Making a Sophisticated Symbolic Solver Available to Different Communities of Users

François Lemaire lemaire@lifl.fr Université Lille 1 France

Marc Moreno Maza moreno@orcca.on.ca Univ. Western Ontario Canada

Yuzhen Xie yxie@orcca.on.ca Univ. Western Ontario Canada

Abstract

Triangular decompositions have become one of the major tools for solving systems of non-linear algebraic or differential equations symbolically. These decompositions display more geometrical information than other symbolic descriptions of polynomial systems. However, their specifications and the algorithms computing them are quite sophisticated. Their implementation in mathematical software, their accessibility, and ease of use for non-expert users are challenges that we discuss in this paper.

We discuss our solutions and illustrate them with the implementation of an algorithm for triangular decompositions, called Triade, in three computer algebra systems: AXIOM, ALDOR, and MAPLE, targeting different communities of users. We believe that these implementations of the same sophisticated mathematical algorithm for different communities of experts, advanced users, and non-experts is a unique experience in the area of symbolic computations which could benefit other algorithms in this field.

Introduction

Triangular decompositions are one of the major tools for solving polynomial systems. For systems of algebraic equations, they provide a convenient way to describe complex solutions and a step toward isolation of real roots or decomposition into irreducible components. They are also used in dynamic evaluation [14, 11], in geometric computations [6], such as implicitization and classification problems [13, 18], automatic theorem proving [7] and for studying dynamical systems in biology [4, 5]. They have interesting properties [10, 24] for developing modular methods and stable numerical techniques.

For systems of partial differential equations, triangular decompositions provide the main practicable way for determining a symbolic description of the solution set. Moreover, techniques from the algebraic case apply to the differential case.

The implementation of triangular decomposition algorithms in mathematical software, their accessibility to expert or non expert users are certainly challenges that we address in this paper. Section 1 presents the main difficulties arising during the conception and the implementation of a polynomial system solver based on triangular decompositions. Among those are:

- the sophisticated notions and rich properties attached to triangular decompositions,

- the prototyping of the algorithm and its sub-routines,

- the validation and the user-interface of such a solver.

These must be addressed differently, depending on the strengths and weaknesses of the implementation environment, the level of expertise and expectation of its community of users,

We report on the implementation of an algorithm for triangular decompositions, called Triade [23], realized during the last eight years, in three computer algebra systems: AXIOM, ALDOR, and MAPLE, targeting different communities of users. We discuss our solutions and compare these three different implementations in Sections 2, 3 and 4 based on the challenges defined in Section 1.

AXIOM has been designed to express the extremely rich and complex variety of structures and algorithms in computer algebra; our AXIOM implementation (65,000 lines of code) of Triade matches the theoretical specifications of the algorithm; this implementation is meant for researchers in the area of symbolic computation, and is available in open source.

ALDOR is an extension of the AXIOM system with focuses on interoperability with other languages and high-performance computing. In ALDOR, the Triade implementation (110,000 lines of code) is available in the form of specialized servers which can solve polynomial systems with frequently used rings of coefficients; these servers have been successfully used by researchers in theoretical physics and algebraic problems. Today, a parallel implementation of Triade in ALDOR is under development on multi-processor machines with shared memory.

MAPLE is a general purpose computer algebra system with users from broad areas (students, engineers, researchers, etc.). The MAPLE implementation of the Triade algorithm (50,000 lines of code) is available since the release 10 of MAPLE as the `RegularChains` library. The non-expert users can access easily a first group of easy-to-use functionalities for computing triangular decompositions and studying the properties of the solutions of polynomial systems. The more expert users can take advantage of the many options of these functionalities. In addition, sub-modules of the `RegularChains` provide advanced features such as automatic case discussion in parametric problems, and linear algebra over non-integral domains.

Section 5 presents a comparison between the three implementations, and highlights their advantages and weaknesses in terms of efficiency, ease of use and targeted audience.

1 Challenges in implementing triangular decompositions

In general, triangular decompositions can reveal geometrical information of the solution sets better than other symbolic descriptions of polynomial systems such as Gröbner bases. However, the specifications and algorithms for computing triangular decompositions are quite sophisticated, which impose great challenges on their implementation in mathematical software, and on their accessibility and ease of use for users with various interest.

For an input system of polynomials F, with rational coefficients, both a Gröbner basis and a triangular decomposition of F give the full set of the complex solutions of F. Consider the polynomial system F_1 with the variables $x > y > z$:

$$\begin{cases} x^3 - 3x^2 + 2x & = & 0 \\ 2yx^2 - x^2 - 3yx + x & = & 0 \\ zx^2 - zx & = & 0 \end{cases}$$

It has lexicographical Gröbner basis:

$$\begin{cases} x^2 - xy - x \\ -xy + xy^2 \\ zxy \end{cases}$$

and triangular decomposition:

$$\{\ x = 0\ \bigcup \begin{cases} x = 1 \\ y = 0 \end{cases} \bigcup \begin{cases} x = 2 \\ y = 1 \\ z = 0 \end{cases}$$

It is clearly shown that it consists of one point ($x = 2, y = 1, z = 0$), one line ($x = 1, y = 0$), and one plane ($x = 0$). This example reveals the first challenge in implementing triangular decompositions, that is, the representation of triangular decompositions. It is a list of list of polynomials with special properties, instead of just a list of polynomials as for a Gröbner basis.

In addition, for the same input polynomial system, there are different possible output triangular decompositions. This makes it harder to specify the results, but in practice these different outputs are of varied benefits. Moreover, there is a canonical form of output (if the system has finitely many solutions), called the equiprojectable decomposition [9], which can be computed from any triangular decomposition of the same system, if needed.

Let us illustrate by an example. For the following input polynomial system F_2,

$$F_2 : \begin{cases} x^2 + y + z = 1 \\ x + y^2 + z = 1 \\ x + y + z^2 = 1 \end{cases}$$

One possible triangular decomposition of the solution set of F_2 is:

$$\begin{cases} z = 0 \\ y = 1 \\ x = 0 \end{cases} \bigcup \begin{cases} z = 0 \\ y = 0 \\ x = 1 \end{cases} \bigcup \begin{cases} z = 1 \\ y = 0 \\ x = 0 \end{cases} \bigcup \begin{cases} z^2 + 2z - 1 = 0 \\ y = z \\ x = z \end{cases}$$

Another one is:

$$\begin{cases} z = 0 \\ y^2 - y = 0 \\ x + y = 1 \end{cases} \bigcup \begin{cases} z^3 + z^2 - 3z = -1 \\ 2y + z^2 = 1 \\ 2x + z^2 = 1 \end{cases}$$

Both results are valid. The second one is the equiprojectable decomposition. As a matter of fact, the second one can be computed from the first one by techniques explained in [9].

Although triangular decompositions display rich geometrical information, the solutions can be hard to read, especially when there is an infinite number of solutions; see Appendix A for an example. Actually, this problem has not been properly solved yet.

In 1987, Wen Tsün Wu [28] introduced the first algorithm computing triangular decompositions of systems of algebraic equations, by means of the so-called "characteristic sets". Kalkbrener [17] provided an algorithm where he considered particular characteristic sets, namely regular chains, leading to theoretical and practical improvements. See also the work of Wang [26], and the work of Lazard and his students [1].

Our study employs the algorithm called Triade, introduced by Moreno Maza [23]. This algorithm relies more intensively on geometrical considerations than the previous ones for computing triangular decompositions, leading to an efficient management of the intermediate computations and control of expression swell. Lazy evaluation techniques and a *task manager* paradigm are also essential tools in this algorithm.

The implementation challenges on Triade are summarized as follows. First of all is the prototyping. Indeed, most operations rely on automatic case discussion, splitting the computations into sub-cases; see Section 2 for this point. Secondly, it has to be decided which functionalities will be provided to the end users, and this will affect the ease of use of the package. This choice depends on the computer algebra system and the different communities of users. Thirdly, code validation is extremely difficult, because checking the computations in the case of triangular decomposition is much harder than for Gröbner bases computations. See Appendix A for detail. Therefore, we need more advanced techniques, such as validating packages, comparison with other software, and large test suites. Performance and optimization of the implementations have special features. Again, they rely largely on the underlying computer algebra system and the requirements of different groups of users. For instance, the data representation for polynomials and regular chains affects the efficiency.

2 The AXIOM implementation

AXIOM designers attempted to overcome the challenges of providing an environment for implementing the extremely rich relationships among mathematical structures [16]. Hence, their design is of somewhat different direction from that of other computer algebra systems.

The AXIOM computer algebra system possesses an interactive mode for user interactions and the SPAD language for building library modules. This language has a two-level object model of *categories* and *domains*, that is similar to *interfaces* and *classes* in Java. They provide a type system that allows the programmer the flexibility to extend or build on existing types, or create new categories and domains, as is usually required in algebra.

The SPAD language has also a functional programming flavor: types and functions can be constructed and manipulated within programs dynamically like the way values are manipulated. This makes it easy to create generic programs in which independently developed components are combined. For instance, one can write a SPAD function q which takes as arguments a commutative ring R and an element $p \in R$ such that $q(R, p)$ implements the quotient ring R/pR.

These features allowed us to implement the Triade algorithm in its full generality, that is without any restrictions w.r.t. the theory presented in [23]. In particular, our code can be used with any multivariate polynomial data-type over any field of coefficients available in AXIOM.

One important characteristic of the algorithms producing triangular decompositions is the fact that the intermediate computations require many polynomial coefficient types leading to potentially many type conversions. More precisely, the typical procedure, say `proc`,

- takes as input a quotient ring Q of the form $\mathbb{K}[X]/I$, where I is an ideal of $\mathbb{K}[X]$, and elements of Q, say f, g, and
- returns a list of pairs $(Q_1, h_1), \ldots, (Q_s, h_s)$ where Q_j is a quotient ring $\mathbb{K}[X]/I_j$ and h_j is an element of Q_j, for all $1 \leq j \leq s$.

In the *Dynamic Evaluation* packages in AXIOM [14, 11] the signature of a function implementing `proc` would match the specializations of `proc` precisely; in particular, the types $Q, Q_1, \ldots, Q_s$ would be instantiated at run-time. In the implementation of the Triade algorithm the quotient rings $Q, Q_1, \ldots, Q_s$ are not built explicitly; instead, they are represented by the ideals $I, I_1, \ldots, I_s$, and $f, g, h_1, \ldots, h_s$ are encoded by representatives in $\mathbb{K}[X]$. This latter approach may look less elegant than the former one. In fact, as reported in [2], it brings performance improvement, by avoiding type instantiations and conversions; in addition, it offers more opportunities for optimizations, by homogenizing the type of the intermediate quantities. This approach was continued in the ALDOR and MAPLE implementations of the Triade algorithm.

As discussed in Section 1, another challenge in implementing triangular decompositions is code validation. Because a given input system $F \subset \mathbb{K}[X]$ may admit different triangular decompositions, it is hard to use one implementation of these decompositions in order to validate another. The safest approach, as mentioned in Appendix A. is through computations based on Gröbner bases. However, this leads to computations which, in practice and in theory, are much more expensive than those of triangular decompositions. This difficulty was resolved by interfacing AXIOM with a high-performance Gröbner engine [12] see [2] for details.

Our AXIOM implementation of the Triade algorithm has been integrated in 1998 in the release 2.2 of AXIOM [15]. Experiments reported in [2] show that it often outperforms comparable solvers. Moreover, combined with another symbolic solver [25], it provides functionalities for isolating real roots of polynomial systems symbolically: AXIOM was the first general purpose computer algebra system offering this feature, which we illustrate by the fragment of AXIOM session shown in Appendix B.

On the contrary of other general purpose computer algebra system like MAPLE, AXIOM is primarily destined to the community of researchers in computer algebra: it requires good programming skills and a strong background in algebra. In particular, every user is potentially an expert and a code developer. As a consequence, the logical organization of the library modules relies simply on the algebraic hierarchies of categories and domains; thus, there is less concern with the "ease of use" than in MAPLE.

To summarize, our AXIOM implementation has reached its goals: providing a generic, reliable and quite efficient polynomial system solver by means of triangular decompositions.

3 The Aldor implementation

ALDOR was designed to be an extension language for the AXIOM computer algebra system. In addition, an ALDOR program can be compiled into: stand-alone executable programs; object libraries in native operating system formats (which can be linked with one another, or with C or Fortran code to form application programs); portable byte code libraries; and C or Lisp source. Aggressive code optimizations by techniques such as program specialization, cross-file procedural integration and data structure elimination, are performed at intermediate stages of compilation [27]. This produces code that is comparable to hand-optimized C.

For these reasons we use ALDOR to develop high-performance implementations of the Triade algorithm since 1999. More recently, we have realized a parallel implementation [20, 21] on a multiprocessor machine using shared memory segments for interprocess data-communication.

Our ALDOR implementation is much less generic than our AXIOM implementation. First, it is limited to particular, and frequently used, coefficient fields, such as $\mathbb{Q}$, the field of rational

numbers, and finite fields. Secondly, it is available in form of executable binary programs, like an operating system command. These "servers" are quite easy to use, but they perform only very specific tasks; in particular, they offer a very limited user interaction. However, their computational power outperforms the AXIOM implementation and they were used to solve difficult problems in theoretical physics [13] and in invariant theory [18].

To summarize, our ALDOR implementation of the Triade algorithm is reaching its main objective: high-performance computing.

4 The RegularChains library in Maple

MAPLE [22] is a general-purpose computer algebra system. It offers an interpreted, dynamically typed programming language. MAPLE has a very large audience among the world. It is used by engineers, researchers as well as students, in much different topics such as engineering, finance, statistics, education, etc. MAPLE is shipped with a wide variety of libraries dealing, for instance, with linear algebra, differential equations solving, numerical computations. MAPLE is intended to be powerful and easy to use for the high end user. We have realized a MAPLE implementation of the Triade algorithm, the `RegularChains` library [19], which is shipped with the MAPLE software since the version 10 of MAPLE, released in 2005.

On the contrary to AXIOM and ALDOR, the MAPLE programming language does not have a strong object oriented flavor. Code organization and validation are, therefore, even more challenging in this context. So, we describe our effort in these directions for implementing the `RegularChains` library. MAPLE libraries are usually organized as follows:

- a user-interface level providing functionalities accessible to the end-user in the interactive mode; those functions usually check the input specifications,
- an internal level providing functionalities accessible only to the library programmers; they are called by the user-interface functionalities for doing the actual computations.

The data structures are quite straightforward. The most complex data used are the multivariate polynomials. We have chosen the native MAPLE polynomials which are directed acyclic graphs (DAG). This choice has been made for simplicity reasons. Indeed, all MAPLE directives manipulating polynomials handle DAG. All other objects, as regular chains, have been implemented with lists. Moreover, all structures have been enriched with extra information of two different kinds. The first kind are cached results which are computed frequently (for example the leading variable of a polynomial). The second kind are flags that help optimizing certain functions (for example, knowing that a regular chain represents an irreducible component helps speeding-up computations).

The source code organization is rather standard too. We have split the library source into different files, each one representing a different class of objects. The objective was to mimic the AXIOM/ALDOR organization into categories and domains. This split is very handy, because it emulates some kind of generic programming. Indeed, if we want to use a different representation for polynomials, we only need to change the file implementing polynomials. This makes it easy to compare the efficiency of two different polynomial representations. As for prototyping, the internal functions have been organized similarly to AXIOM/ALDOR.

The `RegularChains` user-interface has been designed to provide ease of use to the non-expert and, advanced functionalities to the expert. The library offers numerous primitives for

computing and manipulating triangular decompositions. For instance, it provides a rich variety of coefficient fields: $\mathbb{Q}$, the field of rational functions, prime fields, fields of rational functions over $\mathbb{Q}$ and prime fields. This is an additional challenge in the MAPLE framework, which has limited support for generic programming.

Combining ease of use and variety of advanced functionalities is achieved by a two-level organization of the user-interface. The first level provides the basic functionalities easy to use for the non-expert. Those functionalities allow to compute triangular decompositions and manipulate polynomials. The second level of the user-interface provides more technical functionalities that are available through optional arguments of the basic functionalities and through two submodules, called `ChainTools` and `MatrixTools`. Those two sub-libraries respectively provide tools for manipulating triangular decompositions and regular chains, and for doing linear algebra over non-integral domains. This makes MAPLE the unique computer algebra system offering automatic case discussion and recombination, as illustrated by the fragment of MAPLE session in Appendix C.

The code validation is made through the MAPLE library test suites. A test suite for `RegularChains` check all the user interface functions, in order to validate any changes that would be made to the code and the user-interface. Also, the primitive computing triangular decomposition, which is a crucial functionality, is tested through a large set of problems. The outputs are partially checked in positive dimension by checking that the radical of the input system is included in the radical of (the saturated ideal of) each regular chain in the output. This ensures that we haven't lost any solution. A complete check in positive dimension has not been done as it has been done for the AXIOM or ALDOR implementations. However, the checking in zero dimension has been done very thoroughly. Indeed, the output decomposition is processed in a special way. We first make the decomposition radical (each regular chain of the output is made radical), which removes multiple roots. Then the decomposition is processed in such a way that all regular chains of the decomposition have distinct roots. This ensures that the total number of solutions n_o without multiplicity of the decomposition is exactly the sum of the number of solutions of each regular chain (which is just the product of the leading degrees). Thus, if the input system is reduced to zero by each regular chain of the decomposition, we know that the solutions of the input are solutions of the decomposition. Therefore, if we know the number of solutions of the input in advance, and if it is equal to n_o, we are sure that no solutions have been lost or added, which means that the decomposition is correct.

5 Discussion

Here are some highlights and additional comments regarding the three implementations of the Triade algorithm in AXIOM, ALDOR and MAPLE.

The AXIOM implementation has been developed in a very general manner in the sense the design is very close to the mathematical theory. This makes it powerful and flexible. The drawback is that it is not suitable for high-performance, and hard to use for non-experts.

The ALDOR implementation is less general than the AXIOM implementation but has several advantages. First of all, the ALDOR compiler produces binaries which can act as servers or regular applications. This makes it easier for interfacing the Triade solver with other software. Moreover, ALDOR provides an efficient interface with the machine resources leading to higher performances. Both ALDOR and AXIOM implementations are organized into cate-

gories and domains, and lots of functionalities can be used and extended. Therefore, they are well adapted for expert users who aim at developing new algorithms and performing advanced experimentations.

The MAPLE implementation `RegularChains` is different from the ALDOR and AXIOM ones, in numerous ways. First of all, MAPLE has a larger audience of users, and is aimed at being user friendly. `RegularChains` is written in this spirit and is very easy to use for non-experts. Advanced users are still able to make more complicated computations by using optional arguments and the two submodules `ChainTools` and `MatrixTools`. Secondly, the MAPLE programming language is interpreted and dynamically typed. The language syntax is straightforward and easy, thus, it is not difficult to write MAPLE code. However, the code validation and maintenance is much harder because type errors are only detected at execution. Therefore, coding requires a lot of care and discipline.

Despite of this difficulty, it appears in practice that contributions from students and collaborators are usually made to the MAPLE implementation rather than to its ALDOR and AXIOM counterparts. This is clearly due to the ease of use of the `RegularChains` library. Consequently, some recent and efficient algorithms have been implemented only in the `RegularChains` library, see for instance [9, 8]. This is why, on some test problems, the `RegularChains` library can outperform the ALDOR and AXIOM implementations of the Triade algorithm.

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] P. Aubry and M. Moreno Maza. Triangular sets for solving polynomial systems: A comparative implementation of four methods. *J. Symb. Comp.*, 28(1-2):125–154, 1999.

[3] T. Becker and V. Weispfenning. *Gröbner Bases: a computational approach to commutative algebra*, volume 141 of *Graduate Texts in Mathematics*. Springer Verlag, 1991.

[4] F. Boulier, L. Denis-Vidal, T. Henin, and F. Lemaire. Lépisme. In *International Conference on Poynomial System Solving*. University of Paris 6, France, 2004.

[5] F. Boulier and F. Lemaire. *Lépisme*. Université de Lille I, France, 2005. http://www.lifl.fr/∼lemaire/lepisme.

[6] F. Chen and D. Wang, editors. *Geometric Computation*. Number 11 in Lecture Notes Series on Computing. World Scientific Publishing Co., Singapore, New Jersey, 2004.

[7] S.C. Chou. *Mechanical Geometry Theorem Proving*. D. Reidel Publ. Comp., Dordrecht, 1988.

[8] X. Dahan, X. Jin, M. Moreno Maza, and É Schost. Change of ordering for regular chains in positive dimension. In Ilias S. Kotsireas, editor, Maple Conference 2006, pages 26–34, 2006.

[9] 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.

[10] X. Dahan and É. Schost. Sharp estimates for triangular sets. In *ISSAC 04*, pages 103–110. ACM, 2004.

[11] D. Duval. Algebraic Numbers: an Example of Dynamic Evaluation. *J. Symb. Comp.*, 18(5):429–446, November 1994.

[12] J.-C. Faugère. A new efficient algorithm for computing Gröbner bases. *J. Pure and Appl. Algebra*, 139(1-3):61–88, 1999.

[13] M.V. Foursov and M. Moreno Maza. On computer-assisted classification of coupled integrable equations. *J. Symb. Comp.*, 33:647–660, 2002.

[14] T. Gómez Díaz. *Quelques applications de l'évaluation dynamique*. PhD thesis, Université de Limoges, 1994.

[15] The Computational Mathematics Group. AXIOM 2.2. NAG Ltd, Oxford, UK, 1998.

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

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

[18] I. A. Kogan and M. Moreno Maza. Computation of canonical forms for ternary cubics. In Teo Mora, editor, *Proc. ISSAC 2002*, pages 151–160. ACM Press, July 2002.

[19] F. Lemaire, M. Moreno Maza, and Y. Xie. The `RegularChains` library. In Ilias S. Kotsireas, editor, Maple Conference 2005, pages 355–368, 2005.

[20] M. Moreno Maza and Y. Xie. An implementation report for parallel triangular decompositions on a shared memory multiprocessor. ACM Press, 2006.

[21] M. Moreno Maza and Y. Xie. Parallelization of triangular decompositions. In *Proc. of Algebraic Geometry and Geometric Modelling'06*, Barcelona, Spain, 2006.

[22] Maplesoft. *Maple 10*. http://www.maplesoft.com/, 2005.

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

[24] M. Moreno Maza, G. Reid, R. Scott, and W. Wu. On approximate triangular decompsoitions i. dimension zero. In D. M. Wang and L. Zhi, editors, *Symbolic-Numeric Computation*, Xi'an, China, 2005.

[25] R. Rioboo. Real algebraic closure of an ordered field, implementation in axiom. In *Proc. ISSAC'92*, pages 206–215. ISSAC, ACM Press, 1992.

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

[27] Stephen M. Watt, Peter A. Broadbery, Samuel S. Dooley, Pietro Iglio, Scott C. Morrison, Jonathan M. Steinbach, and Robert S. Sutor. A first report on the a# compiler. In *ISSAC '94: Proceedings of the international symposium on Symbolic and algebraic computation*, New York, NY, USA, 1994. ACM Press.

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

APPENDIX A

Let $\mathbb{K}$ be a field and $X = x_1 < \cdots < x_n$ be ordered variables. For a non-constant polynomial $p \in \mathbb{K}[X]$, the *main variable* of p, denoted by $\mathsf{mvar}(p)$, is the greatest variable of p; the *initial* of p, denoted by $\mathsf{init}(p)$ is the leading coefficient of p w.r.t. $\mathsf{mvar}(p)$. For example, for $p_1 = (2x_1 - 1){x_2}^2 - 3x_1x_2 + x_2$, $\mathsf{mvar}(p_1) = x_2$, and $\mathsf{init}(p_1) = 2x_1 - 1$.

Let $F \subset \mathbb{K}[X]$ be any set of polynomials with coefficients in $\mathbb{K}$ and variables in X. We denote by $\langle F \rangle$ the ideal generated by F in $\mathbb{K}[X]$ and by $\sqrt{\langle F \rangle}$ its radical. We denote by $V(F)$ the zero set of F in the affine space $\overline{\mathbb{K}}^n$ where $\overline{\mathbb{K}}$ is an algebraically closed field containing $\mathbb{K}$. Usually $\mathbb{K} = \mathbb{Q}$, the field of rational numbers and $\overline{\mathbb{K}} = \mathbb{C}$, the field of complex numbers.

For a subset $W \subset \overline{\mathbb{K}}^n$, we denote by $\overline{W}$ the *Zariski closure* of W w.r.t. $\mathbb{K}$, that is the intersection of the $V(F)$ containing W, for all $F \subset \mathbb{K}[X]$.

Let I be a proper ideal of $\mathbb{K}[X]$. We say that a polynomial $p \in \mathbb{K}[X]$ is *regular* modulo I if for every prime ideal $\mathcal{P}$ associated with I we have $p \notin \mathcal{P}$, equivalently, this means that p is neither null modulo I, nor a zero-divisor modulo I.

Let $h \in \mathbb{K}[X]$. We denote by $I : h^\infty$ the set of the polynomials p such that there exists a non-negative integer e such that $h^e p$ belongs to I. Let $T = t_1, \ldots, t_s$ be non-constant polynomials in $\mathbb{K}[X]$ with respective (pairwise distinct) main variables $\mathsf{mvar}(t_1) < \cdots < \mathsf{mvar}(t_s)$. The *saturated ideal* of T is defined by

$$\mathbf{Sat}(T) = \langle T \rangle : {h_T}^\infty,$$

where h_T is the product of the initials of the polynomials of T.

The *quasi-component* of T is the subset $W(T)$ of the zero-set $V(T)$ consisting of all the points that do not cancel any of the initials of the polynomials of T. We have the following important property, which realizes a bridge between the "algebraic" notion of a saturated ideal and the "geometric" notion of a quasi-component. It is, in fact, a consequence of Hilbert's theorem of zeros:

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

We can now define one of the two central concepts in the theory of triangular decompositions: the set T is a *regular chain* if for all $i = 2 \cdots s$ the initial of t_i is regular modulo the saturated ideal of $t_1, \ldots, t_{i-1}$.

We can now define the notion of a triangular decomposition of a polynomial system. This notion comes actually in two flavors.

Definition 1 *Let $F \subset \mathbb{K}[X]$ be a polynomial set. A set $C_1, \ldots, C_s$ of regular chains in $\mathbb{K}[X]$ is a* triangular decomposition *of F in the sense of*

- Kalkbrener *if we have* $\sqrt{\langle F \rangle} = \cap_{1 \leq i \leq s} \mathbf{Sat}(C_i)$,
- Lazard *if we have* $V(F) = \cup_{1 \leq i \leq s} W(C_i)$.

It is easy to check that a triangular decomposition of F in the sense of Lazard is also a triangular decomposition of F in the sense of Kalkbrener. The converse is true if $V(F)$ is finite. However, it is false, in general, when $V(F)$ is infinite.

In broad words, a triangular decomposition of F in the sense of Kalkbrener describes "only" the generic zeros of $V(F)$ whereas a triangular decomposition of F in the sense of Lazard describes all the zeros of $V(F)$. The following example illustrates the differences between these two kinds of triangular decompositions.

Given a polynomial system F_1 having two polynomials with variable order of $x > y > a > b > c > d > e > f$:

$$\begin{cases} ax+cy-e &= 0 \\ bx+dy-f &= 0 \end{cases}$$

The triangular decomposition of F_1 in the sense of Kalkbrener consists of one regular chain, which is

$$\begin{cases} bx+dy-f \\ (da-cb)y-fa+eb \end{cases}$$

The triangular decomposition of F_1 in the sense of Lazard consists of eleven regular chains, which are

$$\begin{cases} bx+dy-f \\ (da-cb)y-fa+eb \end{cases}, \begin{cases} ax+cy-e \\ dy-f \\ b \end{cases}, \begin{cases} bx+dy-f \\ da-cb \\ fc-ed \end{cases}, \begin{cases} dy-f \\ a \\ b \\ fc-ed \end{cases}, \begin{cases} bx-f \\ fa-eb \\ c \\ d \end{cases},$$

$$\begin{cases} ax+cy-e \\ b \\ d \\ f \end{cases}, \begin{cases} bx+dy \\ da-cb \\ e \\ f \end{cases}, \begin{cases} cy-e \\ a \\ b \\ d \\ f \end{cases}, \begin{cases} y \\ a \\ b \\ e \\ f \end{cases}, \begin{cases} x \\ c \\ d \\ e \\ f \end{cases}, \begin{cases} a \\ b \\ c \\ d \\ e \\ f \end{cases}.$$

Let $C_1, \ldots, C_s$ be regular chains in $\mathbb{K}[X]$ and $F \subset \mathbb{K}[X]$ be a polynomial set. We explain how to check whether $C_1, \ldots, C_s$ is a triangular decomposition of F in the sense of Kalkbrener. (Up to now, there is no algorithms for the case of Lazard decompositions.) The procedure is as follows and relies on well-known ideal operations, see for instance [3] for details:

1. for all $1 \leq i \leq s$ replace C_i by D_i such that $\mathbf{Sat}(D_i)$ is radical and equals to $\sqrt{\mathbf{Sat}(C_i)}$.
2. for all $1 \leq i \leq s$, for all $f \in F$, check that $f \in \mathbf{Sat}(D_i)$ holds, that is, f reduces to 0 by pseudo-division w.r.t. D_i.
3. compute a set of generators $\{g_1, \ldots, g_t\}$ of the intersection of $\mathbf{Sat}(D_1), \ldots, \mathbf{Sat}(D_s)$.
4. for all $1 \leq i \leq t$ check that g_i belongs to $\sqrt{\langle F \rangle}$.

APPENDIX B

```
Axiom Computer Algebra System (Release 2.3)

(8) -> lp := [p1, p2, p3]

                  2          2          2
   (8)  [z + y + x  - 1,z + y  + x - 1,z  + y + x - 1]
                                    Type: List Polynomial Integer
(9) -> triangSolve(lp)$pack

          2
   (9)  [{z  + 2z - 1,y - z,x - z},{z,y - 1,x},{z,y,x - 1},{z - 1,y,x}]
                           Type: List RegularChain(Integer,[x,y,z])
(10) -> lpr := positiveSolve(lp)$pack

   (10)  [[%B2,%B2,%B2]]
                       Type: List List RealClosure Fraction Integer
(11) -> [[approximate(r, 1/100000) for r in pr] for pr in lpr]

           108583 108583 108583
   (11)  [[------,------,------]]
           262144 262144 262144
                                 Type: List List Fraction Integer
```

APPENDIX C

```
    |\^/|     Maple 10 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2005
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> with(Triade): with(Triade:-MatrixTools): with(Triade:-ChainTools):
> R := PolynomialRing([y,z]);
                             R := polynomial_ring

> rc := Chain([z^4 +1, y^2 -z^2], Empty(R), R): Equations(rc, R);
                                 2    2   4
                               [y  - z , z  + 1]

> m := Matrix([[1, y+z], [0, y-z]]);
                                     [1    y + z]
                                m := [          ]
                                     [0    y - z]

> MatrixInverse(m,rc,R);
     [1     0  ]
     [         ]                                [1    y + z]
```

```
  [[[[       3 ], regular_chain]], [["no inv", [          ], regular_chain]]]
     [      z  ]                              [0    y - z]
     [0    ----]
     [      2  ]

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

> mim := MatrixInverse(m,rc,R);
mim :=

       [ 1      0  ]                   [  0       1/2 ]
       [           ]                   [              ]
    [[[[         3 ], regular_chain], [[    3       3 ], regular_chain]], []]
       [  3     z  ]                   [   z       z  ]
       [-z     ----]                   [- ----    ----]
       [        2  ]                   [   2       4  ]

> m1 := mim [1][1][1]: rc1 := mim [1][1][2]: Equations(rc1, R);
                                      4
                                [y + z, z  + 1]

> m2 := mim [1][2][1]: rc2 := mim [1][2][2]: Equations(rc2, R);
                                      4
                                [y - z, z  + 1]

> mc := MatrixCombine([rc1, rc2], R, [m1, m2]);
              [     3                     3           ]
              [    z  y                  z  y         ]
              [    ---- + 1/2          - ---- + 1/4   ]
      mc := [[[     2                     4           ], regular_chain]]
              [                                       ]
              [      3        2           3        2  ]
              [-3/4 z  + 1/4 z  y    3/8 z  - 1/8 z  y]

> Equations(mc[1][2], R);
                                   2    2   4
                                 [y  - z , z  + 1]
```