Quantifier Elimination by Cylindrical Algebraic Decomposition Based on Regular Chains

Changbo Chen
Chongqing Key Laboratory of Automated Reasoning and Cognition, Chongqing Institute of Green and Intelligent Technology, CAS
changbo.chen@hotmail.com

Marc Moreno Maza
ORCCA, University of Western Ontario
moreno@csd.uwo.ca

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.

Categories and Subject Descriptors

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

General Terms

Algorithms, Experimentation, Theory

Keywords

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 [18, 12, 6, 3] 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 one free variable and all polynomials in the quantified input are linear, see the paper [5]. 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 [20, 2]. Despite of these theoretical results, the practical efficiency and the range of the applications of CAD-based QE have kept improving regularly since Collins' landmark paper [10]. 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].

In [9], together with B. Xia and L. Yang, we presented a different way of computing CADs, based on triangular decomposition of polynomial systems. Our scheme relies on the concept of *cylindrical decomposition of the complex space* (CCD), from which a CAD can be easily derived. Since regular chains 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 [8], 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 [17], Hong produced simple solution formula constructions, assuming that the available polynomials in a CAD are sufficient to generate output formulae.

In his PhD thesis [4], Brown then introduced ways to add

ISSAC '14, July 23 - 25, 2014, Kobe, Japan.

ACM 978-1-4503-2501-1/14/07 ...$15.00.
http://dx.doi.org/10.1145/2608628.2608666.

[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/`

polynomials in an incremental manner and proposed a complete algorithm which can produce 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 CAD 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 [8]). The PL-CAD-concept of *projection-definable* CAD [4] means, in the context of RC-CAD, that the signs of polynomials in the tree T suffice to solve the targeted QE problem,

Despite of the many differences between the PL-CAD and RC-CAD schemes for constructing CADs, we manage in Section 4 to adapt Brown's techniques, based on his notion of *conflicting pair*, to the RC-CAD context. Once a quantifier-free formula is obtained, as in the PL-CAD scheme, simplification strategies are needed. In Section 5, we explain how we do it in the context of RC-CAD. 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 7.

2. PRELIMINARY

In this section, we review some necessary notions for stating the main results of this paper.

Zero sets of constraints. Let $\mathbf{x} = x_1 \prec \cdots \prec x_n$ be a sequence for 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 $\mathbb{C}$ or $\mathbb{R}$. 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 [9, 8]. 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 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 in its main variable) does not vanish at α; (ii) the univariate polynomials $p(\alpha, x_n) \in \mathbb{Q}[x_n]$, for all $p \in P$, are squarefree and pairwise coprime.

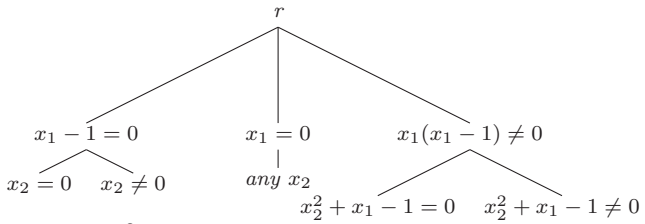
Complex cylindrical tree [9, 8]. Let T be a rooted tree of height n where each node of depth i, for $i = 1, \ldots, n$, being labelled by a polynomial constraint of the type "any x_i" (with zero set defined as $\mathbb{C}^n$), or $p = 0$, or $p \neq 0$, where $p \in \mathbb{Q}[x_1, \ldots, x_i]$. For 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. 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 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 satifies the following:

- if $n = 1$, then either T has only one leaf which is labelled "any x_1", or, for some $s \geq 1$, it has $s+1$ leaves labelled 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 labelled $p_1 = 0, \ldots, p_s = 0, \prod_{i=1}^{s} p_i \neq 0$, 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 from [9, 8] 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.*



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

The following theorem established in [9] allows one to build a CAD from a complete CCT. The CAD can also be organized naturally in a tree data structure. The procedure MakeSemiAlgebraic in [9, 8] builds a CAD tree RT from a complete CCT T.

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.*

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 $Q_i \in \{\exists, \forall\}$, $k+1 \leq i \leq n$. 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).

3. THEORY

To present the algorithm, in this section, we revise the concepts of *projection factor set* and *projection definable*, which were originally introduced in [4].

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 labelled by equational constraints; this includes V itself if it is labelled 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 projection factor sets of all its paths. Let RT be a 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. 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. 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 a 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 $\{r, x_1 = 0,$ any $x_2\}$. Moreover, it is easy to verify that T and RT are projection definable.*

Let $p := x^2 - 2$, whose zero set natually defines a CAD tree RT of $\mathbb{R}^1$. Suppose that the only true cells of RT are $x = -\sqrt{2}$ and $x = \sqrt{2}$. Then RT is not projection definable.

Definition 1. *Let F be a set of nonconstant univariate polynomials in $\mathbb{R}[x]$. We say F is* derivative closed *(w.r.t. factorization) if for any $f \in F$, where $\deg(f) > 1$, $der(f)$ 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 $Z_{\mathbb{R}}(F_\sigma) := \cap_{f \in F} Z_{\mathbb{R}}(f\ \sigma(f)\ 0)$.

Lemma 1 (Thom's Lemma [11]). *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.*

Remark. The formulation of Thom's lemma presented here is slightly different from its original version [11], although it can be proved using exactly the same arugments (by induction on the number of polynomials in F). Such a formulation is often implicitly used in implementations related to Thom's Lemma. An explicit treatment can be found, for example, in [4, 21].

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 $F_\sigma(\alpha)$ belonging 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 demonstrate how to do quantifier elimination via RC-CAD. Algorithm 1 presents 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)[6];
4   RT := MakeSemiAlgebraic(T);
5   AttachTruthValue(FF, RT);
6   PropagateTruthValue(PF, RT);
7   MakeProjectionDefinable_k(PF, RT);
8   SF := GenerateSolutionFormula_k(RT_k);
9 end
```

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

- Projection: Line 3 of Algorithm 1.
- Stack construction: Liness 4, 5, 6 of Algorithm 1.
- Solution formula construction: Liness 7, 8 of Algorithm 1.

[6] If FF is a pure conjunctive formula, the theory of [8] allows F to be a set of polynomial constraints. The detail is omitted here for simplicity.

Note that these phases do not map exactly to those of QE based on PL-CAD. The projection here computes a complex cylindrical tree instead of a projection factor set. For the stack construction phase, its first step MakeSemiAlgebraic, recalled in Section 2, is different from the real root isolation routines used by PL-CAD. The other two steps AttachTruthValue and PropagateTruthValue are the same as their PL-CAD counterparts. We recall them below.

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 a prenex formula $PF := (Q_{k+1}x_{k+1}\cdots Q_n x_n)FF(x_1,\ldots,x_n)$ and a CAD tree RT, each leaf V of which is attached with a truth value of FF evaluated at $V.samplepoint$, as input, and 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 $V_k.samplepoint$. The algorithm 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 be 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 be true if and only if the truth value of at least one child of V is true. The algorithm then makes recursive call with $(Q_{k+1},\ldots,Q_{n-1})$ and RT_{n-1} as input and terminates when all Q_i, $i = k+1,\ldots,n$, have been examined.

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

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.

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

definable, we will present two strategies to address this. The first one, presented in Algorithm 5, is a theoretical solution. This is an adaptation of the augmented projection, widely used in PL-CAD, to RC-CAD. The second one, presented in Algorithm 6 and used in our implementation, is a much more practical solution. This is an adaptation of making CAD projection definable, used in PL-CAD [4], to RC-CAD. Our adaptation is based on the IntersectPath operation of [8]. This operation takes as input a polynomial p, a CCT T and a path Γ of T, returning a refinement of T such that p is sign-invariant on each path derived from Γ. Another operation `NextPathToDo` from [8] is also used in our algorithm. It takes a CCT T as input, for a fixed traversal order of T, returning the first "ToDo" 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. Assume that c_i is of the form $f_i = 0$, $i = 1,\ldots,s$. We say Γ_{k-1} is derivative closed if for any $\alpha \in Z_{\mathbb{C}}(\Gamma_{k-1})$, the set of univariate polynomials $\{f_i(\alpha, x_k)\}$ is derivative closed.*

Algorithm 3: 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 Γ_{k-1} is derivative closed, return false. Otherwise, some progress is made to guarantee that Γ_{k-1} becomes derivative closed after this algorithm is called finitely many times.

```
begin
    V := Γ_{k-1}.leaf;
    let S be the set of children of V in T_k such that:
    each c ∈ S is of the form f = 0, where deg(f) > 1
    and c.derivative is undefined;
    if S = ∅ then return false;
    let c ∈ S such that deg(f) is the smallest;
    let Γ_k be the subtree of T_k which induces Γ_{k-1};
    while C := NextPathToDo_k(Γ_k \ (Γ_{k-1} ∪ c)) do
        IntersectPath_k(der(f, x_k), C, T_k);
    c.derivative := der(f, x_k);
    return true;
end
```

Algorithm 4: 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.

```
begin
    if k = 0 then return;
    while Γ := NextPathToDo_{k-1}(T) do
        todo := true;
        while todo do
            todo := RefineNextChild_k(Γ, T);
    RefineTree_{k-1}(T);
end
```

PROPOSITION 1. *Algorithm 3 and 4 are as specified.*

PROOF. We prove Algorithm 4 by induction. A proof of Algorithm 3 will be supplied in between. Algorithm 4 clearly holds for $k = 0$. Assume that RefineTree$_{k-1}(T)$ makes T_{k-1} projection definable. Then it suffices to show that when Algorithm 4 terminates, each path Γ_{k-1} of T_{k-1} is derivative closed. Equivalently, it is enough to prove that when Algorithm 3 returns false, Γ_{k-1} is derivative closed.

Before the first call to Algorithm 3 is made, for any child c of $\Gamma_{k-1}.leaf$, $c.derivative$ is unassigned. Each time when Algorithm 3 is called, a vertex c of the form $f = 0$, where $c.derivative$ is unassigned and $\deg(f, x_k)$ is the smallest, is chosen. By calling the operation IntersectPath, the children nodes of Γ_{k-1} are refined into new ones, above each of which

$\mathrm{der}(f, x_k)$ is sign invariant. Let $c_1, \ldots, c_s$ be all the equation children of $\Gamma_{k-1}.leaf$. 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 Γ_{k-1}, $\mathrm{der}(f, x_k)(\alpha, x_k)$ is a product of some $f_i(\alpha, x_k)$ times a constant.

Since each time a vertex c is chosen such that $c.derivative$ turns assigned, while meantime for each new added c_i into the tree, $\mathrm{der}(f_i, x_k)$ is strictly less than $\deg(f, x_k)$, we know that Algorithm 3 will return false after being called finitely many times. When false is returned, by Definition 2, Γ_{k-1} is clearly derivative closed. □

Note that Algorithm 4 may generate much more than enough polynomials for the purpose of solution formula construction. Nevertheless, such an algorithm allows a simple solution for making a CAD tree projection definable, see Algorithm 5.

Algorithm 5: 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);
7 end
```

To help present the practical strategy for making a CAD tree projection definable, we revise the notion of "conflicting pair", which was initially introduced for PL-CAD in [4].

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 *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.*

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 of C and D of the largest level, say $i-1$. Let C_i and D_i be respectively the ancestor of C and D of level i. Clearly C_i and D_i share the same parent A and form a conflicting pair. We call C_i and D_i the *conflicting pair associated with* C and D. We call C and D an *extension* of C_i and D_i. Note that for PL-CAD, only (CP_1), (CP_3) and (CP_4) are required. In [4], 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.

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

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;
12 end
```

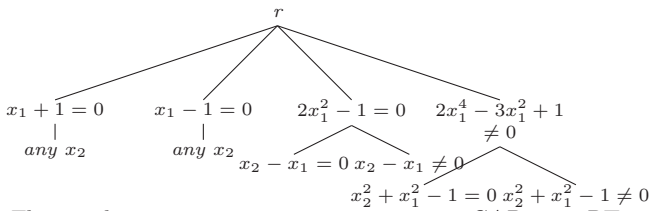
Proof. It is enough to prove the following two claims:
(1) If $CPS = \emptyset$, then RT_k is projection definable.
(2) CPS becomes empty after finitely many steps.
Note that $CPS = \emptyset$ implies that there does not exist cells C and D which satisfy (CP_i), $i = 2, 3, 4$. In other words, if for any two cells C and D derived from the same complex path, thus having the same projection factor set (say P), their attached true values are different, then the signs of polynomials in P are sufficient to distinguish them. So (1) is proved.

Let i be the highest level of conflicting pairs in RT_k. To prove (2), it suffices to prove the following three claims.
(2.1) Any Γ_{i-1} will become projection definable after finitely many steps.
(2.2) When all Γ_{i-1} become projection definable, there will exist no conflicting pairs of level i.
(2.3) When RT gets refined, no conflicting pairs of level higher than i will be generated.
The correctness of (2.1) follows from Proposition 1. 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 children of Γ_{i-1}, and all i-level polynomials in their projection factor set evaluate at C_i and D_i into the same signs. 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. □

We conclude this section by illustrating our algorithm with a simple example. This example is modified from the one in [16, 4], 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 a 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*,* $(-1, -\frac{\sqrt{2}}{2})$*,* $-\frac{\sqrt{2}}{2}$*,* $(-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$*,* $\frac{\sqrt{2}}{2}$*,* $(\frac{\sqrt{2}}{2}, 1)$*,* 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*. Refine* 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:*

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

5. CONSTRUCTION OF SIMPLE QFFS

In last section, a complete quantifier elimination algorithm is provided. By "complete", we mean that it can always generate a solution formula. In this section, we provide some heuristic strategies to enhance Algorithm 2 such that simpler solution formula can be generated with 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 formula is that every polynomial in P is guaranteed to be sign-invariant on each cell of RT (assume no partial CAD tricks are 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.

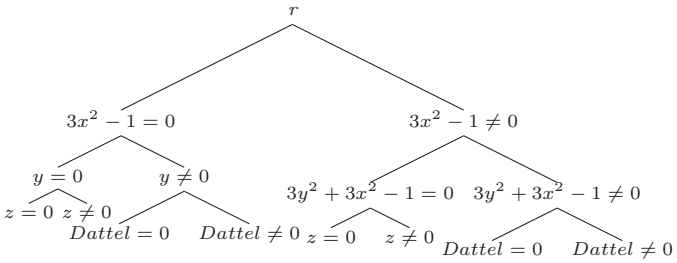
On the other hand, let Γ be the right most path of T, we observe that in many cases, the polynomials in P_Γ are sign-invariant on each path of T, and thus also sign-invariant on every cell of RT (although counter examples exist, see Example 1). Let P' be a subset of P_Γ such that each $p \in P'$ is sign-invariant on T. If RT is projection definable w.r.t. P', then algorithms in [17, 4] 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 [17, 4] 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 [17, 4] 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 [8] on the first test.

LEMMA 3. *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* $prem(p, R) = 0$*.*

REMARK 1. *To test* $Z_{\mathbb{C}}(\Gamma) \cap Z_{\mathbb{C}}(p) = \emptyset$*, it is equivalent to test* $Z_{\mathbb{C}}(p, R, H) := Z_C(p) \cap Z_{\mathbb{C}}(R, H) = \emptyset$*. Efficient operation exists for such test, see Lemma 6 of [7] for details.*

EXAMPLE 4. *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 5. *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:

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

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

6. IMPLEMENTATION

We have implemented our algorithm in the `RegularChains` library in MAPLE. For constructing simple solution formula, the techniques of [17, 4] have not been integrated yet. In this section, we illustrate different aspects of our implementation by examples. The experimental results are obtained on an Ubuntu desktop (2.40GHz Intel Core 2 Quad CPU, 8.0Gb total memory).

There is no doubt that a user friendly interface is important for the applications of quantifier elimination. We have developed the interface of our QE procedure based on the `Logic` package of MAPLE. The following MAPLE session shows how to use our procedure.

EXAMPLE 6 (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 [8], we have shown that our RC-CAD implementation is competitive to the 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 within 1-hour time limit. Our QE implementation directly benefits from the efficiency of RC-CAD. Here we provide the timing and output for three examples.

EXAMPLE 7 (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 8 (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 9 (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:

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

7. 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 a given 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 [15].

In order to clarify this question, we briefly provide some background material. The *polyhedron model* [1] 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 in 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 [14]. 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 [13] 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:

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

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 [19] 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 [13], 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 [13] since the bounds of the control variables are defined by simpler max/min expressions.

Acknowledgements

This work was partially supported by NSFC (11301524), NKBRPC (2011CB302400) and CSTC (cstc2013jjys0002).

References

[1] C. Bastoul. 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, pages 7–16, Washington, DC, USA, 2004. IEEE Computer Society.

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

[3] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in real algebraic geometry*, volume 10 of *Algorithms and Computations in Mathematics*. Springer-Verlag, 2006.

[4] C. W. Brown. *Solution Formula Construction for Truth Invariant CAD's*. PhD thesis, University of Delaware, 1999.

[5] C. W. Brown and J. H. Davenport. The complexity of quantifier elimination and cylindrical algebraic decomposition. In Dongming Wang, editor, *ISSAC*, pages 54–60. ACM, 2007.

[6] B. Caviness and J. Johnson, editors. *Quantifier Elimination and Cylindical Algebraic Decomposition, Texts and Mongraphs in Symbolic Computation*. Springer, 1998.

[7] C. Chen, O. Golubitsky, F. Lemaire, M. Moreno Maza, and W. Pan. Comprehensive triangular decomposition. In *Proc. of CASC'07*, volume 4770 of *Lecture Notes in Computer Science*, pages 73–101. Springer Verlag, 2007.

[8] C. Chen and M. Moreno Maza. An incremental algorithm for computing cylindrical algebraic decompositions. *Proc. ASCM '12*, Oct. 2012.

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

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

[11] M. Coste and M-F. Roy. 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, 1988.

[12] A. Dolzmann, A. Seidl, and T. Sturm, editors. *Algorithmic Algebra and Logic. Proceedings of the A3L 2005, April 3-6, Passau, Germany; Conference in Honor of the 60th Birthday of Volker Weispfenning*. Books on Demand, 2005.

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

[14] K. Högstedt, L. Carter, and J. Ferrante. Determining the idle time of a tiling. In *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '97, pages 160–173, New York, NY, USA, 1997. ACM.

[15] J. Holewinski, L. Pouchet, and P. Sadayappan. High-performance code generation for stencil computations on GPU architectures. In *Proceedings of the 26th ACM International Conference on Supercomputing*, ICS '12, pages 311–320, New York, NY, USA, 2012. ACM.

[16] H. Hong. *Improvements in CAD-based Quantifier Elimination*. PhD thesis, The Ohio State University, 1990.

[17] H. Hong. Simple solution formula construction in cylindrical algebraic decomposition based quantifier elimination. In *ISSAC*, pages 177–188, 1992.

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

[19] L. Khachiyan. Fourier-Motzkin elimination method. In C. A. Floudas and P. M. Pardalos, editors, *Encyclopedia of Optimization*, pages 1074–1077. Springer, 2009.

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

[21] R. Xiao. *Parametric Polynomial System Solving*. PhD thesis, Peking University, Beijing, 2009.