \mathbf{k}[\mathbf{x}]$, we denote by $\dim(\mathcal{I})$ its dimension. A polynomial is *regular* modulo $\mathcal{I}$ if it is neither zero, nor a zerodivisor modulo $\mathcal{I}$. Denote by $V(F)$ the *zero set* (or algebraic variety) of F in $\mathbf{K}^n$. Let $h \in \mathbf{k}[\mathbf{x}]$. The *saturated ideal* of $\mathcal{I}$ w.r.t. h, denoted by $\mathcal{I} : h^\infty$, is the ideal $\{q \in \mathbf{k}[\mathbf{x}] \mid \exists m \in \mathbb{N} \text{ s.t. } h^m q \in \mathcal{I}\}$.

Triangular set. Let $T \subset \mathbf{k}[\mathbf{x}]$ be a *triangular set*, that is, a set of non-constant polynomials with pairwise distinct main variables. The set of main variables and the set of ranks of the polynomials in T are denoted by $\mathrm{mvar}(T)$ and $\mathrm{rank}(T)$, respectively. A variable in $\mathbf{x}$ is called *algebraic* w.r.t. T if it belongs to $\mathrm{mvar}(T)$, otherwise it is said *free* w.r.t. T. For $v \in \mathrm{mvar}(T)$, denote by T_v the polynomial in T with main variable v. For $v \in \mathbf{x}$, we denote by $T_{<v}$ (resp. $T_{\geq v}$) the set of polynomials $t \in T$ such that $\mathrm{mvar}(t) < v$ (resp. $\mathrm{mvar}(t) \geq v$) holds. Let h_T be the product of the initials of the polynomials in T. We denote by $\mathrm{sat}(T)$ the *saturated ideal* of T defined as follows: if T is empty then $\mathrm{sat}(T)$ is the trivial ideal $\langle 0 \rangle$, otherwise it is the ideal $\langle T \rangle : h_T^\infty$. The *quasi-component* $W(T)$ of T is defined as $V(T) \setminus V(h_T)$. Denote $\overline{W(T)} = V(\mathrm{sat}(T))$ as the Zariski closure of $W(T)$. For $F \subset \mathbf{k}[\mathbf{x}]$, we write $Z(F, T) := V(F) \cap W(T)$.

Rank of a triangular set. Let $S \subset \mathbf{k}[\mathbf{x}]$ be a triangular set. We say that T has smaller rank than S and write $T \prec S$ if there exists $v \in \mathrm{mvar}(T)$ such that $\mathrm{rank}(T_{<v}) = \mathrm{rank}(S_{<v})$ holds and: (*i*) either $v \notin \mathrm{mvar}(S)$; (*ii*) or $v \in \mathrm{mvar}(S)$ and $T_v \prec S_v$. We write $T \sim S$ if $\mathrm{rank}(T) = \mathrm{rank}(S)$.

Iterated resultant. Let $p, q \in \mathbf{k}[\mathbf{x}]$. Assume q is nonconstant and let $v = \mathrm{mvar}(q)$. We define $\mathrm{res}(p, q, v)$ as follows: if the degree $\deg(p, v)$ of p in v is null, then $\mathrm{res}(p, q, v) = p$; otherwise $\mathrm{res}(p, q, v)$ is the resultant of p and q w.r.t. v. Let T be a triangular set of $\mathbf{k}[\mathbf{x}]$. We define $\mathrm{res}(p, T)$ by induction: if $T = \varnothing$, then $\mathrm{res}(p, T) = p$; otherwise let v be greatest variable appearing in T, then $\mathrm{res}(p, T) = \mathrm{res}(\mathrm{res}(p, T_v, v), T_{<v})$.

Regular chain. A triangular set $T \subset \mathbf{k}[\mathbf{x}]$ is a *regular chain* if: (*i*) either T is empty; (*ii*) or $T \setminus \{T_{\max}\}$ is a regular chain, where $T_{\max}$ is the polynomial in T with maximum rank, and the initial of $T_{\max}$ is regular w.r.t. $\mathrm{sat}(T \setminus \{T_{\max}\})$. The empty regular chain is simply denoted by $\varnothing$.

Triangular decomposition. Let $F \subset \mathbf{k}[\mathbf{x}]$ be finite. Let

$\mathfrak{T} := \{T_1, \ldots, T_e\}$ be a finite set of regular chains of $\mathbf{k}[\mathbf{x}]$. We call $\mathfrak{T}$ a *Kalkbrener triangular decomposition* of $V(F)$ if we have $V(F) = \cup_{i=1}^{e} \overline{W(T_i)}$. We call $\mathfrak{T}$ a *Lazard-Wu triangular decomposition* of $V(F)$ if we have $V(F) = \cup_{i=1}^{e} W(T_i)$.

PROPOSITION 1 ([1]). *Let p and T be respectively a polynomial and a regular chain of $\mathbf{k}[\mathbf{x}]$. Then, $prem(p, T) = 0$ holds if and only if $p \in sat(T)$ holds.*

PROPOSITION 2 ([17]). *Let T and T' be two regular chains of $\mathbf{k}[\mathbf{x}]$ such that $\sqrt{sat(T)} \subseteq \sqrt{sat(T')}$ and $\dim(sat(T)) = \dim(sat(T'))$ hold. Let $p \in \mathbf{k}[\mathbf{x}]$ such that p is regular w.r.t. $sat(T)$. Then p is also regular w.r.t. $sat(T')$.*

PROPOSITION 3 ([1]). *Let $p \in \mathbf{k}[\mathbf{x}]$ and $T \subset \mathbf{k}[\mathbf{x}]$ be a regular chain. Let $v = mvar(p)$ and $r = prem(p, T_{\geq v})$ such that $r \in \sqrt{sat(T_{<v})}$ holds. Then, we have $p \in \sqrt{sat(T)}$.*

COROLLARY 1. *Let T, T' be regular chains of $\mathbf{k}[x_1, \ldots, x_k]$, for $1 \leq k < n$. Let $p \in \mathbf{k}[\mathbf{x}]$ with $mvar(p) = x_{k+1}$ such that $init(p)$ is regular w.r.t. $sat(T)$ and $sat(T')$. We have: $\sqrt{sat(T)} \subseteq \sqrt{sat(T')} \implies \sqrt{sat(T \cup p)} \subseteq \sqrt{sat(T' \cup p)}$.*

PROPOSITION 4 ([3]). *Let $p \in \mathbf{k}[\mathbf{x}]$. Let $T \subset \mathbf{k}[\mathbf{x}]$ be a regular chain. Then the following statements are equivalent:*
- (i) *the polynomial p is regular w.r.t. $sat(T)$,*
- (ii) *for each prime $\mathfrak{p}$ associated with $sat(T)$, we have $p \notin \mathfrak{p}$,*
- (iii) *the iterated resultant $res(p, T)$ is not zero.*

COROLLARY 2. *Let $p \in \mathbf{k}[\mathbf{x}]$ and $T \subset \mathbf{k}[\mathbf{x}]$ be a regular chain. Let $v := mvar(p)$ and $r := res(p, T_{\geq v})$. We have:*
- (1) *the polynomial p is regular w.r.t. $sat(T)$ if and only if r is regular w.r.t. $sat(T_{<v})$;*
- (2) *if $v \notin mvar(T)$ and $init(p)$ is regular w.r.t. $sat(T)$, then p is regular w.r.t. $sat(T)$.*

3. REGULAR GCDS

Definition 1 was introduced in [17] as part of a formal framework for algorithms manipulating regular chains [8, 13, 6, 12, 24]. In the present paper, the ring $\mathbb{A}$ will always be of the form $\mathbf{k}[\mathbf{x}]/\sqrt{\mathrm{sat}(T)}$. Thus, a regular GCD of p, t in $\mathbb{A}[y]$ is also called a regular GCD of p, t modulo $\sqrt{\mathrm{sat}(T)}$.

DEFINITION 1. *Let $\mathbb{A}$ be a commutative ring with unity. Let $p, t, g \in \mathbb{A}[y]$ with $t \neq 0$ and $g \neq 0$. We say that $g \in \mathbb{A}[y]$ is a* regular GCD *of p, t if:*
- (R_1) *the leading coefficient of g in y is a regular element;*
- (R_2) *g belongs to the ideal generated by p and t in $\mathbb{A}[y]$;*
- (R_3) *if $\deg(g, y) > 0$, then g pseudo-divides both p and t, that is, $prem(p, g) = prem(t, g) = 0$.*

PROPOSITION 5. *For $1 \leq k \leq n$, let $T \subset \mathbf{k}[x_1, \ldots, x_{k-1}]$ be a regular chain, possibly empty. Let $p, t, g \in \mathbf{k}[x_1, \ldots, x_k]$ with main variable x_k. Assume $T \cup \{t\}$ is a regular chain and g is a regular GCD of p, t modulo $\sqrt{sat(T)}$. We have:*
- (i) *if $mdeg(g) = mdeg(t)$, then $\sqrt{sat(T \cup t)} = \sqrt{sat(T \cup g)}$ and $W(T \cup t) \subseteq Z(h_g, T \cup t) \cup W(T \cup g)$ both hold,*
- (ii) *if $mdeg(g) < mdeg(t)$, let $q = pquo(t, g)$, then $T \cup q$ is a regular chain and the following two relations hold:*
  - $(ii.a)$ $\sqrt{sat(T \cup t)} = \sqrt{sat(T \cup g)} \cap \sqrt{sat(T \cup q)}$,
  - $(ii.b)$ $W(T \cup t) \subseteq Z(h_g, T \cup t) \cup W(T \cup g) \cup W(T \cup q)$,
- (iii) $W(T \cup g) \subseteq V(p)$,
- (iv) $Z(p, T \cup t) \subseteq W(T \cup g) \cup Z(\{p, h_g\}, T \cup t)$.

PROOF. We first establish a relation between p, t and g. By definition of pseudo-division, there exist polynomials q, r and a nonnegtive integer e_0 such that

$$h_g^{e_0} t = qg + r \quad \text{and} \quad r \in \sqrt{\mathrm{sat}(T)} \qquad (1)$$

both hold. Hence, there exists an integer $e_1 \geq 0$ such that:

$$(h_T)^{e_1} (h_g^{e_0} t - qg)^{e_1} \in \langle T \rangle \qquad (2)$$

holds, which implies: $t \in \sqrt{\mathrm{sat}(T \cup g)}$. We first prove (i). Since $\mathrm{mdeg}(t) = \mathrm{mdeg}(g)$ holds, we have $q \in \mathbf{k}[x_1, \ldots, x_{k-1}]$, and thus $h_g^{e_0} h_t = q h_g$ holds. Since h_t and h_g are regular modulo $\mathrm{sat}(T)$, the same property holds for q. With (2), we obtain $g \in \sqrt{\mathrm{sat}(T \cup t)}$. Therefore $\sqrt{\mathrm{sat}(T \cup t)} = \sqrt{\mathrm{sat}(T \cup g)}$. The inclusion relation in (i) follows from (1).

We prove (ii). Assume $\mathrm{mdeg}(t) > \mathrm{mdeg}(g)$. With (1) and (2), this hypothesis implies that $T \cup q$ is a regular chain and $t \in \sqrt{\mathrm{sat}(T \cup q)}$ holds. Since $t \in \sqrt{\mathrm{sat}(T \cup g)}$ also holds, $\sqrt{\mathrm{sat}(T \cup t)}$ is contained in $\sqrt{\mathrm{sat}(T \cup g)} \cap \sqrt{\mathrm{sat}(T \cup q)}$. Conversely, for any $f \in \sqrt{\mathrm{sat}(T \cup g)} \cap \sqrt{\mathrm{sat}(T \cup q)}$, there exists an integer $e_2 \geq 0$ and $a \in \mathbf{k}[\mathbf{x}]$ such that $(h_g h_q)^{e_2} f^{e_2} - aqg \in \mathrm{sat}(T)$ holds. With (1) we deduce that $f \in \sqrt{\mathrm{sat}(T \cup t)}$ holds and so does $(ii.a)$. With (1), we have $(ii.b)$ holds.

We prove (iii) and (iv). Definition 1 implies: $\mathrm{prem}(p, g) \in \sqrt{\mathrm{sat}(T)}$. Thus $p \in \sqrt{\mathrm{sat}(T \cup g)}$ holds, that is, $\overline{W(T \cup g)} \subseteq V(p)$, which implies (iii). Moreover, since $g \in \langle p, t, \sqrt{\mathrm{sat}(T)} \rangle$, we have $Z(p, T \cup t) \subseteq V(g)$, so we deduce (iv). □

Let p, t be two polynomials of $\mathbf{k}[x_1, \ldots, x_k]$, for $k \geq 1$. Let $m = \deg(p, x_k)$, $n = \mathrm{mdeg}(t, x_k)$. Assume that $m, n \geq 1$. Let $\lambda = \min(m, n)$. Let T be a regular chain of $\mathbf{k}[x_1, \ldots, x_{k-1}]$. Let $\mathbb{B} = \mathbf{k}[x_1, \ldots, x_{k-1}]$ and $\mathbb{A} = \mathbb{B}/\sqrt{\mathrm{sat}(T)}$.

Let $S_0, \ldots, S_{\lambda-1}$ be the subresulant polynomials [16, 9] of p and t w.r.t. x_k in $\mathbb{B}[x_k]$. Let $s_i = \mathrm{coeff}(S_i, x_k^i)$ be the principle subresultant coefficient of S_i, for $0 \leq i \leq \lambda - 1$. If $m \geq n$, we define $S_\lambda = t$, $S_{\lambda+1} = p$, $s_\lambda = \mathrm{init}(t)$ and $s_{\lambda+1} = \mathrm{init}(p)$. If $m < n$, we define $S_\lambda = p$, $S_{\lambda+1} = t$, $s_\lambda = \mathrm{init}(p)$ and $s_{\lambda+1} = \mathrm{init}(t)$.

The following theorem provides sufficient conditions for S_j (with $1 \leq j \leq \lambda+1$) to be a regular GCD of p and t in $\mathbb{A}[x_k]$.

THEOREM 1. *Let j be an integer, with $1 \leq j \leq \lambda + 1$, such that s_j is a regular element of $\mathbb{A}$ and such that for any $0 \leq i < j$, we have $s_i = 0$ in $\mathbb{A}$. Then S_j is a regular GCD of p and t in $\mathbb{A}[x_k]$.*

PROOF. By Definition 1, it suffices to prove that both $\mathrm{prem}(p, S_j, x_k) = 0$ and $\mathrm{prem}(t, S_j, x_k) = 0$ hold in $\mathbb{A}$. By symmetry we only prove the former equality.

Let $\mathfrak{p}$ be any prime ideal associated with $\mathrm{sat}(T)$. Define $\mathbb{D} = \mathbf{k}[x_1, \ldots, x_{k-1}]/\mathfrak{p}$ and let $\mathbb{L}$ be the fraction field of the integral domain $\mathbb{D}$. Let ϕ be the homomorphism from $\mathbb{B}$ to $\mathbb{L}$. By Theorem 4 in the Appendix of [4], we know that $\phi(S_j)$ is a GCD of $\phi(p)$ and $\phi(t)$ in $\mathbb{L}[x_k]$. Therefore there exists a polynomial q of $\mathbb{L}[x_k]$ such that $p = qS_j$ in $\mathbb{L}[x_k]$, which implies that there exists a nonzero element a of $\mathbb{D}$ and a polynomial q' of $\mathbb{D}[x_k]$ such that $ap = q'S_j$ in $\mathbb{D}[x_k]$. Therefore $\mathrm{prem}(ap, S_j) = 0$ in $\mathbb{D}[x_k]$, which implies that $\mathrm{prem}(p, S_j) = 0$ in $\mathbb{D}[x_k]$. Hence $\mathrm{prem}(p, S_j)$ belongs to $\mathfrak{p}$ and thus to $\sqrt{\mathrm{sat}(T)}$. So $\mathrm{prem}(p, S_j, x_k) = 0$ in $\mathbb{A}$. □

4. THE INCREMENTAL ALGORITHM

In this section, we present an algorithm to compute Lazard-Wu triangular decompositions in an incremental manner.

We recall the concepts of a *process* and a *regular (delayed) split*, which were introduced as Definitions 9 and 11 in [17]. To serve our purpose, we modify the definitions as below.

Algorithm 1: Intersect(p, T, R)

```
if prem(p, T) = 0 then return {T}
if p ∈ k then return { }
r := p; P := {r}; S := { }
while mvar(r) ∈ mvar(T) do
    v := mvar(r); src := SubresultantChain(r, T_v, v, R)
    S := S ∪ {src}; r := resultant(src)
    if r = 0 then break
    if r ∈ k then return { }
    P := P ∪ {r}
𝔗 := {∅}; 𝔗' := { }; i := 1
while i ≤ n do
    for C ∈ 𝔗 do
        if x_i ∉ mvar(P) and x_i ∉ mvar(T) then
            𝔗' := 𝔗' ∪ CleanChain(C, T, x_{i+1}, R)
        else if x_i ∉ mvar(P) then
            𝔗' := 𝔗' ∪ CleanChain(C ∪ T_{x_i}, T, x_{i+1}, R)
        else if x_i ∉ mvar(T) then
            for D ∈ IntersectFree(P_{x_i}, x_i, C, R) do
                𝔗' := 𝔗' ∪ CleanChain(D, T, x_{i+1}, R)
        else
            for D ∈ IntersectAlgebraic(P_{x_i}, T, x_i, S_{x_i}, C, R)
            do
                𝔗' := 𝔗' ∪ CleanChain(D, T, x_{i+1}, R)
    𝔗 := 𝔗'; 𝔗' := { }; i := i + 1
return 𝔗
```

Algorithm 2: RegularGcd(p, q, v, S, T, R)

```
𝔗 := {(T, 1)}
while 𝔗 ≠ ∅ do
    let (C, i) ∈ 𝔗; 𝔗 := 𝔗 \ {(C, i)}
    for [f, D] ∈ Regularize(s_i, C, R) do
        if dim D < dim C then  output [0, D]
        else if f = 0 then  𝔗 := 𝔗 ∪ {(D, i + 1)}
        else  output [S_i, D]
```

Definition 2. *A* process *of* $\mathbf{k}[\mathbf{x}]$ *is a pair* (p, T)*, where* $p \in \mathbf{k}[\mathbf{x}]$ *is a polynomial and* $T \subset \mathbf{k}[\mathbf{x}]$ *is a regular chain. The process* $(0, T)$ *is also written as* T *for short. Given two processes* (p, T) *and* (p', T')*, let* v *and* v' *be respectively the greatest variable appearing in* (p, T) *and* (p', T')*. We say* $(p, T) \prec (p', T')$ *if: (i) either* $v < v'$*; (ii) or* $v = v'$ *and* $\dim T < \dim T'$*; (iii) or* $v = v'$*,* $\dim T = \dim T'$ *and* $T \prec T'$*; (iv) or* $v = v'$*,* $\dim T = \dim T'$*,* $T \sim T'$ *and* $p \prec p'$*. We write* $(p, T) \sim (p', T')$ *if neither* $(p, T) \prec (p', T')$ *nor* $(p', T') \prec (p, T)$ *hold. Clearly any sequence of processes which is strictly decreasing w.r.t.* $\prec$ *is finite.*

Definition 3. *Let* T_i*,* $1 \le i \le e$*, be regular chains of* $\mathbf{k}[\mathbf{x}]$*. Let* $p \in \mathbf{k}[\mathbf{x}]$*. We call* $T_1, \ldots, T_e$ a regular split *of* (p, T) *whenever we have*

(L_1) $\sqrt{sat(T)} \subseteq \sqrt{sat(T_i)}$

(L_2) $W(T_i) \subseteq V(p)$ *(or equivalently* $p \in \sqrt{sat(T_i)}$*)*

Algorithm 3: IntersectFree(p, x_i, C, R)

```
for [f, D] ∈ Regularize(init(p), C, R) do
    if f = 0 then  output Intersect(tail(p), D, R)
    else
        output D ∪ p
        for E ∈ Intersect(init(p), D, R) do
            output Intersect(tail(p), E, R)
```

Algorithm 4: IntersectAlgebraic(p, T, x_i, S, C, R)

```
for [g, D] ∈ RegularGcd(p, T_{x_i}, x_i, S, C, R) do
    if dim D < dim C then
        for E ∈ CleanChain(D, T, x_i, R) do
            output IntersectAlgebraic(p, T, x_i, S, E, R)
    else
        output D ∪ g
        for E ∈ Intersect(init(g), D, R) do
            for F ∈ CleanChain(E, T, x_i, R) do
                output IntersectAlgebraic(p, T, x_i, S, F, R)
```

Algorithm 5: Regularize(p, T, R)

```
if p ∈ k or T = ∅ then return [p, T]
v := mvar(p)
if v ∉ mvar(T) then
    for [f, C] ∈ Regularize(init(p), T, R) do
        if f = 0 then  output Regularize(tail(p), C, R)
        else  output [p, C]
else
    src := SubresultantChain(p, T_v, v, R);
    r := resultant(src)
    for [f, C] ∈ Regularize(r, T_{<v}, R) do
        if dim C < dim T_{<v} then
            for D ∈ Extend(C, T, v, R) do
                output Regularize(p, D, R)
        else if f ≠ 0 then  output [p, C ∪ T_{≥v}]
        else
            for [g, D] ∈ RegularGcd(p, T_v, v, src, C, R) do
                if dim D < dim C then
                    for E ∈ Extend(D, T, v, R) do
                        output Regularize(p, E, R)
                else
                    if mdeg(g) = mdeg(T_v) then output
                    [0, D ∪ T_{≥v}]; next
                    output [0, D ∪ g ∪ T_{>v}]
                    q := pquo(T_v, g)
                    output Regularize(p, D ∪ q ∪ T_{>v}, R)
                    for E ∈ Intersect(h_g, D, R) do
                        for F ∈ Extend(E, T, v, R) do
                            output Regularize(p, F, R)
```

Algorithm 6: $\mathsf{Extend}(C, T, x_i, R)$

if $T_{\geq x_i} = \varnothing$ **then** return C;
let $p \in T$ with greatest main variable; $T' := T \setminus \{p\}$;
for $D \in \mathsf{Extend}(C, T', x_i, R)$ **do**
  for $[f, E] \in \mathsf{Regularize}(\mathit{init}(p), D)$ **do**
    if $f \neq 0$ **then** output $E \cup p$;

Algorithm 7: $\mathsf{CleanChain}(C, T, x_i, R)$

if $x_i \notin \mathrm{mvar}(T)$ **or** $\dim C = \dim T_{<x_i}$ **then** return C
for $[f, D] \in \mathsf{Regularize}(\mathit{init}(T_{x_i}), C, R)$ **do**
  if $f \neq 0$ **then** output D

(L_3) $V(p) \cap W(T) \subseteq \cup_{i=1}^{e} W(T_i)$
We write as $(p, T) \longrightarrow T_1, \ldots, T_e$. *Observe that the above three conditions are equivalent to the following relation.*

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

Geometrically, this means that we may compute a little more than $V(p) \cap W(T)$*; however,* $W(T_1) \cup \cdots \cup W(T_e)$ *is a "sharp" approximation of the intersection of* $V(p)$ *and* $W(T)$.

Next we list the specifications of our triangular decomposition algorithm and its subroutines. We denote by R the polynomial ring $\mathbf{k}[\mathbf{x}]$, where $\mathbf{x} = x_1 < \cdots < x_n$.

$\mathsf{Triangularize}(F, R)$
- **Input:** F, a finite set of polynomials of R
- **Output:** A Lazard-Wu triangular decomposition of $V(F)$.

$\mathsf{Intersect}(p, T, R)$
- **Input:** p, a polynomial of R; T, a regular chain of R
- **Output:** a set of regular chains $\{T_1, \ldots, T_e\}$ such chat $(p, T) \longrightarrow T_1, \ldots, T_e$.

$\mathsf{Regularize}(p, T, R)$
- **Input:** p, a polynomial of R; T, a regular chain of R.
- **Output:** a set of pairs $\{[p_1, T_1], \ldots, [p_e, T_e]\}$ such that for each $i, 1 \leq i \leq e$: (1) T_i is a regular chain; (2) $p = p_i \bmod \sqrt{\mathrm{sat}(T_i)}$; (3) if $p_i = 0$, then $p_i \in \sqrt{\mathrm{sat}(T_i)}$ otherwise p_i is regular modulo $\sqrt{\mathrm{sat}(T_i)}$; moreover we have $T \longrightarrow T_1, \ldots, T_e$.

$\mathsf{SubresultantChain}(p, q, v, R)$
- **Input:** v, a variable of $\{x_1, \ldots, x_n\}$; p and q, polynomials of R, whose main variables are both v.
- **Output:** a list of polynomials $(S_0, \ldots, S_\lambda)$, where $\lambda = \min(\mathrm{mdeg}(p), \mathrm{mdeg}(q))$, such that S_i is the i-th subresultant of p and q w.r.t. v.

$\mathsf{RegularGcd}(p, q, v, S, T, R)$
- **Input:** v, a variable of $\{x_1, \ldots, x_n\}$,
  - T, a regular chain of R such that $\mathrm{mvar}(T) < v$,
  - p and q, polynomials of R with the same main variable v such that: $\mathrm{init}(q)$ is regular modulo $\sqrt{\mathrm{sat}(T)}$; $\mathrm{res}(p, q, v)$ belongs to $\sqrt{\mathrm{sat}(T)}$,
  - S, the subresultant chain of p and q w.r.t. v.
- **Output:** a set of pairs $\{[g_1, T_1], \ldots, [g_e, T_e]\}$ such that $T \longrightarrow T_1, \ldots, T_e$ and for each T_i: if $\dim T = \dim T_i$, then g_i is a regular GCD of p and q modulo $\sqrt{\mathrm{sat}(T_i)}$; otherwise $g_i = 0$, which means undefined.

Algorithm 8: $\mathsf{Triangularize}(F, R)$

if $F = \{\ \}$ **then** return $\{\varnothing\}$
Choose a polynomial $p \in F$ with maximal rank
for $T \in \mathsf{Triangularize}(F \setminus \{p\}, R)$ **do**
  output $\mathsf{Intersect}(p, T, R)$

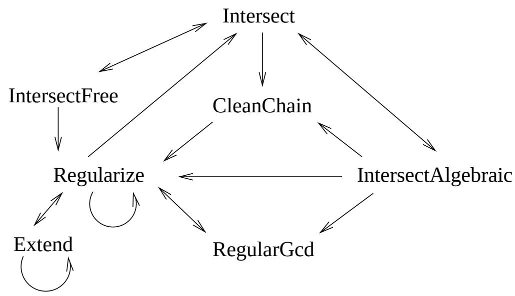

Figure 1: Flow graph of the Algorithms

$\mathsf{IntersectFree}(p, x_i, C, R)$
- **Input:** x_i, a variable of $\mathbf{x}$; p, a polynomial of R with main variable x_i; C, a regular chain of $\mathbf{k}[x_1, \ldots, x_{i-1}]$.
- **Output:** a set of regular chains $\{T_1, \ldots, T_e\}$ such that $(p, C) \longrightarrow (T_1, \ldots, T_e)$.

$\mathsf{IntersectAlgebraic}(p, T, x_i, S, C, R)$
- **Input:** p, a polynomial of R with main variable x_i,
  - T, a regular chain of R, where $x_i \in \mathrm{mvar}(T)$,
  - S, the subresultant chain of p and T_{x_i} w.r.t. x_i,
  - C, a regular chain of $\mathbf{k}[x_1, \ldots, x_{i-1}]$, such that: $\mathrm{init}(T_{x_i})$ is regular modulo $\sqrt{\mathrm{sat}(C)}$; the resultant of p and T_{x_i}, which is S_0, belongs to $\sqrt{\mathrm{sat}(C)}$.
- **Output:** a set of regular chains $T_1, \ldots, T_e$ such that $(p, C \cup T_{x_i}) \longrightarrow T_1, \ldots, T_e$.

$\mathsf{CleanChain}(C, T, x_i, R)$
- **Input:** T, a regular chain of R; C, a regular chain of $\mathbf{k}[x_1, \ldots, x_{i-1}]$ such that $\sqrt{\mathrm{sat}(T_{<x_i})} \subseteq \sqrt{\mathrm{sat}(C)}$.
- **Output:** if $x_i \notin \mathrm{mvar}(T)$, return C; otherwise return a set of regular chains $\{T_1, \ldots, T_e\}$ such that $\mathrm{init}(T_{x_i})$ is regular modulo each $\mathrm{sat}(T_j)$, $\sqrt{\mathrm{sat}(C)} \subseteq \sqrt{\mathrm{sat}(T_j)}$ and $W(C) \setminus V(\mathrm{init}(T_{x_i})) \subseteq \cup_{j=1}^{e} W(T_j)$.

$\mathsf{Extend}(C, T, x_i, R)$
- **Input:** C, is a regular chain of $\mathbf{k}[x_1, \ldots, x_{i-1}]$. T, a regular chain of R such that $\sqrt{\mathrm{sat}(T_{<x_i})} \subseteq \sqrt{\mathrm{sat}(C)}$.
- **Output:** Regular chains $T_1, \ldots, T_e$ of R such that $W(C \cup T_{\geq x_i}) \subseteq \cup_{j=1}^{e} W(T_j)$ and $\sqrt{\mathrm{sat}(T)} \subseteq \sqrt{\mathrm{sat}(T_j)}$.

Algorithm $\mathsf{SubresultantChain}$ is standard, see [9]. The algorithm $\mathsf{Triangularize}$ is a *principle algorithm* which was first presented in [17]. We use the following conventions in our pseudo-code: the keyword **return** yields a result and terminates the current function call while the keyword **output** yields a result and keeps executing the current function call.

5. PROOF OF THE ALGORITHMS

THEOREM 2. *All the algorithms in Fig. 1 terminate.*

PROOF. The key observation is that the flow graph of Fig. 1 can be transformed into an equivalent flow graph satisfying the following properties: (1) the algorithms $\mathsf{Intersect}$ and $\mathsf{Regularize}$ only call each other or themselves; (2) all the other

	sys	Input size				Output size				
		#v	#e	deg	dim	GL	GS	GD	TL	TK
1	4corps-1parameter-homog	4	3	8	1	-	-	21863	-	30738
2	8-3-config-Li	12	7	2	7	67965	-	72698	7538	1384
3	Alonso-Li	7	4	4	3	1270	-	614	2050	374
4	Bezier	5	3	6	2	-	-	32054	-	114109
5	Cheaters-homotopy-1	7	3	7	4	26387452	-	17297	-	285
7	childDraw-2	10	10	2	0	938846	-	157765	-	-
8	Cinquin-Demongeot-3-3	4	3	4	1	1652062	-	680	2065	895
9	Cinquin-Demongeot-3-4	4	3	5	1	-	-	690	-	2322
10	collins-jsc02	5	4	3	1	-	-	28720	2770	1290
11	f-744	12	12	3	1	102082	-	83559	4509	4510
12	Haas5	4	2	10	2	-	-	28	-	548
14	Lichtblau	3	2	11	1	6600095	-	224647	110332	5243
16	Liu-Lorenz	5	4	2	1	47688	123965	712	2339	938
17	Mehta2	11	8	3	3	-	-	1374931	5347	5097
18	Mehta3	13	10	3	3	-	-	-	25951	25537
19	Mehta4	15	12	3	3	-	-	-	71675	71239
21	p3p-isosceles	7	3	3	4	56701	-	1453	9253	840
22	p3p	8	3	3	5	160567	-	1768	-	1712
23	Pavelle	8	4	2	4	17990	-	1552	3351	1086
24	Solotareff-4b	5	4	3	1	2903124	-	14810	2438	872
25	Wang93	5	4	3	1	2772	56383	1377	1016	391
26	Xia	6	3	4	3	63083	2711	672	1647	441
27	xy-5-7-2	6	3	3	3	12750	-	599	-	3267

Table 1 The input and output sizes of systems

algorithms only call either Intersect or Regularize. Therefore, it suffices to show that Intersect and Regularize terminate.

Note that the input of both functions is a process, say (p, T). One can check that, while executing a call with (p, T) as input, any subsequent call to either functions Intersect or Regularize will take a process (p', T') as input such that $(p', T') \prec (p, T)$ holds. Since a descending chain of processes is necessarily finite, both algorithms terminate. □

Since all algorithms terminate, and following the flow graph of Fig. 1, each call to one of our algorithms unfold to a finite dynamic acyclic graph (DAG) where each vertex is a call to one of our algorithms. Therefore, proving the correctness of these algorithms reduces to prove the following two points.

- *Base:* each algorithm call, which makes no subsequent calls to another algorithm or to itself, is correct.
- *Induction:* each algorithm call, which makes subsequent calls to another algorithm or to itself, is correct, as soon as all subsequent calls are themselves correct.

For all algorithms in Fig. 1, proving the base cases is straightforward. Hence we focus on the induction steps.

PROPOSITION 6. IntersectFree *satisfies its specification.*

PROOF. We have the following two key observations:

- $C \longrightarrow D_1, \ldots, D_s$, where D_i are the regular chains in the output of Regularize.
- $V(p) \cap W(D) = W(D, p) \cup V(\mathrm{init}(p), \mathrm{tail}(p)) \cap W(D)$.

Then it is not hard to conclude that $(p, C) \longrightarrow T_1, \ldots, T_e$. □

PROPOSITION 7. IntersectAlgebraic *is correct.*

PROOF. We need to prove: $(p, C \cup T_{x_i}) \longrightarrow T_1, \ldots, T_e$. Let us prove (L_1) now, that is, for each regular chain T_j in the output, we have $\sqrt{\mathrm{sat}(C \cup T_{x_i})} \subseteq \sqrt{\mathrm{sat}(T_j)}$. First by the specifications of the called functions, we have $\sqrt{\mathrm{sat}(C)} \subseteq \sqrt{\mathrm{sat}(D)} \subseteq \sqrt{\mathrm{sat}(E)}$, thus, $\sqrt{\mathrm{sat}(C \cup T_{x_i})} \subseteq \sqrt{\mathrm{sat}(E \cup T_{x_i})}$ by Corollary 1, since $\mathrm{init}(T_{x_i})$ is regular modulo both $\mathrm{sat}(C)$ and $\mathrm{sat}(E)$. Secondly, since g is a regular GCD of p and T_{x_i} modulo $\sqrt{\mathrm{sat}(D)}$, we have $\sqrt{\mathrm{sat}(C \cup T_{x_i})} \subseteq \sqrt{\mathrm{sat}(D \cup g)}$ by Corollaries 1 and Proposition 5.

Next we prove (L_2). It suffices to prove that $W(D \cup g) \subseteq V(p)$ holds. Since g is a regular GCD of p and T_{x_i} modulo $\sqrt{\mathrm{sat}(D)}$, the conclusion follows from (iii) in Proposition 5.

Finally we prove (L_3), that is $Z(p, C \cup T_{x_i}) \subseteq \bigcup_{j=1}^{e} W(T_j)$. Let $D_1, \ldots, D_s$ be the regular chains returned from Algorithm RegularGcd. We have $C \longrightarrow D_1, \ldots, D_s$, which implies $Z(p, C \cup T_{x_i}) \subseteq \cup_{j=1}^{e} Z(p, D_j \cup T_{x_i})$. Next since g is a regular GCD of p and T_{x_i} modulo $\sqrt{\mathrm{sat}(D_j)}$, the conclusion follows from point (iv) of Proposition 5. □

PROPOSITION 8. Intersect *satisfies its specification.*

PROOF. The first while loop can be seen as a projection process. We claim that it produces a nonempty triangular set P such that $V(p) \cap W(T) = V(P) \cap W(T)$. The claim holds before staring the while loop. For each iteration, let P' be the set of polynomials obtained at the previous iteration. We then compute a polynomial r, which is the resultant of a polynomial in P' and a polynomial in T. So $r \in \langle P', T \rangle$. By induction, we have $\langle p, T \rangle = \langle P, T \rangle$. So the claim holds.

Next, we claim that the elements in $\mathfrak{T}$ satisfy the following invariants: at the beginning of the i-th iteration of the second while loop, we have

(1) each $C \in \mathfrak{T}$ is a regular chain; if T_{x_i} exists, then $\mathrm{init}(T_{x_i})$ is regular modulo $\mathrm{sat}(C)$,
(2) for each $C \in \mathfrak{T}$, we have $\sqrt{\mathrm{sat}(T_{<x_i})} \subseteq \sqrt{\mathrm{sat}(C)}$,
(3) for each $C \in \mathfrak{T}$, we have $\overline{W(C)} \subseteq V(P_{<x_i})$,
(4) $V(p) \cap W(T) \subseteq \bigcup_{C \in \mathfrak{T}} Z(P_{\geq x_i}, C \cup T_{\geq x_i})$.

When $i = n+1$, we then have $\sqrt{\mathrm{sat}(T)} \subseteq \sqrt{\mathrm{sat}(C)}$, $W(C) \subseteq V(P) \subseteq V(p)$ for each $C \in \mathfrak{T}$ and $V(p) \cap W(T) \subseteq \cup_{C \in \mathfrak{T}} W(C)$. So $(L_1), (L_2), (L_3)$ of Definition 3 all hold. This concludes the correctness of the algorithm.

Now we prove the above claims (1), (2), (3), (4) by induction. The claims clearly hold when $i = 1$ since $C = \varnothing$ and $V(p) \cap W(T) = V(P) \cap W(T)$. Now assume that the loop invariants hold at the beginning of the i-th iteration. We need to prove that it still holds at the beginning of the $(i+1)$-th iteration. Let $C \in \mathfrak{T}$ be an element picked up at the beginning of i-th iteration and let L be the set of the new elements of $\mathfrak{T}'$ generated from C.

Then for any $C' \in L$, claim (1) clearly holds by specification of CleanChain. Next we prove (2).

- if $x_i \notin \mathrm{mvar}(T)$, then $T_{<x_{i+1}} = T_{<x_i}$. By induction

and specifications of called functions, we have

$$\sqrt{\operatorname{sat}(T_{<x_{i+1}})} \subseteq \sqrt{\operatorname{sat}(C)} \subseteq \sqrt{\operatorname{sat}(C')}.$$

- if $x_i \in \operatorname{mvar}(T)$, by induction we have $\sqrt{\operatorname{sat}(T_{<x_i})} \subseteq \sqrt{\operatorname{sat}(C)}$ and $\operatorname{init}(T_{x_i})$ is regular modulo both $\operatorname{sat}(C)$ and $\operatorname{sat}(T_{<x_i})$. By Corollary 1 we have

$$\sqrt{\operatorname{sat}(T_{<x_{i+1}})} \subseteq \sqrt{\operatorname{sat}(C \cup T_{x_i})} \subseteq \sqrt{\operatorname{sat}(C')}.$$

Therefore (2) holds. Next we prove claim (3). By induction and the specifications of called functions, we have $\overline{W(C')} \subseteq \overline{W(C \cup T_{x_i})} \subseteq V(P_{<x_i})$. Secondly, we have $\overline{W(C')} \subseteq V(P_{x_i})$. Therefore $\overline{W(C')} \subseteq V(P_{<x_{i+1}})$, that is (3) holds. Finally, since $V(P_{x_i}) \cap W(C \cup T_{x_i}) \setminus V(\operatorname{init}(T_{x_{i+1}})) \subseteq \cup_{C' \in L} W(C')$, we have $Z(P_{\geq x_i}, C \cup T_{\geq x_i}) \subseteq \cup_{C' \in L} Z(P_{\geq x_{i+1}}, C' \cup T_{\geq x_{i+1}})$, which implies that (4) holds. This completes the proof. □

PROPOSITION 9. Regularize *satisfies its specification.*

PROOF. If $v \notin \operatorname{mvar}(T)$, the conclusion follows directly from point (2) of Corollary 2. From now on, assume $v \in \operatorname{mvar}(T)$. Let $\mathfrak{L}$ be the set of pairs $[p', T']$ in the output. We aim to prove the following facts

(1) each T' is a regular chain,
(2) if $p' = 0$, then p is zero modulo $\sqrt{\operatorname{sat}(T')}$, otherwise p is regular modulo $\operatorname{sat}(T)$,
(3) we have $\sqrt{\operatorname{sat}(T)} \subseteq \sqrt{\operatorname{sat}(T')}$,
(4) we have $W(T) \subseteq \cup_{T' \in \mathfrak{L}} W(T')$.

Statement (1) is due to Proposition 2. Next we prove (2). First, when there are recursive calls, the conclusion is obvious. Let $[f, C]$ be a pair in the output of Regularize$(r, T_{<v}, R)$. If $f \neq 0$, the conclusion follows directly from point (1) of Corollary 2. Otherwise, let $[g, D]$ be a pair in the output of the algorithm RegularGcd(p, T_v, v, src, C, R). If $\operatorname{mdeg}(g) = \operatorname{mdeg}(T_v)$, then by the algorithm of RegularGcd, $g = T_v$. Therefore we have $\operatorname{prem}(p, T_v) \in \sqrt{\operatorname{sat}(C)}$, which implies that $p \in \sqrt{\operatorname{sat}(C \cup T_{\geq v})}$ by Proposition 3.

Next we prove (3). Whenever Extend is called, (3) holds immediately. Otherwise, let $[f, C]$ be a pair returned by Regularize$(r, T_{<v}, R)$. When $f \neq 0$, since $\sqrt{\operatorname{sat}(T_{<v})} \subseteq \sqrt{\operatorname{sat}(C)}$ holds, we conclude $\sqrt{\operatorname{sat}(T)} \subseteq \sqrt{\operatorname{sat}(C \cup T_{\geq v})}$ by Corollary 1. Let $[g, D] \in$ RegularGcd(p, T_v, v, src, C, R). Corollary 1 and point (ii) of Proposition 5 imply that $\sqrt{\operatorname{sat}(T)} \subseteq \sqrt{\operatorname{sat}(D \cup T_{\geq v})}$, $\sqrt{\operatorname{sat}(T)} \subseteq \sqrt{\operatorname{sat}(D \cup g \cup T_{>v})}$ together with $\sqrt{\operatorname{sat}(T)} \subseteq \sqrt{\operatorname{sat}(D \cup q \cup T_{>v})}$ hold. Hence (3) holds.

Finally by point $(ii.b)$ of Proposition 5, we have $W(D \cup T_v) \subseteq Z(h_g, D \cup T_v) \cup W(D \cup g) \cup W(D \cup q)$. So (4) holds. □

PROPOSITION 10. Extend *satisfies its specification.*

PROOF. It clearly holds when $T_{\geq x_i} = \varnothing$, which is the base case. By induction and the specification of Regularize, we know that $\sqrt{\operatorname{sat}(T')} \subseteq \sqrt{\operatorname{sat}(E)}$. Since $\operatorname{init}(p)$ is regular modulo both $\operatorname{sat}(T')$ and $\operatorname{sat}(E)$, by Corollary 1, we have $\sqrt{\operatorname{sat}(T)} \subseteq \sqrt{\operatorname{sat}(E \cup p)}$. On the other hand, we have $W(C \cup T'_{\geq x_i}) \subseteq \cup W(D)$ and $W(D) \setminus V(h_p) \subseteq \cup\, W(E)$. Therefore $W(C \cup T_{\geq x_i}) \subseteq \cup_{j=1}^{e} W(T_j)$, where $T_1, \ldots, T_e$ are the regular chains in the output. □

PROPOSITION 11. CleanChain *satisfies its specification.*

PROOF. It follows directly from Proposition 2. □

PROPOSITION 12. RegularGcd *satisfies its specification.*

PROOF. Let $[g_i, T_i]$, $i = 1, \ldots, e$, be the output. First from the specification of Regularize, we have $T \longrightarrow T_1, \ldots, T_e$. When $\dim T_i = \dim T$, by Proposition 2 and Theorem 1, g_i is a regular GCD of p and q modulo $\sqrt{\operatorname{sat}(T)}$. □

6. KALKBRENER DECOMPOSITION

In this section, we adapt the Algorithm Triangularize (Algorithm 8), in order to compute efficiently a Kalkbrener triangular decomposition. The basic technique we rely on follows from Krull's principle ideal theorem.

THEOREM 3. *Let $F \subset \mathbf{k}[\mathbf{x}]$ be finite, with cardinality $\#(F)$. Assume F generates a proper ideal of $\mathbf{k}[\mathbf{x}]$. Then, for any minimal prime ideal $\mathfrak{p}$ associated with $\langle F \rangle$, the height of $\mathfrak{p}$ is less than or equal to $\#(F)$.*

COROLLARY 3. *Let $\mathfrak{T}$ be a Kalkbrener triangular decomposition of $V(F)$. Let T be a regular chain of $\mathfrak{T}$, the height of which is greater than $\#(F)$. Then $\mathfrak{T} \setminus \{T\}$ is also a Kalkbrener triangular decomposition of $V(F)$.*

Based on this corollary, we prune the decomposition tree generated during the computation of a Lazard-Wu triangular decomposition and remove the computation branches in which the height of every generated regular chain is greater than the number of polynomials in F.

Next we explain how to implement this tree pruning technique to the algorithms of Section 4. Inside Triangularize, define $A = \#(F)$ and pass it to every call to Intersect in order to signal Intersect to output only regular chains with height no greater than A. Next, in the second while loop of Intersect, for the i-th iteration, we pass the height $A - \#(T_{\geq x_{i+1}})$ to CleanChain, IntersectFree and IntersectAlgebraic.

In IntersectFree, we pass its input height A to every function call. Besides, Lines 5 to 6 are executed only if the height of D is strictly less than A, since otherwise we would obtain regular chains of height greater than A. In other algorithms, we apply similar strategies as in Intersect and IntersectFree.

7. EXPERIMENTATION

Part of the algorithms presented in this paper are implemented in MAPLE14 while all of them are present in the current development version of MAPLE. Tables 1 and 2 report on our comparison between Triangularize and other MAPLE solvers. The notations used in these tables are defined below.

Notation for Triangularize. We denote by TK and TL the latest implementation of Triangularize for computing, respectively, Kalkbrener and Lazard-Wu decompositions, in the current version of MAPLE. Denote by TK14 and TL14 the corresponding implementation in MAPLE14. Denote by TK13, TL13 the implementation based on the algorithm of [17] in MAPLE13. Finally, STK and STL are versions of TK and TL, enforcing all computed regular chains to be squarefree, by means of the algorithms in the Appendix of [4].

Notation for the other solvers. Denote by GL, GS, GD, respectively the function Groebner:-Basis (plex order), Groebner:-Solve, Groebner:-Basis (tdeg order) in current beta version of MAPLE. Denote by WS the function wsolve of the package `Wsolve` [20], which decomposes a variety as a union of quasi-components of Wu Characteristic Sets.

The tests were launched on a machine with Intel Core 2 Quad CPU (2.40GHz) and 3.0Gb total memory. The time-out is set as 3600 seconds. The memory usage is limited to

sys	Triangularize								Triangularize versus other solvers				
	TK13	TK14	TK	TL13	TL14	TL	STK	STL	GL	GS	WS	TL	TK
1	-	241.7	36.9	-	-	-	62.8	-	-	-	-	-	36.9
2	8.7	5.3	5.9	29.7	24.1	25.8	6.0	26.6	108.7	-	27.8	25.8	5.9
3	0.3	0.3	0.4	14.0	2.4	2.1	0.4	2.2	3.4	-	7.9	2.1	0.4
4	-	-	88.2	-	-	-	-	-	-	-	-	-	88.2
5	0.4	0.5	0.7	-	-	-	451.8	-	2609.5	-	-	-	0.7
7	-	-	-	-	-	-	1326.8	1437.1	19.3	-	-	-	-
8	3.2	0.7	0.6	-	55.9	7.1	0.7	8.8	63.6	-	-	7.1	0.6
9	166.1	5.0	3.1	-	-	-	3.3	-	-	-	-	-	3.1
10	5.8	0.4	0.4	-	1.5	1.5	0.4	1.5	-	-	0.8	1.5	0.4
11	-	29.1	12.7	-	27.7	14.8	12.9	15.1	30.8	-	-	14.8	12.7
12	452.3	454.1	0.3	-	-	-	0.3	-	-	-	-	-	0.3
14	0.7	0.7	0.3	801.7	226.5	143.5	0.3	531.3	125.9	-	-	143.5	0.3
16	0.4	0.4	0.4	4.7	2.6	2.3	0.4	4.4	3.2	2160.1	40.2	2.3	0.4
17	-	2.1	2.2	-	4.5	4.5	2.2	6.2	-	-	5.7	4.5	2.2
18	-	15.6	14.4	-	126.2	51.1	14.5	63.1	-	-	-	51.1	14.4
19	-	871.1	859.4	-	1987.5	1756.3	859.2	1761.8	-	-	-	1756.3	859.4
21	1.2	0.6	0.3	-	1303.1	352.5	0.3	-	6.2	-	792.8	352.5	0.3
22	168.8	5.5	0.3	-	-	-	0.3	-	33.6	-	-	-	0.3
23	0.8	0.9	0.5	-	10.3	7.0	0.4	12.6	1.8	-	-	7.0	0.5
24	1.5	0.7	0.8	-	1.9	1.9	0.9	2.0	35.2	-	9.1	1.9	0.8
25	0.5	0.6	0.7	0.6	0.8	0.8	0.8	0.9	0.2	1580.0	0.8	0.8	0.7
26	0.2	0.3	0.4	4.0	1.9	1.9	0.5	2.7	4.7	0.1	12.5	1.9	0.4
27	3.3	0.9	0.6	-	-	-	0.7	-	0.3	-	-	-	0.6

Table 2 Timings of Triangularize versus other solvers

60% of total memory. In both Table 1 and 2, the symbol "-" means either time or memory exceeds the limit we set.

The examples are mainly in positive dimension since other triangular decomposition algorithms are specialized to dimension zero [7]. All examples are in characteristic zero.

In Table 1, we provide characteristics of the input systems and the sizes of the output obtained by different solvers. For each polynomial system $F \subset \mathbb{Q}[\mathbf{x}]$, the number of variables appearing in F, the number of polynomials in F, the maximum total degree of a polynomial in F, the dimension of the algebraic variety $V(F)$ are denoted respectively by $\#v$, $\#e$, deg, dim. For each solver, the size of its output is measured by the total number of characters in the output. To be precise, let "dec" and "gb" be respectively the output of the Triangularize and Groebner functions. The MAPLE command we use are length(convert(map(Equations, dec, R), string)) and length(convert(gb, string)). From Table 1, it is clear that Triangularize produces much smaller output than commands based on Gröbner basis computations.

TK, TL, GS, WS (and, to some extent, GL) can all be seen as polynomial system solvers in the sense of that they provide equidimensional decompositions where components are represented by triangular sets. Moreover, they are implemented in MAPLE (with the support of efficient C code in the case of GS and GL). The specification of TK are close to those of GS while TL is related to WS, though the triangular sets returned by WS are not necessarily regular chains.

In Table 2, we provide the timings of different versions of Triangularize and other solvers. From this table, it is clear that the implementations of Triangularize, based on the algorithms presented in this paper (that is TK14, TL14, TK, TL) outperform the previous versions (TK13, TL13), based on [17], by several orders of magnitude. We observe also that TK outperforms GS and GL while TL outperforms WS.

8. REFERENCES

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

[2] F. Boulier, D. Lazard, F. Ollivier, and M. Petitot. Representation for the radical of a finitely generated differential ideal. In *proceedings of ISSAC'95*, pages 158–166, 1995.

[3] C. Chen, O. Golubitsky, F. Lemaire, M. Moreno Maza, and W. Pan. *Comprehensive Triangular Decomposition*, volume 4770 of *LNCS*, pages 73–101. Springer Verlag, 2007.

[4] C. Chen and M. Moreno Maza. Algorithms for Computing Triangular Decompositions of Polynomial System. *CoRR*, 2011.

[5] C. Chen, M. Moreno Maza, B. Xia, and L. Yang. Computing cylindrical algebraic decomposition via triangular decomposition. In *ISSAC'09*, pages 95–102, 2009.

[6] S.C. Chou and X.S. Gao. Solving parametric algebraic systems. In *Proc. ISSAC'92*, pages 335–341, 1992.

[7] X. Dahan, M. Moreno Maza, É. Schost, W. Wu, and Y. Xie. Lifting techniques for triangular decompositions. In *ISSAC'05*, pages 108–115. ACM Press, 2005.

[8] J. Della Dora, C. Dicrescenzo, and D. Duval. About a new method for computing in algebraic number fields. In *Proc. EUROCAL 85 Vol. 2*, pages 289–290. Springer-Verlag, 1985.

[9] L. Ducos. Optimizations of the subresultant algorithm. *Journal of Pure and Applied Algebra*, 145:149–163, 2000.

[10] X.-S. Gao, J. Van der Hoeven, Y. Luo, and C. Yuan. Characteristic set method for differential-difference polynomial systems. *J. Symb. Comput.*, 44:1137–1163, 2009.

[11] É. Hubert. Factorization free decomposition algorithms in differential algebra. *J. Symb. Comp.*, 29(4-5):641–662, 2000.

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

[13] D. Lazard. A new method for solving algebraic systems of positive dimension. *Discr. App. Math*, 33:147–160, 1991.

[14] G. Lecerf. Computing the equidimensional decomposition of an algebraic closed set by means of lifting fibers. *J. Complexity*, 19(4):564–596, 2003.

[15] X. Li, M. Moreno Maza, and W. Pan. Computations modulo regular chains. In *Proc. ISSAC'09*, pages 239–246, New York, NY, USA, 2009. ACM Press.

[16] B. Mishra. *Algorithmic Algebra*. Springer-Verlag, 1993.

[17] M. Moreno Maza. On triangular decompositions of algebraic varieties. Technical Report TR 4/99, NAG Ltd, Oxford, UK, 1999. Presented at the MEGA-2000 Conference, Bath, England.

[18] T. Shimoyama and K. Yokoyama. Localization and primary decomposition of polynomial ideals. *J. Symb. Comput.*, 22(3):247–277, 1996.

[19] A.J. Sommese, J. Verschelde, and C. W. Wampler. Solving polynomial systems equation by equation. In *Algorithms in Algebraic Geometry*, pages 133–152. Springer-Verlag, 2008.

[20] D. K. Wang. The `Wsolve` package. http://www.mmrc.iss.ac.cn/∼dwang/wsolve.txt.

[21] D. M. Wang. *Elimination Methods*. Springer, New York, 2000.

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

[23] L. Yang, X.R. Hou, and B. Xia. A complete algorithm for

automated discovering of a class of inequality-type theorems. *Science in China, Series* **F**, 44(6):33–49, 2001.

[24] L. Yang and J. Zhang. Searching dependency between algebraic equations: an algorithm applied to automated reasoning. Technical Report IC/89/263, International Atomic Energy Agency, Miramare, Trieste, Italy, 1991.