Quantifier Elimination by Cylindrical Algebraic Decomposition Based on Regular Chains

Changbo Chen [a], Marc Moreno Maza [a,b]

[a] *Chongqing Key Laboratory of Automated Reasoning and Cognition, Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences*

[b] *ORCCA, Western University*

Abstract

A quantifier elimination algorithm by cylindrical algebraic decomposition based on regular chains is presented. The main idea is to refine a complex cylindrical tree until the signs of polynomials appearing in the tree are sufficient to distinguish the true and false cells. We report on an implementation of our algorithm in the `RegularChains` library in Maple and illustrate its effectiveness by examples.

Key words: quantifier elimination, cylindrical algebraic decomposition, regular chains, triangular decomposition

1. Introduction

Quantifier elimination over real closed fields (QE) has been applied successfully to many areas in mathematical sciences and engineering. The following textbooks and journal special issues Hong (1993); Dolzmann et al. (2005); Caviness and Johnson (1998); Basu et al. (2006) demonstrate that QE is one of the major applications of symbolic computation.

It is well known that the worst-case running time for real quantifier elimination is doubly exponential in the number of variables of the input formula, even if there is only

⋆ This research was partly supported by NSFC (11301524, 11471307) and CSTC (cstc2015jcyjys40001, cstc2012ggB40004).

Email addresses: `changbo.chen@hotmail.com` (Changbo Chen), `moreno@csd.uwo.ca` (Marc Moreno Maza).

URLs: `http://www.orcca.on.ca/~cchen` (Changbo Chen), `http://www.csd.uwo.ca/~moreno` (Marc Moreno Maza).

one free variable and all polynomials in the quantified input are linear, see J.H. Davenport and C.W. Brown (Brown and Davenport, 2007). It is also well-known that QE based on Cylindrical Algebraic Decomposition (CAD) has a worst-case doubly exponential running time, even when the number of quantifier alternations is constant, meanwhile other QE algorithms are only doubly exponential in the number of quantifier alternations, see J. Renegar (Renegar, 1992) and S. Basu (Basu, 1999).

Despite of these theoretical results, the practical efficiency and the range of the applications of CAD-based QE have kept improving regularly since G.E. Collins' landmark paper (Collins, 1975). Today, CAD-based QE is available to scientists and engineers thanks to different software namely QEPCAD [1], Mathematica [2], REDLOG [3], SyNRAC [4], `RegularChains` [5].

The work presented here contributes to this effort of making CAD-based QE practically efficient and widely available to the community. The corresponding algorithms were first proposed in our ISSAC 2014 paper Chen and Moreno Maza (2014c). The novelty is the use of the theory of regular chains in the context of QE while the implementation in MAPLE can be freely downloaded at the `RegularChains` library website. This work extends our previous results on CAD, which we summarize now.

In (Chen et al., 2009), together with B. Xia and L. Yang, we presented a different way of computing CADs, based on triangular decomposition of polynomial systems and therefore on the theory of regular chains. Our scheme relies on the concept of *cylindrical decomposition of the complex space* (CCD), from which a CAD can be easily derived. Since regular chain theory is at the center of this new scheme, we call it RC-CAD. Meanwhile, we shall denote by PL-CAD Collins' projection-lifting scheme for CAD construction.

In (Chen and Moreno Maza, 2014a), we substantially improved the practical efficiency of the RC-CAD scheme by means of an incremental algorithm for computing CADs; an implementation of this new algorithm, realized within the `RegularChains` library, outperforms PL-CAD-based solvers on many examples taken from the literature.

The purpose of the present paper is to show that RC-CAD, supported by this incremental algorithm, can serve the purpose of real QE. In addition, our implementation of RC-CAD-based QE is competitive with software implementing PL-CAD-based QE.

We turn our attention to the theoretical implication of performing QE by RC-CAD. If extended Tarski formulae are allowed, then deriving QE from a RC-CAD is a straightforward procedure, hence, we shall not discuss it here. In the rest of this paper, for both input and output of QE problems, only polynomial constraints (with rational number coefficients) will be allowed, thus excluding the use of algebraic expressions containing radicals.

In Collins' original work, the augmented projection operator was introduced in order to find a sufficiently large set of polynomials such that their signs alone could distinguish *true* and *false* cells. In (Hong, 1992), H. Hong produced simple solution formula constructions, assuming that the available polynomials in a CAD were sufficient to generate output formulae.

---

[1] QEPCAD: `http://www.usna.edu/CS/~qepcad/B/QEPCAD.html`

[2] Mathematica: `http://www.wolfram.com/mathematica/`

[3] REDLOG: `http://www.redlog.eu/`

[4] SyNRAC: `http://jp.fujitsu.com/group/labs/techinfo/freeware/synrac/`

[5] `RegularChains`: `http://www.regularchains.org/`

In his PhD thesis (Brown, 1999), C.W. Brown then introduced ways to add polynomials in an incremental manner and proposed a complete algorithm for producing simple formulae.

It was desirable to adapt Brown's ideas to the context of CADs based on regular chains. However, the many differences between the PL-CAD and RC-CAD schemes were making this adaptation challenging. In the PL-CAD scheme, the key data structure is a set P of projection factors, called the *projection factor set*, meanwhile, in the RC-CAD scheme, it is a tree T encoding the associated CCD (cylindrical decomposition of the complex space). Adding a polynomial f to P corresponds to refining T w.r.t. f (as defined by Algorithm 6 in (Chen and Moreno Maza, 2014a)). The PL-CAD-property of a *projection-definable* CAD was another key toward practical efficiency in the work of C.W. Brown (Brown, 1999). Its adaptation to the context of RC-CAD, implies that the signs of polynomials in the tree T suffice to solve the targeted QE problem.

After reviewing in Section 2 the basic notions related to RC-CAD (a complete account of which can be found in (Chen and Moreno Maza, 2014a)) we first adapt in Section 3, the concepts of *projection factor set* and *projection definable*, which were originally introduced by C.W. Brown (Brown, 1999). In Section 4, we first state a "theoretical" solution to the problem of performing quantifier elimination via RC-CAD. Then, after adapting Brown's notion of a *conflicting pair*, we present a more "practical" solution. We view this result as the first main contribution of our work.

Once a quantifier-free formula is obtained, as in the PL-CAD scheme, formula simplification is an important feature. In Section 5, we explain how we do it in the context of RC-CAD. Another desirable optimization is, of course, to obtain running time speedup by slightly weakening the specifications of the QE algorithm or those of one of its subroutines. An instance of that strategy was proposed by H. Hong and M. Safey El Din in (Hong and Safey El Din, 2012). In Section 6, we show that the techniques introduced in (Bradford et al., 2014) for computing truth table invariant CAD (a notion studied by R. Bradford, J.H. Davenport, M. England, S. McCallum, D. Wilson in (Bradford et al., 2013)) can bypass the computation of sign invariant CAD, while retaining the ability to produce truth invariant and projection definable CAD of $\mathbb{R}^k$, thus providing a more efficient but still complete QE procedure. This other contribution of our work is a new development with respect to our ISSAC 2014 paper (Chen and Moreno Maza, 2014c).

In Section 7, we report on our implementation of RC-CAD-based QE using a few examples and comparing it with QEPCAD. Finally, an application of CAD-based QE to automatic generation of parametrized parallel programs is presented in Section 8. This application was suggested by A. Größlinger, M. Griebl and C. Lengauer in (Größlinger et al., 2006) and we use some of the examples proposed in that paper.

2. Preliminary

In this section, we review basic notions related to CAD and in particular RC-CAD, as introduced in (Chen et al., 2009; Chen and Moreno Maza, 2014a). Those will be used throughout the paper.

Zero sets. Let $\mathbf{x} = x_1 \prec \cdots \prec x_n$ be a sequence of n ordered variables. Let $F \subset \mathbb{Q}[\mathbf{x}]$ be finite. Let σ_1 be a map from F to $\{=, \neq\}$ and σ_2 be a map from F to $\{=, \neq, <, >, \leq, \geq\}$. Let $\mathbf{K}$ be the fields $\mathbb{C}$ of complex number or the field $\mathbb{R}$ of real numbers. For $f \in F$

we denote by $Z_{\mathbf{K}}(f)$ the zero set of f in $\mathbf{K}^n$. Denote by $Z_{\mathbb{C}}(f\ \sigma_1(f)\ 0)$ the zero set of $f\ \sigma_1(f)\ 0$ in $\mathbb{C}^n$ and by $Z_{\mathbb{R}}(f\ \sigma_2(f)\ 0)$ the zero set of $f\ \sigma_2(f)\ 0$ in $\mathbb{R}^n$.

Separation (Chen et al., 2009; Chen and Moreno Maza, 2014a). Let C be a subset of $\mathbf{K}^{n-1}$ and $P \subset \mathbb{Q}[x_1 \prec \cdots \prec x_n]$ be a finite set of polynomials, all with the same main variable x_n. We say that P *separates above* C if for each $\alpha \in C$: (i) for each $p \in P$, the polynomial $\mathrm{init}(p)$ (which is the leading coefficient of p w.r.t. to its main variable) does not vanish at α; (ii) the univariate polynomials $p(\alpha, x_n) \in \mathbf{K}[x_n]$, for all $p \in P$, are squarefree and pairwise coprime.

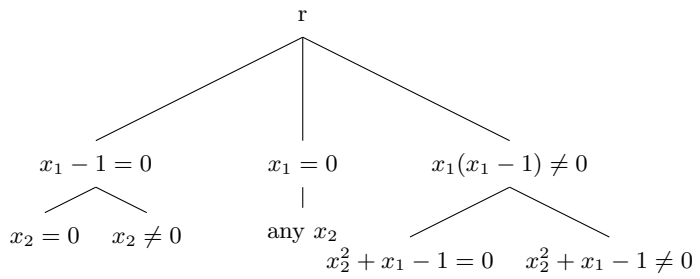
Complex cylindrical tree (CCT) (Chen et al., 2009; Chen and Moreno Maza, 2014a). Let T be a rooted tree of height n where each node of depth i, for $i = 1, \ldots, n$, is labeled by a polynomial constraint of the type "any x_i" (with zero set defined as $\mathbb{C}^n$), or $p = 0$, or $p \neq 0$, where $p \in \mathbb{Q}[x_1, \ldots, x_i]$. For any $i = 1, \ldots, n$, we denote by T_i the induced subtree of T with depth i. Let Γ be a path of T from the root to a leaf[6]. Its zero set $Z_{\mathbb{C}}(\Gamma)$ is defined as the intersection of the zero sets of its nodes. The zero set of T, denoted by $Z_{\mathbb{C}}(T)$, is defined as the union of the zero sets of its paths. We call T a *complete complex cylindrical tree* (complete CCT) of $\mathbb{Q}[x_1, \ldots, x_n]$ whenever it satisfies the following:

- if $n = 1$, then either T has only one leaf which is labeled "any x_1", or, for some $s \geq 1$, it has $s+1$ leaves labeled respectively $p_1 = 0, \ldots, p_s = 0, \prod_{i=1}^{s} p_i \neq 0$, where $p_1, \ldots, p_s \in \mathbb{Q}[x_1]$ are squarefree and pairwise coprime polynomials;
- if $n > 1$, then the induced subtree T_{n-1} of T is a complete CCT and for any given path Γ of T_{n-1}, either its leaf V has only one child in T of type "any x_n", or, for some $s \geq 1$, V has $s+1$ children labeled $p_1 = 0, \ldots, p_s = 0, \prod_{i=1}^{s} p_i \neq 0$, where $p_1, \ldots, p_s \in \mathbb{Q}[\mathbf{x}]$ are polynomials which separate above $Z_{\mathbb{C}}(\Gamma)$.

The set $\{Z_{\mathbb{C}}(\Gamma) \mid \Gamma \text{ is a path of } T\}$ is called a *complex cylindrical decomposition* (CCD) of $\mathbb{C}^n$ associated with T. Note that for a complete CCT, we have $Z_{\mathbb{C}}(T) = \mathbb{C}^n$. A proper subtree rooted at the root node of T of depth n is called a *partial CCT* of $\mathbb{Q}[x_1, \ldots, x_n]$. We use CCT to refer to either a complete or partial CCT.

Let $F \subset \mathbb{Q}[\mathbf{x}]$ be finite. Let Γ be a path of a CCT T. Note that the polynomial constraints along Γ form a regular system, called the *associated regular system* of Γ. Let $p \in F$. We say p is *sign-invariant* on Γ if either $Z_{\mathbb{C}}(\Gamma) \subset Z_{\mathbb{C}}(p)$ or $Z_{\mathbb{C}}(\Gamma) \cap Z_{\mathbb{C}}(p) = \emptyset$ holds. We say that p is sign-invariant on T if p is sign-invariant on every path of T. We say T is sign-invariant w.r.t. F if each $p \in F$ is sign-invariant on T. The procedure CylindricalDecompose introduced in (Chen et al., 2009; Chen and Moreno Maza, 2014a) takes F as input and builds a CCT T which is sign-invariant w.r.t. F.

Example 1. The following tree T is a CCT.



[6] In the rest of this paper, a path of a tree always means a path from the root to a leaf.

Let $p := x_1(x_2^2 + x_1 - 1)$. Then p is sign-invariant on T.

Next we review the notion of cylindrical algebraic decomposition (CAD) of $\mathbb{R}^n$ and highlight the relation between CAD and CCT. Let π_{n-1} be the standard projection from $\mathbb{R}^n$ to $\mathbb{R}^{n-1}$ that maps $(x_1, \ldots, x_{n-1}, x_n)$ onto $(x_1, \ldots, x_{n-1})$.

Stack over a connected semi-algebraic set. Let S be a connected semi-algebraic subset of $\mathbb{R}^{n-1}$. The *cylinder* over S in $\mathbb{R}^n$ is defined as $Z_{\mathbb{R}}(S) := S \times \mathbb{R}$. Let $\theta_1 < \cdots < \theta_s$ be continuous semi-algebraic functions defined on S. Denote $\theta_0 = -\infty$ and $\theta_{s+1} := \infty$. The intersection of the graph of θ_i with $Z_{\mathbb{R}}(S)$ is called the *θ_i-section* of $Z_{\mathbb{R}}(S)$. The set of points between θ_i-section and θ_{i+1}-section, $0 \leq i \leq s$, of $Z_{\mathbb{R}}(S)$ is a connected semi-algebraic subset of $\mathbb{R}^n$, called a *(θ_i, θ_{i+1})-sector* of $Z_{\mathbb{R}}(S)$. The sequence (θ_0, θ_1)-sector, θ_1-section, (θ_1, θ_2)-sector, ..., θ_s-section, (θ_s, θ_{s+1})-sector form a disjoint decomposition of $Z_{\mathbb{R}}(S)$, called a *stack* over S, which is uniquely defined for given functions $\theta_1 < \cdots < \theta_s$.

Cylindrical algebraic decomposition (CAD). A finite partition $\mathcal{D}$ of $\mathbb{R}^n$ is called a *cylindrical algebraic decomposition* (CAD) of $\mathbb{R}^n$ if one of the following properties holds.

- either $n = 1$ and $\mathcal{D}$ is a stack over $\mathbb{R}^0$.
- or the set of $\{\pi_{n-1}(D) \mid D \in \mathcal{D}\}$ is a CAD of $\mathbb{R}^{n-1}$ and each $D \in \mathcal{D}$ is a section or sector of the stack over $\pi_{n-1}(D)$.

When this holds, the elements of $\mathcal{D}$ are called *cells*. The set $\{\pi_{n-1}(D) \mid D \in \mathcal{D}\}$ is called the *induced* CAD of $\mathcal{D}$. A CAD $\mathcal{D}$ of $\mathbb{R}^n$ can be encoded by a tree, called a CAD tree (denoted by RT), as below. The root node, denoted by r, is $\mathbb{R}^0$. The children nodes of r are exactly the elements of the stack over $\mathbb{R}^0$. Let RT_{n-1} be a CAD tree of the induced CAD of $\mathcal{D}$ in $\mathbb{R}^{n-1}$. For any leaf node C of RT_{n-1}, its children nodes are exactly the elements of the stack over C.

Delineability. Let p be a polynomial of $\mathbb{R}[x_1, \ldots, x_n]$ and let S be a connected semi-algebraic set of $\mathbb{R}^{n-1}$. We say that p is *delineable* on S if the real zeros of p define continuous semi-algebraic functions $\theta_1, \ldots, \theta_s$ such that, for all $\alpha \in S$ we have $\theta_1(\alpha) < \cdots < \theta_s(\alpha)$. Note that if p is delineable on S, then the real zeros of p uniquely define a stack over S.

We recall the following Theorem introduced in (Chen et al., 2009).

Theorem 1. Let $P = \{p_1, \ldots, p_r\}$ be a finite set of polynomials in $\mathbb{Q}[x_1 \prec \cdots \prec x_n]$ with the same main variable x_n. Let S be a connected semi-algebraic subset of $\mathbb{R}^{n-1}$. If P *separates* above S, then each p_i is delineable on S. Moreover, the product of the $p_1, \ldots, p_r$ is also delineable on S.

Note that the real zeros of the polynomials in P uniquely define a stack over S. Let T be a CCT of $\mathbb{Q}[x_1 < \cdots < x_n]$. By Theorem 1, the polynomials in the induced CCT T_1 uniquely defines a stack over $\mathbb{R}^0$, which can be represented by a CAD tree of $\mathbb{R}^1$. Such a CAD tree is called the *CAD tree derived* from T_1. Continuing in an inductive manner, let RT_{n-1} be the CAD tree derived from T_{n-1}. Let Γ be a path of T_{n-1}. The zero set of Γ in $\mathbb{R}^{n-1}$ is a disjoint union of cells in RT_{n-1}. Let C be one of them. Let P be the set of polynomials appearing in the children of Γ. By Theorem 1, the real zero set of the polynomials in P uniquely defines a stack over C. The union of all these stacks uniquely determine a CAD tree RT of $\mathbb{R}^n$, called the *CAD tree derived* from T. We call T the *associated CCT* of RT. The procedure MakeSemiAlgebraic in (Chen et al., 2009; Chen and Moreno Maza, 2014a) builds the derived CAD tree RT from a complete CCT T.

Quantifier elimination. Let $f \in \mathbb{Q}[x_1 \prec \cdots \prec x_n]$. Let $\sigma \in \{<, >, \leq, \geq, =, \neq\}$. We call a formula of the form $f\ \sigma\ 0$ a (standard) atomic formula. Let $FF(x_1, \ldots, x_n)$ be a quantifier-free formula in disjunctive normal form (DNF). For an integer $k = 0, \ldots, n-1$, let

$$PF := (Q_{k+1}x_{k+1} \cdots Q_n x_n)FF(x_1, \ldots, x_n)$$

be a prenex formula, where Q_i is either the universal quantifier $\forall$ or the existential quantifier $\exists$, for all for $k+1 \leq i \leq n$. A quantifier-free formula SF in $\mathbb{Q}[x_1, \ldots, x_k]$ which is equivalent to PF is called a *solution formula* of PF. A process obtaining SF from PF is called quantifier elimination (QE).

Style of the pseudo-code. The algorithms stated in the sequel of this paper follow the same style as that used in (Chen and Moreno Maza, 2014a). In particular and since attributes of nodes (like *c.derivative* in Algorithm 3) are used in our pseudo-code, we adhere to the standard "dot" notation of object oriented programming languages. Another important feature of our pseudo-code is the fact that certain procedures may modify their input arguments. For instance, Algorithm 3 will modify its input cylindrical tree T, unless the children of Γ_{k-1} are derivative closed. The recursive nature of our data-structures and algorithms motivates this feature. Efficient considerations, discussed in (Chen and Moreno Maza, 2014a), are other motivations.

3. Theory

To present our algorithm in Section 4, we revisit below the concepts of *projection factor set* and *projection definable*, which were originally introduced in (Brown, 1999).

Projection factor set. Let T be a complete CCT in $\mathbb{Q}[\mathbf{x}]$. Let V be a node in T, different from the root-node. Let $V_1, \ldots, V_s$ be all the siblings of V which are labeled by equational constraints; this includes V itself if it is labeled by an equational constraint. Assume that V_i is of the form $f_i = 0$. The set $\{f_1, \ldots, f_s\}$ is called the *projection factor set* of V. Let Γ be a path of T. The union of the projection factor sets of all the nodes along Γ is called a *projection factor set* of Γ. The projection factor set of T is then defined as the union of the projection factor sets of all its paths. Let RT be the CAD tree derived from T. Its projection factor set is defined as that of T. Let C be a cell of RT derived from a path Γ of T. The projection factor set of C is defined as that of Γ.

Projection definable. Let RT be a CAD tree attached with truth values. We call a quantifier free formula SF a *solution formula* for RT if SF defines the same semi-algebraic set as the union of all cells of RT whose attached truth values are true. The tree RT is called *projection definable* if there exists a solution formula formed by the signs of polynomials in its projection factor set. Let T be a complete CCT. Let RT be the CAD tree derived from T. We say that T is *projection definable* if RT is projection definable no matter which truth values are attached to RT.

Remark. The concept of projection factor set has different meanings for PL-CAD and RC-CAD. To see this, let RT be a CAD tree and P be its projection factor set. If P is a projection factor set in the PL-CAD sense, any polynomial in P is sign invariant above any cell of RT. This is not necessarily true for RC-CAD. To be precise, let T be the associated complex cylindrical tree of RT and let Γ be a path of T. Let C be any CAD cell derived from Γ. It is guaranteed that any polynomial in the projection factor set P_Γ of Γ is sign invariant above C. However, it is possible that a polynomial of P_Γ is not sign invariant above the CAD cells derived from other paths of T. See below for an example.

Example 2. Let T be the CCT in Example 1. Let RT be the CAD tree derived from T. Let Γ be the right most path of T. The projection factor set of Γ is $\{x_1, x_1-1, x_2^2+x_1-1\}$. The projection factor set of T and RT is $\{x_1, x_1-1, x_2^2+x_1-1, x_2\}$. We notice that neither x_2 nor $x_2^2+x_1-1$ is sign-invariant on the path of T consisting of nodes $\{r, x_1 = 0, \text{ any } x_2\}$. Moreover, it is easy to verify that T and RT are projection definable.

Let $p := x^2 - 2$, whose zero set naturally defines a CAD tree RT of $\mathbb{R}^1$. Suppose that the only true cell of RT is either $x = -\sqrt{2}$ or $x = \sqrt{2}$ (but not both). Then RT is not projection definable.

Definition 1. Let F be a set of non-constant univariate polynomials in $\mathbb{R}[x]$. We say F is *derivative closed* (w.r.t. factorization) if for any $f \in F$, where $\deg(f) > 1$, the polynomial $\mathrm{der}(f)$ is a product of some polynomials in F and some constant $c \in \mathbb{R}$.

Let $F \subset \mathbb{R}[\mathbf{x}]$ be finite. Let σ be a map from F to $\{<, >, =\}$. Let $F_\sigma := \wedge_{f\in F} f\ \sigma(f)\ 0$. Define $Z_{\mathbb{R}}(F_\sigma) := \cap_{f\in F} Z_{\mathbb{R}}(f\ \sigma(f)\ 0)$.

The formulation of Thom's lemma presented in Lemma 1 is slightly different from its original version (Coste and Roy, 1988), although the proof relies on exactly the same arguments. Such a formulation is often implicitly used in implementations related to Thom's Lemma. An explicit treatment can be found, for example, in (Brown, 1999; Xiao, 2009).

Lemma 1 (Thom's Lemma (Coste and Roy, 1988))**.** Assume that $n = 1$. If F is derivative closed (w.r.t. factorization), then the set defined by F_σ is either an empty set, a point or an open interval.

Proof. Let m be the number of polynomials in F. We prove the claim by induction on m. If $m = 1$, let f be the only polynomial in F. Then $\deg(f) = 1$ and the claim obviously holds. Assume now that $m \geq 2$ holds. Let f be a polynomial in F with maximum degree. Then $F^* := F \setminus \{f\}$ is also derivative closed (w.r.t. factorization). By induction, $Z_{\mathbb{R}}(F^*_\sigma)$ is either the empty set, a point or an open interval. If $Z_{\mathbb{R}}(F^*_\sigma)$ is the empty set or a point, then the claim clearly holds. Now we assume that $Z_{\mathbb{R}}(F^*_\sigma)$ defines an open interval. Then $\mathrm{der}(f) \neq 0$ at any point of $Z_{\mathbb{R}}(F^*_\sigma)$. In other words, f is monotone above $Z_{\mathbb{R}}(F^*_\sigma)$. Thus $Z_{\mathbb{R}}(F^*_\sigma) \cap Z_{\mathbb{R}}(f\ \sigma(f)\ 0)$ is empty, a point or an open interval. □

Lemma 2. Let C be a region of $\mathbb{R}^{n-1}$. let F be a set of polynomials in $\mathbb{Q}[x_1 \prec \cdots \prec x_n]$ with the same main variable x_n. We assume that F separates above C and that for each point α of C, the set of univariate polynomials $\{p(\alpha, x_n) \mid p \in F\}$ is derivative closed (w.r.t. factorization). Then, the set $C \times \mathbb{R}^1 \cap Z_{\mathbb{R}}(F_\sigma)$ is either empty, or a section, or a sector above C.

Proof. Assume that $C \times \mathbb{R}^1 \cap Z_{\mathbb{R}}(F_\sigma)$ is not empty. Since F separates above C, by Theorem 1, F is delineable above C. Thus $C \times \mathbb{R}^1 \cap Z_{\mathbb{R}}(F_\sigma)$ is either a union of sections or a union of sectors. The former (resp. the latter) holds if and only if there exist at least one (resp. no) equational formulae in F_σ.

Let α be a point of C. Denote $F_\sigma(\alpha) := Z_{\mathbb{R}}(F(\alpha)_\sigma)$. Since $\{p(\alpha, x_n) \mid p \in F\}$ is derivative closed (w.r.t. factorization), by Thom's Lemma, the set $F_\sigma(\alpha)$ is either a point belonging to a section or an open interval contained in a sector. If C has no other points, the theorem clearly holds. If $F_\sigma(\alpha)$ is a point belonging to a section S, it is enough to prove that for any other given point of C, say β, $F_\sigma(\beta)$ belongs to the same section as

$F_\sigma(\alpha)$. Assume that this does not hold, since $F_\sigma(\beta)$ is non-empty by delineability, then $F_\sigma(\beta)$ belongs to a sector or is contained in a different section, say S'. Since S' is a connected semi-algebraic set, there exists $x_n^*, x_n^{**} \in \mathbb{R}$ such that $(\alpha, x_n^*), (\beta, x_n^{**}) \in S'$, and both $F(\alpha, x_n^*)_\sigma$ and $F(\beta, x_n^{**})_\sigma$ are true. This contradicts to the fact that $F_\sigma(\alpha)$ belongs to S. By similar arguments, we can prove that the theorem also holds when $F_\sigma(\alpha)$ is an open interval. □

4. Algorithm

In this section, we explain how to perform quantifier elimination via RC-CAD. Our top-level procedure is Algorithm 1 which lists the main steps of QE based on RC-CAD.

Algorithm 1: QuantifierElimination(PF)

Input: A prenex formula $PF := (Q_{k+1}x_{k+1} \cdots Q_n x_n)FF(x_1, \ldots, x_n)$.
Output: A solution formula of PF.

```
1 begin
2     Let F be the set of polynomials appearing in FF;
3     T := CylindricalDecompose(F);
4     RT := MakeSemiAlgebraic(T);
5     AttachTruthValue(FF, RT);
6     PropagateTruthValue(PF, RT);
7     MakeProjectionDefinable_k(PF, RT);
8     SF := GenerateSolutionFormula_k(RT_k);
```

Recall that QE based on PL-CAD consists of three phases: projection, stack construction and solution formula generation. The steps of Algorithm 1 can also be grouped into these three phases:

- Projection: Line 3 of Algorithm 1.
- Stack construction: Lines 4 to 6 of Algorithm 1.
- Solution formula generation: Lines 7 and 8 of Algorithm 1.

However, these phases do not correspond exactly to those of QE based on RC-CAD. Indeed, the projection phases of RC-CAD-based QE and PL-CAD-based QE compute respectively a complex cylindrical tree and a projection factor set. For the stack construction phase, the procedure MakeSemiAlgebraic (specified in Section 2) of RC-CAD-based QE is different from the real root isolation routines used by PL-CAD-based QE.

The other two steps, namely AttachTruthValue and PropagateTruthValue, are similar for both QE algorithms. We recall these two steps in the case of RC-CAD-based QE. The operation AttachTruthValue takes a DNF formula FF and a CAD tree RT as input. For each path A of RT, it assigns $A.leaf.truthvalue$ the truth value of FF evaluated at $A.leaf.samplepoint$.

The operation PropagateTruthValue takes as input: (1) a prenex formula $PF := (Q_{k+1}x_{k+1} \cdots Q_n x_n)FF(x_1, \ldots, x_n)$ and (2) a CAD tree RT, each leaf V of which is attached with a truth value of FF evaluated at $V.samplepoint$; then it outputs the induced subtree RT_k of RT in $\mathbb{Q}[x_1, \ldots, x_k]$ such that each leaf V_k of RT_k is attached with a truth value of PF evaluated at $V_k.samplepoint$. This operation proceeds as follows. It starts by checking whether Q_n is the universal quantifier $\forall$ or the existential quantifier

$\exists$. For each leaf V of RT_{n-1}, if Q_n is $\forall$, then $V.truthvalue$ is set to true if and only if the truth values of the children of V are all true; if Q_n is $\exists$, then $V.truthvalue$ is set to true if and only if the truth value of at least one child of V is true. The operation then makes recursive call with $(Q_{k+1}, \ldots, Q_{n-1})$ and RT_{n-1} as input and terminates when all Q_i, for $i = k+1, \ldots, n$, have been examined.

The third phase, namely the solution formula generation, is the main focus of this section. If the CAD tree RT is projection definable (see Section 3), then the solution formula generation is straightforward, see Algorithm 2. If RT is not projection definable,

Algorithm 2: GenerateSolutionFormula$_k(RT)$

Input: A projection definable CAD tree RT in $\mathbb{Q}[x_1 \prec \cdots \prec x_k]$ attached with truth values.

Output: A solution formula for RT.

```
1 begin
2     D := ∅;
3     for each cell c of RT whose truth value is true do
4         let P_c be the projection factor set of c;
5         evaluate the polynomials in P_c at a sample point of c and determine their
          signs;
6         let D_c be the resulting conjunction formula;
7         D := D ∪ {D_c}
8     return the disjunction of conjunction formulae in D
```

we will present two strategies to address this situation.

The first one, presented in Algorithm 5, is a theoretical solution. This is an adaptation to RC-CAD of the augmented projection, widely used in PL-CAD. The second one, presented in Algorithm 6 and used in our implementation, is a much more practical solution. This is an adaptation to RC-CAD of the technique of making CAD projection definable, used in PL-CAD (Brown, 1999). Our adaptation is based on the IntersectPath operation of introduced in (Chen and Moreno Maza, 2014a). This operation takes as input a polynomial p, a CCT T and a path Γ of T, then returning a CCT refining of T such that p is sign-invariant on each path derived from Γ. Another operation `NextPathToDo` from (Chen and Moreno Maza, 2014a) is also used in our algorithm. It takes a CCT T as input, for a fixed traversal order of the tree T, returning the first "to-be-visited" path Γ of T. To better state the algorithms, we introduce the following notion.

Definition 2. Let T_k be a complete CCT of $\mathbb{Q}[x_1, \ldots, x_k]$. Let T_{k-1} be the induced subtree of T_k in $\mathbb{Q}[x_1, \ldots, x_{k-1}]$. Let Γ_{k-1} be a path of T_{k-1}. Let $c_1, \ldots, c_s$ be all the equation children of $\Gamma_{k-1}.leaf$ in T_k and let $f_1, \ldots, f_s$ be polynomials such that c_i is $f_i = 0$, for $i = 1, \ldots, s$. We say that the children of Γ_{k-1} are *derivative closed* if

- either we have $k = 1$ and the set of polynomials $\{f_i \mid i = 1, \ldots, s\}$ is derivative closed,
- or $k \geq 2$ and for any $\alpha \in Z_{\mathbb{C}}(\Gamma_{k-1})$, the set of univariate polynomials $\{f_i(\alpha, x_k) \mid i = 1, \ldots, s\}$ is derivative closed.

Proceeding by induction, we say that the tree T_k is *derivative closed* if

- either $k = 0$,

- or $k \geq 1$, T_{k-1} is derivative closed and the children of every path Γ_{k-1} of T_{k-1} are derivative closed.

Lemma 3. Let T_k be a complete CCT of $\mathbb{Q}[x_1, \ldots, x_k]$. If T_k is derivative closed, then T_k is projection definable.

Proof. Let RT_k be the CAD tree derived from T_k. By definition, to prove that T_k is projection definable, it is equivalent to show that for an arbitrary truth value assignment to the cells of RT_k, RT_k is projection definable. If RT_k has no true cells, then RT_k is clearly projection definable. Assume RT_k has at least one true cell and let C be one of them. Let Γ be the path of T_k such that C is derived from Γ. Let $P = \{p_1, \ldots, p_m\}$ be the projection factor set of Γ. Since P is sign invariant above C, there exists a map σ from P to $\{>, <, =\}$ such that $C \subseteq Z_{\mathbb{R}}(\wedge_{i=1}^{m} p_i \ \sigma(p_i)\ 0)$. On the other hand, since T_k is derivative closed, by Definition 2 and Lemma 2, we deduce that $C = Z_{\mathbb{R}}(\wedge_{i=1}^{m} p_i \ \sigma(p_i)\ 0)$. Thus RT_k is projection definable. □

Algorithm 3: $\mathsf{RefineNextChild}_k(\Gamma_{k-1}, T)$

Input: A cylindrical tree T in $\mathbb{Q}[x_1 \prec \cdots \prec x_n]$. A path Γ_{k-1} of T_{k-1} in $\mathbb{Q}[x_1 \prec \cdots \prec x_n]$.

Output: If the children of Γ_{k-1} are derivative closed, return false. Otherwise, some progress is made to guarantee that the children of Γ_{k-1} become derivative closed after this algorithm is called finitely many times.

```
1  begin
2      V := Γ_{k-1}.leaf;
3      let S be the set of children c of V in T_k of the form f = 0, for f ∈ Q[x_1, ..., x_k],
       with deg(f) > 1 such that c.derivative is undefined;
4      if S = ∅ then return false ;
5      let c ∈ S such that deg(f) is minimum;
6      c.derivative := der(f, x_k);
7      let Γ_k be the subtree of T_k which induces Γ_{k-1};
8      while C := NextPathToDo_k(Γ_k \ (Γ_{k-1} ∪ c)) do
9          IntersectPath_k(der(f, x_k), C, T_k);
10     return true;
```

Proposition 1. Algorithms 3 and 4 satisfy their specifications.

Proof. By Lemma 3, in order to prove Algorithm 4, it suffices to show that, when Algorithm 4 returns, the refinement of T_k is derivative closed. To do this, we proceed by induction on k; as a byproduct, we also prove Algorithm 3.

Algorithm 4 clearly holds for $k = 0$. Next, we assume that $\mathsf{RefineTree}_{k-1}(T)$ makes T_{k-1} derivative closed. Hence, following the pseudo-code of Algorithm 4, we assume that Line **7** executes as specified. Then, it suffices to show that, when Algorithm 4 terminates, the children of each path Γ_{k-1} of T_{k-1} are derivative closed.

We can assume that, before making the initial call to Algorithm 3, the flag *c.derivative* is unassigned, for each child c of $\Gamma_{k-1}.leaf$. After that, each time Algorithm 3 is called

Algorithm 4: $\mathsf{RefineTree}_k(T)$

Input: A complete complex cylindrical tree T of $\mathbb{Q}[x_1 \prec \ldots \prec x_n]$.
Output: Refine T to make its induced subtree T_k projection definable.

```
1 begin
2     if k = 0 then return;
3     while Γ := NextPathToDo_{k-1}(T) do
4         todo := true;
5         while todo do
6             todo := RefineNextChild_k(Γ, T);
7     RefineTree_{k-1}(T);
```

and does not return false, one chooses a vertex c of the form $f = 0$, where $c.derivative$ is unassigned and $\deg(f, x_k)$ is minimum. By calling the operation $\mathsf{IntersectPath}$, the child nodes of Γ_{k-1} are refined into new ones, above each of which $\mathrm{der}(f, x_k)$ becomes sign invariant. Let $c_1, \ldots, c_s$ be all the equation children of $\Gamma_{k-1}.leaf$ after this refinement process is completed; assume that c_i is of the form $f_i = 0$. The sign invariance of $\mathrm{der}(f, x_k)$ above each c_i implies that for any α of $Z_{\mathbb{R}}(\Gamma_{k-1})$, the polynomial $\mathrm{der}(f, x_k)(\alpha, x_k)$ is (up to a constant multiplicative factor) a product of some of the polynomial $f_i(\alpha, x_k)$.

Finally, observe that each time a vertex c is chosen (and $c.derivative$ turns assigned), for any new child c_i (of the form $f_i = 0$) of $\Gamma_{k-1}.leaf$ added by Algorithm 3, we have $\deg(f_i, x_k) < deg(f, x_k)$. It follows that Algorithm 3 will return false after being called finitely many times. When false is returned, by Definition 2, the children of Γ_{k-1} are clearly derivative closed. □

Note that Algorithm 4 may generate much more polynomials than necessary for the purpose of solution formula generation. Nevertheless, this algorithm leads to a simple solution for the problem of refining a given CAD into one which is projection definable. Algorithm 5 states such a solution.

Algorithm 5: $\mathsf{MakeProjectionDefinable}_k(PF, RT, theoretical)$

Input: $PF := (Q_{k+1}x_{k+1} \cdots Q_n x_n)FF(x_1, \ldots, x_n)$ is a prenex formula. An FF-invariant CAD tree RT of $\mathbb{Q}[x_1 \prec \cdots \prec x_n]$, each k-level cell of which is attached with a truth value of PF.
Output: Refine RT to make its induced subtree RT_k projection definable.

```
1 begin
2     Let T be the associated CCT of RT;
3     RefineTree_k(T);
4     RT := MakeSemiAlgebraic(T);
5     AttachTruthValue(FF, RT);
6     PropagateTruthValue(PF, RT);
```

We turn our attention now to a more practically efficient strategy for making a CAD tree projection definable. To this end, we revise the notion of "conflicting pair", which was initially introduced for PL-CAD in (Brown, 1999).

Definition 3 (Conflicting pair)**.** Let RT_k be a CAD tree of $\mathbb{R}^k$ attached with truth values. Let T_k be the associated CCT of RT_k. For $1 \leq i \leq k$, we call two distinct i-level cells C_i and D_i in the same stack an *i-level conflicting pair* (or simply a *conflicting pair*) if there exist k-level cells C and D such that

(CP_1) C_i and D_i are respectively the projections of C and D onto $\mathbb{R}^i$,

(CP_2) C and D are derived from the same path of T_k,

(CP_3) above C and D, every polynomial in their common projection factor set P has the same sign,

(CP_4) C and D have opposite attached truth values.

Remark 1. Let C and D be two k-level cells satisfying (CP_2), (CP_3) and (CP_4). Let A be the lowest common ancestor of C and D, that is, the ancestor in RT_k of C and D of the largest level, say $i-1$ in RT_k. Let C_i and D_i be the respective ancestors of C and D of level i. Clearly, the cells C_i and D_i share the same parent A and form an i-level conflicting pair. We call C_i, D_i the *conflicting pair associated with* C, D. We call the pair C, D an *extension* of the pair C_i, D_i in RT_k. Note that for PL-CAD, only the conditions (CP_1), (CP_3) and (CP_4) are required. In (Brown, 1999), it was proved that a CAD is projection definable if and only if it contains no conflicting pairs. Motivated by this result, we propose Algorithm 6.

Algorithm 6: MakeProjectionDefinable$_k(PF, RT, practical)$

Input: Same as Algorithm 5.
Output: Same as Algorithm 5.

```
1  begin
2      let CPS be the set of all conflicting pairs of RT_k;
3      while CPS ≠ ∅ do
4          let CP be a pair in CPS of highest level, say i;
5          let T be the associated CCT of RT;
6          let Γ be the path of T_i, where CP is derived;
7          call RefineNextChild_i(Γ_{i-1}, T) to refine T;
8          RT := MakeSemiAlgebraic(T);
9          AttachTruthValue(FF, RT);
10         PropagateTruthValue(PF, RT);
11         let CPS be the set of all conflicting pairs of RT;
```

Theorem 2. Algorithm 6 constructs a projection definable CAD tree.

Proof. The correctness and termination of Algorithm 6 will result from the following two claims:

(1) If $CPS = \emptyset$, then RT_k is projection definable.

(2) CPS becomes empty after finitely many steps.

We first prove (1). Note that $CPS = \emptyset$ implies that no pairs of cells C, D in RT_k satisfy (CP_i), $i = 2, 3, 4$. In other words, for any two cells C, D derived from the same complex path (thus having the same projection factor set, say P) and attached with

different truth-values, then the signs of the polynomials in P are sufficient to distinguish C from D. Therefore, (1) clearly holds.

Next, we prove (2). We assume that CPS is not empty. We observe that (2) results from the conjunction of the following properties holds. Let i be the highest level of a conflicting pair in RT_k, as defined at Line **4** of Algorithm 6.

(2.1) The children of any path Γ_{i-1} in T_k will become derivative closed after finitely many iterations of the while-loop in Algorithm 6.

(2.2) When the children of any Γ_{i-1} in T_k become derivative closed, there will exist no conflicting pairs of level i.

(2.3) When RT gets refined by executing Lines **7** and **8**, no conflicting pairs of level higher than i are generated.

Property (2.1) and the correctness of Algorithm 4 are essentially the same fact, which was proved with Proposition 1. We prove Property (2.2) by contradiction. If (2.2) does not hold, then there exist a path Γ_{i-1} and two cells C_i, D_i such that C_i and D_i are derived from some child of Γ_{i-1} and every i-level polynomial in their projection factor set has the same sign on both C_i and D_i. This is a contradiction to Lemma 2.

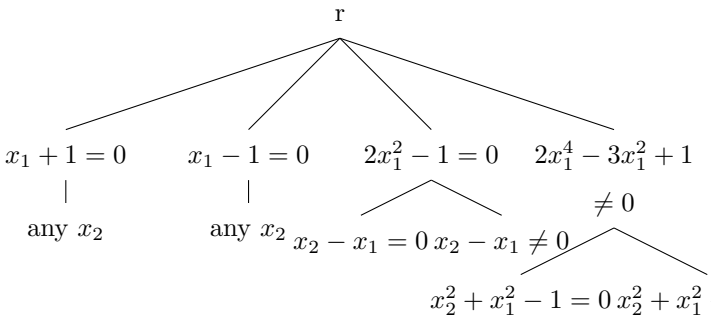
Next we prove (2.3). Assume that RT refines into a new tree RT'. Let C'_j and D'_j be a j-level conflicting pair in RT'_k. Let C' and D' be their extension in RT'_k. Let C and D be two cells of RT_k such that C' is derived from C and D' is derived from D. Note that C and D satisfy (CP_2), (CP_3) and (CP_4). Moreover, the projection of C and D onto $\mathbb{R}^j$ must have the same parent. Thus there exists a conflicting pair associated with C and D in RT_k of level at least j. Since the highest level of conflicting pairs in RT_k is i, (2.3) is proved. □

Proposition 2. For a prenex formula $(Q_{k+1}x_{k+1}\cdots Q_n x_n)FF(x_1,\ldots,x_n)$, Algorithm 1 returns its solution formula.

Proof. It follows directly from the specification of all its subroutines. □

We conclude this section by illustrating our algorithm with two examples. The first example is modified from the one in (Hong, 1990; Brown, 1999), where it was used to demonstrate that PL-CAD based QE may generate a CAD tree that is not projection definable.

Example 3. Let $PF := (\exists x_2)\ (x_1^2 + x_2^2 - 1 = 0) \wedge (x_1 + x_2 < 0) \wedge (x_1 > -1) \wedge (x_1 < 1)$. The projection stage generates the following CCT T:



The stack construction stage computes a CAD tree RT of $\mathbb{Q}[x_1, x_2]$. The induced CAD tree of RT in $\mathbb{Q}[x_1]$ has the following cells $(-\infty, -1)$, -1, $(-1, -\frac{\sqrt{2}}{2})$, $-\frac{\sqrt{2}}{2}$, $(-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$,

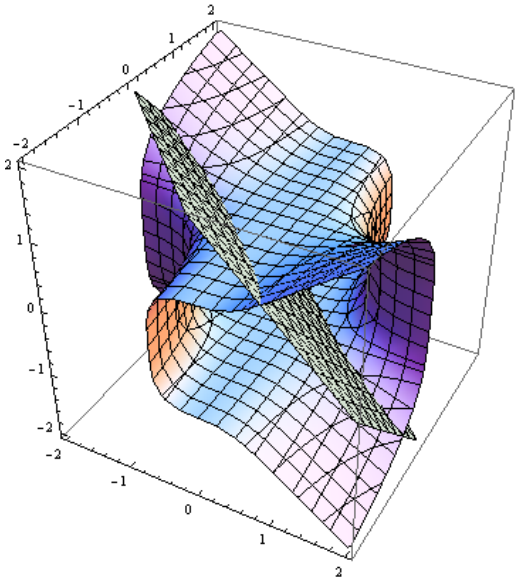


Fig. 1. Plots of $2z^4 + 2x^3y - 1 = 0$ and $x + y + z = 0$.

$\frac{\sqrt{2}}{2}$, $(\frac{\sqrt{2}}{2}, 1)$, 1, and $(1, +\infty)$. Among them, the true cells are the third, the fourth and the fifth cells. The cells $-\frac{\sqrt{2}}{2}$ and $\frac{\sqrt{2}}{2}$ is a conflicting pair. The two cells are derived from the complex path $\Gamma := [r, 2x_1^2 - 1 = 0]$ of T_1. Refining Γ w.r.t. the derivative of $2x_1^2 - 1 = 0$ generates a projection definable CCT, which allows us to obtain the solution formula of PF:

$$(x_1 = 0) \vee (x_1 < 0 \wedge 0 < x_1 + 1) \vee (0 < x_1 \wedge 2x_1^2 < 1).$$

Example 4. Let $f := 2z^4 + 2x^3y - 1$ and $h := x + y + z$. Consider the quantifier elimination problem $(\exists\ z)(f < 0 \wedge h < 0 \wedge x > 0)$. The plots of $f = 0$ and $h = 0$ are depicted in Fig. 1.

The solution set is the blue region in Fig. 2, where the red curve is the locus of $p := 2x^4 + 10x^3y + 12x^2y^2 + 8xy^3 + 2y^4 - 1$, that is, the resultant of f and h w.r.t. z. The solution set is exactly the set of (x, y) such that $x > 0$ and $y < \mathit{RealRoot}_2(p, y)$. Obviously, this region cannot be described just by the sign of p.

To describe the blue region by a QFF, the derivative of p, namely $q := 10x^3 + 24x^2y + 24xy^2 + 8y^3$, is introduced. The locus of q is the blue curve in Fig. 3. Note that the blue region is the union of the green region $(x > 0 \wedge q < 0)$, the blue curve $(x > 0 \wedge q = 0)$ and the pink region $(x > 0 \wedge p < 0 \wedge q > 0)$. Thus the solution formula of the QE problem is

$$(x > 0 \wedge q < 0) \vee (x > 0 \wedge q = 0) \vee (x > 0 \wedge p < 0 \wedge q > 0).$$

5. Construction of simpler QFFs

In Section 4, a complete quantifier elimination algorithm was presented. By "complete", we mean that this algorithm can always generate a solution formula. In the

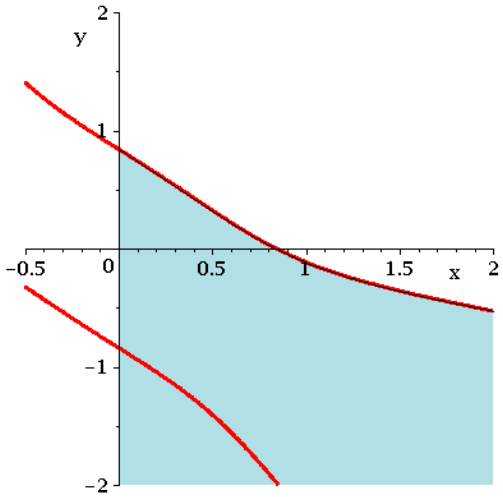


Fig. 2. Projection of $f < 0 \wedge h < 0 \wedge x > 0$ onto (x, y) space.

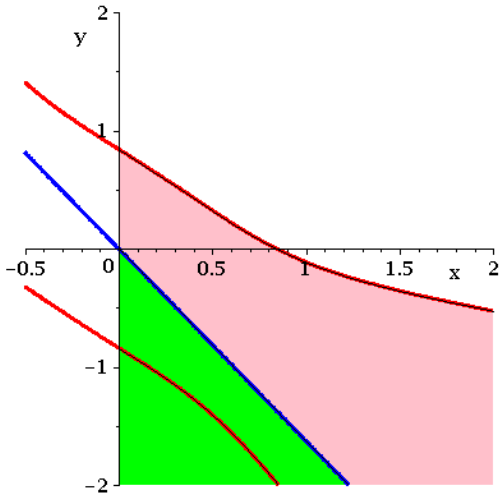


Fig. 3. The zero sets of $x > 0 \wedge q < 0$ (green region), $x > 0 \wedge q = 0$ (blue curve) and $x > 0 \wedge p < 0 \wedge q > 0$ (pink region).

present section, we develop a heuristic strategy improving Algorithm 2 such that simpler solution formulae can be generated at reasonable cost.

Let RT be a CAD tree of $\mathbb{R}^n$ and let P be the associated projection factor set of RT in the sense of PL-CAD. One key reason why PL-CAD succeeds in generating simple solution formulae is that every polynomial in P is guaranteed to be sign-invariant on each cell of RT (assuming that techniques based on partially built CAD are not used).

Let T be a CCT of $\mathbb{Q}[\mathbf{x}]$ and let RT be a CAD tree deduced from T. Let Γ be a path of T and let S be a subset of cells of RT derived from Γ. Let P_Γ be the projection factor set of Γ. It is known that any polynomial p of P_Γ is sign-invariant on each cell

of S, but p may not be sign-invariant on other cells of RT. For instance, in Example 1, the polynomial x_2 is not sign-invariant on CAD cells derived from the third path of the CCT.

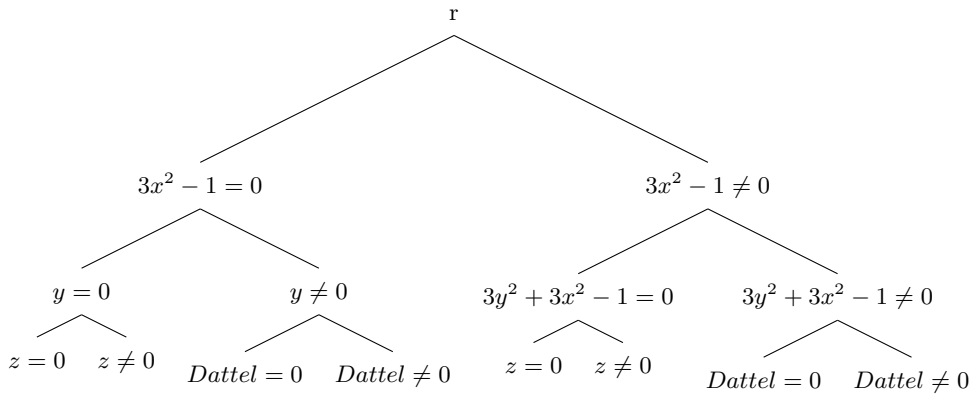
On the other hand, let Γ be the right most path of T, we observe that in many cases, the polynomials in P_Γ are sign-invariant on each path of T, and thus also sign-invariant on every cell of RT (although counter examples exist, see Example 1). Let P' be a subset of P_Γ such that each $p \in P'$ is sign-invariant on T. If RT is projection definable w.r.t. P', then algorithms in (Hong, 1992; Brown, 1999) can be used to generate simple solution formula. If not, the cells of RT derived from the same path of T are grouped together. For each group, they have the same projection factor set. So algorithms in (Hong, 1992; Brown, 1999) can be used again to do the simplification. Let Φ be the resulting formula. If Φ is not simple enough, we can gather polynomials in Φ together into a set, say A, and compute an sign-invariant CAD defined by A and apply algorithms in (Hong, 1992; Brown, 1999) to do the simplification.

Next we show how to test if p is sign-invariant on Γ. By definition, a polynomial p is sign-invariant on Γ if and only if either $Z_{\mathbb{C}}(\Gamma) \subset Z_{\mathbb{C}}(p)$ or $Z_{\mathbb{C}}(\Gamma) \cap Z_{\mathbb{C}}(p) = \emptyset$ holds. Such tests boils down to set-theoretical operations on constructible sets. In particular, we have the following result from (Chen and Moreno Maza, 2014a) on the first test.

Lemma 4. Let Γ be a path of CCT. Let $p \in \mathbb{Q}[\mathbf{x}]$. Let $[R, H]$ be the associated regular system of Γ. Then $Z_{\mathbb{C}}(\Gamma) \subset Z_{\mathbb{C}}(p)$ if and only if $\text{prem}(p, R) = 0$.

Remark 2. To test $Z_{\mathbb{C}}(\Gamma) \cap Z_{\mathbb{C}}(p) = \emptyset$, it is equivalent to test $Z_{\mathbb{C}}(p, R, H) := Z_{\mathbb{C}}(p) \cap Z_{\mathbb{C}}(R, H) = \emptyset$. Efficient operation exists for such test, see Lemma 6 of (Chen et al., 2007) for details.

Example 5. Let $Dattel := z^2 + 3y^2 + 3x^2 - 1$. Let $f := (\exists z)\ Dattel = 0$ be the input formula. A sign-invariant CCT defined by p is described as below.



Algorithm 2 generates the following solution formula:

$$(3x^2 < 1 \wedge 3y^2 + 3x^2 < 1) \vee (3x^2 - 1 = 0 \wedge y = 0) \vee (3x^2 < 1 \wedge 3y^2 + 3x^2 = 1).$$

The sign-invariance of $3y^2 + 3x^2 - 1$ on the CCT allows us to obtain a simpler output formula:

$$3y^2 + 3x^2 < 1 \vee 3y^2 + 3x^2 = 1.$$

Example 6. Consider another input formula

$$(\exists x_1)\ 3x_1 - u_1(1+x_1^3) = 0 \wedge 3x_1^2 - u_2(1+x_1^3) = 0.$$

A variant of Algorithm 2 (removing redundant atomic formula in each conjunction) generates:

$$\begin{aligned}
&(u_2 < 0 \wedge u_1^3 + u_2^3 - 3u_1u_2 = 0) \vee (u_2 = 0 \wedge u_1 = 0)\\
&\vee(u_2^3 - 4 = 0 \wedge u_1u_2 - 2 = 0) \vee (u_2^3 - 4 = 0 \wedge u_1u_2 + 4 = 0)\\
&\vee(0 < u_2^3 - 4 \wedge u_1^3 + u_2^3 - 3u_1u_2 = 0)\\
&\vee(0 < u_2 \wedge u_2^3 < 4 \wedge u_1^3 + u_2^3 - 3u_1u_2 = 0)).
\end{aligned}$$

Using ideas presented here, we obtain $u_1^3 + u_2^3 - 3u_1u_2 = 0$.

6. Speeding up QE by computing truth table invariant CAD

In Section 4, we have presented a complete algorithm, namely Algorithm 1, for doing QE via RC-CAD. Let $PF := (Q_{k+1}x_{k+1}\cdots Q_nx_n)FF(x_1,\ldots,x_n)$, where FF is a quantifier free formula in disjunctive normal form. Let F be the set of non-constant polynomials appearing in FF. To get the solution formula of PF, a PF-truth invariant CAD RT_k of $\mathbb{R}^k$ is required. To make the presentation simple, Algorithm 1 computes an F-sign invariant CAD, which is sufficient but often not necessary to produce a CAD of $\mathbb{R}^k$ which is truth invariant w.r.t. PF. In this section, we show that the techniques introduced in (Bradford et al., 2014) for computing truth table invariant CAD (Bradford et al., 2013) can bypass the sign invariance while retaining the ability to produce truth invariant and projection definable CAD of $\mathbb{R}^k$, thus providing a more efficient but still complete QE procedure.

Let $\Phi := \vee_{i=1}^s \phi_i$ be a DNF. For simplicity of presentation and without lost of generality, we assume that each polynomial constraint appearing in Φ is of the type $p = 0$ or $p > 0$. The *corresponding complex formula* of Φ, denoted by $cf(\Phi)$, is the formula formed by replacing each atomic formula of the type $p > 0$ by $p \neq 0$. Let C be a subset of $\mathbb{R}^n$ (resp. $\mathbb{C}^n$), we say Φ (resp. $cf(\Phi)$) is *truth invariant* on C if the truth value of $\Phi(c)$ is the same for all $c \in C$. We say Φ (resp. $cf(\Phi)$) is *truth table invariant* on C if each conjunctive clause of Φ is truth invariant on C. Let RT (resp. T) be a CAD tree (resp. CCT). We say RT (resp. T) is *truth (table) invariant* for Φ (resp. $cf(\Phi)$) if Φ (resp. $cf(\Phi)$) is truth (table) invariant on each path of RT (resp. T).

It is clear that truth table invariance implies truth invariance. But a truth invariant Φ may not be truth table invariant. For example $x > 0 \vee x < 1$ is truth invariant on $\mathbb{R}$, but not truth table invariant on $\mathbb{R}$.

Proposition 3. Let Φ be a DNF. Let T be a truth table invariant CCT for $cf(\Phi)$. Let RT be the CAD derived from T. Then RT is truth table invariant for Φ. In particular, RT is truth invariant for Φ.

Proof. Suppose that RT is not truth table invariant for Φ. Let C be such a cell such that Φ is not truth table invariant. Then there exists a ϕ_i, $1 \leq i \leq s$, such that ϕ_i is not truth invariant on C. Let c_1, c_2 be two points of C such that $\phi_i(c_1)$ and $\phi_i(c_2)$ be

respectively true and false. There exists an atomic formula $p\ \sigma\ 0$, $\sigma \in \{>, =\}$, in ϕ_i whose truth value is false at c_2. If σ is $=$, then the truth value of $cf(p\ \sigma\ 0)$ is clearly false at c_2, which implies that $cf(\phi_i)(c_2)$ is false. If σ is $>$, then the sign of p at c_2 has to be zero or negative. In the former case, $cf(\phi_i)(c_2)$ is false. In the latter case, by the connectiveness of C, there exists a point c_3 such that p vanishes at c_3. Thus $cf(\phi_i)(c_3)$ is false. Note that the truth value of $cf(\phi_i(c_1))$ is true. Hence, $cf(\phi_i)$ is not truth invariant on C, which contradicts to the fact that T is truth table invariant for $cf(\Phi)$. □

Remark 3. Note that from a truth invariant CCT, one may not be able to produce a truth invariant CAD. Consider the following simple example. Let $\Phi := (x < 0) \vee (y < 0)$. The corresponding complex formula is $cf(\Phi) := (x \neq 0) \vee (y \neq 0)$. Consider the CCT $\{\{x = 0, y = 0\}, \{x = 0, y \neq 0\}, \{x \neq 0\}\}$, which is truth invariant for $cf(\Phi)$. A CAD derived from it is

$$\{\{x = 0, y < 0\}, \{x = 0, y = 0\}, \{x = 0, y > 0\}, \{x < 0\}, \{x > 0\}\}.$$

It is clear that above the cell $\{x > 0\}$, Φ is not truth invariant.

Consider now the QE problem $(Q_{k+1}x_{k+1} \cdots Q_n x_n)FF(x_1, \ldots, x_n)$, where FF is a DNF formula. The paper Bradford et al. (2014) presented an algorithm TTICCD to compute a truth table invariant CCT for $cf(FF)$. In Algorithm 1, let us extend the specification of CylindricalDecompose such that if a complex formula $cf(FF)$ is supplied as input, it calls TTICCD to produce a truth table invariant CCT.

Proposition 4. Algorithm 1 still satisfies its specification if we replace the subroutine CylindricalDecompose(F) by CylindricalDecompose($cf(FF)$).

Proof. Let $T :=$ CylindricalDecompose(FF) and $RT :=$ MakeSemiAlgebraic(T). By Proposition 3, RT is truth invariant for FF. Thus RT_k is truth invariant for PF. If RT_k is projection definable, the proposition clearly holds. Suppose that RT_k is not projection definable. Then Algorithm MakeProjectionDefinable refines T into a new CCT T', from which a CAD is derived, denoted still by RT, such that RT_k is projection definable. Since T is truth table invariant for $cf(FF)$, T' is also truth table invariant for $cf(FF)$, which implies that the new RT is truth invariant for FF. Thus RT_k is truth invariant for PF. So the proposition holds. □

7. Implementation

We have implemented the algorithms and optimization techniques described in the previous sections. The corresponding code is available through the `QuantifierElimination` command of the `RegularChains` library in MAPLE. The latest version of this library is freely available from the download page of `www.regularchains.org`. More details about the implementation and usage of `QuantifierElimination` can be found in (Chen and Moreno Maza, 2014b) and (Chen and Moreno Maza, 2014d).

Many additional improvements are planned for this command. For instance, the techniques of (Hong, 1992; Brown, 1999) for constructing simple solution formulae have not been integrated yet in our implementation

Nevertheless, the current version of our `QuantifierElimination` command is already very promising. We illustrate it in the sequel of this section with different examples.

These experimental results were obtained on an Ubuntu desktop (2.40GHz Intel Core 2 Quad CPU, 8.0Gb total memory).

In addition to performance, our implementation work is motivated by the desire of providing a user friendly interface for the command `QuantifierElimination`. We have built the interface of our QE procedure on top of the Logic package of MAPLE. The following MAPLE session shows how to use our command.

Example 7 (Davenport-Heintz)**.** The interface:

```
> f := &E([c]), &A([b, a]), ((a=d) &and (b=c))
             &or ((a=c) &and (b=1)) &implies (a^2=b):
> QuantifierElimination(f);
             (d - 1 = 0) &or (d + 1 = 0)
```

In (Chen and Moreno Maza, 2014a; Bradford et al., 2014), it is shown that the RC-CAD implementation can compete in terms of running time with state-of-art CAD implementations, such as QEPCAD and MATHEMATICA. In particular, RC-CAD is usually more efficient than its competitors as the number of equational constraints increases. For instance, neither QEPCAD nor MATHEMATICA can solve the examples blood-coagulation-2 and MontesS10 (see below) within 1-hour time limit. Our QE implementation directly benefits from the efficiency of RC-CAD. Hereafter, we provide the timing and output for three examples.

Example 8 (blood-coagulation-2)**.** It takes about 6 seconds.

```
f := &E([x, y, z]), (1/200*x*s*(1 - 1/400*x)
       + y*s*(1 - 1/400*x) - 35/2*x=0)
       &and (250*x*s*(1 - 1/600*y )*(z + 3/250) - 55/2*y=0)
       &and (500*(y + 1/20*x)*(1 - 1/700*z) - 5*z=0);
QuantifierElimination(f);
true
```

Example 9 (MontesS10)**.** It takes about 26 seconds.

```
f := &E([c2,s2,c1,s1]),
      (r-c1+l*(s1*s2-c1*c2)=0) &and (z-s1-l*(s1*c2+s2*c1)=0)
      &and (s1^2+c1^2-1=0) &and (s2^2+c2^2-1=0);
QuantifierElimination(f);
      2    2    2
((((-r  - z  + l  - 2 l + 1 = 0) &or

   2    2    2                     2    2    2
((l  - r  - z  - 2 l < -1) &and (-r  - z  + l  + 2 l + 1 = 0))) &or

   2    2    2                          2    2    2
((l  - r  - z  - 2 l < -1) &and (0 < -r  - z  + l  + 2 l + 1))) &or

       2    2    2                   2    2    2
((0 < -r  - z  + l  - 2 l + 1) &and (l  - r  - z  + 2 l < -1))) &or

       2    2    2                   2    2    2
((0 < -r  - z  + l  - 2 l + 1) &and (-r  - z  + l  + 2 l + 1 = 0))
```

Consider a new example on algebraic surfaces.

Example 10 (Sattel-Dattel-Zitrus)**.** It takes about 3 seconds while QEPCAD cannot solve it in 30 minutes.

```
Sattel := x^2+y^2*z+z^3;
Dattel := 3*x^2+3*y^2+z^2-1;
Zitrus := x^2+z^2-y^3*(y-1)^3;
f := &E([y, z]), (Sattel=0) &and (Dattel=0) &and (Zitrus<0);
QuantifierElimination(f);
```

The output is the inequality:

$$
\begin{aligned}
&387420489\,x^{36} + 473513931\,x^{34} + 1615049199\,x^{32}\\
&-5422961745\,x^{30} + 2179233963\,x^{28} - 14860773459\,x^{26}\\
&+43317737551\,x^{24} - 45925857657\,x^{22} + 60356422059\,x^{20}\\
&-126478283472\,x^{18} + 164389796305\,x^{16} - 121571730573\,x^{14}\\
&+54842719755\,x^{12} - 16059214980\,x^{10} + 3210573925\,x^{8}\\
&-446456947\,x^{6} + 43657673\,x^{4} - 1631864\,x^{2} < 40328.
\end{aligned}
$$

8. Automatic generation of parametrized parallel programs

The general purpose of automatic parallelization is to convert sequential computer programs into multi-threaded or vectorized code. We are interested in the following specific question. Given a theoretically good algorithm (e.g. divide-and-conquer matrix multiplication) and given a type of hardware that depends on various parameters (e.g. a GPGPU with amount S of shared memory per streaming multiprocessor, maximum number P of threads supported by each streaming multiprocessor, etc.) we aim at automatically generating code that depends on the hardware parameters (S, P, etc.) which, then, do not need to be known at compile-time. In contrast, current technology requires the knowledge of machine and program (size of a thread block, etc.) parameters at the time of generating the GPGPU code, see (Holewinski et al., 2012).

In order to clarify this question, we briefly provide some background material. The *polyhedron model* (Bastoul, 2004) is a powerful geometrical tool for analyzing the relation (w.r.t. data locality or parallelization) between the iterations of nested for-loops. Once the polyhedron representing the *iteration space* of a loop nest is calculated, techniques of linear algebra and linear programming can transform it into another polyhedron encoding the loop steps into a coordinate system based on time and space (processors). From there, a parallel program can be generated. For example, for the following code computing the product of two univariate polynomials `a` and `b`, both of degree `n`, and writing the result to `c`,

```
for(i=0; i<=n; i++) {c[i] = 0; c[i+n] = 0;}
for(i=0; i<=n; i++) {
  for(j=0; j<=n; j++)
    c[i+j] += a[i] * b[j];
}
```

elementary dependence analysis suggests to set $t(i,j) = n - j$ and $p(i,j) = i + j$, where t and p represent time and processor respectively. Using Fourier-Motzkin elimination, projecting all constraints on the (t,p)-plane yields the following asynchronous schedule of the above code:

```
parallel_for (p=0; p<=2*n; p++){
  c[p]=0;
  for (t=max(0,n-p); t<= min(n,2*n-p);t++)
    c[p] = c[p] + a[t+p-n] * b[n-t];
}
```

To be practically efficient, one should avoid a too fine-grained parallelization; this is achieved by grouping loop steps into so-called *tiles*, which are generally trapezoids (Högstedt et al., 1997). It is also desirable for the generated code to depend on parameters such as tile and cache sizes, number of processors, etc. These extensions lead, however, to the manipulation of systems of non-linear polynomial equations and the use of techniques like quantifier elimination (QE). This was noticed by the authors of (Größlinger et al., 2006) who observed also that work remained to be done for adapting QE tools to the needs of automatic parallelization.

To illustrate these observations, we return to the above example and use a tiling approach: we consider a one-dimensional grid of blocks where each block is in charge of updating at most B coefficients of the polynomial `c`. Therefore, we introduce three variables `B`, `b` and `u` where the latter two represent a block index and an update (or thread) index (within a block). This brings the following additional relations:

$$\begin{cases} 0 \leq b \\ 0 \leq u < B \\ p = bB + u, \end{cases} \tag{1}$$

to the previous system

$$\begin{cases} 0 < n \\ 0 \leq i \leq n \\ 0 \leq j \leq n \\ t = n - j \\ p = i + j. \end{cases} \tag{2}$$

To determine the target program, one needs to eliminate the variables `i` and `j`. In this case, Fourier-Motzkin elimination (FME) does not apply any more, due to the presence of non-linear constraints. Using quantifier elimination code presented in this paper, we

obtain the following:

$$\begin{cases} B > 0 \\ n > 0 \\ 0 \leq b \leq 2n/B \\ 0 \leq u < B \\ 0 \leq u \leq 2n - Bb \\ p = bB + u, \\ 0 \leq t \leq n, \\ n - p \leq t \leq 2n - p, \end{cases} \tag{3}$$

from where we derive the following program:

```
for (p=0; p<=2*n; p++) c[p]=0;
parallel_for (b=0; b<= 2 n / B; b++) {
    for (u=0; u<=min(B-1, 2*n - B * b); u++) {
       p = b * B + u;
       for (t=max(0,n-p); t<=min(n,2*n-p) ;t++)
           c[p] = c[p] + a[t+p-n] * b[n-t];
   }
}
```

Of course, one could enhance FME with a case discussion mechanism, but this enhancement would be limited to non-linear constraints where each variable appears in degree zero or one. (Otherwise an algorithm for solving semi-algebraic systems would need to support FME, which cannot really be considered as FME anymore.) Moreover, this enhanced and parametric FME would no longer be able to rely on numerical methods for linear programming (Khachiyan, 2009) thus loosing a lot of practical efficiency.

For these reasons, CAD-based QE becomes an attractive alternative. In fact, for more advanced automatic parallelization examples, such as the one of Fig. 5 in (Größlinger et al., 2006), our QE code returns a disjunction of conjunctions of clauses, where most conjunctions can be merged by the techniques presented in Section 5. Each of the remaining conjunctions of clauses leads to a specialized program corresponding to particular configuration like $n < B$. These specialized programs are actually less expensive to evaluate than the one of Fig. 5 in (Größlinger et al., 2006) since the bounds of the control variables are defined by simpler max/min expressions.

9. Conclusion

We introduced a complete real quantifier elimination procedure based on the theory of regular chains and cylindrical algebraic decompositions. Given a prenex formula PF in the first order theory of the reals, the procedure first constructs a complex cylindrical tree T for the corresponding complex formula of PF. From T, one builds a truth invariant CAD tree RT for PF. If RT is projection definable, that is the signs of polynomials in T are sufficient to distinguish the true and false cells in RT, one could immediately produce an equivalent quantifier free formula to PF from RT. For the case that RT is not projection definable, we proposed a method to refine T such that the new CAD tree

RT derived from T becomes projection definable. We discussed briefly how to generate a simpler QFF from RT. The effectiveness of our method was illustrated by several examples and an application on automatic loop parallelization.

Acknowledgements

The authors would like to thank anonymous reviewers for their helpful comments.

References

Bastoul, C., 2004. Code generation in the polyhedral model is easier than you think. In: Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques. PACT '04. IEEE Computer Society, pp. 7–16.

Basu, S., 1999. New results on quantifier elimination over real closed fields and applications to constraint databases. Journal of the ACM 46 (4), 537–555.

Basu, S., Pollack, R., Roy, M.-F., 2006. Algorithms in real algebraic geometry. Vol. 10 of Algorithms and Computations in Mathematics. Springer-Verlag.

Bradford, R., Chen, C., Davenport, J., England, M., Moreno Maza, M., Wilson, D., 2014. Truth table invariant cylindrical algebraic decomposition by regular chains. In: Proc. of CASC 2014. pp. 44–58.

Bradford, R., Davenport, J., England, M., McCallum, S., Wilson, D., 2013. Cylindrical algebraic decompositions for boolean combinations. In: Proc. of ISSAC'13. pp. 125–132.

Brown, C. W., 1999. Solution Formula Construction for Truth Invariant CAD's. Ph.D. thesis, University of Delaware.

Brown, C. W., Davenport, J. H., 2007. The complexity of quantifier elimination and cylindrical algebraic decomposition. In: Proc. of ISSAC 2007. pp. 54–60.

Caviness, B., Johnson, J. (Eds.), 1998. Quantifier Elimination and Cylindical Algebraic Decomposition, Texts and Mongraphs in Symbolic Computation. Springer.

Chen, C., Golubitsky, O., Lemaire, F., Moreno Maza, M., Pan, W., 2007. Comprehensive triangular decomposition. In: Proc. of CASC 2007. Vol. 4770 of Lecture Notes in Computer Science. Springer Verlag, pp. 73–101.

Chen, C., Moreno Maza, M., 2014. An incremental algorithm for computing cylindrical algebraic decompositions. Computer Mathematics: Proc. of ASCM '12, 199–222.

Chen, C., Moreno Maza, M., 2014. Cylindrical algebraic decomposition in the regularchains library. In: Proc. of ICMS 2014. pp. 425–433.

Chen, C., Moreno Maza, M., 2014. Quantifier elimination by cylindrical algebraic decomposition based on regular chains. In: Proc. of ISSAC 2014. pp. 91–98.

Chen, C., Moreno Maza, M., 2014. Real quantifier elimination in the regularchains library. In: Proc. of ICMS 2014. pp. 283–290.

Chen, C., Moreno Maza, M., Xia, B., Yang, L., 2009. Computing cylindrical algebraic decomposition via triangular decomposition. In: Proc. of ISSAC 2009. pp. 95–102.

Collins, G. E., 1975. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. Springer Lecture Notes in Computer Science 33, 515–532.

Coste, M., Roy, M.-F., 1988. Thom's lemma, the coding of real algebraic numbers and the computation of the topology of semi-algebraic sets. J. Symb. Comput. 5 (1-2), 121–129.

Dolzmann, A., Seidl, A., Sturm, T. (Eds.), 2005. Algorithmic Algebra and Logic. Proceedings of the A3L 2005.

Größlinger, A., Griebl, M., Lengauer, C., 2006. Quantifier elimination in automatic loop parallelization. J. Symb. Comput. 41 (11), 1206–1221.

Högstedt, K., Carter, L., Ferrante, J., 1997. Determining the idle time of a tiling. In: Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. POPL '97. pp. 160–173.

Holewinski, J., Pouchet, L., Sadayappan, P., 2012. High-performance code generation for stencil computations on GPU architectures. In: Proceedings of the 26th ACM International Conference on Supercomputing. ICS '12. pp. 311–320.

Hong, H., 1990. Improvements in CAD-based quantifier elimination. Ph.D. thesis, The Ohio State University.

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

Hong, H., 1993. Special issue editorial: Computational quantifier elimination. Comput. J. 36 (5), 399.

Hong, H., Safey El Din, M., 2012. Variant quantifier elimination. Journal of Symbolic Computation 47 (7), 883 – 901.

Khachiyan, L., 2009. Fourier-Motzkin elimination method. In: Encyclopedia of Optimization. pp. 1074–1077.

Renegar, J., 1992. On the computational complexity and geometry of the first-order theory of the reals. parts I–III. J. Symb. Comput. 13 (3), 255–299.

Xiao, R., 2009. Parametric polynomial system solving. Ph.D. thesis, Peking University, Beijing.