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 text books and journal special issues [15] [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].

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 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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2014 ACM 978-1-4503-2501-1/14/07 ...$15.00.

1QEPCAD: http://www.usna.edu/CS/~qepcad/B/QEPCAD.html
2Mathematica: http://www.wolfram.com/mathematica/
3REDLOG: http://www.redlog.eu/
4SyNRAC: http://jp.fujitsu.com/group/labs/techinfo/freeware/synrac/
5RegularChains: 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 \([4]\)). The PL-CAD-concept of projection-definable CAD \([1]\) 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 to the RC-CAD context Brown’s techniques based on his notion of conflicting pair. 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 \( x = x_1 < \cdots < x_n \) be a sequence for ordered variables. Let \( F \subseteq \mathbb{Q}[x] \) be finite. Let \( \sigma_1 \) be a map from \( F \) to \( \{=, \neq\} \) and \( \sigma_2 \) be a map from \( F \) to \( \{=, \neq, <, >, \leq, \geq\} \). Let \( K \subseteq \mathbb{C} \) or \( \mathbb{R} \). For \( f \in F \) we denote by \( \mathbb{Z}_K(f) \) the zero set of \( f \) in \( K^n \). Denote by \( \mathbb{Z}_K(f, \sigma_1(f)) \) the zero set of \( f, \sigma_1(f) \) in \( K^n \) and by \( \mathbb{Z}_K(f, \sigma_2(f)) \) the zero set of \( f, \sigma_2(f) \) in \( \mathbb{R}^n \).

Separation \([9, 8]\). Let \( C \) be a subset of \( K^{n-1} \) and \( P \subseteq \mathbb{Q}[x_1 < \cdots < x_n] \) 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 init(\( p \)) does not vanish at \( \alpha; (ii) \) the univariate polynomials \( \text{p}(\alpha, x_n) \in \mathbb{Q}[x_n] \), 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 \( C^n \)), 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 \( \Gamma \) be a path of \( T \) from the root to a leaf. Its zero set \( \mathbb{Z}_K(\Gamma) \) is defined as the intersection of the zero sets of its nodes. The zero set \( T \) of \( T \), denoted by \( \mathbb{Z}_K(T) \), is defined as the union of zero sets of its paths. We call \( T \) a complete complex cylindrical tree (complete CCT) of \( \mathbb{Q}[x_1, \ldots, x_n] \), if it is defined recursively as below:

- 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 \( \Gamma \) 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 \( p_1, \ldots, p_s \in \mathbb{Q}[x] \) are polynomials which separate above \( \mathbb{Z}_K(\Gamma) \).

The set \( \{ \mathbb{Z}_K(\Gamma) \setminus \Gamma \mid T \} \) is called a complex cylindrical decomposition (CCD) of \( \mathbb{C}^n \) associated with \( T \). Note that for a complete CCT, we have \( \mathbb{Z}_K(T) = \mathbb{C}^n \).

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 \( F \subseteq \mathbb{Q}[x] \) be finite. Let \( \Gamma \) be a path of a CCT \( T \). Note that the polynomial constraints along \( \Gamma \) form a regular system, called the associated regular system of \( T \). Let \( p \in F \).

We say \( p \) is sign-invariant on \( \Gamma \) if either \( \mathbb{Z}_K(\Gamma) \subseteq \mathbb{Z}_K(p) \) or \( \mathbb{Z}_K(\Gamma) \cap \mathbb{Z}_K(p) = \emptyset \) holds. We say \( p \) 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.

```
          r
         /|
        / |  
      x_1 = 0  x_1(x_1 - 1) \neq 0
     /  |   
    x_2 = 0  x_2 \neq 0
```

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

The following theorem established in \([9, 8]\) 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 \( \mathbb{Q}[x_1 < \cdots < x_n] \) 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 definable on \( S \). Moreover, the product of the \( p_1, \ldots, p_r \) is also definable on \( S \).

Quantifier elimination. Let \( f \in \mathbb{Q}[x_1 < \cdots < x_n] \). Let \( \sigma \in \{<, >, \leq, \geq, =, \neq\} \). We call a formula of the form \( f(\sigma) \) a (standard) atomic formula. Let \( FF(1, \ldots, x_n) \) be a quantifier free formula in disjointive 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 prefix formula, where \( Q_i \in \{\exists, \forall\} \), \( k + 1 \leq i \leq n \).

A quantifier-free formula \( SF \) in \( \mathbb{Q}[x_1, \ldots, x_n] \) 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 \( \mathbb{Q}[x] \). 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 $V_i$. Let $\Gamma$ be a path of $T$. The union of the projection factor sets of all the nodes along $\Gamma$ is called a projection factor set of $\Gamma$. 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 $\Gamma$ of $T$. The projection factor set of $C$ is defined as that of $\Gamma$.

**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 $\Gamma$ be a path of $T$. Let $C$ be any CAD cell derived from $\Gamma$. It is guaranteed that any polynomial in the projection factor set $P_T$ of $\Gamma$ is sign invariant above $C$. However, it is possible that a polynomial of $P_T$ 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 7. Let $RT$ be a CAD tree derived from $T$. Let $\Gamma$ be the right most path of $T$. The projection factor set of $\Gamma$ is $\{x_1, x_1 - 1, x_2 + x_1 - 1\}$. The projection factor set of $T$ and $RT$ is $\{x_1, x_1 - 1, x_2 + x_1 - 1, x_2\}$. We notice that neither $x_2$ nor $x_2^2 + x_1 - 1$ is sign invariant on the path $T$ consisting of nodes $[r, x_1 = 0, \text{any } x_2]$. Moreover, it is easy to verify that $T$ and $RT$ are projection definable.

**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 $f \in F$, when $\deg(f) > 1$, $\text{der}(f)$ is a product of some polynomials in $F$ and some constant $c \in \mathbb{R}$.

Let $F \subseteq \mathbb{R}[x]$ be finite. Let $\sigma$ be a map from $F$ to $\{<, >, =\}$. Let $F_\sigma := \{f \in F \mid \sigma(f) = 0\}$. Define $Z_\sigma(F_\sigma) := \bigcap_{f \in F} Z(f) \sigma(f) = 0$.

**Lemma 1** (Thom’s Lemma [11]). Assume that $n = 1$. If $F$ is derivative closed (w.r.t. factorization), then the set $F_\sigma$ defined by $F_\sigma$ 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 arguments (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 [12].

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

**Proof.** Assume that $C \times \mathbb{R}^1 \cap Z_\sigma(F_\sigma)$ is not empty. Since $F$ separates above $C$, by Theorem 1 $F$ is derivative closed above $C$. Thus $C \times \mathbb{R}^1 \cap Z_\sigma(F_\sigma)$ 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 $F_\sigma$.

Let $\alpha$ be a point of $C$. Denote $F_\sigma(\alpha) := Z_\sigma(F(\alpha))$. 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 $\beta$, $F_\sigma(\beta)$ 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 $x_0^*, x_1^* \in S'$ such that $(\alpha, x_0^*)$, $(\beta, x_1^*) \in S'$, and both $F(\alpha, x_0^*)$ and $F(\beta, x_1^*)$ 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 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

$$PF := (Q_{k+1}x_{k+1} \cdots Q_nx_n)F(x_1, \ldots, x_n).$$

**Output:** A solution formula of $PF$.

```
begin
1. Let $F$ be the set of polynomials appearing in $FF$;
2. $T := \text{CylindricalDecompose}(F)$;  
3. $RT := \text{MakeSemiAlgebraic}(T)$;
4. $SF := \text{GenerateSolutionFormula}_{k}(RT_k)$;
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: Lines 4, 5, 6 of Algorithm 1
- Solution formula construction: Lines 7, 8 of Algorithm 1

If $PF$ is a pure conjunctive formula, the theory of [5] 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 $\exists \mathbf{x} (Q_{k+1} \mathbf{x}_{k+1} \ldots Q_n \mathbf{x}_n) F(x_1, \ldots, x_n)$ 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 $RT_1$ of RT in $\mathbb{Q}[x_1, \ldots, x_k]$ such that each leaf $V_k$ of $RT_1$ is attached with a truth value of FF evaluated at $V_k$.samplepoint. The algorithm starts by checking whether $Q_k$ is the universal quantifier $\forall$ or the existential quantifier $\exists$. For each leaf $V$ of $RT_1$, if $Q_k$ is $\forall$, then $\mathbf{V}.truthvalue$ is set to be true if and only if the truth values of the children of $V$ are all true; if $Q_k$ is $\exists$, then $\mathbf{V}.truthvalue$ is set to be true if and only if the truth value of the 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 definable, we will present two strategies to address this. The first one, presented in Algorithm 3, 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 4, and used in our implementation, is a much more practical solution. This is an adaptation of making CAD projection definable, used in PL-CAD, to RC-CAD. Our adaptation is based on the IntersectPath operation of [S]. This operation takes as input a polynomial $p$, a CCT $T$ and a path $\Gamma$ of $T$, returning a refinement of $T$ such that $p$ is sign-invariant on each path derived from $\Gamma$. Another operation NextPathToDo from [S] 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 $\Gamma$ 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 $T_k$ be the induced subtree of $T_k$ in $\mathbb{Q}[x_1, \ldots, x_k]$. Let $\Gamma_{k-1}$ be a path of $T_k$. Let $c_1, \ldots, c_n$ be all the equation children of $\Gamma_{k-1}$, leaf $f$ in $T_k$. Assume that $c_i$ is of the form $f_i = 0, i = 1, \ldots, n$. We say $\Gamma_{k-1}$ is derivative closed if for any $a \in \mathbb{Z}(\Gamma_{k-1})$, the set of univariate polynomials $\{f_i(\alpha, x_k)\}$ is derivative closed.

**Algorithm 3: RefineNextChildk(Γk−1, T).**

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

Output: If $\Gamma_{k-1}$ is derivative closed, return false. Otherwise, some progress is made to guarantee that $\Gamma_{k-1}$ becomes derivative closed after this algorithm is called finitely many times.

1 begin
2 $V := \Gamma_{k-1}.leaf$;
3 let $S$ be the set of children of $V$ in $T_k$ such that:
4 each $c \in S$ is of the form $f = 0$, where $\deg(f) > 1$
5 and $c$.derivative is undefined;
6 if $S = \emptyset$ then return false;
7 let $c \in S$ such that $\deg(f)$ is the smallest;
8 let $\Gamma_k$ be the subtree of $T_k$ which induces $\Gamma_{k-1}$;
9 while $C := \text{NextPathToDo}(\Gamma_k \setminus (\Gamma_{k-1} \cup c))$ do
10 \{ IntersectPath$(\text{der}(f, x_k), C, T_k)$; \}
11 c.derivative := $\text{der}(f, x_k)$;
12 return true;
11 end

**Algorithm 4: RefineTreek(T).**

Input: A complete complex cylindrical tree $T$ of $\mathbb{Q}[x_1 < \cdots < x_n]$.

Output: Refine $T$ to make its induced subtree $T_k$ projection definable.

1 begin
2 if $k = 0$ then return;
3 while $\Gamma := \text{NextPathToDo}_{k-1}(T)$ do
4 todo := true;
5 while todo do
6 todo := RefineNextChildk$(\Gamma, T)$;
7 RefineTree$(\Gamma_{k-1}, T)$;
8 end

**Proposition 1.** Algorithms 3 and 4 are as specified.

**Proof.** We prove Algorithm 3 by induction. A proof of Algorithm 4 will be supplied in between. Algorithm 3 clearly holds for $k = 0$. Assume that RefineTree$(\Gamma_{k-1}, T)$ makes $\Gamma_{k-1}$ projection definable. Then it suffices to show that when Algorithm 3 terminates, each path $\Gamma_{k-1}$ of $\Gamma_{k-1}$ is derivative closed. Equivalently, it is enough to prove that when Algorithm 3 returns false, $\Gamma_{k-1}$ 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 $\Gamma_{k-1}$ are refined into new ones, above each of which
der\((f, x_k)\) is sign invariant. Let \(c_1, \ldots, c_6\) be all the equation children of \(\Gamma_{k-1, \text{leaf}}\). Assume that \(c_1\) is of the form \(f_1 = 0\). The sign invariance of der\((f, x_k)\) above each \(c_i\) implies that for any \(\alpha\) of \(\Gamma_{k-1}\), der\((f, x_k)(\alpha, x_k)\) is a product of some \(f_i(x, x_k)\) times a constant.

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

Note that Algorithm 3 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 4.

**Algorithm 4: MakeProjectionDefinable\((PF, RT, \text{practical})\)**

*Input:* Same as Algorithm 3

*Output:* Same as Algorithm 5

1. let \(CPS\) be the set of all conflicting pairs of \(RT_k\);
2. while \(CPS \neq \emptyset\) do
3. let \(CP\) be a pair in \(CPS\) of highest level, say \(i\);
4. let \(T\) be the associated CCT of \(RT\);
5. let \(\Gamma\) be the path of \(T_k\), where \(CP\) is derived;
6. call \(\text{RefineNextChild}(\Gamma_{i-1}, T)\) to refine \(T\);
7. \(RT \leftarrow \text{MakeSemiAlgebraic}(T)\);
8. \(\text{AttachTruthValue}(PF, RT)\);
9. \(\text{PropagateTruthValue}(PF, RT)\);
10. let \(CPS\) be the set of all conflicting pairs of \(RT_i\);
11. 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

\[\begin{array}{l}
\text{(CP}_1) 
C_i \text{ and } D_i \text{ are respectively the projections of } C \text{ and } D \\
\text{ onto } \mathbb{R}^i. \\
\text{(CP}_2) 
C \text{ and } D \text{ are derived from the same path of } T_k. \\
\text{(CP}_3) 
\text{above } C \text{ and } D, \text{ every polynomial in their common projection factor set } P \text{ has the same sign.} \\
\text{(CP}_4) 
C \text{ and } D \text{ have opposite attached truth values.}
\end{array}\]

Let \(C\) and \(D\) be two \(k\)-level cells satisfying (CP\(_1\)), (CP\(_2\)), and (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\(_2\)) and (CP\(_3\)) 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 5.

**Theorem 2.** Algorithm 5 constructs a projection definable CAD tree.

**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 truth 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.

1. Any \(\Gamma_{i-1}\) will become projection definable after finitely many steps.
2. When \(RT\) gets refined, no conflicting pairs of level \(i\) higher than \(i\) will be generated.
3. When \(RT\) gets refined, no conflicting pairs of level \(i\) higher than \(i\) will be generated.

The correctness of (2.1) follows from Proposition 2. If (2.2) does not hold, then there exist a path \(\Gamma_{i-1}\) and two cells \(C_i\), \(D_i\) such that \(C_i\) and \(D_i\) are derived from some children of \(\Gamma_{i-1}\), and all \(i\)-level polynomials in their projection factor set evaluate at \(C_i\) and \(D_i\) into the same sign. This is a contradiction to Lemma 2.

Next we prove (2.3). Assume that \(RT\) refines into a new tree \(RT'\). Let \(C_i'\) and \(D_i'\) be a \(j\)-level conflicting pair in \(RT'\). Let \(C' \text{ and } D'\) be their extension in \(RT'\). 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], where it was used to demonstrate that PL-CAD based QE may generate a CAD tree that is not projection definable.

**Example 3.** Let \(PF := (3x_2)(x_1^3 + x_2^3 - 1 = 0) \land (x_1 + x_2 < 0) \land (x_1 > -1) \land (x_1 < 1)\). The projection stage generates a CCT \(T\):
CCT, which allows us to obtain the solution formula of $PF$ of $P$. By definition, a polynomial $RT$ of the third, the fourth and the fifth cells. The cells $x_1 = 0$ are grouped together. For each group, they can be used again to do the simplification. Let $Φ$ be the result formula. If $Φ$ is not simple enough, we can gather $RT$ of $P$ together into a set, say $A$, then algorithms in $[17, 4]$ can be used to generate simple $RT$. If $Φ$ is not sign-invariant on other cells of $RT$. On the other hand, let $Γ$ be the right most path of $P$. It is known that any polynomial $p$ of $P_t$ 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 $P$, we observe that in many cases, the polynomials in $P_t$ are sign-invariant on each path of $P_t$, and thus also sign-invariant on every cell of $RT$ (although counter examples exist, see Example 4). Let $P_t$ be a path of $P_t$ such that each $p ∈ P_t$ is sign-invariant on $T$. $Γ$ is a CAD tree deduced from $Γ$. Let $P_t$ be the projection factor set of $Γ$. 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 $Z_c(Γ) ⊂ Z_c(p)$ or $Z_c(Γ) ∩ Z_c(p) = ∅$ holds. Such tests boils down to set-theoretical operations on constructible sets. In particular, we have the following result from $[5]$ on the first test.

**Lemma 3.** Let $Γ$ be a path of $CCT$. Let $p ∈ Q[x]$. Let $[R, H]$ be the associated regular system of $Γ$. Then $Z_c(Γ) ⊂ Z_c(p) if and only if $prem(p, R) = 0$.

**Remark 1.** To test $Z_c(Γ) ∩ Z_c(p) = ∅$, it is equivalent to test $Z_c(p, R, H) := Z_c(p) ∩ Z_c(R, H) = ∅$. Efficient operation exists for such test, see Lemma 6 of $[9]$ for details.

**Example 4.** Let $Dattel := z^2 + 3y^2 + 3x^2 - 1$. Let $f := (z^2) Dattel = 0 be the input formula. A sign-invariant CCT defined by $p$ is described as below.

$$x_1 = 0 \lor (x_1 < 0 \land 0 < x_1 + 1) \lor (0 < x_1 < 2x_1^2 < 1).$$

**Example 5.** Consider another input formula

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

A variant of Algorithm 3 (removing redundant atomic formula in each conjunction) generates:

$$(u_2 < 0 \land u_2^3 + u_2 - 3u_1u_2 = 0) \lor (u_2 = 0 \land u_1 = 0) \lor (u_2^3 - 4 \land u_1u_2 - 2 = 0) \lor (u_2^3 - 4 = 0 \land u_1u_2 + 4 = 0) \lor (0 < u_2^3 - 4 \land u_1 + u_2 - 3u_1u_2 = 0) \lor (0 < u_2^3 - u_1 - 4 \land u_1 + u_2 - 3u_1u_2 = 0)).$$

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 a 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 application of quantifier elimination. We have developed the interface of our QE procedure based on the `Logic` package of MAPLE. The following example shows how to use our procedure.

**Example 6 (Davenport-Heintz). The interface:**

$$f := \&\& \{c(x) \land (a \& b, a, b), \&\& ((a \& d) \&\& (b \& c))
\&\& (a = c) \&\& (b = 1) \implies (a \& b)^2\$$

QuantifierElimination(f);

$$(d - 1 = 0) \&\& (d + 1 = 0)$$
In [3], we have shown that our RC-CAD implementation is competitive to the state of art CAD implementation, such as QEPCAD and MATHEMATICA. In particular, RC-CAD is usually more efficient than them when there are more equational constraints to be exploited. For instance, neither QEPCAD nor MATHEMATICA can solve the examples blood-coagulation-2 and MontesS10 in 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.

\[
f := \forall (x, y, z), (1/200*x*z*(1 - 1/400*x) + y*z*x*(1 - 1/400*x) - 35/2*y^2 = 0) \& \land (500*(y + 1/200*x)^2*(1 - 1/700*z) - 5*z = 0);
\]

QuantifierElimination(f);

Example 8 (MontesS10). It takes about 26 seconds.

\[
f := \forall ((c2, s2, c1, s1),
\]

\[
(r-c1^2*(s1^2+c1^2)-c2^2)/2 \& \land (s1^2+c1^2-1=0) \& \land (s2^2+c2^2-1=0);
\]

QuantifierElimination(f);

\[
true
\]

Elementary dependence analysis suggests to set \((i, j) = n - j\) 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 \((i=0; i<n; i++)\) \{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];
\}

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^2+3;
Dattel := 3*x^2+3*y^2+z^2-1;
Zitrus := x^2*y^2*z^2-1;
\]

\[
f := \forall ((y, z), (Sattel=0) \& \land (Dattel=0) \& \land (Zitrus<0));
\]

QuantifierElimination(f);

\[
The output is the inequality:
\]

\[
387420489 x^{36} + 473513931 x^{34} + 1615049199 x^{32} - 522961745 x^{30} + 2179233963 x^{28} - 14860773459 x^{26} + 4331773751 x^{24} - 4592587657 x^{22} + 60356422059 x^{20} - 12647823472 x^{18} + 164389796305 x^{16} - 121571730573 x^{14} + 5482719755 x^{12} - 16059214980 x^{10} + 3210573925 x^8 - 446506947 x^6 + 43657673 x^4 - 1631864 x^2 - 40328.
\]

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 the shared memory per streaming multiprocessor, maximum number \(P\) of threads supported by each streaming multiprocessor, etc.) we aim at automatically generate 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 \(c\),

\[
for(i=0; i<n; i++) \{ c[i] = 0; c[i+n] = 0;\}
\]

\[
f(i=0; i<n; i++) \{for(j=0; j<n, j++)\}

\[
c[i+j] += a[i] * b[j];
\]

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^2+3;
Dattel := 3*x^2+3*y^2+z^2-1;
Zitrus := x^2*y^2*z^2-1;
\]

\[
f := \forall ((y, z), (Sattel=0) \& \land (Dattel=0) \& \land (Zitrus<0));
\]

QuantifierElimination(f);

\[
The output is the inequality:
\]

\[
387420489 x^{36} + 473513931 x^{34} + 1615049199 x^{32} - 522961745 x^{30} + 2179233963 x^{28} - 14860773459 x^{26} + 4331773751 x^{24} - 4592587657 x^{22} + 60356422059 x^{20} - 12647823472 x^{18} + 164389796305 x^{16} - 121571730573 x^{14} + 5482719755 x^{12} - 16059214980 x^{10} + 3210573925 x^8 - 446506947 x^6 + 43657673 x^4 - 1631864 x^2 - 40328.
\]
to the previous system

\[
\begin{align*}
0 < n \\
0 \leq i \leq n \\
0 \leq j \leq n \\
i = n - j \\
p = i + j.
\end{align*}
\]

(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:

\[
\begin{align*}
B &> 0 \\
n &> 0 \\
0 \leq b \leq 2n/B \\
0 \leq u < B \\
0 \leq u \leq 2n - Bb \\
p &= bB + u,
\end{align*}
\]

from where we derive the following program:

```cpp
for (p=0; p<=2*n; p++) c[p]=0;
parallel_for (b=0; b<= 2 n / B; b++) {
    for (p=0; p<=2*n; p++) c[p]=0;
    for (u=min(0,n-p); u<=min(n,2*n-p); u++) {
        p = b * B + u;
        for (t=0; 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 all variables appear in degree zero or one. (Otherwise an algorithm for solving semi-algebraic systems needs 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 \[4\]. Each of the remaining conjunctions of clauses leads to a specialized program corresponding to particular configuration like \(n \leq B\). 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 cstc2013jjys0002.

References


