Cylindrical Decomposition Under Application-Oriented Paradigms[1]

Andreas Seidl[2]

March 3, 2006

[1] Doctoral Dissertation submitted to the *Fakultät für Mathematik und Informatik* at the *Universität Passau.*

[2] http://andreasseidl.com

Contents

Abstract

Quantifier elimination (QE) is a powerful tool for problem solving. Once a problem is expressed as a formula, such a method converts it to a simpler, quantifier-free equivalent, thus solving the problem. Particularly many problems live in the domain of real numbers, which makes real QE very interesting. Among the so far implemented methods, QE by cylindrical algebraic decomposition (CAD) is the most important complete method. The aim of this thesis is to develop CAD-based algorithms, which can solve more problems in practice and/or provide more interesting information as output.

An algorithm that satisfies these standards would concentrate on *generic* cases and postpone special and degenerated ones to be treated separately or to be abandoned completely. It would give a solution, which is *locally* correct for a region the user is interested in. It would give *answers*, which can provide much valuable information in particular for decision problems. It would *combine* these methods with more specialized ones, for subcases that allow for. It would exploit degrees of freedom in the algorithms by deciding to proceed in a way that promises to be *efficient*. It is the focus of this dissertation to treat these challenges.

Algorithms described here are implemented in the computer logic system REDLOG and ship with the computer algebra system REDUCE.

Introduction

The results presented in this thesis are application-oriented extensions and modifications of the cylindrical algebraic decomposition algorithm (CAD) for real quantifier elimination (QE).

A Tool for Problem Solving

Quantifier elimination (QE) is a versatile tool for solving a wide range of problems. It proceeds in two steps:

$$\text{Problem} \xrightarrow{\text{Modelling}} \text{Formula} \xrightarrow{\text{QE}} \text{Solution}$$

In a first step, the problem is modeled by a formula. This step is usually straightforward and can sometimes be performed automatically, e.g. in case of electrical network analysis, or schematically, e.g. in case of geometrical theorem proving. A formulation is usually easy to comprehend, but not yet useful. Quantifier elimination is the tool to find an equivalent, but simpler formulation. Simpler here means not necessarily shorter. Instead it means that the quantifiers, which can be thought of as an infinite disjunction or conjunction ranging over all real numbers, are replaced by something finite. So quantifier elimination is a process of starting with seemingly infinite cases, but then exposing the finite cases that there really are. This sounds a little bit like a wonder, and in fact it is: only for certain very special situations in algebra and logic it is possible. The field of reals with the language of ordered rings is such a lucky situation. This setting, shortly called *real quantifier elimination*, is the setting with which we will be concerned.

History and Importance of CAD

Some historic remarks are in order. Tarski was the first to realize the existence of a method for real quantifier elimination in the 1930s, but his method was not published until the late 1940s[1]. Tarski's method was not only too complicated to implement on computers in those days, but also its theoretical complexity was less than optimal. In

[1] cf. [Tar48], for a reprint see [CJ89]

1973 Collins found a method called cylindrical algebraic decomposition (CAD). Today this is the most important implemented general method for real quantifier elimination. To substantiate the importance of the method, here is a list of applications:

- The analysis of partial differential equations is one of the best studied application area. This includes stability analysis [HLS97] of PDEs and a method for deciding them to be elliptic [SW03].
- Applications of quantifier elimination in control theory [ADY$^+$96, Jir97] and in the analysis of hybrid systems, in particular for the computation of the reachability space, are also extensively studied [ADY$^+$96, Jir97].
- Ioakimidies has applied QE in the area of theoretical mechanics [Ioa99].
- Quantifier elimination and simplification methods can also be applied in the area of Hopf bifurcations [KW00].
- Motion planning for one or more robots in a time dependent environment was also studied [Wei01b, Wei01a, DW].
- Sturm suggested to use QE for the design, analysis and diagnosis of electrical networks [Stu99b].
- Dolzmann has described in his doctoral dissertation how to solve scheduling problems for the traditional dedicated machine model and for project networks by applying the extended quantifier elimination [Dol00].
- Problems in the area of geometry and computer aided design may be also solved by quantifier elimination. [Stu00, Stu99a].

Challenge

Looking at the above list of application areas one might wonder, why not everybody is using the method extensively. There are two main reasons for this:

1. The flexibility of the method comes at a price. This price is the theoretical complexity, which is doubly exponential. As a result the method does not scale well in practice.
2. Most engineers or mathematicians, who have applications for the method, are either unaware of the method, or put off by the difficulties of learning to formulate the problem in the QE framework and to use existing implementations.

These issues were addressed in the past in several ways:

1. Many improvements were suggested to the method to avoid unnecessary computations. These improvements cover every aspect of the method. There has been 30 years of progress and active research.
2. By restricting the method to input of a special kind, or by adopting the method to the specific problem, the overhead for a particular class of examples or problems can be reduced.
3. Within recent years, several implementations became available. An interesting approach is the package SYNRAC, which is developed by a group at the Fujitsu Laboratories. It allows the engineers to use QE without actually having to know something about logic and formulas.

Contents of this Thesis

Let us now focus on the scope of this thesis. In the past, three paradigms for QE were developed, originally for the virtual substitution (VS) method, which is an alternative, but restricted, real QE method.

1. *Generic Quantifier Elimination* is based on the observation that much work is done for special cases, which might not be of much interest. Thus such cases are excluded, but kept track of.
2. *Local Quantifier Elimination* caters for situation where values of interest for some or all parameters are known. Then a result is returned that is correct for a certain area where the points of interest reside in.
3. *Quantifier Elimination with Answers* provides parametric sample points, in case an existential request can be satisfied, or parametric counter examples, in case a universal request cannot be satisfied.

This thesis makes three core contributions to deal with the issues above and to push the applicability and accessability of the CAD method forward:

1. We show that the aforementioned paradigms generic QE, local QE, and QE with answers can be successfully transferred to the CAD method.
2. We investigate the effects of changing the projection order and a linear transformation of variables on the CAD method and show that these methods can be utilized to dramatically improve the practical complexity.
3. We present an implementation as part of the computer algebra system REDUCE.

Let us now summarize how the content is organized in the main part of this thesis.

Chapter 1 clarifies notation and introduces other basic tools.

Chapter 2 introduces CAD in the relevant detail. This establishes the theoretical framework on which this work is based.

Chapter 3 introduces the concept of cylindrical subdecompositions. This provides an abstraction layer, from which generic CAD and local CAD are derived as applications.

Chapter 4 deals with applications of cylindrical subdecompositions. Most notably these applications include generic CAD and local CAD. In addition we show that these variants can be combined.

Chapter 5 extends the CAD algorithm to give parametric answers and parametric counter-examples.

Chapter 6 shows how efficient projection orders can be constructed.

Chapter 7 introduces and investigates the regularization of formulas.

Chapter 8 discusses the author's implementation.

Chapter 9 gives the overall conclusions.

Acknowledgment

The author has been supported by the DFG, Project WE 566/5, and by the European RTNetwork RAAG, contract no. HPRN-CT-2001-00271.

Chapter 1

Preliminaries

The aim of this chapter is to introduce some notions and to synchronize the reader's notation with the notation used in this thesis. In particular, we will mention some general conventions and notations, look at algebra where we give algorithms to compute subresultants, and introduce some concepts in logic.

1.1 General Conventions

Before we start some general conventions should be mentioned. The natural numbers $\mathbb{N}$ include zero. An algorithm terminates, if a Return command is encountered. The notion $\mathbf{a}$ is a shorthand for $a_1, \ldots, a_n$ for some n which is either known from the context, or which does not matter. If $f : A \to B$ is a map and $S \subseteq A$, then $f[S] \subseteq B$ denotes the image of S under f.

1.2 Algebra

Ring always denotes a commutative ring with unit. In this section, D will be a ring. $\mathbb{A}$ denotes the set of all real algebraic numbers.

We assume that the reader is familiar with the notion of a *polynomial ring in* $X_1, \ldots, X_n$ *over* D, denoted as $D[X_1, \ldots, X_n]$. As we will only consider polynomials over infinite integral domains we need not make a difference between polynomials and polynomial functions.

1.2.1 Univariate Notions on Multivariate Polynomials

1.1 Definition
Let f be a polynomial in $x_1 \ldots, x_j$ ($j \geq 1$), i.e. $f \in D[x_1 \ldots, x_j]$. We can view f as element of $D[x_1 \ldots, x_{j-1}][x_j]$. There is a unique $n \geq -1$ and unique $p_n, \ldots, p_0 \in$

$D[x_1 \ldots, x_{j-1}]$ such that

$$f = \sum_{i=0}^{n} p_i x_j^i$$

and $p_n \neq 0$, if $n \geq 0$. We define the following functions:

- The *degree of f in x_j*, denoted as $\deg(f, x_j)$ is n. In particular, this defines $\deg(0, x_j) := -1$. If $\deg(f, x_j) \leq 0$ then we call f a *constant polynomial in x_j*.
- The leading coefficient of f in x_j, denoted as $\mathrm{lc}(f, x_j)$ is defined to be p_n, if $f \neq 0$. For the zero polynomial, define $\mathrm{lc}(0, x_j) := 0$.
- The *reductum of f in x_j*, denoted as $\mathrm{red}(f, x_j)$ is $\sum_{i=0}^{n-1} p_i x^i$. In particular, if $n \in \{-1, 0\}$, i.e. if f is a constant polynomial in x_j, then $\mathrm{red}(f, x_j)$ equals the empty sum, which is 0.
- The *(formal) derivation of f in x_j*, denoted as $\mathrm{der}(f, x_j)$ is $\sum_{i=1}^{n} np_n x^{n-1}$. In particular, if f is constant in x_j, then $\mathrm{der}(f, x_j) = 0$.
- The list $(p_n, \ldots, p_0)$ is called the *list of coefficients of f wrt. x_j*. In particular, the empty list is the list of coefficients of 0.

For improved readability we allow the notations $\deg_x(f)$, $\mathrm{lc}_x(f)$, $\mathrm{red}_x(f)$ and $\mathrm{der}_x(f)$ instead of $\deg(f, x)$, $\mathrm{lc}(f, x)$, $\mathrm{red}(f, x)$ and $\mathrm{der}(f, x)$ as well. The variable can be omitted, if it is known from the context.

1.2 Remark
With these definitions we have in any case:

$$f = \mathrm{lc}(f, x_j) x_j^{\deg(f, x_j)} + \mathrm{red}(f, x_j)$$

This gives a recursion scheme on polynomials, as the leading coefficient has smaller degree and as the reductum is shorter than the original polynomial.

1.2.2 Multivariate Notions on Multivariate Polynomials

1.3 Definition
For variables $(\mathbf{x}) = (x_1, \ldots, x_r)$ and a polynomial $f \in D[\mathbf{x}]$ choose a minimal (wrt. subset ordering) $E \subseteq \mathbb{N}^r$ and a family $(d_e)_{e \in E}$ in $D \setminus \{0\}$ such that

$$f = \sum_{\mathbf{e} \in E} d_{\mathbf{e}} \prod_{i=1}^{r} x_i^{e_i}.$$

Define:

- The *monomials of f wrt.* $\mathbf{x}$ to be[1]
$$\operatorname{mons}_{\mathbf{x}}(f) := \left\{ d_{\mathbf{e}} \prod_{i=1}^{r} x_i^{e_i} \,\middle|\, e \in E \right\}.$$
- The *terms of f wrt.* $\mathbf{x}$ to be
$$\operatorname{terms}_{\mathbf{x}}(f) := \left\{ \prod_{i=1}^{r} x_i^{e_i} \,\middle|\, e \in E \right\}.$$
- The *coefficients of f wrt.* $\mathbf{x}$ to be
$$\operatorname{coeffs}_{\mathbf{x}}(f) := \{ d_{\mathbf{e}} \mid e \in E \}.$$
- The *number of monomials of f wrt.* $\mathbf{x}$ to be
$$\operatorname{nom}_{\mathbf{x}}(f) := |\operatorname{mons}_{\mathbf{x}}(f)|.$$
Note that the number of monomials equals $|\operatorname{terms}_{\mathbf{x}}(f)|$, the number of terms.
- The *total degree of f wrt.* $\mathbf{x}$ to be
$$\operatorname{tdeg}_{\mathbf{x}}(f) := \begin{cases} \max_{\mathbf{e} \in E} \left(\sum_{j=1}^{r} e_j \right), \text{ if } E \neq \emptyset \\ -1, \text{ otherwise.} \end{cases}$$

Regarding the total degree, $\operatorname{tdeg}(f)$ denotes the total degree of f wrt. all variables actually occurring in f, if they are known from the context.

1.2.3 Resultants and Discriminants

Elimination Theory considers the following question (cf. [Usp48], p.277, [MS99], p.33):

> If f and g are two univariate polynomials with coefficients in an integral domain A, find necessary and sufficient conditions (on the coefficients) for f and g to have common roots in an extension of the domain A.

Resultants are the tool of choice for this task. A common way to introduce the resultant is via the Sylvester matrix.

1.4 Definition (Sylvester matrix)
Let $f = \sum_{i=0}^{m} a_i x^i$, $g = \sum_{i=0}^{n} b_i x^i$ with $m = \deg_x(f)$, $n = \deg_x(g)$ be two polynomials in x. If $f = 0$ or $g = 0$ then $\operatorname{SYL}(f, g, x) = \emptyset$, the 0×0-matrix. Otherwise, $\operatorname{SYL}(f, g, x)$ is defined as the following $m + n \times m + n$-matrix:

$$(l, c) \mapsto \begin{cases} a_{m-(c-l)} & \text{if } 1 \leq l \leq m \text{ and } l \leq c \leq l + m \\ b_{l-c} & \text{if } n + 1 \leq l \leq m + n \text{ and } l - n \leq c \leq l \\ 0 & \text{otherwise.} \end{cases}$$

[1] Note that what we call monomials is called terms in some places.

This definition leads to a simple algorithm to generate the Sylvester matrix.

1.5 Algorithm (Sylvester matrix)

$$S \longleftarrow \mathrm{SYL}(f, g, x)$$

Input: Univariate polynomials f, g in x over some ring D
Output: A square Matrix S over D with $\deg_x(f) + \deg_x(g)$ rows.

1. $m := \deg_x(f)$, $n := \deg_x(g)$, $f = \sum_{i=0}^{m} a_i x^i$, $g = \sum_{i=0}^{n} b_i x^i$.
2. Initialize S with $0 \in R^{m+n \times m+n}$.
3. For l from 1 to m do for c from l to $l + m$ do $S_{l,c} := a_{m-(c-l)}$.
4. For l from $n + 1$ to $m + n$ do for c from $l - n$ to l do $S_{l,c} := b_{l-c}$.
5. Return S.

1.6 Definition (resultant)
Let f, g be two polynomials in x. Then the *resultant of f and g wrt. x* is the determinant of the Sylvester matrix of f and g wrt. x:

$$\mathrm{res}(f, g, x) := \det(\mathrm{SYL}(f, g, x)).$$

The following facts about the resultant are taken from [Mig91]:

1.7 Theorem (properties of the resultant)
With notation from Algorithm SYL*:*

1. *res$(f, 0, x) = 0$*
2. *res$(g, f, x) = (-1)^{mn}$res(f, g, x)*
3. *If $\deg(f, x) = m \leq n = \deg(g, x)$ then res$(f, g, x) = a_m^{n-m}$res$(f, \mathsf{rem}(g, f, x), x)$.*
4. *If $f \neq 0$ and $g \neq 0$ then res$(f, g, x) = 0$ iff f and g have a common nontrivial factor.*
5. *Suppose f and g are polynomials over an integral domain and $\alpha_1, \ldots \alpha_m$ are the roots of f and $\beta_1, \ldots, \beta_n$ are the roots of g (in a suitable extension), then*

$$\textit{res}(f, g, x) = a_m^n \prod_{i=1}^{m} g(\alpha_i) = (-1)^{mn} b_n^m \prod_{j=1}^{n} f(\beta_j) = a_m^n b_n^m \prod_{i=1}^{m} \prod_{j=1}^{n} (\alpha_i - \beta_j).$$

Assertion 4 of the last Theorem is sometimes called the *standard theorem on resultants*, see e.g. [Loo82], p. 178.

Resultants can be used to find intersections of curves. A curve, however, can intersect itself. Here discriminants are useful.

1.8 Definition (discriminant)
Let f be a polynomial in x over D, then the *discriminant of* f *wrt.* x is defined to be:

$$\mathrm{dis}(f,x) := \mathrm{res}(f, \mathrm{der}(f,x), x).$$

1.2.4 Subresultants and Their Coefficients

By removing some rows and columns from the Sylvester matrix $\mathrm{SYL}(f,g,x)$, we define a modified Sylvester matrix as follows:

1.9 Algorithm (modified Sylvester matrix)

$$M \longleftarrow \mathrm{SYLMOD}(f,g,x,i,j)$$

Input: Polynomials f, g in x over D, natural numbers i, j.
Output: A square Matrix S over D.

```
1. M := SYL($f,g,x$)
2. For $r$ from $m+n$ downto $(m+n)-j+1$ do remove the $r$-th row from $M$.
3. For $r$ from $n$ downto $n-j+1$ do remove the $r$-th row from $M$.
4. For $c$ from $m+n$ downto $m+n-i-j+1$ do remove the $c$-th column from $M$.
5. For $c$ from $m+n-i-j-1$ downto $m+n-2j$ do remove the $c$-th column from $M$.
6. Return $M$.
```

The Sylvester matrix is a square $m+n \times m+n$-matrix. The modified matrix has $m+n-2j$ rows and columns, hence it is a square matrix again, and the following definitions are well defined.

1.10 Definition (subresultant, principal subresultant coefficient)
Define the j-th principal subresultant coefficient and the j-th subresultant of two polynomials $f, g \in D[x]$ as follows:

- $\mathrm{psc}(f,g,x,j) := \det(\mathrm{SYLMOD}(f,g,x,j,j))$, the *$j$-th principal subresultant coefficient of f and g wrt. x.*

- $\mathrm{sres}(f,g,x,j) := \sum_{i=0}^{j} \det(\mathrm{SYLMOD}(f,g,x,j,i))x^i$, the *$j$-th subresultant of f and g wrt. x.*

We note some properties of psc and sres.

1.11 Lemma

1. $psc_0(f,g,x) = sres_0(f,g,x) = res(f,g,x)$
2. $psc_j(f,g,x) = lc_x(sres_j(f,g,x))$
3. $\deg_x(sres(f,g,x,j)) = j$

The notions introduced and discussed in this section are needed in particular to define the projection operator for CAD.

1.3 First-Order Predicate Logic

In order to practice logic, one needs to provide the syntactical means first, i.e. language, terms, atomic formulas and formulas. Based on these notions one can give calculi to deduce formally from a given set of formulas new formulas. Investigation of this is called *proof theory.* One can then proceed to consider *structures*, i.e. a collection of a language, a given set called *universe* and semantic functions for every symbol of the language. Investigation of mathematical concepts based on these notions is called *model theory.* We will recall some needed notions, as in practice many different—but quite equivalent—ways of defining things abound. See [EFT78] or [SW] for a thorough introduction.

We start with the language, the terms, the atomic formulas, and the formulas we will deal with, i.e. the *syntax.*

1. We focus on the language $(0^{(0)}, 1^{(0)}, -^{(1)}, +^{(2)}, \cdot^{(2)}; <^{(2)})$ of ordered rings. (The arity of the function and relation symbols are given in braces in the exponent.)
2. Terms over this language can be considered as polynomial expressions with integer coefficients.
3. Atomic formulas are polynomial equations or order inequalities.
4. First-order formulas, or short: formulas, are the smallest set with the properties
   (a) true and false are formulas,
   (b) atomic formulas are formulas,
   (c) negation, conjunction, disjunction, implication, and equivalence of formulas are formulas, and
   (d) existential or universal quantification of formulas are formulas.

Strictly speaking, terms and atomic formulas are strings, where the following kinds of symbols are used: *special symbols* consisting of a opening round bracket "(", a closing round bracket ")", a comma "," and an equation symbol "=", an infinite but countable set $\mathcal{V}$ of variables that is assumed to be fixed, *logical operators* $\{\mathsf{true}, \mathsf{false}, \neg, \wedge, \vee, \longrightarrow, \longleftrightarrow\}$, and *quantifier* symbols $\{\forall, \exists\}$. It is assumed that these sets of symbols together with $\{0, 1, -, +, \cdot, <\}$ are pairwise disjoint.

Each formula falls into exactly one of the cases (a)–(d) above. This gives a natural recursion scheme on formulas. Several definitions for formulas can be made either by structural induction, or by viewing a formula as a string.

Without loss of generality, the atomic formulas can be written as $p = 0$ or $p < 0$. Although p is strictly speaking a string, we identify p with the denoted polynomial, the *polynomial of the corresponding atomic formula.* Then the *polynomials of a formula* is the set of all polynomials in all atomic subformulas.

The variables occurring in a formula φ are denoted with $\mathcal{V}(\varphi)$. The free variables $\mathcal{V}_f(\varphi)$ of φ are the elements x of $\mathcal{V}(\varphi)$ that occur in a position that is not within the range of a quantifier $\mathsf{Q}x$, where $\mathsf{Q} \in \{\exists, \forall\}$. A formula is called quantifier-free, if it contains neither the symbol $\exists$ nor $\forall$,

For a term t and a variable x we denote with $\varphi[t/x]$ the result of substituting t for x in φ. Consequently, $\varphi[t_1/x_1, \ldots, t_n/x_n]$ denotes the simultaneous substitution of the terms $t_1, \ldots, t_n$ for the variables $x_1, \ldots, x_n$ in φ.

To define the semantics of a formula (wrt. $\mathbb{R}$), one can augment the formula by a list of pairwise distinct variables $(x_1, \ldots, x_k)$. This gives an *extended formula.* For such an extended formula one can define a semantic map $\mathbb{R}^k \to \{0, 1\}$ that gives information whether a formula is valid or not at a certain point. In addition, extended formulas define a certain subspace. See Section 3.1 for more details.

There are normal forms for formulas. The normal form which is of most interest to us is the *prenex normal form* (PNF). Here the formula starts with quantifiers, and after that there is a quantifier-free part, called the *matrix* of the formula. When working with formulas in PNF, the following notations can be useful:

1.12 Definition (variable block, quantifier block)
For a list $(\mathbf{y}) = (y_1, \ldots, y_m)$ of variables, $\mathsf{Q} \in \{\forall, \exists\}$, and a formula ψ let $\mathsf{Q}\mathbf{y}(\psi)$ denote the formula $\mathsf{Q}y_1(\ldots(\mathsf{Q}y_m(\psi))\ldots)$. In this context, we will call $(\mathbf{y})$ a variable block (or: block of variables), and $\mathsf{Q}\mathbf{y}$ a quantifier block (or: block of quantifiers).

1.13 Definition (variable block list)
Consider an extended prenex formula

$$\varphi(x_1, \ldots, x_k) := \mathsf{Q}_{k+1}x_{k+1}(\ldots \mathsf{Q}_r x_r(\psi)\ldots).$$

There exists a unique $n \in \mathbb{N}$ and unique integers $i_1, \ldots, i_n$ such that with the additional definition $i_0 := 1$, $i_{n+1} := r + 1$, $\mathsf{Q}_{r+1} := \emptyset$ the following holds for all $1 \leq j < n$:

1. $k + 1 = i_1 < \ldots < i_n \leq r$

2. $\mathsf{Q}_{i_j} = \ldots = \mathsf{Q}_{i_{j+1}-1}$
3. $\mathsf{Q}_{i_{j+1}-1} \neq \mathsf{Q}_{i_{j+1}}$

Then we can write with the notation introduced above

$$\varphi(x_1,\ldots,x_k) := \mathsf{Q}_{i_1}x_{i_1}\ldots x_{i_2-1}(\ldots \mathsf{Q}_{i_n}x_{i_n}\ldots x_{i_{n+1}-1}(\psi)\ldots),$$

or, with $(\mathbf{x}^{(j)}) := (x_{i_j},\ldots,x_{i_{j+1}-1})$ for $0 \leq j \leq n$

$$\varphi(\mathbf{x}^{(0)}) := \mathsf{Q}_{i_1}\mathbf{x}^{(1)}(\ldots \mathsf{Q}_{i_n}\mathbf{x}^{(n)}(\psi)\ldots).$$

Then $((\mathbf{x}^{(0)}),\ldots,(\mathbf{x}^{(n)}))$ is called the *variable block list of* $\varphi(x_1,\ldots,x_k)$.

1.4 Summary

This preliminary chapter featured some general conventions, some definitions from algebra, which are used for defining a projection operator later, and some notions for logic, which are given to allow the reader to quickly synchronize with the notions used in this thesis.

Chapter 2

Cylindrical Algebraic Decomposition

This thesis will deal with modifications of the cylindrical algebraic decomposition (CAD) algorithm. In order to have a basis for this work, it is necessary to recall how the algorithm works. This establishes the notational and algorithmical framework, upon which the main part of this thesis builds. The plan of this chapter is as follows:

1. We introduce the notions regarding CAD. These are: region, cylinder, section and sector of a cylinder, decomposition, cell, algebraic cell, algebraic decomposition, stack, cylindrical decomposition, cylindrical algebraic decomposition, and sign-invariance.

2. We give the specification of a CAD algorithm and discuss how data can be represented. At this point we can only guess and verify that a mathematical object is a CAD, we have not yet an algorithm to construct a CAD constructively.

3. We introduce the important concept of delinability for one polynomial and for a set of polynomials and show how it can be utilized for sign-invariant stack construction.

4. We show how conditions for delineability can be given by means of Collins' projection operator. We give the specification of a projection operator and introduce projection sets.

5. We combine the application of a projection operator with stack construction to give a recursive CAD algorithm. Preferably this is done into two phases, the projection phase, where the projection set is computed, and the extension phase.

6. We specify a CADQE algorithm that performs real quantifier elimination (QE). We show how a CAD algorithm can be extended to a CADQE algorithm.

2.1 Notions Regarding CAD

In this section we define the notion *cylindrical algebraic decomposition* and specify what an algorithm should provide.

2.1 Definition (real variety and common zeros)
For variables $(\mathbf{x}) = (x_1, \ldots, x_j)$ we define the shorthand $I_j := \mathbb{Z}[\mathbf{x}]$. For a set of polynomials $A \subseteq I_j$ let $V_\mathbf{x}(A)$ denote the *real variety* of A, i.e. the set

$$V_\mathbf{x}(A) = \left\{(\mathbf{a}) \in \mathbb{R}^j \,\middle|\, \text{for all } f \in A : f(\mathbf{a}) = 0\right\}$$

of all common zeros of polynomials in A. Dually, the set of all *real zeros* of polynomials in A is defined as

$$\mathrm{Zeros}_\mathbf{x}(A) = \left\{(\mathbf{a}) \in \mathbb{R}^j \,\middle|\, \text{exists } f \in A : f(\mathbf{a}) = 0\right\}.$$

2.2 Remark
It holds that $\mathrm{Zeros}_\mathbf{x}(A) = V_\mathbf{x}(\prod A)$. This would be an alternative definition.

2.3 Definition (connected, region)
A non-empty subset of $\mathbb{R}^j$ is called *connected*, if it is connected in a topological sense wrt. the topology induced on $\mathbb{R}^j$ by the Euclidean metric. We call a connected subset a *region*.

2.4 Remark
Note that a region is non-empty. An open region is pathwise connected [Que01].

2.5 Definition (cylinder, section, sector)
If S is a region in $\mathbb{R}^j$, then:

1. $\mathrm{cyl}(S) := S \times \mathbb{R}$ denotes the *cylinder* over S.

2. The graph of a continuous map $f : S \to \mathbb{R}$, i.e. $\{(s, f(s)) \mid s \in S\}$, is called the *$f$-section* (of the cylinder over S). A subset C of $\mathbb{R}^{j+1}$ is called a section, if there is a $S \subseteq \mathbb{R}^j$ and a map $f : S \to \mathbb{R}$ such that C is the f-section.

3. Let f_1 be either a continuous map $S \to \mathbb{R}$ or the unique map $-\infty : S \to \{-\infty\}$ and f_2 be either a continuous map $S \to \mathbb{R}$ or the unique map $\infty : S \to \{\infty\}$. If $f_1 < f_2$ then the set $\{(s,t) \mid s \in S, f_1(s) < t < f_2(s)\}$ is called *(f_1, f_2)-sector* (of the cylinder over S). A subset C of $\mathbb{R}^{j+1}$ is called a sector, if there is a $S \subset \mathbb{R}^j$ and maps f_1, f_2 such that C is the (f_1, f_2)-sector.

2.6 Remark

1. If $Z \subseteq \mathbb{R}^{j+1}$ is a cylinder, then the set $S \subseteq \mathbb{R}^j$ such that $Z = \mathrm{cyl}(S)$ is uniquely determined. If C is a section, then S, $f : S \to \mathbb{R}$ and $S \times \mathbb{R}$, such that C is the f-section (of the cylinder $S \times \mathbb{R}$), are uniquely determined. The analogous result holds for sectors.

2. If $f_1 = -\infty$ or $f_2 = \infty$, then the (f_1, f_2)-sector has infinite diameter. But even for $f_1, f_2 \notin \{-\infty, \infty\}$ can the (f_1, f_2)-sector have infinite diameter. E.g. consider the (f_1, f_2)-sector for $f_1 :]-\infty, 0[\to \mathbb{R} : x \mapsto 0$ and $f_2 :]-\infty, 0[\to \mathbb{R} : x \mapsto x^2$.

2.7 Definition (decomposition, cell)
A *decomposition* U of $R \subseteq \mathbb{R}^j$ is a finite set of disjoint regions such that the union of U results in R. The elements of U are called *cells*.

2.8 Remark

1. $\emptyset$ is a decomposition for $\emptyset$.

2. $\{\{\emptyset\}\}$ is a decomposition for $\mathbb{R}^0 = \{\emptyset\}$. In fact, there is no other decomposition for $\mathbb{R}^0$.

3. Not for every $R \subseteq \mathbb{R}^j$ exists a decomposition. E.g. $j = 1$, $R = \mathbb{Q}$.

4. A decomposition differs from the well known set-theoretic notion of partition in that a partition is not required to be finite, and the elements are not required to be connected. But both, an element of a partition and a cell in a decomposition, are required to be non-empty.

5. The notion of cell is not simply a synonym for a region. Consider

$$R := \left\{(x, y) \in \mathbb{R}^2 \;\middle|\; y = 0 \text{ or } x \in \mathbb{Z}\right\}.$$

   R is a region, but $\complement R$ consists of infinitely many connected components. So R cannot be a cell in a decomposition of $\mathbb{R}^2$.

2.9 Definition (algebraic cell, algebraic decomposition)
A cell $C \subseteq \mathbb{R}^j$ is called *algebraic*, if there is an extended formula $\delta_C(x_1, \ldots, x_j)$ over the language of ordered rings such that C is the $\mathbb{R}$-realization of $\delta_C(x_1, \ldots, x_j)$. That is, $C = \mathbb{R}_{\delta_C(\mathbf{x})}$, where $\mathbb{R}_{\delta_C(\mathbf{x})} = \{(\mathbf{a}) \in \mathbb{R}^j | \mathbb{R} \models \delta_C(\mathbf{a})\}$. A decomposition is algebraic, if every cell is algebraic.

2.10 Remark

1. Note that an algebraic cell should more precisely be called *semi-algebraic*, but within CAD context it is a convention to just use the term *algebraic*.

2. δ_C can be assumed to be a quantifier-free formula due to the famous result by Tarski.

2.11 Definition (stack, cylindrical decomposition, CAD)

1. A *stack* over a cell C is a decomposition of $\mathrm{cyl}(C)$ such that the projection, which drops the last component, maps each cell of this decomposition to C.

2. A decomposition D_j of $\mathbb{R}^j$ is called *cylindric*, if $0 \leq j \leq 1$ or if $j > 1$ and D_j can be partitioned into stacks over cells of a cylindrical decomposition of $\mathbb{R}^{j-1}$, the *induced* cylindrical decomposition of D_j.

3. A decomposition that is cylindric and algebraic is called *cylindrical algebraic decomposition* (CAD).

2.12 Remark

1. The projection from $\mathbb{R}^j$ onto $\mathbb{R}^{j-1}$, $j \geq 1$, which drops the last component, can be denoted with $\pi_{(1,\ldots,j-1)}$. Then a decomposition S of $C \times \mathbb{R}$ is a stack over C iff
$$\left\{\pi_{(1\ldots,j-1)}[s] \,\middle|\, s \in S\right\} = \{C\}$$

2. The union of the cells of a stack forms a cylinder.

3. D_{j-1}, the induced cylindrical decomposition of D_j is uniquely determined: We have that $D_{j-1} = \left\{\pi_{(1,\ldots,j-1)}[C] \,\middle|\, C \in D_j\right\}$. Thus speaking of *the* induced decomposition is justified. Furthermore, by recursion, cylindrical decompositions $D_{j-2}, \ldots, D_0$ are induced and uniquely determined.

2.13 Definition (sign-invariant)
Let $A \subseteq I_j$ be a set of integral polynomials over $(\mathbf{x}) = (x_1, \ldots, x_j)$. An set $S \subseteq \mathbb{R}^j$ is called sign-invariant wrt. A and $(\mathbf{x})$, if there exists a family $(\sigma_f)_{f \in A}$ in $\{-1, 0, 1\}$ such that for all $s \in S$ and $f \in A$ $\text{sign}(f(s)) = \sigma_f$. If $(\mathbf{x})$ is known from the context, then we simply say S is sign-invariant wrt. A, or S is A-sign-invariant. A decomposition or stack is called A-sign-invariant, or sign-invariant wrt. A, if every cell is A-sign-invariant.

2.2 Algorithm Specification and Data Representation

With the definitions from the previous section, the basic specification of a CAD algorithm is:

2.14 (specification of a CAD algorithm)

$$D \longleftarrow CAD(A, (\mathbf{x})).$$

Input: A finite subset A of I_r and variables $(\mathbf{x}) = (x_1, \ldots, x_r)$, for a $r \geq 0$.
Output: A A-sign-invariant cylindrical algebraic decomposition D of $\mathbb{R}^r$ wrt. $(\mathbf{x})$, from which can be read:

1. The number and arrangement of the cells.

2. The sign of each element of A on each cell.

2.15 Definition (CAD problem, solution for a CAD problem)
We call a pair $(A, (\mathbf{x}))$ as specified as input in 2.14 a *CAD problem* and an A-sign-invariant decomposition D as specified as output in 2.14 a *solution* of a CAD problem.

In order to give an algorithm it has to be clear how the objects that are dealt with are represented. This is folklore for objects like identifiers, numbers, polynomials, and finite collections. Such representations have usually the property that the representant uniquely determines the represented object. (In general, the converse is not true: E.g. there is a machine representation of the integers where 0 has two representants. Distributive representation of polynomials is an additional example.)

Regarding CAD a straightforward approach would be to (a) represent cells by a Tarski formula and (b) represent a CAD as a list of cells.

Such an approach, however, is not suitable for our needs specified in (1) and (2) above. On the one hand, there is too little information. From a Tarski formula it is difficult to find a point in the described area. On the other hand, there is too much information. In fact, a sample point in each cell suffices for most quantifier elimination needs.

Thus let us state the following very clearly: The algorithms for CAD constructions for quantifier elimination will use a CAD representation that contains enough information for quantifier elimination purposes, but insufficient information to uniquely determine the mathematically underlying CAD. A cell will mainly be represented by a sample point of the cell. Instead of a CAD, more precisely a *cylindrical algebraic sample* (CAS) will be computed. This allows to compute the invariant sign of each polynomial of A on each cell.

Here is an example that demonstrates that by only taking sample points the original problem cannot be reconstructed: For $A = \left\{x_1^2 - x_2\right\}$ an A-invariant CAD wrt. (x_1, x_2) can be represented with the sample points $((0, -1), (0, 0), (0, 1))$, but this represents also an A-invariant CAD for $A = \left\{x_1^3 - x_2\right\}$ or $A = \left\{-x_1^2 - x_2\right\}$ wrt. (x_1, x_2).

So a sample point, possibly additional information, like for each polynomial of A the invariant sign on the cell, a describing formula for the cell, or a truth value, can be included in the representation.

The requirement about the cylindrical arrangement of the cells in the specification of an CAD algorithm can be satisfied by representing a decomposition as a tree, instead of a plain set. This preserves the information about the cylindrical arrangement of the cells. Alternatively, a cell can contain an index, which gives information about the position of the cell in the according CAD tree. Then a CAD can be represented as a plain list. Both approaches are equivalent.

Altogether we will formulate the algorithms in a mathematical way, i.e. they construct a CAD, but only the information needed is effectively constructed. This abstracts from representation details.

2.3 Delinability and Sign-invariant Stack Construction

Consider the following situation: For a set $S \subseteq \mathbb{R}^{j-1}$ and a set of polynomials $F \subseteq I_j$ the portion of $V(F)$ lying in the cylinder above S consists of disjoint sections, such that there is an F-sign-invariant decomposition of the cylinder above S into sections and sectors of roots of polynomials from F.

First we will introduce the notion of delinability to give a criterion for the just described situation. This notion was introduced by Collins in his original paper and slightly modified later on. We keep faithful to this notion and repeat it with some clarifications. By restricting our attention to regions we can give simpler conditions for delineability.

2.3.1 Delineability for one Polynomial

2.16 Definition
For a real polynomial $f(x_1, \ldots, x_r)$, $r \geq 2$ and a subset S of $\mathbb{R}^{r-1}$ we say that *the roots of f are delinable on S*, if the following holds:

1. There exists an $m \geq 0$ and positive integers $e_1, \ldots, e_m$ such that for all $(\mathbf{a}) \in S$ univariate $f(\mathbf{a}, x_r)$ has exactly m distinct roots with multiplicities $e_1, \ldots, e_m$.

Furthermore, there exists a $k \geq 0$ and maps $f_1, \ldots, f_k$ from S to $\mathbb{R}$, such that

2. $f_1 < \ldots < f_k$ are continuous.

3. For all $(\mathbf{a}) \in S$ and all $1 \leq i \leq k$ we have that $f_i(\mathbf{a})$ is a root of univariate $f(\mathbf{a}, x_r)$ of multiplicity e_i.

4. For all $(\mathbf{a}) \in S$, $b \in \mathbb{R}$ we have that $f(\mathbf{a}, b) = 0$ implies that there exists a $1 \leq i \leq k$ such that $b = f_i(\mathbf{a})$.

In this context we say that $f_1, \ldots, f_k$, *delineate the real roots of f on S*. One also simply says *f is delineable over S* or *f has the delineability property over S*. This defines delineability to be a ternary predicate.

2.17 Remark
With the notation from Definition 2.16:

1. If f is the zero polynomial, then $f(\mathbf{a}, x_r)$ has an infinite number of roots for each $(\mathbf{a}) \in S$. Thus it cannot be delineable, as no finite k exists as required in the definition.

2. If f vanishes at some point in S, then f is not delineable on S. In particular, if f vanishes identically on $S \neq \emptyset$, then f is not delineable on S.

   Proof. Choose $(\mathbf{a}) = (a_1, \ldots, a_{r-1}) \in S$ such that $f(\mathbf{a}, x_j) = 0$. Given any $m \geq 0$ and integers $e_1, \ldots, e_m$ we note that univariate $f(\mathbf{a}, x_r)$ has not exactly m distinct roots with multiplicities $e_1, \dots, e_m$, as in fact it has infinitely many roots. Thus condition 1 in the definition does not hold. □

3. If f is a constant, but non-zero polynomial, then $f(\mathbf{a}, x_r)$ has no root for each $(\mathbf{a}) \in S$. Thus it is delineable, with $k = 0$ and $m = 0$.

4. Consider $S = \emptyset$. Then for any choice of m, $\mathbf{e}$ condition 1 is satisfied. Moreover, for $k = 0$ (or, for $k = 1$ and f_1 being the empty map) conditions 2–4 are satisfied. Hence f is delineable over S.

5. Let us drop the condition $r \geq 2$ and consider $r = 1$. Then, as a subset of $\mathbb{R}^0$, we have either $S = \emptyset$, in which case f is delineable over S. Or, we have $S = \{\emptyset\}$. Then condition 1 is easily verified, and, for k being the number of distinct roots of f, and, for f_i mapping $\emptyset$ to the ith root, conditions 2–4 hold as well.

6. If we assume $S \neq \emptyset$, m is uniquely determined. So are k and $f_1, \dots, f_k$.

7. Let $g(\mathbf{x})$ be another real polynomial. If $V_{\mathbf{x}}(f) \cap (S \times \mathbb{R}) = V_{\mathbf{x}}(g) \cap (S \times \mathbb{R})$ then f is delineable over S iff g is delineable over S. I.e. changing a polynomial does not change the delineability property, if the variety is not changed. Thus we can e.g. perform content elimination (divide a polynomial by its domain content).

2.18 Example
Consider $f = x_1^2 + x_2^2 - 1$, $S =]-1, 1]$ and $(\mathbf{x}) = (x_1, x_2)$. Then $f(0, x_2)$ has two real roots, each of multiplicity 1, and $f(1, x_2)$ has one real root of multiplicity 2. Thus the first condition cannot be satisfied and f cannot be delineable over S.

Now consider $S =]-1, 1[$. Set $m := 2$, $(\mathbf{e}) := (1, 1)$. The first condition is satisfied. Set $k := 2$ and $f_1 : S \to \mathbb{R} : a \mapsto -\sqrt{1 - a^2}$, $f_2 : S \to \mathbb{R} : a \mapsto \sqrt{1 - a^2}$. This satisfies conditions 2–4. Thus f is delineable over S.

Let us consider, with the notation of Definition 2.16, two conditions:

(D1) The leading coefficient of f does not vanish on S. I.e. for all $(\mathbf{a}) \in S$ we have $(\mathrm{lc}_{x_r}(f))(\mathbf{a}) \neq 0$.

(D2) f has a constant number of (complex) roots on S. I.e. there exists a $n \in \mathbb{N}$ such that for all $(\mathbf{a}) \in S$ univariate $f(\mathbf{a}, x_r)$ has n distinct roots.

2.19 Lemma (necessary conditions for delineability)
Let $f(x_1, \dots, x_r)$ be a real polynomial with $r \geq 2$ and $\emptyset \neq S \subseteq \mathbb{R}^{r-1}$. If the roots of f are delineable on S, then for each $(\mathbf{a}) \in S$ we have that conditions (D1) and (D2) hold.

Proof. As $S \neq \emptyset$, and as the zero polynomial is not delineable on non-empty sets, as remarked in 2.17, we have that f is not the zero polynomial. Thus, $f(\mathbf{a})$ has a constant number $m > 0$ of roots, each with constant multiplicity $e_i > 0$. We have that

$deg_{x_r}(f(\mathbf{a})) = \sum_{i=1}^{m} e_i > 0$ is constant on S, i.e. the leading coefficient cannot vanish and (D1) holds.

In addition, (D2) follows immediately from the definition. □

Conversely, if we add the condition of S being a connected set, then these two conditions are sufficient for delineability.

2.20 Lemma (sufficient conditions for delineability)
Let $f(x_1, \ldots, x_r)$ be a real polynomial with $r \geq 2$ and $S \subseteq \mathbb{R}^{r-1}$. Then the roots of f are delineable on S, if S is connected and conditions (D1) and (D2) hold.

For the proof see Theorem 1 in [Col75]. These lemmata underline the importance of connected sets. This motivates why such objects deserve a special name: *regions*. Altogether we have as an immediate consequence:

2.21 Theorem (characterization of delineability on regions)
Let $f(x_1, \ldots, x_r)$ be a real polynomial with $r \geq 2$ and $S \subseteq \mathbb{R}^{r-1}$ a region. Then the roots of f are delineable on S, iff conditions (D1) and (D2) hold.

2.3.2 Delineability for Sets of Polynomials

We need to extend the definition of delineability for a polynomial to delineability for sets of polynomials.

2.22 Definition
For a set $A = \{f_1, \ldots, f_n\} \subseteq I_r$ of polynomials we say the roots of A are delineable on $S \subseteq \mathbb{R}^{r-1}$, if the roots of $\prod_{i=1}^{n} f_i$ are delineable on S.

2.23 Remark
With the notation from Definition 2.22:

1. If A contains the zero polynomial, then A is not delineable.
2. If a polynomial in A vanishes identically on S, then A is not delineable on S.
3. If A is empty, then $\prod_{i=1}^{0} f_i = 1$, and hence A is delineable.
4. Instead of using the product of all polynomials, we could have defined delineability for sets of polynomials in a different, but equivalent way: A is delineable over S iff the following two conditions hold:
   (a) Every element of A is delineable on S
   (b) The sections of cyl(S) belonging to different $f, g \in A$ are either disjoint or identical.

5. Let $B = \{g_1(\mathbf{x}), \ldots, g_l(\mathbf{x})\}$ be another set of real polynomials. If $V_{\mathbf{x}}(\prod_{i=1}^{n} A) \cap S \times \mathbb{R} = V_{\mathbf{x}}(\prod_{i=1}^{l} g_i) \cap S \times \mathbb{R}$ then A is delineable over S iff B is delineable over S. I.e. changing a set of polynomials does not change the delineability property, if the variety of the product polynomial is not changed. Thus we can e.g. remove constant polynomials or perform factorization.

2.3.3 Stack Construction

The notion of delineability was introduced to allow for the construction of a sign-invariant stack over a base cell.

2.24 Algorithm (sign-invariant algebraic stack construction)

$$(\mathbf{C}) \longleftarrow \mathrm{STACK}(B, F, (\mathbf{x}))$$

Input: A finite set F of integral polynomials in $(\mathbf{x}) = (x_1, \ldots, x_j)$, $j \geq 1$, and an algebraic cell $B = (B, (\alpha_1, \ldots, \alpha_{j-1}), \delta_B, (\iota_1, \ldots, \iota_{j-1})))$ in $\mathbb{R}^{j-1}$ such that F^* is delineable on B. Here F^* denotes the polynomials in F, which do not vanish identically on B.
Output: An F-sign-invariant algebraic stack $(\mathbf{C})$ over B.

1. Substitution of $\alpha_1, \ldots, \alpha_{j-1}$ for $x_1, \ldots, x_{j-1}$ for any polynomial in F results in a set $F_{(\alpha)}$ of univariate polynomials in $\mathbb{A}[x_j]$. Removal of zero polynomials results in a set $F^*_{(\alpha)}$ of univariate polynomials over $\mathbb{A}$ in x_j.

2. Root isolation. Find the n real roots of $F^*_{(\alpha)}$ and denote them with $\alpha_j^{(2i)}$ for $1 \leq i \leq n$, such that $\alpha_j^{(2)} < \ldots < \alpha_j^{(2n)}$. There exists unique maps $\rho_1 < \ldots < \rho_m : B \rightarrow \mathbb{R}$, which delineate the real roots of $\prod F^*$.

3. Cells. Define C_{2i} to be the ρ_i-section, for $1 \leq i \leq m$. Define C_{2i-1} to be the (ρ_{i-1}, ρ_i)-sector for $1 \leq i \leq m+1$, where $\rho_0 := -\infty$ and $\rho_{2m+1} := \infty$. This gives cells $C_1, \ldots, C_{2m+1}$.

4. Sample points. Find $m+1$ rational numbers $\alpha_j^{(2i-1)}$ for $1 \leq i \leq m+1$, such that
$$\alpha_j^{(1)} < \alpha_j^{(2)} < \cdots < \alpha_j^{(2m)} < \alpha_j^{(2m+1)}.$$
Define $2m+1$ points $\alpha_{C_i} := (\alpha_1, \ldots, \alpha_{j-1}, \alpha_j^{(i)})$ for $1 \leq i \leq 2m+1$. For each $1 \leq i \leq 2m+1$ the point $\alpha^{(i)}$ lies in the cell C_i.

5. Describing formulas. For every $1 \leq i \leq n$, choose a polynomial $f_i \in F^*$ such that $f(\alpha_{C_{2i}}) = 0$. Say, $\alpha_j^{(2i)}$ is the k_i-th root of $f(\alpha_B, x_j)$. Define $\delta_{C_{2i}} := \delta_B \wedge \varphi_{\text{root of},f_i,k_i}$ for $1 \leq i \leq n$. Define $\delta_{C_{2i-1}} := \delta_B \wedge \exists q, s(\varphi_{\text{root of},f_{i-1},k_{i-1}}[q/x_j] \wedge \varphi_{\text{root of},f_i,k_i}[s/x_j] \wedge q < x_j < s)$, for $2 \leq i \leq m$. Furthermore, define $\delta_{C_1} := \delta_B \wedge \exists s(\varphi_{\text{root of},f_1,k_1}[s/x_j] \wedge x_j < s)$ and $\delta_{C_{2n+1}} := \delta_B \wedge \exists q(\varphi_{\text{root of},f_n,k_n}[q/x_j] \wedge q < x_j)$. See the remark below for a definition of $\varphi_{\text{root of},f,k}$.

6. Index. Define the index $\iota_{C_i} := (\iota_1, \ldots, \iota_{j-1}, i)$.

7. Return $(C_1, \ldots, C_{2n+1})$, where $C_i = (C_i, \alpha_{C_i}, \delta_{C_i}, \iota_{C_i})$

After we have given the algorithm, we want to justify and discuss the approach in greater detail.

2.25 Remark

1. The set F is split into two sets: F^* and $F^0 := F \setminus F^*$. Note that for any polynomial in F^0 any stack over B will be F^0-sign-invariant. Thus it suffices to consider the polynomials from F^*, and find an F^*-sign-invariant stack. The vanishing polynomials are sorted out in Step 1 of the algorithm. When the sample point of B is substituted into a polynomial f, and $f(\alpha, x_j) = 0$ then this implies by Remark 2.17,(2) that f is not delineable over S, and hence, by assumption on F, f vanishes identically on S. Thus, by removing zero polynomials after substitution, we in fact concentrate on polynomials $f(\alpha, x_j)$ with $f \in F^*$.

2. The existence of $\rho_1 < \ldots < \rho_n : B \to \mathbb{R}$ follows from the definition of delineability. For $1 \leq i \leq n$ we have that $\rho_i(\alpha_1, \ldots, \alpha_{j-1}) = \alpha_j^{(2i)}$.

3. The cylinder over B can now be decomposed into n sections, and into $n+1$ sectors, as determined by the ρ_i. This makes $2n + 1$ cells. We have that $B \times \mathbb{R}$ is the disjoint union of $C_1, \ldots, C_{2n+1}$

4. The roots we isolated give us the last component of the sample points of the sections. The last components of the sample points for the sectors are found by choosing a rational numbers in between two roots. For C_1 or, respectively, C_{2m+1}, a rational number smaller than the smallest root, or, respectively, greater than the greatest root can be chosen. The sample points are now defined by enlarging the sample point of the base cell by the last component just found. Thus all sample points differ only in the last component.

5. For $(\mathbf{x}) = (x_1, \ldots, x_{j-1})$, an integral polynomial f in $(\mathbf{x}, x_j)$, and $k \in \mathbb{N}_1$, define the formula $\varphi_{\text{root of},f,k} := \exists r_1, \ldots, r_{k-1}(r_1 < \ldots < r_{k-1} < x_j \wedge f(\mathbf{x}, r_1) = 0 \wedge \ldots \wedge f(\mathbf{x}, r_{k-1}) = 0 \wedge f(\mathbf{x}, x_j) = 0 \wedge \forall r(f(\mathbf{x}, r) = 0 \longrightarrow (r = r_1 \vee \ldots r = r_{k-1} \vee r = x \vee r > x)))$. Then $\varphi_{\text{root of},f,k}(x_1, \ldots, x_j)$ is an extended formula. For all $(\mathbf{a}) \in \mathbb{R}^{j-1}$ and $a_j \in \mathbb{R}$ we have: $\mathbb{R} \models \varphi_{\text{root of},f,k}(\mathbf{a}, a_j)$ iff a_j is the k-th root of $f(\mathbf{a}, x_j)$.

   With the definition of the algorithm we have $C_i = \mathbb{R}_{\delta_{C_i}(\mathbf{x}, x_j)}$ for $1 \leq i \leq 2n + 1$. This shows that the cells are indeed algebraic.

6. Later the index can be used to determine the position of the cell within a CAD tree. It is of further use to determine the dimension of a cell.

The following lemma summarizes the remarks made and hints that a stack plays an important role in eliminating a quantifier.

2.26 Lemma (properties of STACK)
With notions and assumptions from Algorithm 2.24

1. $(\mathbf{C})$ *is an* F*-sign-invariant non-empty algebraic stack over* B.

2. *Let* $\psi(\mathbf{x})$ *be an extended formula that has constant truth value* v_C *on each cell in* $(\mathbf{C})$. *Then for* $\mathsf{Q} \in \{\exists, \forall\}$ *the formula* $\mathsf{Q}x_j\psi$ *has constant truth value* v_B *on* B, *where* $v_B = \max\{v_C \mid C \in (\mathbf{C})\}$, *if* $\mathsf{Q} = \exists$, *and* $v_B = \min\{v_C \mid C \in (\mathbf{C})\}$, *if* $\mathsf{Q} = \forall$.

Proof. The first claim is clear from the remark above. We show the second claim by case distinction.

Case 1: For all $C \in (\mathbf{C})$: $v_C = 0$. Then $(\mathsf{Q}x_j\psi)^{\mathbb{R}}(\mathbf{a}) = 0 = \min\{v_C \mid C \in (\mathbf{C})\} = \max\{v_C \mid C \in (\mathbf{C})\}$ for all $(\mathbf{a}) \in B$ and $\mathsf{Q} \in \{\exists, \forall\}$.

Case 2: Exists $C \in (\mathbf{C})$: $v_C = 1$. Then $(\exists x_j\psi)^{\mathbb{R}}(\mathbf{a}) = 1 = \max\{v_C \mid C \in (\mathbf{C})\}$ for all $(\mathbf{a}) \in B$. For the universal quantifier we need one more distinction. Case 2.1: Exists $C' \in (\mathbf{C})$: $v_{C'} = 0$. Then $(\forall x_j\psi)^{\mathbb{R}}(\mathbf{a}) = 0 = \min\{v_C \mid C \in (\mathbf{C})\}$ for all $(\mathbf{a}) \in B$. Case 2.2: For all $C' \in (\mathbf{C})$: $v_{C'} = 1$. Then $(\forall x_j\psi)^{\mathbb{R}}(\mathbf{a}) = 1 = \min\{v_C \mid C \in (\mathbf{C})\}$ for all $(\mathbf{a}) \in B$. □

Being able to construct sign-invariant stacks is an important first milestone on our way to CAD construction, as a CAD is a union of stacks. Still two components are missing: more practical conditions to ensure delineability and recursion.

2.4 Collins' Approach to Projection

If we are given a region and a set of polynomials that is delineable on this set, we can construct a sign-invariant stack. So far, however, we lack practical means to find and check regions for delieability. Theorem 2.21 gives, for a given region, conditions on the polynomial to be delineable. We can improve on this in two ways:

1. First, we make (D2) more practical. This is achieved in Theorem 2.29.

2. Second, we want to have a similar result for sets of polynomials. This is done in Theorem 2.30.

Then we relax the assumptions. Theorem 2.31 improves on Theorem 2.29 and Theorem 2.32 improves on Theorem 2.30.

The conditions to ensure delineability will be in terms of coefficients. This is done by a projection operator, which maps a set of polynomials to a set of polynomials in one less variables.

We conclude this section by giving a general specification of a projection operator.

2.4.1 Ensuring Delineability on Regions

2.27 Lemma
The number of distinct roots of a non-constant univariate real polynomial $f(x)$ is

$$\deg_x(f) - \deg_x(\gcd(f, der_x(f))).$$

2.28 Lemma
Let $f(x)$ and $g(x)$ be non-zero polynomials over a unique factorization domain. Then

$$\deg_x(\gcd(f,g)) = \min \left\{ j \in \mathbb{N} \,\middle|\, psc_j(f,g,x) \neq 0 \right\}.$$

The proofs can be found as the proofs of Theorem 3 and Theorem 2, respectively, in [Col75].

Based on these lemmata, we can give a sufficient condition based on a polynomial's coefficients (or, based on coefficients of a set of polynomials) that we have delineability over a region. As this result is not stated explicitly in the literature we give the simple proof.

2.29 Theorem (conditions on a region for delineability of a polynomial)
Let $f \in I_j$ be a non-zero polynomial and S be a region in D such that the leading coefficient of f does not vanish and such that $res_{x_j}(f, der_{x_j}(f))$ does not vanish at some point in this region. Then f is delineable on S.

Proof. Note that $\mathrm{res}_{x_j}(f, \mathrm{der}_{x_j}(f))$ equals $\mathrm{psc}_0(f, \mathrm{der}_{x_j}(f), x)$. It follows from Lemma 2.28 that $\deg_{x_j}(\gcd(f, \mathrm{der}_{x_j}(f)))$ is invariant on S. From Lemma 2.27 it follows that the number of distinct roots of f is invariant on S. Finally our claim follows with Lemma 2.20 and with the assumption that leading coefficient of f does not vanish on S. □

2.30 Theorem (conditions on a region for delineability of a set of poly.)
Let $A \subseteq I_j$ be a set of non-zero polynomials and S be a region in $\mathbb{R}^{j-1}$ such that for all $f \in A$ we have that $lc_{x_j}(f)$ does not vanish, $res_{x_j}(f, der_{x_j}(f))$ does not vanish, and for all $f, g \in A$ we have that $res_{x_j}(f, g)$ does not vanish at some point in this region. Then A is delineable on S.

2.4.2 Collins' Projection Operator

We will need the following notation: If $F \subseteq I_j$ and a region $R \subseteq \mathbb{R}^j$ is known from the context, then F^* denotes the set of all polynomials from F that do not vanish at some point in R.

By moving to a larger set of polynomials than just leading coefficients, discriminants, and resultants, we can get a more general condition. We need some more notation. For $A \subseteq D[x]$ define

$$B(A,x) \quad := \quad \left\{ \mathrm{red}^k(a,x) \,\middle|\, a \in A, k \geq 0, \mathrm{red}^k(a,x)) \neq 0 \right\},$$

the set[1] of all non-zero redukta of elements of A wrt. x. In case A consists of a singleton element f we allow us to write $B(f, x)$ instead of $B(\{f\}, x)$.

$$L(A, x) := \{\mathrm{lc}_x(a) \mid b \in A\},$$

the set of all leading coefficients of elements of A wrt. x.

$$S_1(A, x) \quad := \quad \{\mathrm{psc}_k(a, \mathrm{der}_x(a), x) \mid a \in A \text{ and } 0 \leq k < \deg_x(der_x(a))\},$$

and

$$\textsc{psc}(f, g, x) \quad := \quad \{\mathrm{psc}_k(f, g, x) \mid 0 \leq k < \min(\deg_x(f), \deg_x(g)\}),$$

a set comprised of all possible principal subresultant coefficients.

$$\begin{aligned} S_2(A, x) \quad &:= \quad \{\mathrm{psc}_k(a_1, a_2, x) \mid a_1, a_2 \in A, 0 \leq k < \min\{\deg_x(a_1), \deg_x(a_2)\}\} \\ &= \quad \bigcup_{a_1, a_2 \in A} \textsc{psc}(a_1, a_2, x), \end{aligned}$$

a set of further subresultants.

The following conditions improve on Theorem 2.29, as cases where the leading coefficients vanish are included as well.

2.31 Theorem (sufficient condition for f to be delineable on S)
Let $f(x_1, \ldots, x_j)$ be a non-zero real polynomial, $j \geq 2$, and S be a region in $\mathbb{R}^{j-1}$ such that f does not vanish on S. Define

$$\begin{aligned} \textsc{projc}_1(A, x_j) \quad &:= \quad L(B(A, x_j), x_j) \cup S_1(B(A, x_j), x_j) \\ &= \quad \bigcup_{h \in B(A, x_j)} (\{lc_{x_j}(h)\} \cup \textsc{psc}(h, der_{x_j}(h), x_j)). \end{aligned}$$

If every element of $\textsc{projc}_1(\{f\}, x_j)$ is invariant on S then the roots of f are delineable on S.

The proof can be found as the proof of Theorem 4 in [Col75]. To lift this result to a set of polynomials, we need to add some more subresultants.

2.32 Theorem (sufficient condition for A to be delineable on S)
Let A be a non-empty set of non-zero real polynomials in $x_1, \ldots, x_j$, $j \geq 2$, and S be a connected subset of $\mathbb{R}^{j-1}$. Let $\textsc{projc}_1(A, x_j)$ be defined as before and define

$$\textsc{projc}_2(A, x_j) := S_2(B(A, x_j), x_j) = \bigcup_{f, g \in B(A, x_j)} \textsc{psc}(f, g, x_j).$$

If every element of $\textsc{projc}_1(A, x_j) \cup \textsc{projc}_2(A, x_j)$ is invariant on S then the roots of A^* are delineable on S.

[1] Note that in the original paper the condition $\deg(\mathrm{red}^k(a)) \geq 1$ was used instead of $\mathrm{red}^k(a, x) \neq 0$. This can lead to misconceptions, as it is only correct if deg is considered to be the total degree.

The proof can be found as the proof of Theorem 5 in [Col75]. We can now define *Collins' projection operator*:

$$\textsc{projc}(A, (\mathbf{x})) := \textsc{projc}_2(A, x_j) \cup \textsc{projc}_2(A, x_j)$$

A projection operator essentially maps a set of multivariate polynomials over the integers to a set of polynomials in one less variable. Input sets, and thus output sets as well, are assumed to be finite.

2.4.3 Specification of a Projection Operator

With PROJC we have defined the first instance of a projection operator. In general, a projection follows this specification:

2.33 (specification of a projection operator)

$$F \longleftarrow \mathrm{PROJ}(A, (\mathbf{x}))$$

Input: Set A of integral polynomials in $(\mathbf{x}) = (x_1, \ldots, x_j)$.
Output: Set F of integral polynomials in $(x_1, \ldots, x_{j-1})$ such that for any F-invariant region R in $\mathbb{R}^{j-1}$ the following two conditions hold:

1. Every element of A is either delineable or identically zero on R.
2. Let A^* denote the polynomials of A, which do not vanish identically on R. The sections of $\mathrm{cyl}(R)$ belonging to different $f, g \in A^*$ are either disjoint or identical.

2.34 Remark

1. Condition (2) is changed slightly to correct a glitch in [Hon90].
2. The property stated for the output F of a projection operator in the specification can be restated in the light of Remark 2.17 as follows:

   > Let R be an F-invariant region in $\mathbb{R}^{j-1}$ and let A^* denote the polynomials of A, which do not vanish identically on R. Then the roots of A^* are delineable on R.

3. PROJC adheres to the specification of a projection operator.
4. Assume PROJ is a projection operator, and PROJ$'$ is a map of same type with $\textsc{proj}'(A, (\mathbf{x})) \supseteq \textsc{proj}(A, (\mathbf{x}))$. Then PROJ$'$ is a projection operator as well. The reason is that a $\textsc{proj}'(A, (\mathbf{x}))$-invariant region is in particular a $\textsc{proj}(A, (\mathbf{x}))$-invariant region.

2.4.4 From Projection Operator to Projection Set

So far we have seen that delineability for polynomials on a region can be ensured by requiring the region to be sign-invariant wrt. a certain set of polynomials in one less variable that can be obtained via application of a projection operator.

We extend now the previous results for ensuring delineability to multiple levels. We want to find for a set A of polynomials in r variables practical conditions on a $(j-1)$-level region ($1 \leq j \leq r$), such that the j-level polynomials from A are delineable on that region.

This will amount to transform the set A into a set F, by performing projection steps repeatedly. We can transform the initial set and the sets occurring after each projection step to equivalent, but nicer ones by replacing them with the set of all irreducible factors.

2.35 Algorithm (projection set)

$$F \longleftarrow \mathrm{PROJSET}^{(\textsc{proj})}(A, (\mathbf{x}))$$

Input: Set A of integral polynomials in $(\mathbf{x}) = (x_1, \ldots, x_r)$. In addition, a projection operator PROJ is known that satisfies Specification 2.33.
Output: Set F of integral polynomials in $(\mathbf{x})$ with properties stated below.

```
1. Let $F$ denote the irreducible factors of $A$.
2. For $j$ from $r$ downto 2 do
   (a) Let $F_j$ denote the $j$-level polynomials from $F$.
   (b) Let $P$ denote the irreducible factors of $\textsc{proj}_{(x_1,\ldots,x_j)}(F_j)$.
   (c) $F := F \cup P$
3. Return $F$.
```

2.36 Remark
With the notations and assumption from Algorithm 2.35:

1. Irreducible factor computation is just an instance of the more general concept of squarefree basis computation. A set A of integral polynomials is called *squarefree basis* if all elements of A have positive degree, are primitive, are square-free and are pairwise relatively prime; alternatively, if is a set of ample, primitive irreducible polynomials of positive degree. We won't go further into this, please cf. to the original paper by Collins. It is best practice to perform squarefree basis computation in form of irreducible factors computation. This is based on observation, and no thorough published investigation of the most efficient transformation during projection is known to the author. In addition, note that this step is optional (for Collins-Hong style projection operators), but in practice it is highly desirable to perform this step.

2. Note that taking irreducible factors includes removing constant polynomials, as a polynomial is irreducible, if it is non-constant, and not the product of two non-constant polynomials.

3. Notation: For a projection set F the underlying variable order $(\mathbf{x})$ can be assumed to be known. Then F_j denotes the set of all j-level polynomials of F wrt. $(\mathbf{x})$.

2.37 Lemma (Properties of algorithm PROJSET)
With the notations and assumption from Algorithm 2.35:

1. *F is closed under* $\textsc{proj}$ *in the following sense: For $1 \leq j \leq r$ every irreducible factor of* $\textsc{proj}_{(x_1,\ldots,x_j)}(F_j)$ *is already in F.*

2. $\mathrm{PROJSET}_{\mathbf{x}}$ *is idempotent, i.e.* $\mathrm{PROJSET}_{\mathbf{x}}(\mathrm{PROJSET}_{\mathbf{x}}(A)) = \mathrm{PROJSET}_{\mathbf{x}}(A)$.

3. *For $1 \leq j \leq r$ and for any F_{j-1}-invariant region R in $\mathbb{R}^{j-1}$: F_j^* is delineable on R.*

2.38 Remark

1. We will, in general, not make a difference between projection polynomials and projection factors, because there is usually no benefit in discerning these notions.

2. Closure of a set under projection. If sets A and A^+ are given such that A is closed under projection, then $\mathrm{PROJSET(A \cup A^+)}$ is the closure of A and A^+ under projection. An algorithm could be devised to compute this set more efficiently as just defined.

2.5 Sign-Invariant Decomposition

We can now put two things together: Projection set computation and sign-invariant stack computation. This allows us to get, on input of finite set of polynomials in r variables and a variable order, a cylindrical algebraic decomposition of real r-space in form of a CAD tree. We assume a sequence of variables $(x_1, \ldots, x_r)$ to be fixed. Usually $0 \leq j \leq r$ will hold.

2.5.1 Trees for CAD Representation

As already mentioned, it is natural to use a trees to represent a CAD.

2.39 Definition (recursive labeled tree)
L a set. Define $\mathcal{R}_L$ to be the smallest set closed with the property:

(R1) $\emptyset \in \mathcal{R}_L$

(R2) If $a \in L$ and $R_1, \ldots, R_n \in \mathcal{R}_L \setminus \{\emptyset\}$ for some $n \in \mathbb{N}$, then $(a, (R_1, \ldots, R_n)) \in \mathcal{R}_L$.

2.40 Remark

1. $\emptyset$ is called the *empty tree.*
2. In particular, if $a \in L$, then $(a, \emptyset) \in \mathcal{R}_L$.
3. A more general way to define labeled trees would be this:
   (a) Let $\mathbb{N}^*$ denote the set of words over the alphabet $\mathbb{N}$. A subset T of $\mathbb{N}^*$ is called *tree*, if the following two conditions are met:
   (T1) If $n_1 \cdots n_k \in T$ then for all $h \leq k$ $n_1 \cdots n_h \in T$.
   (T2) If $t \in T$ then for all $t' \in \mathbb{N}^*$: if t' is lexicographically smaller than t, then $t' \in T$.

   The elements of T are called nodes. If $n_1 \cdots n_h$ is a node, then $(n_1, \ldots, n_h) \in \mathbb{N}^h$ is called the *index* of the node.

   With this definition, we can remark: $\emptyset$ is a tree, the empty tree. (T1) says that T is closed under prefixes, (T2) says that T is closed under lexicographic smaller elements. If $T \neq \emptyset$, then $\epsilon \in T$. This follows immediately from (T1). ϵ is called the root of T.

   (b) T a tree, L a set and $l : T \to L$ a map. Then the pair (T, l) is called a *labeled tree (with labels from L).* The elements of T are called nodes of (T, l). The label of a node t is $l(t)$.

   As a remark: The pair $(\emptyset, \emptyset : \emptyset \to L)$ is a labeled tree, the empty labeled tree.

   The benefit of this definition is that the notion of index occurs naturally. Every recursive labeled tree can be represented as a labeled tree, but not vice versa.

2.41 Definition (depth)

Define for a recursive labeled tree $R = (a, (R_1, \ldots, R_n))$:

$$\operatorname{depth}((a, (R_1, \ldots, R_n))) := \begin{cases} 0, \text{ if } n = 0, \\ 1 + \max\{\operatorname{depth}(R_i) \mid 1 \leq i \leq n\}, \text{ otherwise} \end{cases}$$

2.5.2 Full CAD

From now on a CAD tree is a recursive labeled tree, where the nodes are labeled with cells. We represent cells in the form $C = (C, \alpha_C, \delta_C, \iota_C)$, where C is a subset of some real j-space, α_C is a sample point, δ_C a describing formula, and ι_C the cell's index.

2.42 Algorithm (CAD tree)

$$D \longleftarrow \text{CADTREE}(A, (\mathbf{x}))$$

Input: A finite set A of polynomials in I_r and a list $(\mathbf{x}) = (x_1, \ldots, x_r)$ of variables.
Output: A CAD tree with properties stated below.

1. Projection. $F := \mathrm{PROJSET}(A, (\mathbf{x}))$. Denote with F_j the projection polynomials of level j.
2. Root of the tree. Let C be the only cell a decomposition of $\mathbb{R}^0$ can have, i.e. $C = (\{\emptyset\}, (), \mathit{true}, ())$.
3. Call subroutine. Return $D := \mathrm{CADTREE1}(C, F, (\mathbf{x})))$

2.43 Algorithm (full CAD tree subroutine)

$$D \longleftarrow \mathrm{CADTREE1}(B, F, (\mathbf{x}))$$

Input: A $(j-1)$-level cell B and a finite set F of integral polynomials in variables $(\mathbf{x}) = (x_1, \ldots, x_r)$ such that B is F_{j-1}-sign-invariant and such that for $j \leq i \leq r$ and for any F_{i-1}-sign-invariant region R in $\mathbb{R}^{j-1}$ F_j^* is delineable on R.
Output: A CAD subtree.

1. Definitions. j is known, as the sample point of B has length $j-1$, and r is known as the length of $(\mathbf{x})$.
2. Base Case. If $j > r$ then B is cell of level r, and hence a leave. Return a tree which only consists of a root labeled with B.
3. Stack construction. $(C_1, \ldots, C_{2n+1}) := \mathrm{STACK}(B, F_j, (x_1, \ldots, x_j))$
4. Recursive call. Define $T_i := \mathrm{CADTREE1}(C_i, F, (\mathbf{x}))$, for $1 \leq i \leq 2n+1$.
5. Return the tree $(B, (T_1, \ldots, T_{2n+1}))$.

2.44 Lemma (properties of CADTREE)
With notation and assumptions from Algorithm 2.42. Let D_j, for $0 \leq j \leq r$, denote the set of all labels of j-level nodes in D. Then D_j is a F_j-sign-invariant CAD of $\mathbb{R}^j$. In particular, D_r is an A-sign-invariant CAD of $\mathbb{R}^r$.

Proof. By induction on j. If $j = 0$ then $D_0 = \{\{\emptyset\}\}$. As $F_0 = \emptyset$ the claim clearly holds. Assume $j > 0$ and that D_{j-1} is a F_{j-1}-sign-invariant CAD of $\mathbb{R}^{j-1}$. It is not difficult to see that

$$D_j = \bigcup \{\mathrm{STACK}(B, F_j, (x_1, \ldots, x_j)) \mid B \in D_{j-1}\}.$$

So it is easy to see that D_j can be partitioned into F_j-sign-invariant algebraic stacks over cells of the cylindrical decomposition D_{j-1} of $\mathbb{R}^{j-1}$. Thus D_j is an F_j-sign-invariant CAD of $\mathbb{R}^j$.

As each irreducible factor of A is in F, D_r is in particular A-sign-invariant. □

2.6 Quantifier Elimination by Cylindrical Decomposition

At this point we move the focus from CAD to its application for quantifier elimination. The aim of the algorithm is to produce on input of a first-order formula an equivalent quantifier-free first-order formula as an output, such that all the output's free variables are free in the input formula as well.

2.6.1 Algorithm Specification

2.45 (specification of CADQE)

$$\varphi' \longleftarrow \mathrm{CADQE}(\varphi)$$

Input: Formula φ.
Output: Quantifier-free formula φ', with $\mathcal{V}(\varphi') \subseteq \mathcal{V}_f(\varphi)$ and $\mathbb{R} \models \varphi \longleftrightarrow \varphi'$.

2.46 Remark

1. Note that while for the CAD algorithm the variable order was given as an input, for the CADQE algorithm no variable order is specified. The reason for this is that for CAD the knowledge about the variable order is crucial; if it is not decided upon input, it would have to be returned on output, as the output is worthless without this knowledge.

2. One could add the variable order as a second input argument for the specification of CADQE. In this case, however, one has to require the input formula to be prenex, as otherwise, in the process of making the input formula prenex, new variables can occur.

2.6.2 Preparing a Formula for CAD

In a first *preparational* step, an order of the input formula's variables is fixed, and the input formula φ is made prenex wrt. this order. After this preparation phase, we have decided on a variable list $(x_1, \ldots, x_r)$, or, equivalently, a variable order $x_r \to \cdots \to x_1$, and have a prenex

$$\varphi(x_1, \ldots, x_k) = \mathsf{Q}_{k+1}x_{k+1} \ldots \mathsf{Q}_r x_r \psi \qquad (\mathsf{Q}_{k+1} \ldots \mathsf{Q}_r \in \{\exists, \forall\})$$

where $\psi(x_1, \ldots, x_r)$ is quantifier-free, and called *matrix* of φ. Furthermore, k is the number of free and r the total number of variables. The set of polynomials of the formula are extracted and denoted A.

We have introduced two formats for writing down variables in an ordered way: As a list $(x_1, \ldots, x_r)$ or with arrow notation $x_r \to \cdots \to x_1$. The latter notation proved to be natural in the context of projection, while the former notation is natural for the other phases of the algorithm.

Note that in general the variable order is not uniquely determined. We will exploit this degree of freedom in the subsequent Chapter 6.

2.6.3 Evaluation and Propagation of Truth Values

For QE purposes we have to tag cells with truth values. Thus we extend the representation of a cell C by a fifth component $v_C \in \{0, 1\}$. We will call a cell with truth value 1 a *true* cell, and a cell with truth value 0 a *false* cell. True and false cells are called solution and non-solution cells in the literature as well.

2.47 Algorithm (evaluation and propagation of truth values)

$$D \longleftarrow \text{TVPROP}(D, (\mathsf{Q}), (\mathbf{x}), \psi)$$

Input: A CAD tree D such that the yield D_r is a decomposition of $\mathbb{R}^r$, a formula ψ such that $\psi(\mathbf{x})$ is an extended formula, variables $(\mathbf{x}) = (x_1, \ldots, x_r)$, such that $\psi(\mathbf{x})$ is truth-invariant on the cells of D_r, and a list of quantifier symbols $(\mathsf{Q}) = (\mathsf{Q}_{k+1}, \ldots, \mathsf{Q}_r)$. Output: A CAD tree D such that the cells in level k to r contain truth values with properties stated below.

1. Base case: evaluation of truth values. If $D = (C, \emptyset)$ and $(\mathbf{s}) \in \mathbb{A}^r$ is the sample point of C, then let C' denote C with its truth value assigned to $\psi^{\mathbb{R}}(\mathbf{s})$. Return the tree $(C', \emptyset)$

2. Recursion case: propagation of truth values. If $D = (C, T_1, \ldots, T_n)$, $n \geq 1$, then compute
$$T_i' := \text{TVPROP}(T_i, (\mathsf{Q}), (\mathbf{x}), \psi).$$
Let $v_1, \ldots, v_n$ denote the truth values of the root labels of $T_1, \ldots, T_n$. Let C' denote the cell C, with its truth value assigned to $\min\{v_1, \ldots, v_n\}$, if $\mathsf{Q}_{r-\text{depth}(D)} = \forall$, $\max\{v_1, \ldots, v_n\}$, otherwise. Note that one can find $r - \text{depth}(D)$ by adding 1 to the length of the sample point of C as well.

2.48 Lemma (properties of TVPROP)
With the notation and assumptions from the algorithm: Let $k \leq j \leq r$. Then on each cell $C \in D_j$

$$\varphi_j(x_1, \ldots, x_j) := \mathsf{Q}_{j+1}x_{j+1} \ldots \mathsf{Q}_r x_r \psi$$

is truth-invariant with truth value v_C as computed by the algorithm. □

Proof. By induction on j. Base case $j = r$: As all polynomials from ψ are sign-invariant, ψ is truth-invariant on C. For the induction step Lemma 2.26,(2) can be used. □

After computing a CAD tree and after evaluation and propagation of truth values we have now a truth invariant decomposition of the free variable space of the input formula. The space described by the input formula is the union of true cells of level k:

$$\mathbb{R}_{\varphi(x_1,\ldots,x_k)} = \bigcup \{C \in D_k \mid v_C = 1\}$$

We are now very close to finding a solution formula φ'. In principle, if we knew for each cell C of D_k a describing *quantifier-free* formula δ_C we could simply use

$$\bigvee_{C \in D_k, v_C = 1} \delta_C$$

as a solution formula. Instead of finding a quantifier-free description of each cell, currently the best practice is to try to construct a signature-based formula.

2.6.4 Solution Formula Construction

So far we have seen how on input of $\varphi(x_1, \ldots, x_k)$ we can find a $\varphi(x_1, \ldots, x_k)$-truth-invariant decomposition D_k of the free variable space $\mathbb{R}^k$. This decomposition is in particular sign-invariant wrt. projection factors of level 1 through k. The idea behind signature-based solution formula construction is to discern true and false cells by the signature of these polynomials. If we can accomplish this task this is the last building block needed to give an CADQE algorithm.

We need a slightly more general version of Hong's solution formula construction. So the presentation of this algorithm belongs to Chapter 3. Here we give only the specification of SFC.

2.49 (specification of SFC)

$$\varphi' \longleftarrow \mathrm{SFC}(Y, F, (\mathbf{x}))$$

Input: A set F of polynomials from I_k with $(\mathbf{x}) = (x_1, \ldots, x_k)$ and an F-sign-invariant CAD Y of $\mathbb{R}^k$ where each cell C has a truth value v_C assigned.
Output: A quantifier-free formula φ' with properties stated below, or the symbol fail.

The property of SFC which we need in this chapter is:

> If D_k is a F-sign-invariant decomposition of $\mathbb{R}^k$, then $\varphi'(x_1, \ldots, x_k)$ as computed by $\mathrm{SFC}(D_k, F, (x_1, \ldots, x_k))$ is a describing formula for the union of true cells in D_k.

This follows from Lemma 3.31.

2.6.5 Algorithm QE by full CAD

We have now discussed all the building blocks, i.e. preparation, projection, sign-invariant CAD construction, evaluation and propagation of truth values, and solution formula construction. This can be summarized in the following algrorithm:

2.50 Algorithm (quantifier elimination by CAD)

$$\varphi' \longleftarrow \mathrm{CADQE}(\varphi)$$

Input and output: As stated in 2.45. In addition, if SFC fails, the output can be the symbol fail as well.

1. Preparation. Find a prenex equivalent formula for φ, i.e. find variables $(x_1, \ldots, x_r)$, an integer $0 \leq k \leq r$, quantifier symbols $(\mathsf{Q}_{k+1}, \ldots, \mathsf{Q}_r)$ and a quantifier-free formula ψ such that $\psi(x_1, \ldots, x_r)$ and $\varphi(x_1, \ldots, x_k)$ are extended formulas, every variable from $\{x_1, \ldots, x_k\}$ occurs freely in φ and such that
$$\mathbb{R} \models \varphi \longleftrightarrow \mathsf{Q}_{k+1}x_{k+1} \ldots \mathsf{Q}_r x_r \psi.$$
2. Projection. $F := \text{PROJSET}(A, (\mathbf{x}))$, where A are the polynomials from ψ.
3. Lifting. Compute an F-sign-invariant CAD of $\mathbb{R}^r$: $D := \text{CADTREE}(F, (\mathbf{x}))$.
4. Evaluation and propagation of truth values. $D' := \text{TVPROP}(D, (\mathsf{Q}), (\mathbf{x}), \psi)$.
5. Solution formula construction. Let D'_k denote the cells of level k in D' and let $F_{(1,\ldots,k)}$ denote the polynomials of level 1 through k from F. Then define $\varphi' := \varphi'' := SFC(D'_k, F_{(1,\ldots,k)}, (x_1, \ldots, x_k))$.
6. Simplification (optional). Find a quantifier-free formula φ' with $\mathcal{V}(\varphi') \subseteq \mathcal{V}(\varphi'')$ such that $\mathbb{R} \models \varphi'' \longleftrightarrow \varphi'$
7. Return φ'.

2.51 Remark

1. The correctness of the algorithm follows by putting the results of this section together. First, for the input formula φ a prenex equivalent formula
$$\bar{\varphi}(x_1, \ldots, x_k) = \mathsf{Q}_{k+1}x_{k+1} \ldots \mathsf{Q}_r x_r \psi$$
is computed without introducing new free variables. Notice, however, that new bounded variables are possibly introduced. Then a $\psi(x_1, \ldots, x_r)$-truth-invariant CAD D of $\mathbb{R}^r$ is computed. After evaluation and propagation of truth values, as stated by Lemma 2.48, a $\bar{\varphi}(x_1, \ldots, x_k)$-truth-invariant CAD D'_k of $\mathbb{R}^k$ is found. Assuming solution formula construction succeeds, a describing formula $\varphi'(x_1, \ldots, x_k)$ for the true-space of $\bar{\varphi}(x_1, \ldots, x_k)$ is constructed, as shown in Lemma 3.31. We have now
$$\mathbb{R} \models \varphi \longleftrightarrow \bar{\varphi} \longleftrightarrow \varphi'.$$
2. Note that after evaluation and propagation of truth values only cells of level k are needed. So it is sound if TVPROP trims the respective subtrees.

Notation	Meaning
φ	input formula, often assumed to be prenex
ψ	matrix of φ, i.e. leading quantifieres stripped off φ
k	number of free variables
r	total number of variables
$(x_1, \ldots, x_r)$	variable order, written as list
$x_r \to \cdots \to x_1$	variable order, arrow notation
$\{x_1, \ldots, x_k\}$	free variables of φ
$\{x_{k+1}, \ldots, x_r\}$	bounded variables of φ
$\{x_1, \ldots, x_r\}$	variables of φ
$\{\mathsf{Q}_{k+1}, \ldots, \mathsf{Q}_r\}$	quantifiers
A	polynomials of the input formula
F	projection factors
$F_1, \ldots, F_r$	projection factors of level $1, \ldots, r$
D	CAD tree
$D_0, \ldots, D_r$	decompositions of $\mathbb{R}^0, \ldots, \mathbb{R}^r$
φ'	quantifier-free output formula, solution formula

Figure 2.1: Ubiquitous notation in the context of QECAD.

2.7 Summary

In this chapter we have presented the CAD framework in a consistent, modern, and implementation-friendly way, reused as much notation (see Figure 2.1 for an overview) from the literature as possible, and to provided a solid basis for the following chapters, in particular for Chapter 3.

Chapter 3

Cylindrical Subdecomposition

As outlined in the introduction, our aim is to cut down on the amount of computation required for real quantifier elimination by cylindrical algebraic decomposition (CADQE), while retaining meaningful semantics. This first part of this chapter is devoted to answer the following question:

> In the setting of CADQE, what algorithmic and semantics benefits and effects are there if we decompose only a subspace?

In response to this question we put forward a generalization of the CAD framework, which we call *cylindrical algebraic subdecomposition* (SCAD), or shorter, cylindrical subdecomposition. In contrast to a CAD problem we restrict our attention to a semi-algebraic subspace, which is given by an extended formula. Corresponding subspaces in lower dimensions are induced. We define the notion of *cylindrical algebraic subdecomposition* (SCAD) and give an algorithm for computing a subdecomposition. We show how subdecompositions can be utilized for real quantifier elimination and give clean semantics for the result. The main result from this section is Algorithm SCADQE (see 3.33) and its semantics (see 3.34).

The approach is to carefully lift and adapt, after some preparations, the notions, algorithms and results of the CAD framework to a setting where one does not decompose the full space. In more detail, the plan of this chapter is as follows:

1. We first turn to semi-algebraic subspaces and their description with extended formulas.

2. We introduce the notions subcylinder, substack, subcylindrical decomposition, and subcylindrical algebraic decomposition (SCAD) and state what we call the SCAD problem.

3. We give an algorithm to construct a sign-invariant substack.

4. We can relax the delineability property of projection sets: It is only required for regions which lie within the corresponding subspace. These relaxed specifications spawn the notions of subprojection operator and subprojection set.

5. We give an algorithm to construct a cylindrical algebraic subdecomposition.

6. Based on the subdecomposition algorithm we give a quantifier elimination algorithm and its semantics.

7. We discuss how SCAD relates to full CAD, to partial CAD, an in how far assumption can be made during the algorithm.

The applications of this approach, in particular generic CAD and local CAD, follow in Chapter 4.

3.1 Subspaces

CAD finds a decomposition of full space. To specify a subspace we need a formula and an order on the free variables. For this we recall the notion of extended formula.

3.1 Definition (extended formula)
For a first-order formula φ and variables $(\mathbf{x}) = (x_1, \ldots, x_k)$ the pair $(\varphi, (\mathbf{x}))$ is called extended first-order formula, if $\mathcal{V}(\varphi) \subseteq \{\mathbf{x}\}$ and the variables from $(\mathbf{x})$ are pairwise distinct. As a convention, we write the pair $(\varphi, (\mathbf{x}))$ more suggestively as $\varphi(\mathbf{x})$.

For each extended formula $\varphi(\mathbf{x})$ one defines by recursion on the structure of φ a map $\mathbb{R}^k \to \{0, 1\}$. Here 0 stands for *true* and 1 stands for *false*. This map is denoted as $\varphi^{\mathbb{R}}$ and the extension of φ is always known from the context.

An extended formula $\varphi(\mathbf{x})$ is said to hold at a point $(\mathbf{a}) = (a_1, \ldots, a_k)$, if $\varphi^{\mathbb{R}}(\mathbf{a}) = 1$. In this situation on writes $\mathbb{R} \models \varphi(\mathbf{a})$. A formula φ is said to hold in $\mathbb{R}$, if for an extension $\varphi(\mathbf{x})$ one has $\mathbb{R} \models \varphi(\mathbf{a})$, for all $\mathbf{a}$. Then one writes $\mathbb{R} \models \varphi$. Instead of $\mathbb{R} \models \varphi \longleftrightarrow \varphi'$ we sometimes use the infix notation $\varphi \equiv_{\mathbb{R}} \varphi'$.

The subspace defined by an extended formula is its $\mathbb{R}$-realization.

3.2 Definition ($\mathbb{R}$-realization)
For an extended formula $\alpha(\mathbf{x})$, where $(\mathbf{x}) = (x_1, \ldots, x_r)$, the set

$$\mathbb{R}_{\alpha(\mathbf{x})} := \{(\mathbf{a}) \in \mathbb{R}^r \mid \mathbb{R} \models \alpha(\mathbf{a})\}$$

is called $\mathbb{R}$-realization of $\alpha(\mathbf{x})$. If the variable list $(\mathbf{x})$ is known from the context, we simply write $\mathbb{R}_\alpha$. In addition, for $0 \leq j \leq r$, define α_j to be a quantifier-free equivalent of $\exists x_{j+1} \cdots \exists x_r \alpha$ and the $\mathbb{R}^j$-realization of $\alpha(\mathbf{x})$ to be

$$\mathbb{R}^j_{\alpha(\mathbf{x})} := \left\{(\mathbf{a}) \in \mathbb{R}^j \,\middle|\, \mathbb{R} \models \alpha_j(\mathbf{a})\right\}.$$

The following lemma says that the $\mathbb{R}^j$-realization can be obtained from the $\mathbb{R}$-realization by projection on j-space.

3.3 Lemma
With the notation from Definition 3.2:

$$\mathbb{R}^j_{\alpha(\mathbf{x})} = \mathbb{R}_{\alpha_j(x_1,\dots,x_j)} = \pi_{(1,\dots,j)}[\mathbb{R}_{\alpha(\mathbf{x})}].$$

Proof. $(x_1, \dots, x_j)$ is an extension for $\exists x_{j+1} \cdots \exists x_r \alpha$. It holds for all $(\mathbf{p}) \in \mathbb{R}^j$:

$$\begin{aligned} (\mathbf{p}) \in \pi_{(1,\dots,j)}[\mathbb{R}_{\varphi(\mathbf{x})}] \quad &\text{iff} \quad \text{exists } p_{j+1}, \dots, p_r \in \mathbb{R} : (p_1, \dots, p_r) \in \mathbb{R}_{\alpha(\mathbf{x})} \\ &\text{iff} \quad \text{exists } p_{j+1}, \dots, p_r \in \mathbb{R} : \mathbb{R} \models \alpha(p_1, \dots, p_r) \\ &\text{iff} \quad \mathbb{R} \models (\exists x_{j+1} \cdots \exists x_r \alpha)(p_1, \dots, p_j) \\ &\text{iff} \quad (\mathbf{p}) \in \mathbb{R}^j_{\alpha(\mathbf{x})} \end{aligned}$$

□

The following example illustrates the effect of the choice of the extension on the described space.

3.4 Example
Consider $\alpha := (x = 0)$. Then $\alpha(x)$, $\alpha(x, y)$, and $\alpha(y, x)$ are extended formulas. The corresponding R-realizations are

$$\mathbb{R}_{\alpha(x)} = \{0\}, \quad \mathbb{R}_{\alpha(x,y)} = \{(0, b) \mid b \in \mathbb{R}\}, \quad \mathbb{R}_{\alpha(y,x)} = \{(a, 0) \mid a \in \mathbb{R}\}.$$

The corresponding $\mathbb{R}^1$-realizations are

$$\mathbb{R}^1_{\alpha(x)} = \{0\}, \quad \mathbb{R}^1_{\alpha(x,y)} = \{0\}, \quad \mathbb{R}^1_{\alpha(y,x)} = \mathbb{R}.$$

3.2 SCAD Notions and Problem Statement

As we have introduced notions for the induced subspaces, let us now see how the other notions regarding CAD have to be adjusted. Unless stated otherwise, $\alpha(x_1, \dots, x_r)$ denotes an extended formula and $0 \leq j \leq r$.

3.5 Remark (region)
The notion of *region* remains unchanged, i.e. $R \subseteq \mathbb{R}^j_{\alpha(\mathbf{x})}$ is a region, iff R is a region in $\mathbb{R}^j$. Thus for the connectedness property of a region the topology of $\mathbb{R}^j$, and not the subspace topology of $\mathbb{R}^j_{\alpha(\mathbf{x})}$ is used.

3.6 Definition (subcylinder, section, sector)
If S is a region in $\mathbb{R}^j_{\alpha(\mathbf{x})}$, then:

1. $\mathrm{subcyl}_{\alpha(\mathbf{x})}(S) := \mathrm{cyl}(S) \cap \mathbb{R}^j_{\alpha(\mathbf{x})}$ denotes the subcylinder over S wrt. $\alpha(\mathbf{x})$. If $\alpha(\mathbf{x})$ is known from the context we write simply $\mathrm{subcyl}(S)$.
2. C is called a section or a sector of $\mathrm{subcyl}(S)$ if C is a sector or a section of the cylinder over S and $C \subseteq \mathrm{subcyl}(S)$.

3.7 Remark (subdecomposition, algebraic cell, algebraic subdecomposition) Definition 2.7 is already general enough, so we need not to formally define a notion of subdecomposition. For a decomposition U of $R \subseteq \mathbb{R}^j_{\alpha(\mathbf{x})}$, however, we may use the term *subdecomposition*, to underline the fact that it is a decomposition of a subspace of $\mathbb{R}^j$. The elements of U are called cells as usual. Analogously the notions algebraic cell and algebraic subdecomposition are defined.

3.8 Definition (substack, subcylindrical decomposition, SCAD)

1. A substack over a cell C wrt. $\alpha(\mathbf{x})$ is a decomposition of $\mathrm{subcyl}(C)$ such that the projection, which drops the last component, maps each cell of this subdecomposition to C.
2. A decomposition D_j of $\mathbb{R}^j_{\alpha(\mathbf{x})}$ is called *subcylindric*, if $j = 0$ or if $j > 0$ and D_j can be partitioned into substacks over cells of a subcylindrical decomposition D_{j-1} of $\mathbb{R}^{j-1}_{\alpha(\mathbf{x})}$, the *induced* subcylindrical decompositon of D_{j-1}.
3. A decomposition that is subcylindric and algebraic is called *subcylindrical algebraic decomposition* (SCAD). In light of the remark above we allow us to say *cylindrical algebraic subdecomposition* instead of subcylindrical algebraic decomposition.

As we have now introduced the necessary notions, we can formulate the SCAD problem.

3.9 (specification of SCAD)

$$D \longleftarrow \mathrm{SCAD}(A, \alpha, (\mathbf{x}))$$

Input: A and $(\mathbf{x})$ as in Specification 2.14. In addition, a formula α, such that $\alpha(\mathbf{x})$ is an extended formula.
Output: An A-sign-invariant cylindrical algebraic subdecomposition D of $\mathbb{R}^r_{\alpha(\mathbf{x})}$, from which can be read:

1. The number and arrangement of the cells.
2. The sign of each element of A on each cell.

3.10 Definition (SCAD problem, solution for a SCAD problem) We call a triple $(A, \alpha, (\mathbf{x}))$ as specified as input in 3.9 a *SCAD problem* and an A-sign-invariant subdecomposition D as specified as output in 3.9 a *solution* of a SCAD problem.

As for a CAD, it is proximate to represent a SCAD by a tree. Here the remarks from the sections 2.2, 2.5.2, and 2.6.3 apply as well.

3.3 Sign-Invariant Substack Construction

3.11 Algorithm (substack construction)

$$(\mathbf{C}) \longleftarrow \mathrm{SUBSTACK}(B, F, \alpha, (\mathbf{x}))$$

Input: $B, F, (\mathbf{x})$ as in Algorithm 2.24. In addition there exist variables $x_{j+1}, \ldots, x_r$ such that $\alpha(\mathbf{x}, x_{j+1}, \ldots, x_r)$ is an extended formula, $B \subseteq \mathbb{R}^{j-1}_{\alpha(\mathbf{x}, x_{j+1}, \ldots, x_r)}$ is an algebraic cell and every cell of $\mathrm{STACK}(B, F, (\mathbf{x}))$ is either a subset of $\mathbb{R}^{j}_{\alpha(\mathbf{x}, x_{j+1}, \ldots, x_r)}$ or $\complement\mathbb{R}^{j}_{\alpha(\mathbf{x}, x_{j+1}, \ldots, x_r)}$.
Output: An F-sign-invariant algebraic substack $(\mathbf{C})$ over B.

1. $(\mathbf{E}) := \mathrm{STACK}(B, F, (\mathbf{x}))$
2. Remove those cells from $(\mathbf{E})$, which are not subset of $\mathbb{R}^{j}_{\alpha(\mathbf{x}, x_{j+1}, \ldots, x_r)}$. Call the result $(\mathbf{C})$.
3. Return $(\mathbf{C})$.

We need some more notation.

3.12 Definition (bounded quantifiers)
For $\mathsf{Q} \in \{\forall, \exists\}$, a variable x, and formulas γ and ξ, define

$$(\mathsf{Q}x)_\gamma \xi := \begin{cases} \exists x(\gamma \wedge \xi), & \text{if } \mathsf{Q} = \exists, \\ \forall x(\gamma \longrightarrow \xi), & \text{otherwise.} \end{cases}$$

3.13 Lemma (properties of SUBSTACK)
With notions and assumptions from Algorithm 3.11

1. *$(\mathbf{C})$ is an F-sign-invariant algebraic substack over B wrt. $\alpha(\mathbf{x})$.*
2. *Let $\psi(\mathbf{x})$ be an extended formula that has constant truth value v_C on each cell C in $(\mathbf{C})$. Then, if $(\mathbf{C}) \neq \emptyset$, for $\mathsf{Q} \in \{\exists, \forall\}$ the formula $(\mathsf{Q}x_j)_{\alpha_j}\psi$ has constant truth value v_B on B, where $v_B = \max\{v_C \mid C \in (\mathbf{C})\}$, if $\mathsf{Q} = \exists$, and $v_B = \min\{v_C \mid C \in (\mathbf{C})\}$, if $\mathsf{Q} = \forall$.*

Proof. Choose $x_{j+1}, \ldots, x_r$ such that $\alpha(\mathbf{x}, x_{j+1}, \ldots, x_r)$ is an extended formula. As a shortcut, define $\beta := \exists x_{j+1} \cdots \exists x_r \alpha$, then $\mathbb{R}_{\beta(\mathbf{x})} = \mathbb{R}^{j}_{\alpha(\mathbf{x}, x_{j+1}, \ldots, x_r)}$.

As $(\mathbf{E})$ is a finite collection of disjoint regions, so is $(\mathbf{C})$. Next we show:

$$\bigcup\{\mathbf{C}\} = \bigcup\left\{E \in (\mathbf{E}) \,\middle|\, E \subseteq \mathbb{R}^{j}_{\beta}\right\} = \mathrm{subcyl}_{\beta(\mathbf{x})}(B).$$

The first equation holds due to Step 2. To show the second equation, let $a \in \bigcup\left\{E \in (\mathbf{E}) \,\middle|\, E \subseteq \mathbb{R}^{j}_{\beta}\right\}$. Choose $E \in (\mathbf{E})$ such that $a \in E$ and $E \subseteq \mathbb{R}^{j}_{\beta}$. As $E \subseteq \mathrm{cyl}(B)$

we have $a \in \operatorname{cyl}(B)$. As in addition $a \in \mathbb{R}^j_\beta$ we have shown $a \in \operatorname{subcyl}_{\beta(\mathbf{x})}(B)$. Conversely, assume $a \in \operatorname{subcyl}_{\beta(\mathbf{x})}(B)$. As $a \in \operatorname{cyl}(B) \cap \mathbb{R}^j_\beta$, choose $E \in (\mathbf{E})$ with $a \in E$. By assumption, either $E \subseteq \mathbb{R}^j_{\beta(\mathbf{x})}$ or $E \subseteq \complement\mathbb{R}^j_{\beta(\mathbf{x})}$. As $a \in E \cap \mathbb{R}^j_\beta$ we have $E \subseteq \mathbb{R}^j_{\beta(\mathbf{x})}$. Thus $a \in \bigcup \left\{ E \in (\mathbf{E}) \,\middle|\, E \subseteq \mathbb{R}^j_\beta \right\}$. This shows that $(\mathbf{C})$ is a decomposition of $\operatorname{subcyl}_{\beta(\mathbf{x})}(B)$. In addition each $C \in (\mathbf{C})$ is algebraic and $\pi_{(1,\ldots,j-1)}[C] = B$, as each cell in $(\mathbf{E})$ has this property. This finishes the proof of the first claim.

To show the second claim we assume an extended formula $\psi(\mathbf{x})$ that has constant truth value v_C on each cell C in $(\mathbf{C})$ to be given. In addition, $(\mathbf{C}) \neq \emptyset$ is assumed. We make a case distinction.

Case 1: For all $C \in (\mathbf{C}) : v_C = 0$. Let $(\mathbf{a}) \in B$. Case 1.1: $\mathsf{Q} = \exists$. Let $a_j \in \mathbb{R}$. If $\beta^{\mathbb{R}}(\mathbf{a}, a_j) = 1$, i.e. $(\mathbf{a}, a_j) \in \mathbb{R}_{\beta(\mathbf{x})}$, then choose $C \in (\mathbf{C})$ with $(\mathbf{a}, a_j) \in C$. Then $\psi^{\mathbb{R}}(\mathbf{a}, a_j) = v_C = 0$ and $(\beta \wedge \psi)^{\mathbb{R}}(\mathbf{a}, a_j) = 0$. If $\beta^{\mathbb{R}}(\mathbf{a}, a_j) = 0$, i.e. $(\mathbf{a}, a_j) \notin \mathbb{R}_{\beta(\mathbf{x})}$, then $(\beta \wedge \psi)^{\mathbb{R}}(\mathbf{a}, a_j) = \min\left\{0, \psi^{\mathbb{R}}(\mathbf{a}, a_j)\right\} = 0$. So for all $a_j \in \mathbb{R}$: $(\beta \wedge \psi)^{\mathbb{R}}(\mathbf{a}, a_j) = 0$ and we have

$$(\exists x_j(\beta \wedge \psi))^{\mathbb{R}}(\mathbf{a}) = 0 = \max\left\{v_c \mid c \in (\mathbf{C})\right\}.$$

Case 1.2: $\mathsf{Q} = \forall$. Choose $C \in (\mathbf{C})$, as $(\mathbf{C}) \neq \emptyset$. Choose $a_j \in \mathbb{R}$ with $(\mathbf{a}, a_j) \in C$. Consequently $\beta^{\mathbb{R}}(\mathbf{a}, a_j) = 1$ and $(\beta \longrightarrow \psi)^{\mathbb{R}}(\mathbf{a}, a_j) = \max\left\{(\neg\beta)^{\mathbb{R}}(\mathbf{a}, a_j), 0\right\} = 0$. So there exists $a_j \in \mathbb{R}$ such that $(\beta \longrightarrow \psi)^{\mathbb{R}}(\mathbf{a}, a_j) = 0$ and we have

$$(\forall x_j(\beta \longrightarrow \psi))^{\mathbb{R}}(\mathbf{a}) = 0 = \min\left\{v_c \mid c \in (\mathbf{C})\right\}.$$

Case 2: There exists $C \in (\mathbf{C})$ such that $v_C = 1$. Then $(\exists x_j(\beta \wedge \psi))^{\mathbb{R}}(\mathbf{a}) = 1 = \max\left\{v_c \mid c \in (\mathbf{C})\right\}$ for all $(\mathbf{a}) \in B$. For the universal quantifier we need one more distinction. Case 2.1: There exists $C' \in (\mathbf{C})$ such that $v_{C'} = 0$. Then $(\forall x_j(\beta \longrightarrow \psi))^{\mathbb{R}}(\mathbf{a}) = 0 = \min\left\{v_c \mid c \in (\mathbf{C})\right\}$ for all $(\mathbf{a}) \in B$. Case 2.2: For all $C \in (\mathbf{C})$: $v_C = 1$. Let $(\mathbf{a}) \in B$ and $a_j \in \mathbb{R}$. If $\beta^{\mathbb{R}}(\mathbf{a}, a_j) = 1$, then choose $C' \in (\mathbf{C})$ with $(\mathbf{a}, a_j) \in C'$. Then $\psi^{\mathbb{R}}(\mathbf{a}, a_j) = v_{C'} = 1$ and $(\beta \longrightarrow \psi)^{\mathbb{R}}(\mathbf{a}, a_j) = 1$. If $\beta^{\mathbb{R}}(\mathbf{a}, a_j) = 0$, then $(\beta \longrightarrow \psi)^{\mathbb{R}}(\mathbf{a}, a_j) = 1$ as well. Thus $(\forall x_j(\beta \longrightarrow \psi))^{\mathbb{R}}(\mathbf{a}) = 1 = \min\left\{v_c \mid c \in (\mathbf{C})\right\}$. □

3.14 Remark
With notation and assumptions from Algorithm 3.11.

1. How decide whether E_i lies within $\mathbb{R}^j_{\alpha(\mathbf{x}, x_{j+1}, \ldots, x_r)}$ or its complement? Let α'_j be a quantifier-free equivalent to α_j. Note that $\alpha'_0, \ldots, \alpha'_{r-1}$ can be found by computing one CAD tree. Then if $(\mathbf{e})$ is a sample point of E_i, $(\alpha'_j)^{\mathbb{R}}(\mathbf{e}) = 1$ iff E_i lies within $\mathbb{R}^j_{\alpha(\mathbf{x}, x_{j+1}, \ldots, x_r)}$.

2. The number and position of deleted cells can only in some cases be reconstructed from the indices of remaining cells. Thus, as an alternative to removing a cell, a cell can be marked to be a *non-lifting* cell. We print such cells white.

3.4 Subprojection Operator and Subprojection Set

In the context of subdecompositions weaker conditions on projection operators and projection sets suffice. As usual, if an A-invariant region R in $\mathbb{R}^{j-1}$ is known, then A^* denotes the polynomials from A that do not vanish.

3.15 (specification of a subprojection operator)

$$F \longleftarrow \mathrm{SPROJ}(A, \alpha, (\mathbf{x}))$$

Input: A and $(\mathbf{x})$ as in Specification 2.33. In addition there exist variables $x_{j+1}, \ldots, x_r$ such that $\alpha(\mathbf{x}, x_{j+1}, \ldots, x_r)$ is an extended formula.
Output: Set F of integral polynomials in $(x_1, \ldots, x_{j-1})$ such that for any F-invariant region R in $\mathbb{R}^{j-1}_{\alpha(\mathbf{x}, x_{j+1}, \ldots, x_r)}$ the following two conditions hold:

1. Every element of A is either delineable or identically zero on R.

2. Let A^* denote the polynomials of A, which do not vanish identically on R. The sections of $\mathrm{cyl}(R)$ belonging to different $f, g \in A^*$ are either disjoint or identical.

3.16 Remark

1. Every projection operator is a subprojection operator. More precisely, if $\textsc{proj}$ satisfies Specification 2.33, then $(A, \alpha, (\mathbf{x})) \mapsto \textsc{proj}(A, (\mathbf{x}))$ satisfies Specification 3.15.

2. In the context of subdecomposition, one wants to design projection operators that exploit the knowledge about the given subspace. The generic projection operator, which is defined in an application section below, is an example for an subprojection operator.

It is straightforward to adapt the algorithm PROJSET to the corresponding algorithm that computes the subprojection set:

3.17 Algorithm (subprojection set)

$$F \longleftarrow \mathrm{SPROJSET}^{(\textsc{sproj})}(A, \alpha, (\mathbf{x}))$$

Input: A and $(\mathbf{x})$ as in Algorithm 2.35. In addition, a formula α, such that $\alpha(\mathbf{x})$ is an extended formula. Furthermore, a subprojection operator $\textsc{sproj}$ is known.
Output: Set F of integral polynomials in $(\mathbf{x})$ with properties stated below.
Proceed as in Algorithm PROJSET, but with the following modification:

1. In line 2,(b), use $\textsc{sproj}_{\alpha,(x_1,\ldots,x_j)}(F_j)$.

Accordingly, the properties of the corresponding projection set are weaker, too:

3.18 Lemma (Properties of algorithm SPROJSET)
With the notations and assumption from Algorithm 3.17:

1. *F is closed under* SPROJ *in the following sense: For $1 \leq j \leq r$ every irreducible factor of* $\text{SPROJ}_{\alpha,(x_1,\dots,x_j)}(F_j)$ *is already in F.*

2. $\text{SPROJSET}_{\alpha,(\mathbf{x})}$ *is idempotent, i.e.*

$$\text{SPROJSET}_{\alpha,(\mathbf{x})}(\text{SPROJSET}_{\alpha,(\mathbf{x})}(A)) = \text{SPROJSET}_{\alpha,(\mathbf{x})}(A).$$

3. *For $1 \leq j \leq r$ and any F_{j-1}-invariant region R in $\mathbb{R}^{j-1}_{\alpha(\mathbf{x})}$: F_j^* is delineable over R.*

3.5 Sign-Invariant Subdecomposition

By combining a subprojection operator with substack construction we can construct a SCAD. With a small modification, the CADTREE algorithm is adapted to the needs of SCAD.

3.19 Algorithm (SCAD tree)

$$D \longleftarrow \text{SCADTREE}(A, \alpha, (\mathbf{x}))$$

Input: A and $(\mathbf{x})$ as in Algorithm 2.42. In addition a formula α such that $\alpha(\mathbf{x})$ is an extended formula.
Output: A SCAD tree with properties stated below.

1. Projection. $F := \text{SPROJSET}(A, \alpha, (\mathbf{x}))$. Denote with F_j the projection polynomials of level j.

2. Empty subspace. If $\exists x_1 \cdots \exists x_r \alpha$ is equivalent to false, then Return $\emptyset$, i.e. the empty tree.

3. Root of the tree. Let C be the only cell a decomposition of $\mathbb{R}^0$ can have, i.e. $C = (\{\emptyset\}, (), \mathit{true}, ())$.

4. Call subroutine. Return $D := \text{SCADTREE1}(C, F, \alpha, (\mathbf{x})))$

3.20 Algorithm (full CAD tree subroutine)

$$D \longleftarrow \text{SCADTREE1}(C, F, \alpha, (\mathbf{x}))$$

Input: C, F, $(\mathbf{x})$ as in Algorithm 2.43. In addition a formula α such that $\alpha(\mathbf{x})$ is an extended formula.
Output: A SCAD subtree.
Proceed as in Algorithm 2.43, but with the following modifications:

- In Step 3, use SUBSTACK$(C, F_j, \alpha, (x_1, \ldots, x_j))$.
- In Step 4, use SCADTREE1$(C_i, F, \alpha, (\mathbf{x}))$.

3.21 Lemma (properties of SCADTREE)
With notation and assumptions from Algorithm 3.19. Let D_j, for $0 \leq j \leq r$, denote the set of all labels of j-level nodes in D. Then D_j is a F_j-sign-invariant SCAD of $\mathbb{R}^j_{\alpha(\mathbf{x})}$. In particular, D_r is an A-sign-invariant SCAD of $\mathbb{R}^r_{\alpha(\mathbf{x})}$.

Proof. With the convention that the j-level nodes of the empty tree are $\emptyset$ the proof can be easily lifted from the proof of Lemma 2.44. □

3.6 Quantifier Elimination by SCAD

With a SCAD algorithm at hand, we can proceed similar to the CAD setting and perform evaluation and propagation of truth values, and give as a solution formula a describing formula of the true cells in free variable space. The semantics of this result is, however, far from clear, and will be given at the end of this section. We start by giving a specification of the algorithm in question.

3.6.1 Algorithm Specification

3.22 (specification of a SCADQE algorithm)

$$\varphi' \longleftarrow \mathrm{SCADQE}(\varphi, \alpha)$$

Input: Formulas φ and α such that φ is prenex, α is quantifier-free, and every variable of α occurs in φ, possibly as a bounded variable.
Output: Quantifier-free formula φ' with properties stated below.

Let us go through the steps the final algorithm will be composed of.

3.6.2 Preparation Phase

As the input formula is already assumed to be prenex there is little work to do, except of deciding on an order $(x_1, \ldots, x_k)$ of the free variables. The order $(x_{k+1}, \ldots, x_r)$ of the bounded variables is given by the formula, but changes could be made within blocks of likely quantified variables. So we write the input formula as:

$$\varphi(x_1, \ldots, x_k) = \mathsf{Q}_{k+1} x_{k+1} \ldots \mathsf{Q}_r x_r \psi \qquad (\mathsf{Q}_{k+1} \ldots \mathsf{Q}_r \in \{\exists, \forall\}).$$

3.6.3 Projection and Subdecomposition Phase

The set of input polynomials A we consider for SCAD needs to be comprised of the polynomials of ψ and α. By including all polynomials of α we ensure that during stack construction cells of level j are α_j-truth-invariant. This is necessary for substack construction. Altogether we end up with a subdecomposition $D := \mathrm{SCAD}(A, \alpha, (\mathbf{x}))$ of $\mathbb{R}^r$.

3.6.4 Evaluation and Propagation of Truth Values

For evaluation and propagation of truth values we re-specify Algorithm 2.47 and care in addition for the case that the decomposition is empty. As the new version is more general than the original one, it shall supersede that definition.

3.23 Algorithm (evaluation and propagation of truth values for subdec.)

$$D \longleftarrow \mathrm{TVPROP}(D, (\mathsf{Q}), (\mathbf{x}), \psi)$$

Input: A SCAD tree D such that the yield D_r is a subdecomposition of $\mathbb{R}^r$, a formula ψ such that $\psi(\mathbf{x})$ is an extended formula, variables $(\mathbf{x}) = (x_1, \dots, x_r)$, such that $\psi(\mathbf{x})$ is truth-invariant on the cells of D_r, and a list of quantifier symbols $(\mathsf{Q}) = (\mathsf{Q}_{k+1}, \dots, \mathsf{Q}_r)$.
Output: A SCAD tree D such that the cells in level k to r contain truth values with properties stated below.
Proceed as in Algorithm 2.47, but add the following case to Step 1:

If $D = \emptyset$ then Return $\emptyset$.

The following lemma gives the semantics of the truth values that are computed for cells of level k through r by the algorithm.

3.24 Lemma (properties of TVPROP with SCAD tree)
With D, (Q), $(\mathbf{x})$, ψ and assumptions from Algorithm 3.23. If $\alpha(\mathbf{x})$ is an extended formula such that D_r is a decomposition of $\mathbb{R}_{\alpha(\mathbf{x})}$, then for each j with $k \leq j \leq r$ and each cell C in D_j the formula

$$\varphi^{(j)}(x_1, \dots, x_j) := (\mathsf{Q}_{j+1} x_{j+1})_{\alpha_{j+1}} \dots (\mathsf{Q}_r x_r)_{\alpha_r} \psi$$

is truth-invariant with truth values v_C as computed by the algorithm.

Proof. Induction on j. For $j = r$ let C be a cell in D_r. According to Lemma 3.21 all polynomials of ψ are invariant on C. Thus the truth value of ψ is invariant on C as well.

Now assume that $k \leq j < r$ and that the assumption holds for $j + 1$. Let C be a cell in D_j. On all cells of the stack above C $\varphi^{(j+1)}(x_1, \dots, x_{j+1})$ has constant truth value by assumption. According to Lemma 3.13,(2), v_C is the invariant truth value of $\varphi^{(j)}(x_1, \dots, x_j)$ □

As an easy corollary we know now what area the true cells and the false cells of level k describe.

3.25 Lemma (semantics of the true cells in free variable space)
With notions and assumptions from Lemma 3.24. The extended formula

$$\check{\varphi}(x_1, \ldots, x_k) := \alpha_k \wedge (\mathsf{Q}_{k+1} x_{k+1})_{\alpha_{k+1}} \ldots (\mathsf{Q}_r x_r)_{\alpha_r} \psi$$

is a describing formula of $\bigcup \{C \in D_k \mid v_C = 1\}$. *Conversely,*

$$\hat{\varphi}(x_1, \ldots, x_k) := \alpha_k \wedge \neg((\mathsf{Q}_{k+1} x_{k+1})_{\alpha_{k+1}} \ldots (\mathsf{Q}_r x_r)_{\alpha_r} \psi)$$

is a describing formula of $\bigcup \{C \in D_k \mid v_C = 0\}$. □

Let us now discuss what occurrences of bounds in $\varphi^{(k)}(x_1, \ldots, x_k)$ are indeed necessary.

3.26 Remark (simpler semantics)
Note that $\mathbb{R} \models \alpha_k \longleftarrow \ldots \longleftarrow \alpha_r$.

1. Let m' denote the level of the highest variable that occurs in α wrt. $(x_1, \ldots, x_r)$, and put $m := \max\{k, m'\}$. Then the bounds $\alpha_{m+1}, \ldots, \alpha_r$ can be dropped. Note that α_k remains. The ultimate reason for this is that in propositional calculus the formula $A \longrightarrow (A \wedge B)$ is equivalent to $A \longrightarrow B$ and that $A \wedge (A \longrightarrow B)$ is equivalent to $A \wedge B$.

2. Among those blocks of like quantifiers, where a bound remains, it suffices to keep only the bound for the highest variable.

3. We do not expect that in general more bounds can be dropped. We give the argument for two blocks of quantifiers. In such a case, say, the formula is equivalent to either

   $$\exists x_{k+1}, \ldots, x_l \forall x_{l+1}, \ldots, x_r \alpha_l \wedge (\alpha_r \longrightarrow \psi)$$

   or

   $$\forall x_{k+1}, \ldots, x_l \exists x_{l+1}, \ldots, x_r \alpha_l \longrightarrow (\alpha_r \wedge \psi)$$

   Let us abstract the matrix of each formula to propositional calculus. This amounts to

   $$A \wedge (B \longrightarrow C), \quad A \longrightarrow (B \wedge C).$$

   Looking at Table 3.1 we see that despite the assumption $A \longleftarrow B$ in $A \longrightarrow (B \wedge C)$ neither A , B, or C is redundant, due to conflicting line pairs $(1, 2)$, $(6, 8)$, and $(4, 8)$. And in $A \wedge (B \longrightarrow C)$ neither A , B, or C is redundant, due to conflicting line pairs $(1, 2)$, $(2, 4)$, and $(4, 8)$.

The following is needed later to argue that the semantics of the CADQE algorithm is independent of the variable order.

A	B	C	$A \longleftarrow B$	$A \longrightarrow (B \wedge C)$	$A \wedge (B \longrightarrow C)$
0	0	0	1	1	0
1	0	0	1	0	1
0	1	0	0	1/dc	0/dc
1	1	0	1	0	0
0	0	1	1	1	0
1	0	1	1	0	1
0	1	1	0	1/dc	0/dc
1	1	1	1	1	1

Table 3.1: No further minimization possible. *dc* stands for *don't care.*

3.27 Lemma (independence of the semantics from the variable order)
Consider the formula

$$\varphi' := (\mathsf{Q}_{k+1}x_{k+1})_{\alpha_{k+1}} \dots (\mathsf{Q}_r x_r)_{\alpha_r}\psi$$

and, for a permutation $(x'_{k+1}, \dots, x'_r)$ *of* $(x_{k+1}, \dots, x_r)$*, that respects blocks of like quantifiers, the formula*

$$\varphi'' := (\mathsf{Q}_{k+1}x'_{k+1})_{\alpha'_{k+1}} \dots (\mathsf{Q}_r x'_r)_{\alpha'_r}\psi,$$

where α'_j *is a quantifier-free equivalent to* $\exists x'_{j+1} \dots \exists x'_r \alpha$*. Then* $\mathbb{R} \models \varphi' \longleftrightarrow \varphi''$*.*

Proof. We show the claim for the case that there is one block of like quantifiers. Let $\bowtie$ denote $\wedge$, if $\mathsf{Q} = \exists$, and $\longrightarrow$, if $\mathsf{Q} = \forall$.

$$\begin{aligned}
(\mathsf{Q}x_{k+1})_{\alpha_{k+1}} \dots (\mathsf{Q}x_r)_{\alpha_r}\psi &\equiv_{\mathbb{R}} \mathsf{Q}x_{k+1} \dots \mathsf{Q}x_r(\alpha_{k+1} \bowtie (\dots(\alpha_r \bowtie \psi)\dots)) \\
&\equiv_{\mathbb{R}} \mathsf{Q}x_{k+1} \dots \mathsf{Q}x_r(\alpha_r \bowtie \psi) \\
&\equiv_{\mathbb{R}} \mathsf{Q}x_{k+1} \dots \mathsf{Q}x_r(\alpha'_r \bowtie \psi) \\
&\equiv_{\mathbb{R}} \mathsf{Q}x'_{k+1} \dots \mathsf{Q}x'_r(\alpha'_r \bowtie \psi) \\
&\equiv_{\mathbb{R}} \mathsf{Q}x'_{k+1} \dots \mathsf{Q}x'_r(\alpha'_{k+1} \bowtie (\dots(\alpha'_r \bowtie \psi)\dots)) \\
&\equiv_{\mathbb{R}} (\mathsf{Q}x'_{k+1})_{\alpha'_{k+1}} \dots (\mathsf{Q}x'_r)_{\alpha'_r}\psi
\end{aligned}$$

The argument is now easily generalized by induction on the number of blocks of like quantifiers and by using the fact that always $\alpha_j \equiv_{\mathbb{R}} \alpha'_j$, if j marks the end of a block of like quantifiers. □

3.6.5 Solution Formula Construction for Subdecomposition

With some small modifications we can reuse the algorithm for signature-based solution formula construction from [Hon90].

3.28 Definition (signature, signature-based formula)
Let $(\mathbf{f}) = (f_1, \dots, f_m)$ be a list of integer polynomials.

1. A signature for $(\mathbf{f})$ is a sign vector $\sigma = (\sigma_1, \dots, \sigma_m) \in \{-1, 0, 1\}^m$.

2. If W is a finite set of signatures of $(\mathbf{f})$, then

$$\varphi_{\text{signbased},W,(\mathbf{f})}(x_1, \dots, x_k) := \bigvee_{\sigma \in W} \bigwedge_{i \in \{1,\dots,m\}} f_i \rho_i 0,$$

   where $\rho_i \in \{<, =, >\}$ is defined to be $<$, if $\sigma_i = -1$, $=$, if $\sigma_i = 0$, and $>$, if $\sigma_i = 1$, is called the W-based formula for $(\mathbf{f})$.

Disjoint sets of signatures result in signature-based formulas, which describe disjoint spaces.

3.29 Lemma
Let W, W' with $W \cap W' = \emptyset$ be two sets of signatures for $(\mathbf{f})$. Let $(\mathbf{x})$ be an extension to $\varphi_{\text{signbased,W},(\mathbf{f})}$ and $\varphi_{\text{signbased,W}',(\mathbf{f})}$. Then

$$\mathbb{R} \not\models \exists \mathbf{x}(\varphi_{\text{signbased,W},(\mathbf{f})} \wedge \varphi_{\text{signbased,W}',(\mathbf{f})})$$

Proof. If $W = \emptyset$ or $W' = \emptyset$ then one of the signature-based formulas is false, and the claim holds. So we assume W and W' to be non-empty.

Suppose $\mathbb{R} \models \exists \mathbf{x}(\varphi_{\text{signbased,W},(\mathbf{f})} \wedge \varphi_{\text{signbased,W}',(\mathbf{f})})$. Choose reals $(\mathbf{a})$ such that $\mathbb{R} \models (\varphi_{\text{signbased,W},(\mathbf{f})} \wedge \varphi_{\text{signbased,W}',(\mathbf{f})})(\mathbf{a})$. Choose $\sigma \in W$, $\sigma' \in W'$ such that

$$\mathbb{R} \models (\varphi_{\text{signbased},\{\sigma\},(\mathbf{f})} \wedge \varphi_{\text{signbased},\{\sigma'\},(\mathbf{f})})(\mathbf{a}).$$

Choose i such that $\sigma_i \neq \sigma'_i$. Now $f_i(\mathbf{a})$ evaluates to a number and has two different signs, a contradiction. □

The following algorithm is used for the special case $Y = D_k$ in Chapter 2. Here, instead of D_k, we more generally assume an $\varphi(x_1, \dots, x_k)$-invariant decomposition Y of a subspace of $\mathbb{R}^k$.

3.30 Algorithm (signature-based solution formula construction)

$$\varphi' \longleftarrow \text{SFC}(Y, F, (\mathbf{x}))$$

Input: A set F of polynomials from I_k with $(\mathbf{x}) = (x_1, \dots, x_k)$ and an F-sign-invariant CAD Y of a subspace of $\mathbb{R}^k$ where each cell C has a truth value v_C assigned.
Output: A quantifier-free formula φ' with properties stated below, or the symbol fail.

1. Decide on an ordering of the polynomials. Choose $(\mathbf{f}) = (f_1, \dots, f_m)$ such that $F = \{\mathbf{f}\}$.

2. For each cell C in Y compute a signature $\sigma^{(C)} := \operatorname{sign}(f(\alpha_C))$.
3. True, false, and conflicting signatures. $W_t := \left\{\sigma^{(C)} \mid C \in D, v_C = 1\right\}$, $W_f := \left\{\sigma^{(C)} \mid C \in D, v_C = 0\right\}$, $W_c := W_t \cap W_f$.
4. Quick win. If $W_t = \emptyset$ then Return false. If $W_f := \emptyset$ then Return true.
5. If $W_c = \emptyset$ then Return $\varphi_{\text{signbased},W_t,(\mathbf{f})}$, else Return fail.

3.31 Lemma (properties of SFC)
With notions and assumptions from Algorithm 3.30. Let $Y^{(t)} := \{C \in Y \mid v_C = 1\}$ and $Y^{(f)} := \{C \in Y \mid v_C = 0\}$. If SFC succeeds, then

$$\bigcup Y^{(t)} \subseteq \mathbb{R}_{\varphi'(x_1,\dots,x_k)} \subseteq \complement \bigcup Y^{(f)}.$$

In particular, if Y is a decomposition of $\mathbb{R}^k$, then $\varphi'(\mathbf{x})$ is a describing formula for $Y^{(t)}$.

Proof. As the algorithm succeeds we have $W_c = \emptyset$. If $W_t = \emptyset$ then

$$\bigcup Y^{(t)} = \emptyset = \mathbb{R}_{\mathsf{false}(x_1,\dots,x_k)} \subseteq \bigcup Y^{(f)}.$$

Conversely, if $W_f = \emptyset$, then

$$\bigcup Y^{(t)} \subseteq \mathbb{R}^k = \mathbb{R}_{\mathsf{true}(x_1,\dots,x_k)} = \bigcup Y^{(f)}.$$

In the following, Lemma 3.29 is used implicitly in some arguments. For $C \in Y$ we have that $C \subseteq \mathbb{R}_{\varphi_{\text{signbased},\{\sigma_C\},(\mathbf{f})}}$ due to the assumption that $(\mathbf{f})$ are C-sign-invariant. Thus

$$\bigcup Y^{(t)} \subseteq \bigcup_{C \in Y^{(t)}} \mathbb{R}_{\varphi_{\text{signbased},\{\sigma_C\},(\mathbf{f})}} = \mathbb{R}_{\bigvee_{C \in Y^{(t)}} \varphi_{\text{signbased},\{\sigma_C\},(\mathbf{f})}} = \mathbb{R}_{\varphi_{\text{signbased},W_t,(\mathbf{f})}}$$

As just seen, and as $W_c = \emptyset$, we have that

$$\mathbb{R}_{\varphi_{\text{signbased},W_t,(\mathbf{f})}} = \bigcup_{C' \in Y^{(t)}} \mathbb{R}_{\varphi_{\text{signbased},\left\{\sigma_{C'}\right\},(\mathbf{f})}} \subseteq \complement C$$

for all $C \in Y^{(f)}$. Thus

$$\begin{aligned} \complement \bigcup Y^{(f)} &= \bigcap \{\complement C \mid C \in Y, v_C = 0\} \\ &\supseteq \bigcap \left\{\mathbb{R}_{\varphi_{\text{signbased},W_t,(\mathbf{f})}} \,\middle|\, C \in Y, v_C = 0\right\} \\ &= \mathbb{R}_{\varphi_{\text{signbased},W_t,(\mathbf{f})}} \end{aligned}$$

If Y is a decomposition of $\mathbb{R}^k$, then $\bigcup Y^{(t)} = \complement \bigcup Y^{(f)}$, thus $Y^{(t)} = \mathbb{R}_{\varphi'(x_1,\dots,x_k)}$. □

3.32 Remark

1. The algorithm SFC can fail in rare cases. Then the true-space is not *projection-definable.* The version given in [Hon90] is less general in that it assumes Y to be a decomposition of $\mathbb{R}^k$, but more sophisticated in that it is shown there how to use multiple-valued logic minimization to further simplify the result.

2. The original paper [Col75] shows how to find a quantifier free describing formula for each cell in free variable space by means of an *augmented projection operator.* This approach turned out to be too costly and is replaced now by signature-based solution formula construction as best practice.

3. Brown [Bro99] shows how a situation with conflicting signature can be remedied by subsequently adding additional input polynomials. Adding polynomials and recomputing the CAD is less costly than using augmented projection beforehand. Due to this result a CAD algorithm with signature-based solution formula construction can be considered complete.

3.6.6 Quantifier Elimination by SCAD

Finally we can give the proposed SCADQE algorithm and clarify, by putting the various results from this section together, its semantics.

3.33 Algorithm (SCADQE)

$$\varphi' \longleftarrow \mathrm{SCADQE}(\varphi, \alpha)$$

Input: Formulas φ and α such that φ is prenex and every free variable of α occurs in φ, possibly as a bounded variable.
Output: Quantifier-free formula φ' with properties stated below.
Proceed as in CADQE (cf. Algorithm 2.50), with the following modifications:

- In Step 2, let A denote the polynomials of ψ and α.
- In Step 3, replace CADTREE$(F, (\mathbf{x}))$ by SCADTREE$(F, \alpha, (\mathbf{x}))$.
- Replace Step 6 by:

  Simplification under theory (optional). Find a φ' such that $\mathbb{R} \models \alpha \longrightarrow (\varphi'' \longleftrightarrow \varphi')$

3.34 Theorem (properties of SCADQE)
With notions φ, α, and φ' and assumptions from Algorithm 3.33 and for any valid choice of variable order $(x_1, \dots, x_r)$ as discussed on page 53 it holds: $\mathcal{V}(\varphi') \subseteq \mathcal{V}_f(\varphi)$ and with formulas α_j as introduced in Definition 3.2:

$$\mathbb{R} \models \alpha_k \longrightarrow (\varphi' \longleftrightarrow (\mathsf{Q}_{k+1} x_{k+1})_{\alpha_{k+1}} \dots (\mathsf{Q}_r x_r)_{\alpha_r} \psi).$$

In particular, if α does not contain any bounded variable from φ, then

$$\mathbb{R} \models \alpha \longrightarrow (\varphi' \longleftrightarrow \varphi).$$

Proof. Let $(x_1, \dots, x_r)$ denote the variable order fixed in Step 1, D denote the SCAD tree computed in Step 3, and D' the SCAD tree with truth values from Step 4 of the algorithm. $(x_1, \dots, x_k)$ is an extension for α_k, φ', and $(\mathsf{Q}_{k+1}x_{k+1})_{\alpha_{k+1}} \dots (\mathsf{Q}_r x_r)_{\alpha_r}\psi$. Let $(\mathbf{a}) \in \mathbb{R}^k$ with $\mathbb{R} \models a_k(\mathbf{a})$. So $(\mathbf{a})$ lies within $\bigcup D_k$. Choose cell $C \in D_k$ with $(\mathbf{a}) \in C$. We have to show

$$(\varphi')^{\mathbb{R}}(\mathbf{a}) = ((\mathsf{Q}_{k+1}x_{k+1})_{\alpha_{k+1}} \dots (\mathsf{Q}_r x_r)_{\alpha_r}\psi)^{\mathbb{R}}(\mathbf{a}).$$

Due to Lemma 3.24, $(\mathsf{Q}_{k+1}x_{k+1})_{\alpha_{k+1}} \dots (\mathsf{Q}_r x_r)_{\alpha_r}\psi)^{\mathbb{R}}(\mathbf{a}) = v_C$. So it suffices to show $(\varphi')^{\mathbb{R}}(\mathbf{a}) = v_C$. If $v_C = 1$, then due to Lemma 3.31 (first inclusion) $(\varphi')^{\mathbb{R}}(\mathbf{a}) = 1$. If $v_C = 0$, then with Lemma 3.31 (second inclusion)

$$(\mathbf{a}) \in C \subseteq \complement\mathbb{R}_{\varphi'(x_1,\dots,x_k)} = \mathbb{R}_{\neg\varphi'(x_1,\dots,x_k)}.$$

So $(\neg\varphi')^{\mathbb{R}}(\mathbf{a}) = 1$ and thus $(\varphi')^{\mathbb{R}}(\mathbf{a}) = 0$.

Altogether we have shown the claim for one particular variable order. Due to Lemma 3.27 the claim holds for any valid variable order $(x_1, \dots, x_r)$. □

3.35 Example
Consider the input formula $\varphi(x, y) := \exists y(x = y^2)$ and the assumption $\alpha(x, y) := (x^2 + y^2 - 1 < 0)$. SCADQE returns $\varphi' = (x^2 + x - 1 < 0 \wedge x \geq 0)$. The $\mathbb{R}$-realization of $\varphi'(x)$ is the half-open interval $[0, -\frac{1}{2} + \frac{\sqrt{5}}{2}[$. As a comparison, the $\mathbb{R}$-realization of the result of CADQE would be $[0, \infty[$.

3.7 Discussion

After the introduction of the subdecomposition framework we want to discuss how it relates to full decomposition, partial CAD, and how it can be modified such that assumptions can be made on the fly as well.

3.7.1 Subdecomposition Versus Decomposition

For a SCAD problem $(A, (\mathbf{x}), \alpha)$ we call the CAD problem $(A, (\mathbf{x}))$ the corresponding CAD problem. Conversely, if α is known from the context, for a CAD problem $(A, (\mathbf{x}))$ the problem $(A, (\mathbf{x}), \alpha)$ is called the corresponding SCAD problem. We note some observations on how a subdecomposition relates to the corresponding decomposition. Let us first consider the situation where by making assumptions no new projection factors are introduced. Then:

1. If the set of projection factors is the same as that of regular CAD, then the subdecomposition will be a subset of the corresponding decomposition. In particular, the nubmer of cells of the subdecomposition will be less than or equal to the number of cells of the corresponding decomposition.

2. If the set of projection factors is a subset of that of regular CAD, then the subdecomposition needs no longer be a subset of the decomposition. What happens is that cells melt together, yielding a coarser decomposition.

3. On the one hand it is additional work to find non-lifting cells. On the other hand one level later there is less work, as there are less stacks to be constructed. We have two expectations: (1) If a decomposition succeeds then a subdecomposition will succeed as well, maybe needing more time in rare cases. (2) In general we expect a subdecomposition to be a time saver.

Let us now consider the situation where the assumptions introduce additional projection factors. Then:

1. A subdecomposition can have more cells than the corresponding decomposition. As an example, consider $A = \{x^2 + y^2 - 1\}$, $\alpha := (x^2 + y^2 \leq 2)$, and the variable order (x, y). The regular CAD algorithm computes a CAD of 13 cells. The corresponding SCAD for this example, however, results in 25 cells.

2. A subdecomposition can have less cells than the corresponding decomposition, but counting white cells there were more. As an example, consider $A := \{4y^2 + 4x^2 - 12x + 5, 4y^2 - 12y + 5 + 4x^2\}$, $\alpha := (x > y)$, and the variable order (x, y). There are two circles, one at $(1.5, 0)$ and the other at $(0, 1.5)$, both with radius 1. These circles do not overlap. A CAD results in $1+3+5+7+9+7+5+3+1 = 41$ cells. A corresponding SCAD consists of $3+5+7+9+11+9+7+5+3 = 59$ cells in total, but 34 white cells have to be removed. So there remain $1+1+1+3+5+5+5+3+1 = 25$ cells.

3. In principle it is possible that it takes much more effort to construct the SCAD than the corresponding CAD. In practice, however, the assumptions made are not arbitrary. They are either made manually by the user who has a certain understanding of the problem, or they are made automatically by an algorithm, as will be seen in later applications. In both cases we expect that in general the SCAD approach will lead to improved results.

3.7.2 Partial CAD Versus SCAD

Partial Cylindrical Algebraic Decomposition (PCAD) was introduced by Collins and Hong [CH91] and consists of three ideas to speed up the construction of a CAD tree or to simplify a CAD tree in the context of quantifier elimination. These are: Partial CAD itself, the first improvement *trial evaluation* (TE), and the second improvement *propagation below free variable space* (PBFVS). Note that PCAD can have two different meanings: It can be used both as a collective term for these three ideas and as a designation for the first idea. We shall discuss the relationship between PCAD and its improvements and SCAD.

The idea of PCAD itself is already sketched in Collins' original paper [Col75]. For ease of presentation we considered two different phases: the construction of a CAD and then, afterwards, evaluation and propagation of truth values. These two phases can of course be intermingled, in that the tree is constructed in a depth-first manner, evaluation of truth values is performed as soon as a leaf is reached, and propagation of truth values takes place as soon as a stack is returned. When doing this, it is easy to see that in certain situations it is not necessary to compute the subtree over every cell in a stack. If the truth values of a stack will be propagated to the base cell by means of an existential quantifier, then as soon as one finds a true cell one can cease to compute subtrees. The dual situation holds for universal quantifiers. As a result, not a full CAD tree is computed, but only a partial one. This can lead to enormous speed-ups.

Let us look at trial evaluation. The idea behind TE is that in some cases the truth value of a cell might be determined by plucking in the sample point into the matrix ψ. If ψ contains atomic formulas for which the partial sample point suffices to find a truth value, then chances are that the overall truth value of ψ can be determined and assigned to the cell. As a result, the subtree over the cell need not be constructed. This integrates nicely into the CADQE framework if done at level k and above. If done within free variable space, then a cell C of level $j < k$, for which a truth value by TE is found, represents the cell $C \times \mathbb{R}^{k-j}$. As a result, a truth-invariant full decomposition of free variable space for the input formula is constructed. This decomposition might no longer be sign-invariant wrt. projection factors of level k, and there are some implications on solution formula construction, which we need not discuss here. Depending on the problem TE can lead to large gains, or to an overhead.

Regarding PBFVS, the idea is that when all cell in a stack on or below free variable space have the same truth value, than this value can be propagated to the base cell, and the children of this base call can be removed from the tree. Similarly as with TE, a leave C of level $j < k$ represents the cell $C \times \mathbb{R}^{k-j}$ of level k, and the decomposition might no longer be sign-invariant, but is still truth-invariant. The advantage of PBFVS is faster solution formula construction and simplification.

Looking at PCAD, TE, and PBFVS, these improvements have the following common characteristics:

1. A full decomposition of free variable space is computed.
2. This decomposition is not necessarily sign-invariant (wrt. projection factors of level 1 through k).
3. Classical semantics: Output is equivalent to input.

In contrast to this, SCAD has the the following properties:

1. A subdecomposition of free variable space is computed.
2. This decomposition is sign-invariant (wrt. projection factors of level 1 through k).
3. Relaxed semantics: Under some assumptions the output is equivalent to the input formula, where some bounds have to be added.

We can conclude that SCAD differs significantly from PCAD and its improvements.

Even better, SCAD can be combined with PCAD. When constructing a substack in quantified variable space we can apply the PCAD idea and cease to compute subtrees in appropriate cases. As a result we end up with a decomposition of free variable space, which is exactly the same as without PCAD applied. If we utilize TE for SCAD, then the following can happen: A cell below free variable space is assigned a truth value, but if we would continue the decomposition, then a white cell would occur in a stack above. As a result, not a decomposition of the appropriate subspace of free variable space, but a decomposition of some superset of it would be computed. This does not affect the semantics of the solution, the only disadvantage might be that solution formula construction unnecessarily fails due to compatible signatures. As for PBFVS, similarly to TE a superset is decomposed. Note that in contrast to TE the extraneous space may bear wrong truth values, but again this does not affect the semantics.

To summarize, the use of PCAD and TE on and above free variable space is save and has no visible effects on the subdecomposition of free variable space that is computed; we can proceed with solution formula construction as described above. TE below free variable space and PBFVS has effects on the decomposition, but the semantics remains intact. Solution formula construction is affected and has to be modified accordingly.

3.7.3 A Priori Assumptions Versus Making Assumptions on the Fly

So far we have considered the situation that assumptions are given beforehand to the algorithm Note that the overall algorithm is good-natured in that in most steps the use the assumptions is optional, and not mandatory. In the following, if we speak about making assumptions on the fly, i.e. during the execution of the algorithm, then we mean assumptions that will fully exclude or include future cells to be generated, and that will fully exclude or include cells that were already generated. E.g. making assumptions in terms of projection factors satisfies this condition.

1. During projection, assumptions may or may not be used to cut down on the projection set. If assumptions are not, or not always used, then redundant projection factors may persist. Such a projection set, however, is in particular a valid subprojection set. Making not full use of the assumptions does not affect the semantics, but “only” leads to superfluous computation, as possibly a more fine-grained decomposition has to be constructed.
2. During extension, assumptions may or may not be used to remove white cells below and on free variable space. Here superfluous cells might result in unnecessary

computation, a cluttered up solution formula, or a failed signature-based solution formula construction, but the semantics for QE is not affected (we can argue similarly as above, where we argued that SCAD is compatible with TE and PBFVS). Note, however, that above free variable space this freedom does not exist. Here a precise substack construction is needed, so evaluation and propagation of truth values leads to correct results.

As a consequence, instead of having assumptions fixed beforehand, we consider the following more flexible approach:

1. During projection we can make assumptions on all variables.

2. After projection we can make assumptions on all variables.

3. During extension we can make assumptions on free variables. If assumptions made include bounded variables, then it has to made clear that none of the cells constructed so far gets turned into a white cell by making these assumptions.

4. After extension, and during solution formula construction, we can make assumptions on free variables.

In any case, we can put the semantics and correctness of such CADQE variant down to the semantics and correctness of the original CADQE algorithm:

1. Make (additional) assumptions α' during the algorithm as allowed and desired. This results in a variant CADQE$'$ of CADQE of the following format:

$$(\alpha', \varphi') \longleftarrow \mathrm{CADQE}'(\varphi, \alpha)$$

2. Argue that φ' can be obtained by $\mathrm{SCAD}(\varphi, \alpha \wedge \alpha')$ by making use of the assumptions in a selective way. This clarifies the semantics of φ'.

Altogether, by relaxing the utilization of assumptions in CADQE within the aforementioned limits we get a flexible framework to build interesting application-oriented variants upon. When doing this, the key questions to be asked are:

1. In which steps should assumptions be allowed to be made?

2. What kind of assumption should be made?

3. Should assumptions be utilized during projection and how can this be done?

4. Is there an efficient way to sort out white cells?

5. Is it ensured that the assumptions are not inconsistent, i.e. equivalent to false?

3.8 Conclusions

We have put forward the framework of cylindrical subdecomposition (SCAD) in order to clarify the effects and benefits of decomposing only a subspace, as opposed to finding a full decomposition. In particular, this framework gives an algorithm and semantics of subdecomposition-based real quantifier elimination (SCADQE), where external assumptions on bounded variables are allowed. The approach is compatible with partial CAD. It provides an abstraction layer on which further application can be realized. By using the SCAD framework one can focus on application ideas, while semantics and correctness is provided by the framework.

Chapter 4

SCAD Applications

This chapter is devoted to applications of the SCAD framework, which was presented in Chapter 3.

1. As a first application for SCAD we give a *generic projection operator*, which exploits the relaxed conditions on projection sets in a subdecomposition setting. By computing non-trivial examples we illustrate the capabilities of this approach.
2. As a second application we show how the paradigm of *local quantifier elimination* can be applied to the CAD setting by means of performing a subdecomposition.
3. We show that generic projection operator and local elimination can be combined for added advantages.
4. We discuss more applications and further work.

4.1 Generic Elimination

This section applies the paradigm of generic quantifier elimination to partial cylindrical algebraic decomposition (PCAD). On input of a first-order formula over the reals generic PCAD outputs a theory and a quantifier-free formula. The theory is a set of negated equations in the parameters of the input formula. The quantifier-free formula is equivalent to the input for all parameter values satisfying the theory. For obtaining this generic elimination procedure, we derive a generic projection operator from the standard Collins–Hong projection operator. Our operator particularly addresses formulas with many parameters. It restricts decomposition to a reasonable subset of the entire space. There is a theory in the form of negated equations generated that describe this subset. The approach is compatible with other improvements in the framework of PCAD. It turns out that the theory contains assumptions that are easily interpretable and that are most often non-degeneracy conditions. The applicability of our generic elimination procedure significantly extends that of the corresponding regular procedure. Our procedure is implemented in the computer logic system REDLOG.

The content of this section is published in [SS03].

4.1.1 Introduction

In [DSW98] *generic quantifier* elimination has been introduced on the basis of real quantifier elimination by virtual substitution. This works as follows: On input of a first-order formula φ over the reals, the generic quantifier elimination procedure has two return values:

1. A *theory* Θ, i.e., a list of negated equations, also called *assumptions*, in the parameters of φ.
2. A quantifier-free formula φ' in the parameters of φ.

The specification of the algorithm is that φ' is a quantifier-free equivalent to φ for all choices of parameters satisfying Θ; formally

$$\mathbb{R} \models \bigwedge \Theta \longrightarrow (\varphi' \longleftrightarrow \varphi).$$

Note that there are never equalities or ordering inequalities assumed, and that there are no Boolean connections other than conjunction possible between assumptions. As a consequence, the exception set, for which φ' is *not* correct, has measure zero within the parameter space. The idea behind generic quantifier elimination is that the assumption of Θ supports the construction of φ' to such an extent that the range of practically feasible problems is significantly extended.

With virtual substitution methods [Wei88] this has without doubt been the case. This has been demonstrated in particular in the area of automated geometry proving [DSW98, Stu99a] and physical network analysis [Stu99b]. In all applications examined so far there have been two more most interesting observations made:

- The assumptions Θ use to have a straightforward interpretation within the real system modeled by the input. For instance, conditions on an electric circuit would not compare voltages with resistances.
- Even more strikingly, the assumptions often provide additional *non-degeneracy* assumptions that are actually necessary to make the input a sufficiently precise model of the real world.

We shall see concrete examples for both these points in Section 4.1.4.

Regarding the second point, in his famous monograph on geometry proving [Cho88], Chou has convincingly demonstrated that for geometric configurations it is not practicable to determine all necessary non-degeneracy conditions in advance.

Consequently, straightforward algebraic formulations of geometric theorems are in most cases “false,” and this would in fact be the result of any regular quantifier elimination procedure. Generic quantifier elimination, in contrast, adds in almost all cases considered so far the missing input specifications to Θ and obtains "true" as φ'. From this point of view, generic quantifier elimination is much more than a weaker form of regular quantifier elimination. It is noteworthy that the Wu–Ritt reduction techniques used by Chou yield conditions in the parameters very similar to our assumptions. In this section, we gain the following results pertaining to generic quantifier elimination by PCAD:

1. We define a *generic projection operator* GPROJ, which is derived from the standard Collins–Hong projection operator PROJH [Col75, Hon90]. This operator is compatible with the common variants of cell decomposition and solution formula construction. On the other hand, it even allows for a specifically optimized decomposition.

2. Our operator GPROJ particularly addresses formulas with many parameters: It will turn out that projection within parameter space systematically allows for assumptions. In bound variable space, in contrast, this requires certain configurations that occur in practice with significant frequency but not systematically.

3. Unlike all other improvements of projection operators discussed in the literature so far, we do not only aim at a coarser decomposition of the entire space but at decomposing only a reasonable subset of this space. Our approach is compatible with other improvements in the framework of PCAD.

4. Our projection operator GPROJ allows to restrict the possible form of valid assumptions: One can, e.g., restrict to monomial assumptions.

5. We have implemented quantifier elimination by PCAD using our generic projection operator GPROJ. This allows us to judge the empirical performance of our approach on practical examples.

6. Furthermore, it is possible with our implementation to choose between generic quantifier elimination using our GPROJ and regular quantifier elimination using PROJH. This ensures perfect comparability of computation times as well as of qualities of results between our approach and the classical one. It turns out that generic PCAD dramatically exceeds the capabilities of regular PCAD.

7. As discussed with other methods above, we obtain assumptions Θ that are easily interpretable and that are most often non-degeneracy conditions.

It is not hard to see that the introduction of our generic projection operator does not lead to any better upper worst-case complexity bounds compared to regular PCAD. We thus focus on demonstrating its applicability by solving with our implementation of generic PCAD examples that are not solvable with its non-generic counterpart. Since real quantifier elimination has during the past years been on the edge between academic examples and real world problems, any step further is most promising.

The plan of this section is a follows: Section 4.1.2 is the technical core. Here we define our generic projection operator, which is derived from Hong's modification in [Hon90] of Collins' original projection operator in [Col75], and prove that it yields a subprojection operator. In Section 4.1.3 we give, based on the SCAD framework, our GCADQE algorithm by using the generic projection operator to obtain assumptions and a subprojection set. In Section 4.1.4 we give computation examples. In Section 4.1.5 we finally summarize and evaluate our work.

4.1.2 Generic Projection Operator

Throughout this section, we consider our prenex input formula to contain variables $x_1, \ldots, x_r$, where $x_1, \ldots, x_k$ are free variables, which we also refer to as *parameters*, and $x_{k+1}, \ldots, x_r$ are variables bound by quantifiers. By I_j we denote $\mathbb{Z}[x_1, \ldots, x_j]$. Consequently, I_r is the set of all polynomials possibly occurring in our formula, and I_k is the set of all polynomials containing only parameters. Let A generally denote a finite subset of I_r.

We recall the definition of Hong's projection operator from [Hon90]:

$$\begin{aligned}
\textsc{projh}(A) &= \textsc{proj}_1(A) \cup \textsc{proj}_2^*(A),\\
\textsc{proj}_1(A) &= \bigcup_{\substack{f\in A\\ f^*\in\mathrm{red}(f)}} \bigl(\{\mathrm{ldcf}(f^*)\} \cup \textsc{psc}(f^*, f^{*\prime})\bigr)\\
\textsc{proj}_2^*(A) &= \bigcup_{\substack{f,g\in A\\ f<g}} \bigcup_{f^*\in\mathrm{red}(f)} \textsc{psc}(f^*, g).
\end{aligned}$$

For the motivation of our approach, consider the leading coefficients added in $\textsc{proj}_1(A)$. In contrast to only adding the leading coefficient of each polynomial in A, there are in addition the leading coefficients of all reducta added. The reason for this is that there will in general be choices for variables such that these leading coefficients vanish. For the relevant properties of the projection sets it is, however, crucial to include leading coefficients for all possible choices of variables including degenerate situations.

From this point of view, the construction of the chain of reducta can be stopped as soon as the first constant leading coefficient appears; a fact, which is well-known in the community. Our idea is now to go one step further: We are going to stop this process as soon as a leading coefficient appears, which contains only parameters. We simply assume this parametric leading coefficient to be nonzero. This assumption, which is formally a negated equation, is added to a *theory* Θ.

Similar observations hold for the chains of principal subresultant coefficients computed in both $\textsc{proj}_1$ and $\textsc{proj}_2^*$.

After our generic projection we continue PCAD. At the end we have obtained on input of a first order formula φ both a quantifier-free formula φ' and the theory Θ mentioned above. The result φ' is correct for all choices of parameters satisfying Θ.

Formally, we have

$$\mathbb{R} \models \bigwedge \Theta \longrightarrow (\varphi' \longleftrightarrow \varphi).$$

From this semantical description it is straightforward that we do not want to admit any assumptions on bound variables. Recall that projection proceeds from bound variable space to parameter space. Within the former it is good luck to find parametric leading coefficients or principal subresultant coefficients; within the latter they occur systematically.

We are now going to formalize our idea and prove its correctness.

For the definition of our generic projection operator, we define a generic set of reducta as follows: If there is some j such that $\mathrm{red}^j(f) \neq 0$ and $\mathrm{ldcf}(\mathrm{red}^j(f)) \in I_k$ and there is some $j < j'$ such that $\mathrm{red}^{j'}(f) \neq 0$, then

$$\mu = \min\bigl\{\, j \bigm| \mathrm{red}^j(f) \neq 0 \text{ and } \mathrm{ldcf}(\mathrm{red}^j(f)) \in I_k \,\bigr\}$$

and

$$\textsc{gred}(f) = \bigl(\{\mathrm{ldcf}(\mathrm{red}^{\mu}(f)) \neq 0\}, \{\, \mathrm{red}^i(f) \mid 0 \leq i \leq \mu \,\}\bigr).$$

Else $\textsc{gred}(f) = \bigl(\emptyset, \mathrm{red}(f)\bigr)$.

Similarly, we have a generic set of principal subresultant coefficients: If there is $j < \min\{\deg(f), \deg(g)\}$ such that $\mathrm{psc}_j(f,g) \in I_k$ and there is $j < j' < \min\{\deg(f), \deg(g)\}$ such that $\textsc{psc}_{j'}(f,g) \neq 0$, then

$$\mu = \min\bigl\{\, j < \min\{\deg(f), \deg(g)\} \bigm| \mathrm{psc}_j(f,g) \in I_k \,\bigr\}$$

and

$$\textsc{gpsc}(f,g) = \bigl(\{\mathrm{psc}_{\mu}(f,g) \neq 0\}, \{\, \mathrm{psc}_i(f,g) \mid 0 \leq i \leq \mu \,\}\bigr).$$

Else $\textsc{gpsc}(f,g) = \bigl(\emptyset, \textsc{psc}(f,g)\bigr)$.

As a final preparational step, we make the conventions that

$$\mathrm{gldcf}(f) = \bigl(\emptyset, \{\mathrm{ldcf}(f)\}\bigr)$$

and that

$$(\Theta, S) \sqcup (\Theta', S') = (\Theta \cup \Theta', S \cup S').$$

This allows us to define our generic projection operator as follows:

$$\begin{aligned}
\textsc{gproj}(A) &= \textsc{gproj}_1(A) \sqcup \textsc{gproj}_2^*(A),\\
\textsc{gproj}_1(A) &= \bigsqcup_{\substack{f \in A\\ f^* \in \textsc{gred}(f)}} \bigl(\{\mathrm{gldcf}(f^*)\} \sqcup \textsc{gpsc}(f^*, {f^*}')\bigr)\\
\textsc{gproj}_2^*(A) &= \bigsqcup_{\substack{f,g \in A\\ f < g}} \ \bigsqcup_{f^* \in \textsc{gred}(f)} \textsc{gpsc}(f^*, g)
\end{aligned}$$

Note that all polynomials occurring as the left hand sides of assumptions are as well part of the projection set. This way, after computation of a sign-invariant decomposition, every assumption will be either constantly valid or constantly invalid over each cell.

We recall further definitions. We denote by $V(A)$ the real variety of A. For $0 \leq j < r$ let S be a connected subset of $\mathbb{R}^j$. Then $Z(S) := S \times \mathbb{R}$ is the *cylinder* over S. We adopt the definition of a *section* of a cylinder from [ACM84]. Then A is *delineable* on S if the portion of $V(A)$ lying in $Z(S)$ consists of n disjoint sections of $Z(S)$ for some $n \geq 0$. This notion of delineability, which is a bit weaker than the original one by Collins, has been introduced in [ACM84]. We shall allow ourselves in the sequel to briefly say *invariant* instead of sign-invariant.

For a set of conditions Θ in variables $x_1, \ldots, x_r$ and $0 \leq j \leq r$ let

$$\mathbb{R}^j_\Theta = \left\{ \underline{x} \in \mathbb{R}^j \;\middle|\; \mathbb{R} \models \bigwedge \Theta(\underline{x}, \underline{y}) \text{ for some } \underline{y} \in \mathbb{R}^{r-j} \right\}.$$

4.1 Lemma
Let A be a finite subset of I_j for $j \geq 2$, and say $\textsc{gproj}_1(A) = (\Theta, P)$. *Let S be a connected subset of $\mathbb{R}^{j-1}_\Theta$ such that every element of P is invariant on S. Then every element of A is either delineable or identically zero on S.*

Proof. This assertion is a modification of Lemma 2 in [Hon90], for which the proof is given in Theorem 4 of [Col75]. We modify this proof to derive a proof of our claim as follows: Fix an element $f(x_1, \ldots, x_j) = \sum_{i=0}^m f_i x_j^i$ of A and a connected subset S of $\mathbb{R}^{j-1}_\Theta$ such that f is not identically zero on S.

Choose $m \geq 1$ maximal, such that $f_m \neq 0$ on S and choose k such that

$$g := \operatorname{red}^k(f) = \sum_{i=0}^m f_i x_j^i.$$

Denote $\textsc{gred}(f) = (\Theta_1, B)$, and assume $\Theta_1 \neq \emptyset$, say

$$\Theta_1 = \bigl\{\operatorname{ldcf}(\operatorname{red}^\mu(f)) \neq 0\bigr\}, \quad B = \bigl\{f, \operatorname{red}(f), \ldots, \operatorname{red}^\mu(f)\bigr\}.$$

Because of $\Theta_1 \subseteq \Theta$ and our assumption on S we have $k \leq \mu$. Hence certainly $g \in B$ and $\operatorname{ldcf}(g) \in P$.

Choose $l \geq 1$ minimal such that $\operatorname{psc}_l(g, g') \neq 0$ on S. Denote $\textsc{gpsc}(g, g') = (\Theta_2, S_1)$. Assume $\Theta_2 \neq \emptyset$, say

$$\Theta_2 = \bigl\{\operatorname{psc}_\nu(g, g') \neq 0\bigr\}, \quad S_1 = \bigl\{\operatorname{psc}_0(g, g'), \ldots, \operatorname{psc}_\nu(g, g')\bigr\}.$$

In the same way as above we conclude that $l \leq \nu$, hence $psc_l(g, g') \in S_1$.

We have shown that whenever we make assumptions and cease to include further polynomials, then all polynomials needed for the original proof that we are modifying are contained in our generic projection set. □

4.2 Lemma
Let A be a finite subset of I_j for $j \geq 2$, and say GPROJ$_1(A) = (\Theta, P)$. *Let S be a connected subset of $\mathbb{R}^{j-1}_{\Theta}$ such that every element of P is invariant on S. Let f and g be any two different polynomials in A. If the least integer k such that*

$$\mathrm{psc}_k\bigl(f(\underline{\alpha}, x_j), g(\underline{\alpha}, x_j)\bigr) \neq 0$$

does not vary for $\underline{\alpha} \in S$, then the sections of $Z(S)$ belonging to f and g are either disjoint or identical. □

This assertion is a modification of Lemma 3 in [Hon90]. The basic proof ideas are given in Theorem 5 of [Col75]. Using similar modifications as in the proof of Lemma 4.1 one obtains a proof for our assertion.

4.3 Theorem (gproj is a subprojection operator)
Let A be a finite subset of I_j for $j \geq 2$, and say GPROJ$_1(A) = (\Theta, P)$. *Let S be a connected subset of $\mathbb{R}^{j-1}_{\Theta}$ such that every element of P is invariant on S. Then the following two conditions hold:*

1. *Every element of A is either delineable or identically zero on S.*
2. *The sections of $Z(S)$ belonging to different f, $g \in A$ are either disjoint or identical.* □

Again, a proof can be obtained by adaption of the proof of Theorem 1 in [Hon90].

It is not hard to see that our generic projection remains correct if we impose restrictions on the form of possible assumptions. From an application point of view this is a most interesting feature. The user might, e.g., wish to obtain only monomial assumptions, require at least one variable to occur only linearly, impose other degree restrictions, or wish to prohibit assumptions on certain parameters.

By successively liberating restrictions on possible assumptions, our generic projection set scales from the Collins–Hong projection set towards Brown's projection set [Bro01]. In fact, Brown implicitly makes assumptions even on bound variables but does not create any theory. The price is that his decomposition, in contrast to ours, can fail.

4.1.3 Decomposition and Quantifier Elimination

The situation for generic CAD differs from SCAD only in so far, as the subspace is not known from the beginning, but found during projection. The implications of making assumptions on the fly were discussed in Section 3.7.3. By writing a small wrapper around SCADQE we obtain the GCADQE algorithm.

4.4 Algorithm (GCADQE)

$$(\Theta, \varphi') \longleftrightarrow \mathrm{GCADQE}(\varphi)$$

Input: Formula φ.
Output: List of negated equations Θ and a quantifier-free formula φ' with properties stated below.

1. Let (Θ, F) denote the result of generic projection of the polynomials of φ. Set $\alpha := \bigwedge \Theta$.

2. Let φ' denote the result of SCADQE(φ, α). When calling SCADQE, use the generic projection operator, such that SCADQE will indeed use the subprojection set F as already computed in Step 1.

3. Return (Θ, φ')

The correctness of GCADQE is now easily verified, as it follows from the results from Section 4.1.2 and the properties of SCADQE.

4.5 Theorem
With notations and assumptions from Algorithm 4.4. Then

$$\mathbb{R} \models \bigwedge \Theta \longrightarrow (\varphi' \longleftrightarrow \varphi).$$

□

4.6 Remark

1. To fit squarely into the SCAD framework, the generic projection set is computed twice in this formulation of GCADQE. Of course, in practice one would compute the result of generic projection only once, and save it for further use.

2. REDLOG contains powerful simplifiers [DS97c]. We use these simplifiers in two different ways: First, instead of blindly making assumptions, we conservatively check whether a desired non-zeroness follows from assumptions already made before. Second, we obtain by simplification a most concise theory for output at the end.

4.1.4 Computation Examples

The computations for the following example have been performed on a 1.5 GHz Intel Pentium M under Windows XP using 128 MB RAM.

Rhomboid

Consider the following geometrical theorem: In a parallelogram, the diagonals intersect each other in the middle. We can attempt to prove this with real QE.

level	CADQE	GCADQE	GCADQE+PCAD
4	693	154	12
3	127	30	30
2	19	10	10
1	5	3	3
Time:	130ms	30ms	20ms

Table 4.1: Number of cells in a full, generic, and generic-partial CAD tree for φ_{pg}

Without loss of generality, we can assume one corner of the parallelogram at $(0,0)$, a second corner at $(1,0)$. The remaining corners are assumed at (u,v), and at $(1+u,v)$. Then a condition for a point (x,y) to lie on both diagonals is

$$\frac{y}{x} = \frac{v}{1+u} \quad \text{and} \quad \frac{-v}{1-u} = \frac{-y}{1-x}.$$

A condition that (x,y) lies on the middle of one diagonal is that the distance from $(0,0)$ to (x,y) equals the distance from (x,y) to $(1+u,v)$, i.e.

$$x^2 + y^2 = (1+u-y)^2 + (v-y)^2.$$

Together this gives the following formulation:

$$\begin{aligned} \varphi_{\mathrm{pg}} \quad := \quad & \forall x \forall y (uy - vx + y = 0 \wedge -uy + vx - v + y = 0 \\ & \qquad \longrightarrow -u^2 + 2ux - 2u - v^2 + 2vy + 2x - 1 = 0) \end{aligned}$$

Applying QE to $\forall u \forall v \varphi_{\mathrm{pg}}$ results in false. This hints a flaw in the formulation, as we know the theorem holds and thus expect true. Applying GCADQE to φ_{pg} results in

$$((u+1 \neq 0, u-1 \neq 0, v \neq 0), \mathsf{true}).$$

In case $v = 0$ the parallelogram degenerates to a line, so $v \neq 0$ is a non-degeneracy condition. The cases $u = 1$ and $u = -1$ are special cases and can be considered separately.

The comparison in Table 4.1 shows that there are considerable savings when using GCADQE. If partial CAD is performed in addition, then there are savings by trial evaluation of level-3 cells.

The following computations have been performed on a 933 MHz Intel Pentium III under Linux using 128 MB RAM. All timings are CPU times including garbage collection times.

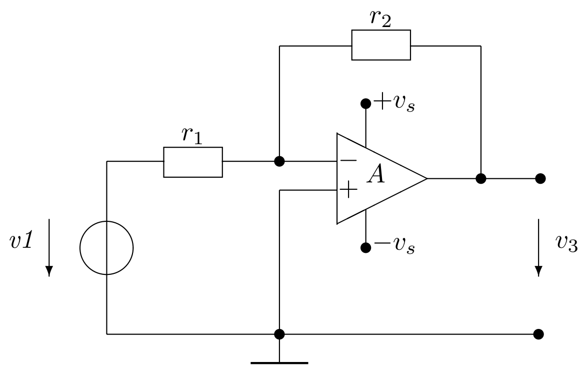


Figure 4.1: An inverting operation amplifier circuit

Inverting Operation Amplifier

We revisit an example from [Hen95, Stu99b]. The inverting operation amplifier in Figure 4.1 is described by the following set of equations. This description has been automatically generated from a description of the circuit using the MATHEMATICA-based system Analog Insydes [HH98]:

$$\begin{aligned}
v_1 &= \mathit{v1}\\
v_2 &= -v_{\mathrm{pm\_op1}}\\
v_3 &= v_{\mathrm{og\_op1}}\\
v_1 + i_{\mathrm{v0}} r_1 &= v_2\\
v_2 r_1 + v_2 r_2 &= v_3 r_1 + v_1 r_2 + i_{\mathrm{pm\_op1}} r_1 r_2\\
v_3 + i_{\mathrm{og\_op1}} r_2 &= v_2\\
v_{\mathrm{og\_op1}} &= v_{\mathrm{pm\_op1}} x_{\mathrm{op1}}^2\\
v_s^2 x_{\mathrm{op1}}^2 + A v_{\mathrm{og\_op1}}^2 &= A v_s^2\\
i_{\mathrm{pm\_op1}} &= 0.
\end{aligned}$$

The aim is to determine the output voltage v_3 as a function of the input voltage *v1*. The amplification factor A, the supply voltage v_s, and the resistances r_1 and r_2 are parameters. All other variables

$$v_{\mathrm{pm\_op1}},\ i_{\mathrm{og\_op1}},\ i_{\mathrm{pm\_op1}},\ i_{\mathrm{v0}},\ v_1,\ v_2,\ v_{\mathrm{og\_op1}},\ x_{\mathrm{op1}}$$

have to be existentially quantified.

In [Hen95] it had been tried to determine a solution via computation of an elimination ideal basis, where it was for principal reasons not possible to exclude certain so-called parasitic solutions. The problem has then been satisfactorily solved in [Stu99b] using generic quantifier elimination by virtual substitution. For our purposes here, we consider it an excellent benchmark example for the decomposition of a 14-dimensional space. The 14 variables partition into 6 existentially quantified variables and 8 parameters; it should be mentioned that the elimination of one of the quantified variables, viz. v_1, is in fact trivial.

The projection order for all computation variants on this example discussed in the sequel is as follows:

$$\begin{gathered}\xrightarrow{v_{\mathrm{pm\_op1}}}\ \xrightarrow{i_{\mathrm{og\_op1}}}\ \xrightarrow{i_{\mathrm{pm\_op1}}}\ \xrightarrow{i_{\mathrm{v0}}}\ \xrightarrow{v_1}\ \xrightarrow{v_2}\\ \xrightarrow{v_{\mathrm{og\_op1}}}\ \xrightarrow{x_{\mathrm{op1}}}\ \mid\ \xrightarrow{A}\ \xrightarrow{v_s}\ \xrightarrow{\mathit{v1}}\ \xrightarrow{r_2}\ \xrightarrow{v_3}\ \xrightarrow{r_1}.\end{gathered}$$

By placing "|" we indicate the beginning of the parameter space.

Using PROJH, the size of the projection set develops as follows:

$$\begin{gathered}9 \to 12 \to 14 \to 16 \to 17 \to 20 \to\\ 30 \to 42 \to\ \mid 78 \to 375 \to \bot \qquad\qquad (> 104\text{ min}).\end{gathered}$$

That is, there are 9 input factors before the projection of $v_{\mathrm{pm\_op1}}$. This first projection step results in 12 factors altogether before the projection of $i_{\mathrm{og\_op1}}$, etc. After 104 min the computation aborts during the projection of v_s due to exceeding the chosen memory size of 128 MB.

Using GPROJ, we successively obtain projection set cardinalities as follows:

$$\begin{gathered}9 \to 12 \to 13 \to 15 \to 15 \to 16 \to\\ 18 \to 26 \to\ \mid 32 \to 35 \to 35 \to 35 \to 35 \qquad (870\text{ ms}).\end{gathered}$$

During this projection, there is in addition to the projection factors the following theory of 13 negated equations generated:

$$\begin{aligned}\{\ 4A^2r_1^2v_3^2 + 8A^2r_1r_2\mathit{v1}\,v_3 + 4A^2r_2^2\mathit{v1}^2 + r_1^2v_s^2 + 2r_1r_2v_s^2 + r_2^2v_s^2 &\neq 0,\\ Ar_1v_3^3 - Ar_1v_3v_s^2 + Ar_2\mathit{v1}\,v_3^2 - Ar_2\mathit{v1}\,v_s^2 + r_1v_3v_s^2 + r_2v_3v_s^2 &\neq 0,\\ Ar_1v_3^3 - Ar_1v_3v_s^2 + Ar_2\mathit{v1}\,v_3^2 - Ar_2\mathit{v1}\,v_s^2 - r_1v_3v_s^2 - r_2v_3v_s^2 &\neq 0,\\ Ar_1v_3 + Ar_2\mathit{v1} + r_1v_3 + r_2v_3 &\neq 0,\\ A &\neq 0,\\ r_1v_3 + r_2\mathit{v1} &\neq 0,\\ r_1 + r_2 &\neq 0,\\ r_1 &\neq 0,\\ r_2 &\neq 0,\\ v_3 + v_s &\neq 0,\\ v_3 - v_s &\neq 0,\\ v_3 &\neq 0,\\ v_s &\neq 0\ \}.\end{aligned}$$

To be more precise, the theories which we give here have undergone some simplification techniques [DS97c]. There have been possibly more than 13 assumptions made during projection.

Among the assumptions in the theory, we find typical non-degeneracy conditions: resistances like r_1, r_2 can, of course, never be zero in reality; the output voltage v_3 can naturally never reach the supply voltage v_s.

The entire generic PCAD finishes after 12 min yielding the quantifier-free result "false." This is a bit surprising. The reason is that the algebraic equation describing the behavior of the circuit is actually the negation of the third assumption in the theory.

Instead of going into details about the physical facts that could be derived in this situation, we continue seeking for a straightforward result. For this we once more use GPROJ but now prohibit non-monomial assumptions in the theory. We successively obtain the following cardinalities of projection sets:

$$\begin{aligned}&9 \to 12 \to 13 \to 15 \to 15 \to 16 \to\\ &\quad 19 \to 26 \to \ | \ 32 \to 35 \to 35 \to 35 \to 35 \qquad (230\text{ ms}).\end{aligned}$$

In addition, we obtain the following theory with 5 negated equations:

$$\{A \neq 0, r_1 \neq 0, r_2 \neq 0, v_3 \neq 0, v_s \neq 0\}.$$

On the basis of this projection, generic PCAD finishes after 12 min. This time we obtain a quantifier-free result containing 408 atomic formulas. Here we obviously suffer from our still preliminary solution formula construction code. Substitution of reasonable values for the parameters A, r_1, r_2 and successive simplifications indicate, however, that this formula actually describes the desired solution derived in [Stu99b].

X-Axis Ellipse Problem

The problem, which has been suggested by Lazard in [Laz88], is to write down conditions such that the ellipse

$$(x-c)^2/a^2 + (y-d)^2/b^2 - 1 = 0$$

is inside the circle $x^2 + y^2 - 1 = 0$. We treat the special case $d = 0$; compare Figure 4.2. The input formula reads as follows:

$$\forall x \forall y \big(b^2(x-c)^2 + a^2y^2 - a^2b^2 = 0 \longrightarrow x^2 + y^2 - 1 \leq 0\big).$$

The projection order is $\xrightarrow{y} \xrightarrow{x} \ | \ \xrightarrow{c} \xrightarrow{b} \xrightarrow{a}$. With the regular projection operator PROJH, we successively obtain intermediate projection sets of the following sizes:

$$2 \to 9 \to \ | \ 18 \to 28 \to 32 \qquad\qquad (20\text{ ms}).$$

On the basis of this projection set, regular PCAD succeeds after 1 min with a quantifier-free equivalent containing 4234 atomic formulas.

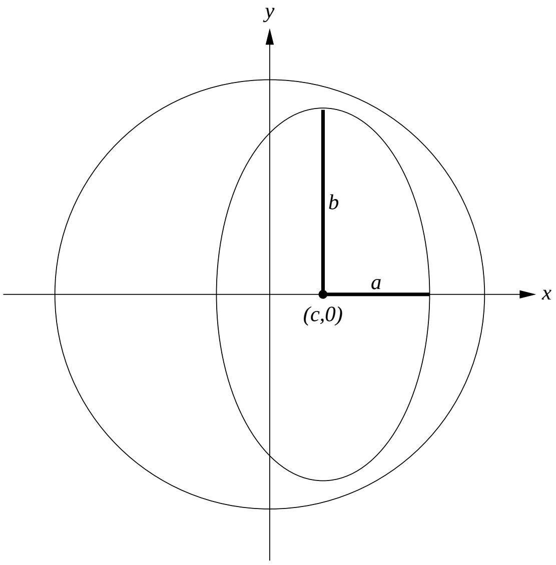


Figure 4.2: The x-axis ellipse problem

With our generic projection operator GPROJ, we obtain, in contrast, the following intermediate projection set sizes:

$$2 \to 9 \to \ | \, 16 \to 24 \to 24 \qquad\qquad (10 \text{ ms}).$$

In addition, we obtain the following theory:

$$\{a + b \neq 0, a - b \neq 0, a \neq 0, b \neq 0\}.$$

On this basis, generic PCAD returns after 7 s a quantifier-free formula containing 448 atomic formulas.

We finally analyze the situation when using GPROJ but admitting only monomial assumptions. Here we obtain the projection set sizes

$$2 \to 9 \to \ | \, 17 \to 25 \to 25 \qquad\qquad (10 \text{ ms})$$

together with the theory

$$\Theta = \{a \neq 0, b \neq 0, c \neq 0\}.$$

On this basis, generic PCAD yields after 8 s a quantifier-free formula φ' with 578 atomic formulas. Note that $a \neq 0$ and $b \neq 0$ are obviously non-degeneracy conditions.

We now use Θ and φ' obtained by this last generic PCAD, and complete it to a quantifier-free equivalent of the input formula. For this we additionly treat the three cases excluded by Θ. Applying regular PCAD to the original problem with 0 substituted for a, we obtain "false" in less than 10 ms. The same holds for the case $b = 0$. For the case $c = 0$ we obtain in 240 ms a quantifier-free equivalent φ'_c with 76 atomic formulas. Combining these results, we have

$$\mathbb{R} \models \Big(\Big(\bigwedge \Theta \wedge \varphi'\Big) \vee (c = 0 \wedge \varphi'_c)\Big) \longleftrightarrow \varphi.$$

Our quantifier-free formula obtained by one generic PCAD plus three regular PCAD's on special cases contains $3 + 578 + 1 + 76 = 658$ atomic formulas. It requires an overall computation time of less than 9 s. This is a considerable improvement compared to the straightforward PCAD with 4234 atomic formulas in 1 min.

4.1.5 Conclusions

We have defined a generic projection operator for PCAD and described how to perform generic quantifier elimination on the basis of this operator. Our techniques are implemented within the computer logic system REDLOG. By means of two highly non-trivial quantifier elimination examples, we have demonstrated that our generic approach has a significantly extended application range compared to regular PCAD. This particularly affects input formulas with polynomials of high degree and many parameters thus filling a present gap in the applicability of real quantifier elimination techniques.

4.2 Local Elimination

Local quantifier elimination was introduced as a variant of real quantifier elimination by virtual substitution (VSQE). For a first-order formula and a real point a quantifier-free formula is computed, which is not only at the given point equivalent to the input, but on a semi-algebraic set that contains the given point. Such a semi-algebraic set is computed by the algorithm as well, and returned represented by a quantifier-free formula. In this section:

1. We introduce the concept of local quantifier elimination in greater detail and motivate its usefulness.
2. We discuss how local quantifier elimination can be realized within the CAD framework and give the Algorithm LCADQE.
3. By examining examples we demonstrate the algorithm and compare the result with regular CAD.

4.2.1 Introduction

Local quantifier elimination was introduced by Dolzmann and Weispfenning in [DW00a]. It caters for situations, where example values of interest for some or all parameters are known. Such values can be values for which empirically a good behavior of an underlying system is known. It is, however, desired to find an environment of one point (or of some points) such that all values from these region have the same good behavior as the given point.

Such problems often occur in engineering. It is the idea behind this method to use the additional information about a sample point to simplify the algorithm such that the computation time is decreased and the output size of the solution is reduced.

4.2.2 Algorithm

As for the algorithm, we get a formula, a variable list and a reference point as input. As a result we derive a quantifier-free equivalent for the input problem with parameters fixed as specified, and, in addition, a description of a subspace wherein these parameters can be varied such that the equivalence still holds. More formally, we have the following specification for the local elimination:

4.7 (specification of LCADQE)

$$(\alpha, \varphi') \longleftarrow \mathrm{LCADQE}(\varphi, (\mathbf{x}), (\mathbf{p}))$$

Input: A prenex formula φ, a variable list $(\mathbf{x}) = (x_1, \ldots, x_k)$ such that $\varphi(\mathbf{x})$ is an extended formula, and a reference point $(\mathbf{p}) = (p_1, \ldots, p_h) \in \mathbb{A}^h$ with $0 \leq h \leq k$.
Output: A pair (α, φ') of quantifier-free formulas, such that α and φ' are formulas with extension $(x_1, \ldots x_k)$,

1. $(\mathbf{p}) \in \mathbb{R}^h_{\alpha(\mathbf{x})}$, and
2. $\mathbb{R} \models \alpha \longrightarrow (\varphi \longleftrightarrow \varphi')$.

Here φ' is called *solution*, and α is called *area* or *range description*. $\mathbb{R}^j_{\alpha(\mathbf{x})}$ is called range of the reference point.

Our plan is to realize this application by means of a subdecomposition. The central question is how the area description comes into being. There is a trade-off: On the one hand one wants to have a small area, as this reduces computational effort. On the other hand, a larger area yields a stronger result.

Consider this approach: Compute the projection set with a standard projection operator. Let C denote the cell of level h of the induced decomposition that contains the reference point. Find a describing formula δ_C for this cell, and define $\alpha := \delta_C$. Let $\varphi' := \mathrm{SCADQE}(\varphi, \alpha, (\mathbf{x}))$ and return (α, φ'). This would be a proximate course of action and limit the area to a small one. The problem with this approach is, however, that finding δ_C is difficult and orthogonal to the usual proceeding of signature-based solution formula construction.

After these considerations the following solution is more suitable: Instead of C, use the smallest superset of C that can be described by a signature-based formula.

Naturally, the formula doing this can be found by using the signs of the projection factors at the sample point.

4.8 Algorithm (LCADQE)

$$(\alpha, \varphi') \longleftarrow \text{LCADQE}(\varphi, (\mathbf{x}), (\mathbf{p}))$$

Input: As specified in 4.7.
Output: As specified in 4.7, or, in rare cases, fail.

1. Projection. Perform projection as given by the variable list $(\mathbf{x}, x_{k+1}, \ldots, x_k)$. This results in the set F.

2. Area definition. Let $f_1, \ldots, f_m$ denote the projection factors from $F_1, \ldots, F_h$. Define the signature

   $$\sigma := (\text{sign}(f_1(\mathbf{p})), \ldots, \text{sign}(f_m(\mathbf{p})))$$

   Define $\alpha := \varphi_{\text{signbased},\{\sigma\},(\mathbf{f})}$ (cf. Definition 3.28).

3. Subdecomposition. $\varphi' := \text{SCADQE}(\varphi, \alpha)$. Make sure that SCADQE uses the projection order $(\mathbf{x}, x_{k+1}, \ldots, x_k)$.

4. If SCADQE succeeds, then Return (α, φ'), else Return fail.

4.9 Remark
We use the notation from Algorithm LCADQE.

1. There is a certain degree of freedom to choose a projection order. The block $(x_{h+1}, \ldots, x_k)$ and every block of like bounded quantified variables can be reordered. In principle, the block $(x_1, \ldots, x_h)$ could be reordered as well, but then this reordered block would have to be returned as output of the algorithm, and the semantics would need to be adjusted straightforwardly.

2. Instead of one point p a set of points P could be given as input. For the points $(p^{(1)}, \ldots, p^{(n)})$ from P and signatures $\sigma^{(i)} := (\text{sign}(f_1(p^{(i)})), \ldots, \text{sign}(f_m(p^{(i)})))$ one would choose then

   $$\alpha := \varphi_{\text{signbased},\left\{\sigma^{(1)},\ldots,\sigma^{(n)}\right\},(\mathbf{f})}.$$

   As a consequence, Property (1) in the specification has to be changed slightly into: for all $p \in P : p \in \mathbb{R}^j_{\alpha(\mathbf{x})}$.

3. The general CADQE algorithm is a special case of LCADQE: Choose $p := ()$, the empty reference point. Then $\alpha = \text{true}$, i.e. the neutral element wrt. $\wedge$. If a set of reference points is allowed, choose $P := \{()\}$.

4. If $(\mathbf{p})$ has length k, then $\varphi' \in \{\mathsf{true}, \mathsf{false}\}$, if the algorithm succeeds.

5. We could have specified $(x_1, \ldots, x_h)$ to be an extension to α, as, looking at the algorithm, indeed $\mathcal{V}(\alpha) \subseteq \{x_1, \ldots, x_h\}$. This was not done due to a later improvement.

4.10 Theorem (properties of LCADQE)
With the notions and assumptions from the algorithm: φ' and α are quantifier-free formulas with $\mathcal{V}(\varphi') \subseteq \{\mathbf{x}\}$ and $\mathcal{V}(\alpha) \subseteq \{x_1, \ldots, x_h\}$. Furthermore it holds:

1. $(\mathbf{p}) \in \mathbb{R}^h_{\alpha(\mathbf{x})}$ *and*

2. $\mathbb{R} \models \alpha \longrightarrow (\varphi \longleftrightarrow \varphi')$.

Proof. By definition in Step 2 it is clear that α is quantifier-free. As the polynomials of α stem from $F_1, \ldots, F_h$, $\mathcal{V}(\alpha) \subseteq \{x_1, \ldots, x_h\}$ is clear as well. Property (1) is clear by the choice of α. $\mathcal{V}(\varphi') \subseteq \{x_1, \ldots, x_k\}$ and Property (2) follows from Theorem 3.34. □

4.2.3 Examples

In the following we will see some (S)CAD trees drawn. There are five kind of cells:

white cell A cell that is computed, but removed from the decomposition. No lifting takes place over such a cell, and it bears no truth value.

gray cell A cell that is not computed due to some improvement.

black cell A regular cell that bears no truth value.

green cell A regular cell, that bears truth value 1.

red cell A regular cell, that bears truth value 0.

These color codes are used in electronic versions of this document. In order to allow to discern gray, red, and green cells in black and white print, we label green cells with “T” and red cells with “F”.

Let us look at the results of the local elimination method on our standard example $\varphi := \exists x(ax^2 + bx + 1)$ with projection order (a, b, x). This yields the projection set $\left\{ax^2 + bx + 1, -4a + b^2, b, a\right\}$.

At first, let us use $(a = 1)$ as a reference point. Figure 4.3 shows the resulting CAD tree. Both in Figure 4.3 and in Figure 4.4 the subtrees of white cells, which have not to be constructed, are displayed in gray. With the notation from the algorithm we have $h = 1$, $m = 1$, $f_1 = a$ and $\sigma = (+1)$. This yields $\alpha = (a > 0)$. Lifting has to occur only over the rightmost subtree in level 1. We get $\varphi' := (-4a + b^2 \geq 0)$ as a solution formula. In total, the algorithm results in $(a > 0, -4a + b^2 \geq 0)$. This tells us that for

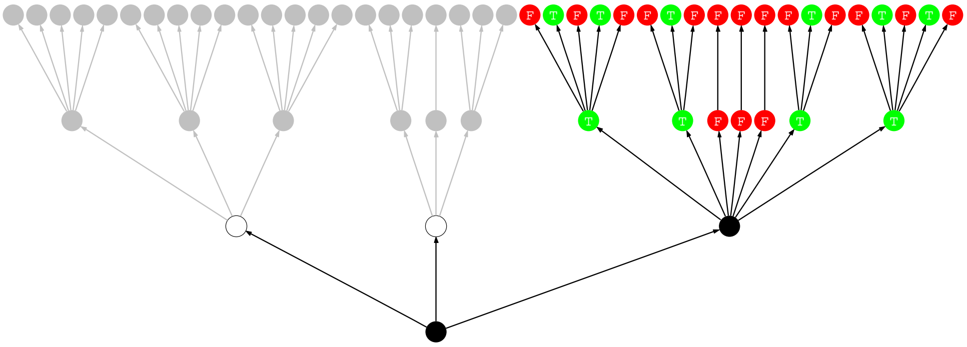


Figure 4.3: Local CAD tree for standard example and reference point $(a = 1)$ as a subtree of the full CAD tree

$a = 1$, and, in fact, for all positive values for a the polynomial $ax^2 + bx + 1$ has a real root if and only if the discriminant $-4a + b^2$ is positive semi-definite.

Let us go one step further and use $(a = 1, b = 3)$ as a reference point. This reference point has maximal length. Figure 4.4 shows the resulting CAD tree. With the notation

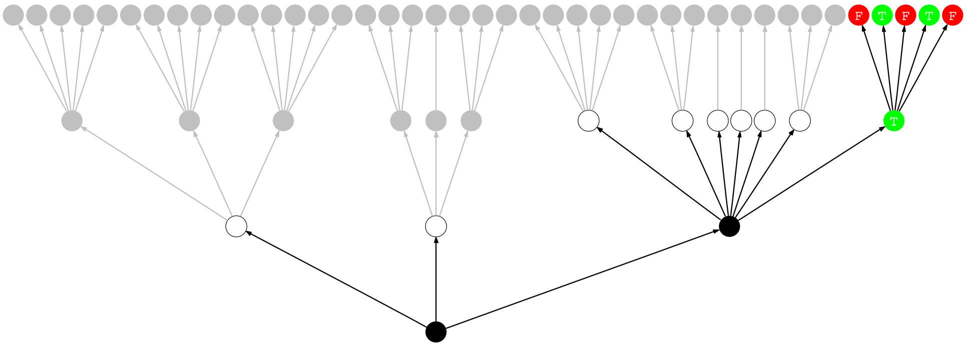


Figure 4.4: Local CAD tree for standard example and reference point $(a = 1, b = 3)$ as subtree of the full CAD tree

from the algorithm we have $h = 2$, $m = 3$, $f_1 = a$, $f_2 = -4a + b^2$, $f_3 = b$, and $\sigma = (+1, +1, +1)$. This yields $\alpha = (a > 0 \wedge -4a + b^2 > 0 \wedge b > 0)$. We can derive the solution formula $\varphi' = \mathsf{true}$. In total, the algorithm results in $(a > 0 \wedge -4a + b^2 > 0 \wedge b > 0, \mathsf{true})$. From this result we gain the following insight: if we fix a positive value for a and choose b such that $b > \sqrt{4a}$, then the univariate polynomial $ax^2 + bx + 1$ has a real root.

We summarize the number of cells in Table 4.2.

4.11 Remark

1. The range the reference point resides in needs not to be an environment in the

level	CADQE	LCADQE ($a = 1$)	LCADQE (a=1,b=3)
3	41	19	5
2	13	7	1
1	3	1	1

Table 4.2: Number of cells in CAD tree of standard example for regular and for local CAD.

sense of topology. I.e. not necessarily is the reference point in the inner part of this set. As an example, consider the standard example with reference point $(a = 0, b = 0)$. Then the algorithm returns (α, false) with $\alpha := (a = 0 \wedge b = 0)$. Now $\mathbb{R}_{\alpha(a,b)}$ is a singleton non-open set that contains the reference point $(0, 0)$.

2. For sample points of maximal length, the solution formula is a truth value. Then, the describing formula for the area is a sufficient condition for the input formula being true or false.

3. The describing formula for the area is in general not a necessary condition for the equivalence of the input and solution formula.

4. The local CAD tree is a subtree of the full CAD tree.

5. The size of the solution of LCADQE will be smaller or equal to the size of the solution of CADQE, as only a subset of the cells of the original CAD tree have to be taken into account for solution formula construction.

4.3 Combination of Generic Projection with Local Elimination

In [DW00b] it was remarked that local and generic QE are closely related for the VS method. We have a similar situation for the CAD method.

4.3.1 Algorithm

Local CAD can be combined with the use of the generic projection operator. We will first discuss the proceedings, and then formulate the overall algorithm.

We get $(\varphi, (\mathbf{x}), (\mathbf{p}))$ as specified for input with LCADQE. If Θ is the list of assumptions generated by generic projection, α is the area description generated for a reference point $(\mathbf{p})$ and variables $(\mathbf{x})$ by local CAD for the generic projection set, then we use SCADQE applied to (φ, β) to compute the solution formula φ', where $\beta := \alpha \wedge \bigwedge \Theta$ and SCADQE is ensured to use the generic projection set and the variable order that was used to compute it. Then (β, φ'), is the result of *local-generic quantifier elimination by CAD* (LGCADQE). We call the CAD tree computed to solve the problem the *local-generic CAD tree*. Similar to local elimination, we want the following two semantical conditions to hold:

1. $(\mathbf{p}) \in \mathbb{R}^h_{\beta(\mathbf{x})}$, and
2. $\mathbb{R} \models \beta \longrightarrow (\varphi \longleftrightarrow \varphi')$.

Instantly the second condition is satisfied. Regarding the first condition there is a problem, which can occur. Consider our standard example and the reference point $a = 0$. Generic projection makes the assumption $a \neq 0$. After projection, the local method makes the assumption $a = 0$. As a result, the level-1 decomposition is comprised only of white cells. Thus $(a = 0 \wedge a \neq 0, \mathsf{false})$ is the result of the algorithm. Although still logically correct, due to $\mathbb{R} \models \mathsf{false} \longrightarrow (\varphi \longleftrightarrow \mathsf{false})$, this is not desired and the first condition is violated. The following shows how it can be ensured already during projection that the first condition is satisfied:

4.12 Remark

1. As argued for the local case, $(\mathbf{p}) \in \mathbb{R}_{\alpha(x_1,\dots,x_h)}$ is ensured by construction of α. This is the argument for the last equivalence in:

$$\begin{aligned}(\mathbf{p}) \in \mathbb{R}^h_{\beta(x_1,\dots,x_k)} &\iff (\exists x_{h+1} \dots \exists x_k \alpha \wedge \bigwedge \Theta)^{\mathbb{R}}(\mathbf{p}) = 1\\ &\iff (\alpha \wedge \exists x_{h+1} \dots \exists x_k \bigwedge \Theta)^{\mathbb{R}}(\mathbf{p}) = 1\\ &\iff (\mathbf{p}) \in \mathbb{R}_{\alpha(x_1,\dots,x_h)} \text{ and } (\mathbf{p}) \in \mathbb{R}^h_{\bigwedge \Theta(x_1,\dots,x_k)}\\ &\iff (\mathbf{p}) \in \mathbb{R}^h_{\bigwedge \Theta(x_1,\dots,x_k)}\end{aligned}$$

   So the condition $(\mathbf{p}) \in \mathbb{R}^h_{\bigwedge \Theta(x_1,\dots,x_k)}$ is necessary and sufficient for Property (1) of the semantics to hold. Still the question is open how it is detected during generic projection whether an assumption can be made.

2. A straightforward solution would be this: At a point during generic projection, when assumptions $\check{\Theta}$ are already made and it must be decided whether to add an assumption ϑ or not, then check $(\mathbf{p}) \in \mathbb{R}^h_{\Theta(x_1,\dots,x_k)}$, where $\Theta := \vartheta \wedge \bigwedge \check{\Theta}$, by applying QE to

$$\exists x_{h+1} \dots \exists x_k \Theta,$$

   which gives $\Theta'(x_1, \dots, x_h)$. The assumption ϑ can be made iff $(\Theta')^{\mathbb{R}}(\mathbf{p}) = 1$. If $(\mathbf{p})$ is a rational point the these values can first be substituted, and then QE can be performed. Iff this result is true then the assumption ϑ can be made.

   As the number of variables for this QE subproblem is smaller than the number of variables of the original problem, and as the set of projection factors for this problem is a small subset of the projection factor set of the original problem, it is expected that this yields not too big an overhead.

3. We can improve on the straightforward approach. It would be much more desirable not to have an accumulating formula to check. We can argue as follows:

$$\begin{aligned}(\mathbf{p}) \in \mathbb{R}^h_{\bigwedge \Theta(x_1,\ldots,x_k)} &\iff (\exists x_{h+1} \ldots \exists x_k \bigwedge \Theta)^{\mathbb{R}}(\mathbf{p}) = 1 \\ &\iff (\bigwedge_{\vartheta \in \Theta} \exists x_{h+1} \ldots \exists x_k \vartheta)^{\mathbb{R}}(\mathbf{p}) = 1 \\ &\iff \text{for all } \vartheta \in \Theta : (\exists x_{h+1} \ldots \exists x_k \vartheta)^{\mathbb{R}}(\mathbf{p}) = 1 \\ &\iff \text{for all } (q \neq 0) \in \Theta : q(\mathbf{p}, x_{h+1}, \ldots, x_k) \neq 0\end{aligned}$$

Here the second equivalence needs more reasoning. In general, existential quantifiers and a conjunction cannot be interchanged. It is, however, not difficult to see that for a family $(q_1, \ldots, q_n)$ of real polynomials in m variables the following conditions are equivalent:

(a) There exists $a \in \mathbb{R}^m$ such that for all $i \in \{1, \ldots, n\} : q_i(a) \neq 0$.

(b) For all $i \in \{1, \ldots, n\}$ there exists $a \in \mathbb{R}^m$ such that $q_i(a) \neq 0$.

Back to the equivalence above, we have now gained a criterion that justifies the following approach: When during generic projection an assumption ϑ, say $\vartheta = (q \neq 0)$ with $q \in I_k$, should be added, then test whether the polynomial $q(\mathbf{p}, x_{h+1}, \ldots, x_k)$, which is a polynomial in $\mathbb{A}[x_{h+1}, \ldots, x_k]$, is not the zero polynomial. The assumption ϑ can be made iff $q(\mathbf{p}, x_{h+1}, \ldots, x_k)$ is not the zero polynomial.

4.3.2 Examples

Let us look at the results of the local elimination method, combined with the generic projection operator, on our standard example. At first, let us use $(a = 1)$ as a reference point. The generic projection operator yields $\Theta = \{a \neq 0\}$. Figure 4.5 shows the resulting local-generic CAD tree. With the notation from the algorithm we have $h = 1$, $m = 1$, $f_1 = a$ and $\sigma = (+1)$. This yields $\alpha = (a > 0)$. By the condition Θ, no lifting has to occur over the middle cell C_2 in Level 1. In addition, the condition α excludes the left cell C_1 of Level 1 as well. C_1 and C_2 are white cells. Therefore, lifting has to occur only over the rightmost cell C_3 in Level 1. Due to the lack of the level-2 projection factor b there are now only five instead of seven cells in the stack above C_3.

We get $\varphi' := (-4a + b^2 \geq 0)$ as a solution formula. In total, The algorithm results in $(a > 0 \wedge a \neq 0, -4a + b^2 \geq 0)$. This tells us that for $a = 1$, and, in fact, for all positive values for a the polynomial $ax^2 + bx + 1$ has a real root if and only if the discriminant $-4a + b^2$ is positive semi-definite.

Let us go one step further and use $(a = 1, b = 3)$ as a reference point. This reference point has maximal length. Figure 4.4 shows the resulting local-generic CAD tree. With the notation from the algorithm we have $h = 2$, $m = 2$, $f_1 = a$, $f_2 = -4a + b^2$ and

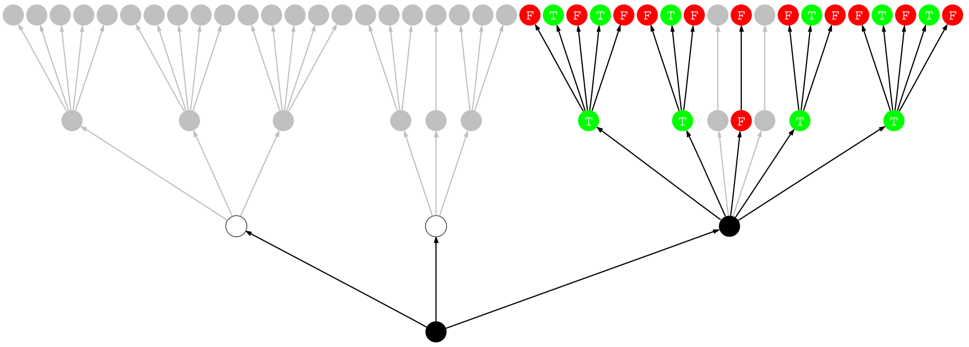


Figure 4.5: Full CAD tree for standard example overlayed with local-generic CAD tree for reference point $(a = 1)$

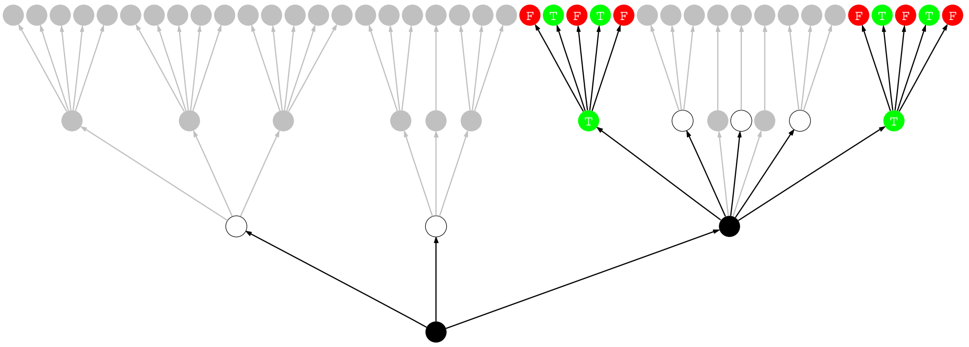


Figure 4.6: Full CAD tree for standard example overlayed with local-generic CAD tree for reference point $(a = 1, b = 3)$

$\sigma = (+1, +1)$. This yields $\alpha = (a > 0 \wedge -4a + b^2)$. We can derive the solution formula true. In total, the algorithm results in $(a > 0 \wedge -4a + b^2, \mathsf{true})$. From this result we gain the following insight: if we fix a positive value for a and choose b such that $|b| > \sqrt{4a}$, then the univariate polynomial $ax^2 + bx + 1$ has a real root.

We summarize the number of cells in Table 4.3:

4.3.3 Observations and Remarks

1. For a local QE problem, the number of cells that have to be computed for the local-generic CAD tree is not necessarily less than the number for the local CAD tree. The reason is that with fewer projection factors the area description gets weaker. Nevertheless, although there are more cells, the roots of the according polynomials are more easily isolated in the local-generic case. This is why one can

level	CADQE	LGCADQE	LGCADQE
		$(a = 1)$	$(a = 1, b = 3)$
3	41	17	10
2	13	5	2
1	3	1	1

Table 4.3: Number of cells in CAD tree of standard example for regular CAD and for local CAD combined with generic projection operator.

expect the local-generic variant to be faster than the purely local variant, even if the former results in more cells to be constructed.

2. The local-generic CAD tree is a subtree of the generic CAD tree.

3. The local-generic CAD tree is in general not a subtree of the full CAD tree. The reason is, that in general the generic CAD tree is not a subtree of the full CAD tree. For the sake of presentation and better comparison with the non-generic case the local-generic trees are drawn over a full CAD tree. Thus, the two gray cells displayed in the stack over C_3 are not part of the generic, and therefore not part of the local-generic CAD tree. The area they are comprised of is in the generic and local-generic case part of the middle cell C_{33} in the stack over C_3.

4.4 Further Applications

We want to sketch some further applications.

1. Salvage signature-based solution formula construction. As pointed out at several locations, signature-based solution formula construction can fail in rare cases due to compatible signatures. A simple example is this:

$$\varphi_{\text{cessfc}} := \exists y(x^2 + y^2 < 1 \land x + y > 0)$$

Projection results in the level-1 projection factors $f_1 := 2x^2$, $f_2 := x-1$, $f_3 := x+1$. Decomposition and evaluation and propagation of truth values results in nine level-1 cells with truth values and signatures for (f_1, f_2, f_3) as listed in Table 4.4.

Signature-based solution formula construction finds sets $W_t = \{\sigma_5, \sigma_6, \sigma_7\}$, $W_f = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_8, \sigma_9\}$, and $W_c = \{\sigma_6, \sigma_7\}$. As $W_c \neq \emptyset$ there are conflicting signatures, and solution formula construction fails, or, as suggested by Hong, returns a sufficient and a necessary solution formula.

By thinking in terms of subdecomposition, we can suggest this alternative approach. Make the assumption $\alpha := \neg\varphi_{\text{signbased},\mathrm{W_c},(\mathbf{f})}$. This removes all cells with conflicting signatures from the decomposition, and SFC can give a result φ'. For

i	C_i	v_i	σ_i
1	$]-\infty, -1[$	0	$(+1, -1, -1)$
2	$\{-1\}$	0	$(+1, -1, 0)$
3	$]-1, -\frac{\sqrt{2}}{2}[$	0	$(+1, -1, +1)$
4	$\left\{-\frac{\sqrt{2}}{2}\right\}$	0	$(0, -1, +1)$
5	$]-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}[$	1	$(-1, -1, +1)$
6	$\left\{\frac{\sqrt{2}}{2}\right\}$	1	$(0, -1, +1)$
7	$]\frac{\sqrt{2}}{2}, 1[$	1	$(+1, -1, +1)$
8	$\{1\}$	0	$(+1, 0, +1)$
9	$]1, \infty[$	0	$(+1, +1, +1)$

Table 4.4: Information about the level-1 cells of φ_{cessfc}

the example φ_{cessfc} we get the solution $\varphi' = (2x^2 - 1 < 0 \land x - 1 < 0 \land x + 1 > 0)$, which is readily simplified to $2x^2 - 1 < 0$.

Note that φ' is equivalent to the sufficient condition, and $\neg\alpha \lor \varphi'$ is equivalent to the necessary condition suggested by Hong. We believe that our suggestion is more intuitive and more concise for the user.

2. CAD with external theory. In [Dol00] it was shown how to aid quantifier elimination by virtual substitution. Our subdecomposition framework shows how this can be done for the CAD method. We can even allow conditions on bounded variables.

3. Salvage decomposition based on Brown-McCallum projection operator. Brown-McCallum projection based decomposition can fail. It is a conjecture that SCAD framework can be adapted for this approach and that in case of failure assumptions can be made to exclude problematic cells. To do this, we need the feature of making assumptions on bounded variables, as provided by our framework.

4. Consider full-dimensional cells only. A full-dimensional cell is a cell where no component of the sample point is found by root isolation. I.e. full-dimensional cells are the cells that have an index where each component is odd. Only considering full-dimensional cells is desirable, as root isolation is needed only for integer polynomials. Avoiding implementation of and computation with algebraic numbers greatly improves implementation and execution speed. Such an approach was only used so far for very special cases. SCAD clarifies the semantics and makes this approach possible in general. Consider the example

$$\varphi := \exists y(x^2 + y^2 - 1 \leq 0).$$

CADQE would return $-1 \leq x \leq 1$ as a result. A SCAD that considers only full-dimensional cells would yield $\varphi' := -1 < x < -1$. Such an approach that yields results which are correct up to measure zero sets is particularly promising in engineering.

4.5 Conclusions

In this chapter we have developed quantifier elimination by generic CAD and local CAD as applications within the SCAD framework. In addition, we have shown how these approaches can be combined with each other, and sketched more applications.

By looking at examples we see that the amount of computation needed is greatly reduced as expected. This is done at the expense of a relaxed semantics. While from a theoretical point of view a weaker semantics might seem inferior at first, it turns out that for both generic CAD and local CAD the generated assumptions consist of interesting information. In addition, from a practical point of view the actual user might be more happy to get an result that excludes certain degenerate or special cases, or that is locally correct, than to wait considerably longer, or indefinitely, for a general result.

Chapter 5

Getting Answers

We show how the cylindrical algebraic decomposition (CAD) method for real quantifier elimination (QE) can be extended to provide answers, i.e. sample solutions for variables in a leading block of existential quantifiers, thus providing more and interesting information.

5.1 Introduction

For the virtual substitution method, an interesting extension has been developed [Wei94]. This method was extended to give sample parametric answers for values of the variables in a leading block of existential quantifiers. This has turned out to be very useful in practice, e.g. for error diagnosis in electrical networks [Stu99b] and for collision problems [Stu99a].

This chapter deals with extending the CAD method for real QE to give, next to a quantifier-free equivalent formula, answers. In this chapter:

1. We motivate that sometimes one would like to get more information out of QE than just an equivalent formula, e.g. sample values for certain variables.

2. We demonstrate that prior progress was made to extend the virtual substitution method for real QE to give answers. Due to limitations on the VS method, however, QE with answers was so far not applicable to all problems over the reals.

3. We give a CAD-based algorithm to show how the CAD method can be exploited to provide answers.

4. By extending the CAD method we lift the limitations on the applicability of extended quantifier elimination.

5. We explain how, by duality, one can get parametric counter-examples for formulas, which have an outermost block of universal quantifiers.

The content of this chapter is published in [Sei04]

5.2 Motivating Examples and Prior Work

We look at some examples to give an impression of how real quantifier elimination can be used to model and solve a problem, to demonstrate prior progress made to extend the virtual substitution method to give answers, and to motivate the suggested improvement.

5.2.1 Solving a Tangram Style Puzzle

Consider the following puzzle. There are four small squares A, B, C, D of diameter 2 and one square of diameter 4. The squares are arranged as can be seen in Figure 5.1. Every edge is parallel to the bisecting line of the first and third, or, of the second and fourth quadrant. More precisely, the position of, say, the square A in the real plane is given by the corner points (a_1, a_2), $(a_1 - 1, a_2 + 1)$, $(a_1, a_2 + 2)$ and $(a_1 + 1, a_2 + 1)$. The position of the big square S is given by $(0, 0)$, $(-2, 2)$, $(0, 4)$ and $(2, 2)$. The challenge is to decide whether the small squares fit by translation into the big one without overlapping each other. Let us formulate this problem in first-order logic. First of all, we need a formula describing whether a point (x_1, x_2) lies within the small square A. Its relative position is given by (a_1, a_2).

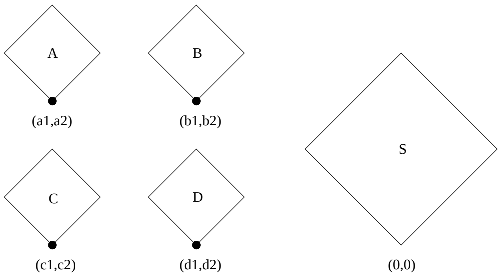


Figure 5.1: Four small squares (left) and a big square (right)

$$\begin{aligned}\varphi_A :\equiv\ & a_1 + a_2 - x_1 - x_2 + 2 > 0 \land a_1 + a_2 - x_1 - x_2 < 0 \land \\ & a_1 - a_2 - x_1 + x_2 - 2 < 0 \land a_1 - a_2 - x_1 + x_2 > 0.\end{aligned}$$

Similarly, one finds formulas φ_B, φ_C, and φ_D. For the big square S

$$\varphi_S :\equiv x_1 + x_2 - 4 < 0 \land x_1 + x_2 > 0 \land x_1 - x_2 + 4 > 0 \land x_1 - x_2 < 0$$

is appropriate. Note that we have to consider *open* objects without border.

The squares A, B, C, D lie within S, for fixed (a_1, a_2), (b_1, b_2), (c_1, c_2), (d_1, d_2), iff

$$(\varphi_A \lor \varphi_B \lor \varphi_C \lor \varphi_D) \longrightarrow \varphi_S$$

for all (x_1, x_2). Furthermore, the squares A, B, C, D do not overlap each other, iff

$$\neg(\varphi_A \wedge \varphi_B) \wedge \neg(\varphi_A \wedge \varphi_C) \wedge \neg(\varphi_A \wedge \varphi_D) \wedge \neg(\varphi_B \wedge \varphi_C) \wedge \neg(\varphi_B \wedge \varphi_D) \wedge \neg(\varphi_C \wedge \varphi_D).$$

Altogether the problem can be stated as follows:

$$\varphi :\equiv \exists a_1 \exists a_2 \exists b_1 \exists b_2 \exists c_1 \exists c_2 \exists d_1 \exists d_2 \forall x_1 \forall x_2 (\neg(a_1+a_2-x_1-x_2+2 > 0 \wedge a_1+a_2-x_1-x_2 < 0 \wedge a_1-a_2-x_1+x_2-2 < 0 \wedge a_1-a_2-x_1+x_2 > 0 \wedge b_1+b_2-x_1-x_2+2 > 0 \wedge b_1+b_2-x_1-x_2 < 0 \wedge b_1-b_2-x_1+x_2-2 < 0 \wedge b_1-b_2-x_1+x_2 > 0) \wedge \neg(a_1+a_2-x_1-x_2+2 > 0 \wedge a_1+a_2-x_1-x_2 < 0 \wedge a_1-a_2-x_1+x_2-2 < 0 \wedge a_1-a_2-x_1+x_2 > 0 \wedge c_1+c_2-x_1-x_2+2 > 0 \wedge c_1+c_2-x_1-x_2 < 0 \wedge c_1-c_2-x_1+x_2-2 < 0 \wedge c_1-c_2-x_1+x_2 > 0) \wedge \neg(a_1+a_2-x_1-x_2+2 > 0 \wedge a_1+a_2-x_1-x_2 < 0 \wedge a_1-a_2-x_1+x_2-2 < 0 \wedge a_1-a_2-x_1+x_2 > 0 \wedge d_1+d_2-x_1-x_2+2 > 0 \wedge d_1+d_2-x_1-x_2 < 0 \wedge d_1-d_2-x_1+x_2-2 < 0 \wedge d_1-d_2-x_1+x_2 > 0) \wedge \neg(b_1+b_2-x_1-x_2+2 > 0 \wedge b_1+b_2-x_1-x_2 < 0 \wedge b_1-b_2-x_1+x_2-2 < 0 \wedge b_1-b_2-x_1+x_2 > 0 \wedge c_1+c_2-x_1-x_2+2 > 0 \wedge c_1+c_2-x_1-x_2 < 0 \wedge c_1-c_2-x_1+x_2-2 < 0 \wedge c_1-c_2-x_1+x_2 > 0) \wedge \neg(b_1+b_2-x_1-x_2+2 > 0 \wedge b_1+b_2-x_1-x_2 < 0 \wedge b_1-b_2-x_1+x_2-2 < 0 \wedge b_1-b_2-x_1+x_2 > 0 \wedge d_1+d_2-x_1-x_2+2 > 0 \wedge d_1+d_2-x_1-x_2 < 0 \wedge d_1-d_2-x_1+x_2-2 < 0 \wedge d_1-d_2-x_1+x_2 > 0) \wedge \neg(c_1+c_2-x_1-x_2+2 > 0 \wedge c_1+c_2-x_1-x_2 < 0 \wedge c_1-c_2-x_1+x_2-2 < 0 \wedge c_1-c_2-x_1+x_2 > 0 \wedge d_1+d_2-x_1-x_2+2 > 0 \wedge d_1+d_2-x_1-x_2 < 0 \wedge d_1-d_2-x_1+x_2-2 < 0 \wedge d_1-d_2-x_1+x_2 > 0) \wedge (((a_1+a_2-x_1-x_2+2 > 0 \wedge a_1+a_2-x_1-x_2 < 0 \wedge a_1-a_2-x_1+x_2-2 < 0 \wedge a_1-a_2-x_1+x_2 > 0) \vee (b_1+b_2-x_1-x_2+2 > 0 \wedge b_1+b_2-x_1-x_2 < 0 \wedge b_1-b_2-x_1+x_2-2 < 0 \wedge b_1-b_2-x_1+x_2 > 0) \vee (c_1+c_2-x_1-x_2+2 > 0 \wedge c_1+c_2-x_1-x_2 < 0 \wedge c_1-c_2-x_1+x_2-2 < 0 \wedge c_1-c_2-x_1+x_2 > 0) \vee (d_1+d_2-x_1-x_2+2 > 0 \wedge d_1+d_2-x_1-x_2 < 0 \wedge d_1-d_2-x_1+x_2-2 < 0 \wedge d_1-d_2-x_1+x_2 > 0)) \longrightarrow (x_1+x_2-4 < 0 \wedge x_1+x_2 > 0 \wedge x_1-x_2+4 > 0 \wedge x_1-x_2 < 0))).$$

Generating such formulas by hand can be a tedious and error-prone task. Therefore the author has implemented a package TANGRAM. This package was written to aid the formulation of tangram style problems. It allows for the easy creation of formulas describing convex objects, which have a polygonal outline. In addition, based on already defined shapes, a formula saying that certain small shapes fit into a bigger shape without overlapping each other, can be produced. The above formulas were generated by this software.

Let us now apply QE to this problem formulation φ. We derive *true* with the virtual substitution method. Although this is the correct answer, we wish to get more information. Where do we have to place the small shapes? This is where *extended* QE, or QE *with answers* comes into play.

For the virtual substitution method, prior research [Wei94] has shown that for the outermost block of quantifiers one can get sample values in the existential case or counter-examples in the universal case for the according variables. For our example, the virtual substitution method yields *true* and the sample values

$$a_1 = 0\ ,\ a_2 = 2\ ,\ b_1 = 0\ ,\ b_2 = 0\ ,\ c_1 = 1\ ,\ c_2 = 1\ ,\ d_1 = -1\ ,\ d_2 = 1.$$

5.2.2 A Parametric Example

Variables, which are not within the scope of a quantifier, are called *free variables* or *parameters*. The preceding example was a *closed* formula, i.e. a formula without free variables. For such an example we can get concrete values as answers. If there are parameters such answers will have to be parametric as well in general.

Past research for the virtual substitution method showed that there is a natural way for this method to deliver the parametric answers [Wei94], thus this method is not restricted to decision problems.

Consider the following example. The formula $\varphi :\equiv \exists x(ax^2 + bx + 1 = 0)$ asks for conditions on the parameters a, b such that the quadratic polynomial $ax^2 + bx + 1$ has a real root. QE by VS with answers returns three guarded points:

$$\left(4a - b^2 \le 0 \land a \ne 0, \left(x = \frac{-\sqrt{-4a + b^2} - b}{2a}\right)\right)$$
$$\left(4a - b^2 \le 0 \land a \ne 0, \left(x = \frac{+\sqrt{-4a + b^2} - b}{2a}\right)\right)$$
$$\left(a = 0 \land b \ne 0, \left(x = \frac{-1}{b}\right)\right).$$

This tells us not only that $4a - b^2 \le 0 \land a \ne 0 \lor a = 0 \land b \ne 0$ is equivalent to $ax^2 + bx + 1$ having a real root. It tells us in addition what such a root x looks like in the two cases $4a - b^2 \le 0 \land a \ne 0$ and $a = 0 \land b \ne 0$.

5.2.3 Finding Extraneous Points

Consider the parametric curve (f_1, g_1) with

$$f_1(t) = -6t^4 - 63, \quad g_1(t) = 92t^3 + 70t^2.$$

By computing the resultant ρ of $x - f_1$ and $y - g_1$ wrt. t one can get an implicit description of this parametric curve. Such a description is often not exact. More precisely, the graph

$$\{(x, y) \in \mathbb{R}^2 | \text{exists } t \in \mathbb{R} \text{ such that } x - f_1(t) = 0 \text{ and } y - g_1(t) = 0\}$$

of the parametric curve is often a proper subset of the real variety of $\{\rho\}$, i.e. the set

$$\{(x, y) \in \mathbb{R}^2 | \rho(x, y) = 0\}.$$

To get an *exact* real implicit representation, we apply real quantifier elimination to

$$\exists t(x = -6t^4 - 63 \text{ and } y = 92t^3 + 70t^2)$$

and derive

$$\begin{aligned} &8954912x^3 - 1777440x^2y + 1710485868x^2 + 44100xy^2 - 223957440xy + \\ &108895082184x + 27y^4 + 2778300y^2 - 7054659360y + 2310620648364 = 0 \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{and } x + 63 \leq 0 \end{aligned}$$

To do this we need the CAD method, as the VS methods fails due to degree restrictions. The result is equivalent to

$$\rho = 0 \text{ and } x \leq -63$$

At this point, quantifier elimination with answers could help us to find possible extraneous points, which lie in the real variety of ρ, but not in the graph of the given parametric curve.

These examples make it obvious that for decision problems, as well as for parametric problems, getting answers out of quantifier elimination in addition to quantifer-free equivalents is highly desirable. Furthermore, getting answers from the CAD method is desirable in particular, due to limitations of the VS method, as the last example showed.

5.3 QE with Parametric Answers

We want to specify now what the result of QE with answers is. From now on we assume the input formula

$$\varphi(x_1, \dots, x_k) \equiv \exists x_{k+1} \cdots \exists x_l \mathsf{Q}_{l+1} x_{l+1} \cdots \mathsf{Q}_r x_r \psi$$

to be in prenex normal form. The input formula has $x_1, \dots, x_k$ as free and $x_{k+1}, \dots, x_r$ as bound variables. For each $k+1 \leq j \leq r$ the symbol Q_j denotes x_j's quantifier. We furthermore assume that Q_{l+1} is a universal quantifier. So $(x_{k+1}, \dots, x_l)$ is the leading block of existentially quantified variables of maximal length.

5.3.1 Specification

The output of QE with answers is specified to be a finite set of guarded points [DS97a]

$$\{(\psi'_i, (x_{k+1} = b_{i,k+1}, \dots, x_l = b_{i,l})) | i \in I\}.$$

Here the ψ_i's are quantifier-free formulas, and $b_{i,j}$ is, for some i and $k+1 \leq j \leq l$, a term which contains at most the variables $x_1, \dots, x_j$. Such a term is an arithmetic expression, which may include a binary predicate of the form *Root*(f, n), where n is a natural number and f a polynomial expression. The variable x_j can only occur within a polynomial expression of a *Root* predicate. Note that within a *Root* predicate the role of x_j could be assumed by any other variable, except from $\{x_1, \ldots, x_{j-1}\}$, without impact on the map γ defined below.

A guarded point $(\psi, (x_{k+1} = b_{k+1}, \ldots, x_l = b_l))$ can be viewed to define a partial map $\gamma : \mathbb{R}^k \to \mathbb{R}^{l-k}$: Given values $(a_1, \ldots, a_k) \in \mathbb{R}^k$, such that $\mathbb{R} \models \psi(a_1, \ldots, a_k)$, we can successively compute $a_{k+1}, \ldots, a_l$ in the following way: Let us assume that $a_{k+1}, \ldots, a_j$ are already computed for a $k \leq j < l$, and now we want to compute a_{j+1}. We substitute in the term b_{j+1} the values $a_1, \ldots, a_j$ for the variables $x_1, \ldots, x_j$. This yields an expression b'_{j+1}, where x_{j+1} is the only variable that can occur. If so, it occurs in the polynomial expression of a *Root* symbol. Now, each occurrence of an expression $\mathit{Root}(f, n)$ in b'_{j+1} can be replaced with the n-th root of the univariate polynomial f. For guarded points constructed by the algorithm below this will always be well defined. We get an expression b''_{j+1} without free variables, which can be evaluated in $\mathbb{R}$ to a_{j+1}.

After detailing the syntactical definition, we specify how the semantics of the output of QE with answers should be. There are two conditions.

(C1) *Quantifier-free formula.* We want to easily construct a quantifier-free formula, which is equivalent to the input formula, from the set of guarded points:

$$\mathbb{R} \models \varphi \leftrightarrow \bigvee_{i \in I} \psi'_i.$$

(C2) *Example solution.* For every guarded point $(\psi, (x_{k+1} = b_{k+1}, \ldots, x_l = b_l))$ in the output set and every $(a_1, \ldots, a_k) \in \mathbb{R}^k$ with $\mathbb{R} \models \psi(a_1, \ldots, a_k)$, we want to have

$$\mathbb{R} \models \mathsf{Q}_{l+1} x_{l+1} \cdots \mathsf{Q}_r x_r \psi(a_1, \ldots, a_l),$$

where $\gamma(a_1, \ldots, a_k) = (a_{k+1}, \ldots, a_l)$.

These two conditions motivate why QE with answers is also called extended QE: The output of a quantifier-free formula, as provided by classical QE, is extended to provide sample solutions in addition.

5.4 Situation for the CAD Method

5.4.1 Algorithm

We now want to investigate if we can achieve similar results for the CAD method. Still the same assumptions on the form of the input formula φ are made as at the beginning of Section 5.3.

Let us furthermore assume that a full CAD tree for this problem has been constructed, and cells of $D_r, \ldots, D_k$ bear a truth value. For each cell $C \in D_k$ let δ_C denote a quantifier-free description of this cell. Such a formula exists, as each cell represents a semi-algebraic set. Let G denote the finite set which is generated by collecting for each true cell $C^{(l)}$ in D_l a guarded point $(\delta_C, x_{k+1} = p_{k+1}, \dots, x_l = p_l)$. Here $C \in D_k$ is the unique predecessor of $C^{(l)}$, and $p_{k+1}, \dots, p_l$ are derived in the following way: Let $C, C^{(k+1)}, \dots, C^{(l)}$ be the path from C to $C^{(l)}$ in the CAD tree and $k+1 \le j \le l$. To define p_j, we look how the last component s_j of $C^{(j)}$'s sample point $(s_1, \dots, s_j)$ was generated. There are four cases.

1. There is a projection polynomial $f \in F_j$ such that s_j is the n-th root of the polynomial $f(s_1, \dots, s_{j-1}, x_j) \in \mathbb{A}[x_j]$. Then $p_j := \mathit{Root}(f, n)$.
2. There are projection polynomials $f, f' \in F_j$ such that s_j lies between the n-th root of the polynomial $f(s_1, \dots, s_{j-1}, x_j)$ and the m-th root of $f(s_1, \dots, s_{j-1}, x_j)$ and there exists no other root in between. Then $p_j := (\mathit{Root}(f, n) + \mathit{Root}(f, m))/2$.
3. If s_j is smaller than the smallest or greater than the biggest root, then $p_j := \mathit{Root}(f, 1) - 1$ or $p_j := \mathit{Root}(f, m) + 1$ for an appropriate $f \in F_j$ and integer m.
4. Otherwise, if there was no root at all, p_k can be simply defined as 0 or as the special symbol *arbitrary*.

After this set of guarded points is defined, we want to allow two ways to manipulate this set to make it more concise.

1. *Combine.* Two guarded points, which only differ in the first part, can be combined:
$$(\psi, b), (\psi', b) \mapsto (\psi \vee \psi', b).$$
2. *Simplify.* The first part of a guarded point can be replaced by an equivalent one.

In contrast to the virtual substitution method, the degree of the polynomials can be arbitrary. Thus we cannot rely on radicals to specify p_j. Instead we had to introduce the symbol *Root*. The semantics of *Root* is self-explanatory. The delineability property of the projection set ensures *Root* to be well defined.

5.4.2 Correctness

We need to check whether the conditions (C1) and (C2) are satisfied. As for (C1) a valid solution formula can be constructed as a disjunction over formulas describing the true cells of D_k.

To see that (C2) holds, note that a full CAD tree D for

$$\mathsf{Q}_{l+1} x_{l+1} \cdots \mathsf{Q}_r x_r \psi$$

is essentially the same as for φ. The only difference is that cells on the levels $D_k, \dots, D_{l-1}$ bear no truth value. If one now computes for a guarded point and for $(a_1, \dots, a_k) \in \mathbb{R}^k$ the values $a_{k+1}, \dots, a_l$, then the point $(a_1, \dots, a_l)$ lies in a true cell of D_l. Hence (C2) holds.

Furthermore, if for a set of guarded points both (C1) and (C2) hold, then combining and simplifying points as described above will preserve these properties.

5.4.3 Examples

Second Example Revisited

We revisit the example $\varphi :\equiv \exists x(ax^2+bx+1=0)$ from Subsection 5.2.2 to demonstrate the algorithm. Figure 5.2 shows the full CAD tree for this problem. Here $k=2$, $l=3$, and $r=3$ with our notation. Let us denote the three cells of D_1 by C_1, C_2, C_3. We name

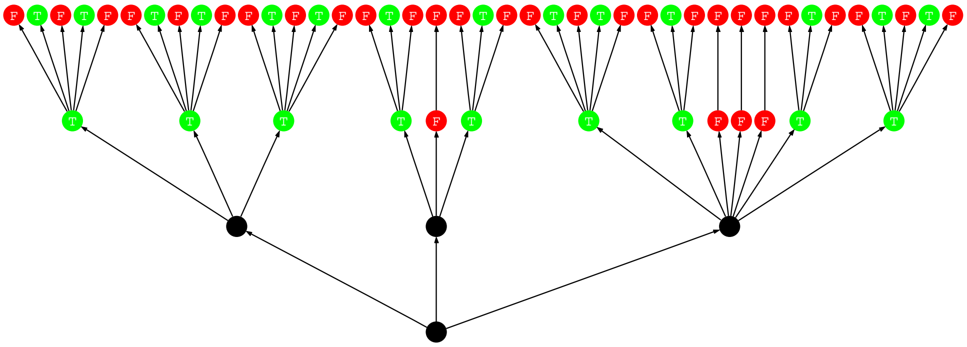


Figure 5.2: Full CAD tree for $\varphi \equiv \exists x(ax^2+bx+1=0)$ wrt. the variable order $x \to b \to a$ and projection sets $F_3 = \left\{ax^2+bx+1\right\}$, $F_2 = \left\{-4a+b^2, b\right\}$, and $F_1 = \{a\}$. Solid black cells bear no truth value.

cells on a higher level similarly, but extend the index of the corresponding base cell. So, e.g., the three children of C_1 in D_2 will be called C_{11}, C_{12}, C_{13} and the five children of C_{11} will be called $C_{111}, \ldots, C_{115}$. There are 14 true cells in D_3. The first true cell is C_{112}. The last component of its sample point was generated as a first root. If δ_{11} is a describing formula for C_{11} then $(\delta_{11}, (x = \mathit{Root}(ax^2+bx+1, 1)))$ is the first guarded point. The last component of the sample point of C_{114} was generated as a second root. Hence $(\delta_{11}, (x = \mathit{Root}(ax^2+bx+1, 2)))$ is the next guarded point to collect. Proceeding with the algorithm we get the following twelve additional guarded points:

$$
\begin{gathered}
(\delta_{12}, (x = \mathit{Root}(ax^2+bx+1, 1))) \\
(\delta_{12}, (x = \mathit{Root}(ax^2+bx+1, 2))) \\
(\delta_{13}, (x = \mathit{Root}(ax^2+bx+1, 1))) \\
(\delta_{13}, (x = \mathit{Root}(ax^2+bx+1, 2))) \\
(\delta_{21}, (x = \mathit{Root}(ax^2+bx+1, 1))) \\
(\delta_{23}, (x = \mathit{Root}(ax^2+bx+1, 1)))
\end{gathered}
$$

$$\begin{aligned}
&(\delta_{31}, (x = \mathit{Root}(ax^2 + bx + 1, 1)))\\
&(\delta_{31}, (x = \mathit{Root}(ax^2 + bx + 1, 2)))\\
&(\delta_{32}, (x = \mathit{Root}(ax^2 + bx + 1, 1)))\\
&(\delta_{36}, (x = \mathit{Root}(ax^2 + bx + 1, 1)))\\
&(\delta_{37}, (x = \mathit{Root}(ax^2 + bx + 1, 1)))\\
&(\delta_{37}, (x = \mathit{Root}(ax^2 + bx + 1, 2)))
\end{aligned}$$

We can reduce the number of points by combining them as follows: As $\delta_{21} \vee \delta_{23}$ is equivalent to $a = 0 \wedge b \neq 0$, we combine point 7 and 8 to

$$(a = 0 \wedge b \neq 0, (x = \mathit{Root}(ax^2 + bx + 1, 1))).$$

The latter could be further simplified by making use of the knowledge $a = 0$. This reduces the quadratic polynomial to a linear one.

Furthermore, as $\delta_{11} \vee \delta_{12} \vee \delta_{13} \vee \delta_{31} \vee \delta_{32} \vee \delta_{36} \vee \delta_{37}$ is equivalent to $4a - b^2 \leq 0 \wedge a \neq 0$, we can combine the points $1, 3, 5, 9, 11, 12, 13$ to

$$(4a - b^2 \leq 0 \wedge a \neq 0, (x = \mathit{Root}(ax^2 + bx + 1, 1))).$$

Finally, as $\delta_{11} \vee \delta_{12} \vee \delta_{13} \vee \delta_{31} \vee \delta_{37}$ is equivalent to $4a - b^2 < 0 \wedge a \neq 0$, we can combine the points $2, 4, 6, 10, 14$ to

$$(4a - b^2 < 0 \wedge a \neq 0, (x = \mathit{Root}(ax^2 + bx + 1, 2))).$$

We finish this example by concluding that we could reduce the 14 guarded points to three, which quite match the three cases found by the VS method. (Actually, the reason why we did not get exactly the same result are the cells C_{322} and C_{362}, where the last component of the sample point is a single root with multiplicity two.)

Third Example Revisited

In Subsection 5.2.3 we found an exact implicit description for a given parametric curve, and ended up with the formula

$$\rho = 0 \text{ and } x \leq -63.$$

This hints that there might be extraneous points in the halfspace $x > -63$. So we apply extended QE by CAD to

$$\exists x \exists y (\rho = 0 \wedge x > 63)$$

and derive *true* and the answer

$$x = \frac{-30758091}{559682}, \quad y = \frac{42875}{529}.$$

We can now convince ourselves that this is the only extraneous point by applying QE to

$$\exists x \exists y(x > -63 \land \rho = 0 \land \neg(559682x = -30758091 \land 529y = 42875))$$

and deriving *false*.

5.5 Dual Answers: Parametric Counter-Examples

In this section we assume the input formula

$$\varphi(x_1, \dots, x_k) \equiv \forall x_{k+1} \cdots \forall x_l \mathsf{Q}_{l+1} x_{l+1} \cdots \mathsf{Q}_r x_r \psi$$

to have a leading block of universal quantifiers. This is dual to the assumption in Section 5.3. By negating this input formula, we get

$$\overline{\varphi}(x_1, \dots, x_k) \equiv \exists x_{k+1} \cdots \exists x_l \overline{\mathsf{Q}_{l+1}} x_{l+1} \cdots \overline{\mathsf{Q}_r} x_r \overline{\psi}.$$

For formulas, overlining is just an alternative notation for negation. For quantifiers, an overlined quantifier denotes the dual one. Let

$$\{(\psi'_i, (x_{k+1} = b_{i,k+1}, \dots, x_l = b_{i,l}))|i \in I\}$$

be the output of the extended algorithm on input of $\overline{\varphi}$. Then, dually to (C1) and (C2) the following two conditions hold.

($\overline{\text{C1}}$) *Quantifier-free formula.* We can easily construct from the set of guarded points a quantifier-free formula, which is equivalent to the input formula:

$$\mathbb{R} \models \varphi \leftrightarrow \bigwedge_{i \in I} \neg \psi'_i.$$

($\overline{\text{C2}}$) *Counter-example.* For every guarded point $(\psi, (x_{k+1} = b_{k+1}, \dots, x_l = b_l))$ in the output set and every $(a_1, \dots, a_k) \in \mathbb{R}^k$ with $\mathbb{R} \models \psi(a_1, \dots, a_k)$, we have

$$\mathbb{R} \not\models \mathsf{Q}_{l+1} x_{l+1} \cdots \mathsf{Q}_r x_r \psi(a_1, \dots, a_l),$$

where $\gamma(a_1, \dots, a_k) = (a_{k+1}, \dots, a_l)$.

In other words: Instead of as a union of true cells, we get the set described by a solution formula as an intersection of complements of false cells. If given parameter values $(a_1, \dots, a_k)$ lie in a false cell, then the appropriate guarded point delivers counter-examples $(a_{k+1}, \dots, a_l)$ such that $\mathsf{Q}_{l+1} x_{l+1} \cdots \mathsf{Q}_r x_r \psi(a_1, \dots, a_l)$ does not hold.

5.6 Conclusions

We have motivated that it is highly desirable to produce answers in addition to a solution formula as output of quantifier elimination for a formula with a leading existential block of quantifiers. We have devised an algorithm to extend the cylindrical algebraic decomposition method to produce such answers. For non-decision problems, these sample solutions are in general parametric. For the dual case, i.e. for formulas with a leading universal block of quantifiers, we get parametric counter-examples. The main advantage over a virtual substitution based approach is that there are no degree restrictions on the input.

Chapter 6

Efficient Projection Orders

We introduce an efficient algorithm for determining a suitable projection order for performing cylindrical algebraic decomposition. Our algorithm is motivated by a statistical analysis of comprehensive test set computations. This analysis introduces several measures on both the projection sets and the entire computation, which turn out to be highly correlated. The statistical data also shows that the orders generated by our algorithm are significantly close to optimal. This improvement is applicable to both, pure CAD and QE by CAD. The content of this chapter, except of Section 6.3, is published in [DSS03].

6.1 Introduction

During the past 30 years there have been considerable research and publications on optimizing CAD. For the application to real quantifier elimination, the introduction of *partial* CAD (PCAD) [CH91] has been one major progress, which affects the extension phase.

The vast majority of improvements, however, extremely focused on improving the projection phase [McC84, McC88, Hon90, Laz94, Bro01, SS03]. Most surprisingly, all these contributions concentrating on improved projection operators never examined the relevance of the *order* in which the variables are projected. If one is only interested in pure CAD, then this order can be chosen completely arbitrarily. For quantifier elimination there are restrictions imposed by projecting unquantified variables last and not interchanging $\exists$ with $\forall$. There is, however, still a considerable degree of freedom.

We are going to demonstrate by means of a small example that the projection order is highly relevant for the practical complexity of the overall procedure: We consider two circles of radius 2,

$$\begin{aligned} c_1 &= (x+3)^2 + (y+1)^2 - 4, \\ c_2 &= (x-3)^2 + (y-1)^2 - 4, \end{aligned}$$

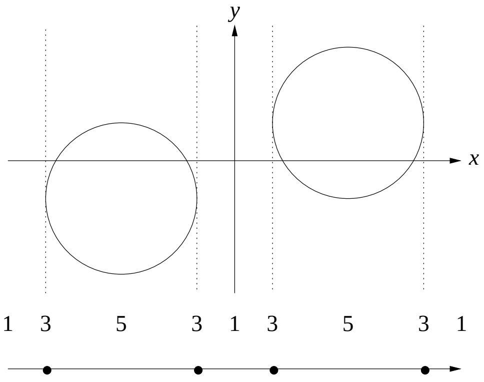


Figure 6.1: CAD wrt. the variable order $y \to x$.

one located at $(-3, -1)$ and the other one located at $(3, 1)$. Figure 6.1 shows a CAD for these circles choosing the projection order $y \to x$. This CAD contains $1+3+5+3+1+3+5+3+1 = 25$ cells.

Figure 6.2 shows, by the way of contrast, a corresponding CAD using the projection order $x \to y$. Note that the y-axis is drawn horizontally here. We obtain considerably more cells: $1+3+5+7+9+7+5+3+1 = 41$.

It is obvious that the computation of this second CAD requires more computational resources while delivering a result of equal quality for most purposes.

This chapter provides results for determining good variable orders from the input beforehand, i.e., without actually constructing any CAD, in order to then construct the desired CAD wrt. such an order.

The plan of the paper is as follows: In Section 6.2, we introduce time and space measures for characterizing the complexity of a particular CAD computation. These measures apply partly to the projection phase and partly to the overall computation. We discuss a comprehensive example set of CAD's wrt. all relevant orders and apply all our measures to all results. We then show on a precise formal basis that for this set of examples all our measures are statistically strongly correlated. In particular there are measures on the projection that are suitable for predicting the complexity of the overall computation.

In Section 6.4, we introduce a heuristic algorithm for efficiently constructing one good projection order wrt. to the relevant measures on the projection phase. This is done *without* trying all relevant projection orders. In fact, it cannot be avoided constructing several alternatives for the projection of each variable, but the number of such construction steps in our algorithm is only quadratic in contrast to exponential in the number of variables. We reuse our example database from Section 6.2 to show two facts: First, our heuristic algorithm yields projection orders that are statistically significantly close to optimal. Second, the overhead originating from constructing the

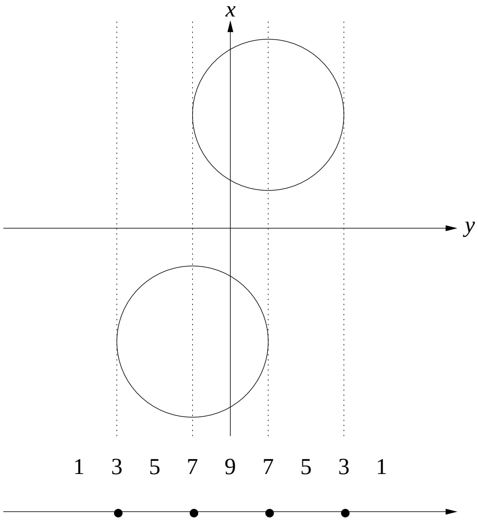


Figure 6.2: CAD wrt. the variable order $x \to y$.

good projection order is negligible while the gain is immense.

In Section 6.5 we conclude our contribution by summarizing and evaluating our results.

Section 6.6 finally contains an appendix listing the variable orders used for the computation of the example set in Section 6.2 such that these computations can be reproduced.

6.2 Measures on CAD Computations

In this section we consider the following situation: There is a set A of polynomials in r variables given, which possibly origin from a prenex formula φ. For a variable order $X = (x_1, \dots, x_r)$, which we also write $x_r \to \dots \to x_1$, the projection results in projection sets F_r, ..., F_1 of projection factors. Based on these sets, the CAD tree D is constructed. The levels D_1, ..., D_r of D are CAD's for $\mathbb{R}^1$, ..., $\mathbb{R}^r$, respectively. This situation is completely specified by (A, X) or (φ, X), respectively. We have developed our methods on the basis of the classical Collins–Hong projection. They are, however, also applicable to the other projection operators discussed in the literature.

Our goal is to find for given A or φ a favorable projection order X at the earliest possible stage. For this we want to be able to draw conclusions on the size of the CAD from properties of the intermediate projection sets.

Therefore, we are going to systematically investigate numerous complete CAD's wrt. all relevant projection orders, consider certain *measures* on the projection sets as well as on the CAD's, and examine statistical correlations between these measures.

6.2.1 Measures

There are six measures that we take into account:

1. The number of projection factors in all the projection sets $F_r, \ldots, F_1$:
$$\operatorname{card}(A, X) = \sum_{i=1}^{r} |F_i|.$$

2. The sum of total degrees of *all* monomials of *all* polynomials in *all* the projection sets $F_r, \ldots, F_1$:
$$\operatorname{sotd}(A, X) = \sum_{i=1}^{r} \sum_{f \in F_i} \sigma(f),$$
where, using the convention $\mathbf{e} = (e_1, \ldots, e_r)$,
$$\sigma\left(\sum_{\mathbf{e} \in E} a_{\mathbf{e}} x_1^{e_1} \cdots x_r^{e_r}\right) = \sum_{\mathbf{e} \in E} \sum_{i=1}^{r} e_i.$$

3. The number of cells in the resulting full CAD:
$$\operatorname{ncad}(A, X) = |D_r|.$$

4. The overall computation time of the full CAD computation in seconds:
$$\operatorname{tcad}(A, X).$$

5. The number of leaves in the partial CAD tree that is generated for quantifier elimination:
$$\operatorname{npcad}(\varphi, X) = \bigl|\bigl\{\, c \in D_k \cup \cdots \cup D_r \mid c \text{ is a leaf} \,\bigr\}\bigr|.$$

6. The overall computation time of the quantifier elimination by partial CAD in seconds:
$$\operatorname{tqe}(\varphi, X).$$

The time measures tcad and tqe depend on the implementation and the machine. The CAD implementation of the author in the REDLOG package [DS97b] of the widespread computer algebra system REDUCE was used. All computations were carried out on a 2.0 GHz Intel Pentium IV using 128 MB of RAM.

The first four measures card, sotd, ncad, and tcad are defined for sets A of polynomials. They can be as well applied to formulas φ by considering the set of polynomials occurring in these formulas. The last two measures npcad and tqe, in contrast, do not make sense outside a quantifier elimination context.

It might appear more natural to consider in the definition of our sotd the total degrees of the corresponding polynomials instead of sums of total degrees of monomials. Experiments have shown that these measures are highly correlated. Our choice has the advantage to favor sparse polynomials.

For the definition of npcad we shortly recall the basic idea of the partial CAD procedure for quantifier elimination [CH91]: When using full CAD, there is the CAD tree D with levels $D_1, \dots, D_r$ computed, and then the matrix formula is evaluated at the sample points of the cells in D_r. The truth values thus obtained are propagated down to the corresponding root cell in D_k according to the types of quantifiers. Hence full CAD computation and its use for quantifier elimination are two isolated subsequent steps. For partial CAD, in contrast, one tries to determine truth values for the matrix formulas *during extension* already for cells in D_1, …, D_{r-1}, i.e., without fixing all variables to real numbers. For instance, $x_1 - 42 > 0 \wedge x_2^3 - 4711x_3 = x_4$ is false for $x_1 = \sqrt{2}$, no matter what x_2, …, x_r are. Whenever one succeeds this way for a cell $c \in D_i$, where $i \in \{1, \dots, r\}$, this cell c need not be further extended. In other words, the partial CAD tree is pruned at this point, and c becomes a leaf.

It is now clear, that it is not reasonable to consider the time for the partial CAD construction as a measure: This construction is not isolated from the quantifier elimination but contains a considerable part of the quantifier elimination work, viz. hard computations with algebraic numbers for trial evaluation of the matrix formula. The other part of the quantifier elimination work, viz. solution formula construction, is, in contrast, still an isolated subsequent step. From that point of view, we consider tqe an appropriate counterpart for tcad.

The first two measures card and sotd can be applied after projection or even during projection on intermediate projection sets. They are candidates for suitable criteria for determining projection orders. The latter four measures ncad, tcad, npcad, and tqe, in contrast, can be applied only after a complete CAD computation or quantifier elimination, respectively. They are going to be used for evaluating the significance of our candidate criteria.

6.2.2 Computation of the Test Set

We are going to discuss a test set consisting of six CAD examples, mostly from the literature. Each example has been computed wrt. a significant number of projection orders. In fact, we have computed a much more comprehensive example set comprising 48 examples, and selected these six examples as a representative subset for this paper.

All our examples are in fact quantifier elimination examples, which allows us to apply all our measures. Note that also from the point of view of pure CAD, it is not at all a restriction to consider only such examples; they can be considered deliverers of interesting sets of polynomials.

Admissible Projection Orders

On the other hand, we consider the quantifier block structure of the examples in order to restrict the set of possible orders to a reasonable subset: Strictly speaking, quantifier elimination by CAD for a prenex formula

$$\varphi = Q_{k+1}x_{k+1}\dots Q_r x_r \psi$$

requires a projection order with two properties: First, all quantified variables x_r, …, x_{k+1} are projected before all unquantified variables x_k, …, x_1.

Second, the quantified variables have to be projected *essentially* in the order $x_r \to \dots \to x_{k+1}$. This requirement for a fixed projection order is weakened by the fact that in φ like neighbored quantifiers can be equivalently interchanged.

For instance, $\exists x \exists y \forall z \psi$ is equivalent to $\exists y \exists x \forall z \psi$ but *not* generally equivalent to $\exists x \forall z \exists y \psi$. Consequently $z \to y \to x$ and $z \to x \to y$ are both possible projection orders in this example, while $y \to z \to x$ is not. This observation suggests to rewrite $\exists \{x, y\} \forall \{z\} \psi$ thus making visible the *quantifier blocks*.

So returning to the general discussion, we can rewrite our prenex formula φ as

$$\varphi = Q_1 B_1 \dots Q_n B_n \psi,$$

where $Q_i \neq Q_{i+1}$ for $i \in \{1, \dots, n-1\}$ and B_1, …, B_n are finite sets of variables. This is the unique *block representation* of a prenex formula. For convenience, let B_0 denote the set of unquantified variables.

From that point of view, *admissible* projection orders for quantifier elimination are characterized by projecting

$$B_n \to \dots \to B_1 \to B_0,$$

while within each block B_i the order can be freely chosen. Obviously, the number of admissible projection orders is given by

$$\prod_{i=0}^{n} |B_i|!.$$

The Quartic Problem

The `quartic` problem has been suggested by Lazard [Laz88]. It asks for necessary and sufficient conditions on the coefficients of a quartic polynomial to be positive semidefinite:

$$\texttt{quartic} = \forall x (px^2 + qx + r + x^4 \geq 0).$$

There are $1! \cdot 3! = 6$ admissible orders. The following table presents the computation results:[1]

[1] For all our examples discussed thoughout this section, the actual variable orders used in each table row are collected in an appendix in Section 6.6.

	card	sotd	ncad	tcad	npcad	tqe
1	*7*	54	445	4.71	251	7.04
2	*7*	54	445	83.39	251	138.18
3	*7*	*50*	*417*	*0.54*	*235*	*0.89*
4	*7*	*50*	*417*	1.64	239	2.55
5	9	66	⊥	>600	⊥	>600
6	9	66	⊥	>600	⊥	>600

We see that card cannot predict differences in ncad, where sotd can. Note that ncad and tcad are surprisingly unrelated here. On the other hand, there is one order, viz. no. 3, that is optimal wrt. all criteria. We have automatically aborted all our computations after 10 minutes. Measures that are unknown due to such unfinished computations are marked with $\bot$.

A Real Implicitization Problem

This example is an exercise on complex implicitization in a textbook [CLO92]. Our formulation asks for a corresponding real implicitization:

$$\texttt{cls7} = \exists u \exists v (x = uv \land y = uv^2 \land z = u^2).$$

The number of admissible orders is $2! \cdot 3! = 12$.

	card	sotd	ncad	tcad	npcad	tqe
1	*9*	*25*	*889*	*0.09*	*266*	0.16
2	*9*	*25*	*889*	*0.09*	*266*	*0.15*
3	*9*	*25*	*889*	0.15	268	0.20
4	*9*	*25*	*889*	0.14	*266*	0.22
5	*9*	*25*	*889*	0.14	268	0.19
6	*9*	*25*	*889*	0.15	*266*	0.19
7	11	36	1571	0.19	508	0.28
8	11	36	1571	0.18	508	0.28
9	11	36	1571	0.17	582	0.29
10	11	36	1571	0.16	580	0.29
11	11	36	1571	0.17	582	0.29
12	11	36	1571	0.17	580	0.28

Compared to the previous example there is much less variation here. Note that a choice of projection order according to card or sotd yields significantly good ncad, tcad, npcad, and tqe.

Range of Lower Bounds

The following formula asks for the possible range of strict lower bounds on the values of a parabola that has no real zeros. The example comes from the context of [Sei01].

$$\begin{aligned}\texttt{as6} \quad = \quad & \forall x \forall a \forall b \forall c \exists x' \big((a > 0 \land a{x'}^2 + bx' + c \neq 0) \longrightarrow \\ & \quad y < ax^2 + bx + c \big).\end{aligned}$$

There are $1! \cdot 4! \cdot 1! = 24$ admissible orders:

	card	sotd	ncad	tcad	npcad	tqe
1	11	42	4199	0.39	283	0.07
2	11	42	4199	0.48	307	0.09
3	12	44	5231	0.55	487	0.15
4	12	44	5231	0.55	487	0.14
5	13	53	6389	1.37	341	0.13
6	12	49	6389	0.38	357	0.09
7	11	50	4007	0.44	*241*	*0.06*
8	11	50	4007	0.55	255	0.08
9	12	50	5027	0.62	395	0.12
10	12	50	5027	0.62	395	0.11
11	12	50	5027	0.58	305	0.10
12	12	50	5027	0.59	321	0.10
13	12	43	5007	0.46	523	0.18
14	12	43	5007	0.40	523	0.17
15	11	39	3975	0.38	423	0.16
16	11	39	3975	0.40	423	0.15
17	11	39	3975	0.28	415	0.16
18	11	39	3975	0.29	415	0.14
19	11	36	3709	0.52	348	0.16
20	11	36	3709	0.21	358	0.11
21	*9*	*28*	*2365*	0.27	272	0.11
22	*9*	*28*	*2365*	0.27	288	0.11
23	*9*	*28*	*2365*	*0.13*	290	0.08
24	*9*	*28*	*2365*	0.14	290	0.08

This is another example with little variation. Here card and sotd do not discover optimal orders wrt. npcad or tqe. The orders that they point at are, however, absolutely acceptable.

Consistency in Strict Inequalities

This problem decides whether the intersection of the open ball with radius 1 centered at the origin and the open cylinder with radius 1 and axis the line $x = 0$, $y + 2 = 2$

is nonempty. It has been introduced by McCallum and used by Collins and Hong for demonstrating PCAD [McC87, CH91]:

$$\texttt{con} = \exists z \exists x \exists y \big(x^2 + y^2 + z^2 < 1 \wedge x^2 + (y + z - 2)^2 < 1\big).$$

There are 3! = 6 admissible orders:

	card	sotd	ncad	tcad	npcad	tqe
1	12	46	251	*0.02*	51	0.01
2	*8*	43	365	2.24	43	0.04
3	11	*33*	*193*	*0.02*	*29*	0.01
4	11	*33*	*193*	*0.02*	37	<*0.01*
5	*8*	43	365	2.90	47	0.07
6	12	46	251	*0.02*	51	0.01

Comparing the lines 2 and 5 with 3 and 4, we observe that card and sotd contradict each other. Following sotd in these cases yields the best values for ncad, tcad, npcad, and tqe.

Parametrized Collision Problem

The following formula asks if two moving objects, a circle and a square, are going to collide at some time t in the future. The circle is moving with constant velocity $(1, 0)$, while the velocity of the square is parameterized with (v_x, v_y). This example has been used by Collins and Hong for several fixed choices of (v_x, v_y) [CH91].

$$\begin{aligned}\texttt{pcol} \quad = \quad & \exists t \exists x \exists y (t > 0 \wedge -1 \leq x - v_x t \leq 1 \wedge \\ & \qquad -9 \leq y - v_y t \leq -7 \wedge (x - t)^2 + y^2 \leq 1).\end{aligned}$$

The number of admissible orders is $3! \cdot 2! = 12$:

	card	sotd	ncad	tcad	npcad	tqe
1	62	250	*144971*	*47.86*	*6969*	*3.78*
2	62	250	*144971*	121.13	*6969*	4.91
3	⊥	⊥	⊥	>600	⊥	>600
4	4678	227337	⊥	>600	⊥	>600
5	62	*248*	149925	58.95	13310	4.53
6	62	*248*	149925	151.50	13310	7.51
7	*57*	323	⊥	>600	⊥	>600
8	*57*	323	⊥	>600	⊥	>600
9	⊥	⊥	⊥	>600	⊥	>600
10	⊥	⊥	⊥	>600	⊥	>600
11	⊥	⊥	⊥	>600	⊥	>600
12	⊥	⊥	⊥	>600	⊥	>600

In this example we observe an immense variety in all measures. In particular, the orders in the lines 3, 9–12 do not even allows to finish projection within the time limit. The probability of failing to finish full as well as partial CAD computation for a random order is 2/3. Minimal card misleadingly points at such failing orders. Minimal sotd does not point at the optimal order but still at acceptable ones.

The X-Axis Ellipse Problem

The X-Axis Ellipse Problem has been firstly stated by Kahan [Kah75]. It has been formulated as a quantifier elimination problem by Lazard [Laz88]. The problem is to write down conditions such that the ellipse

$$\frac{(x-c)^2}{a^2} + \frac{(y-d)^2}{b^2} = 1$$

is inside the circle $x^2 + y^2 = 1$. We treat the special case $d = 0$:

$$\texttt{ell} = \forall x \forall y \big(b^2(x-c)^2 + a^2y^2 = a^2b^2 \longrightarrow x^2 + y^2 \leq 1\big).$$

There are $2! \cdot 3! = 12$ admissible orders:

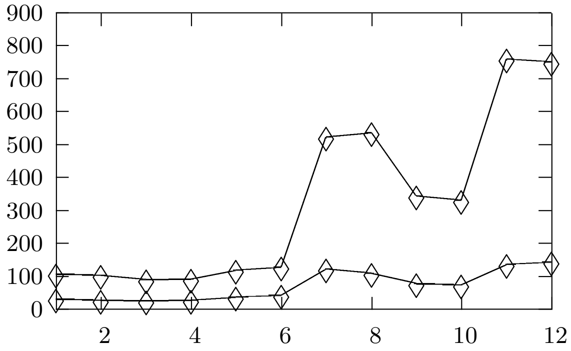
	card	sotd	ncad	tcad	npcad	tqe
1	32	107	114541	46.64	37883	29.68
2	28	103	*51477*	45.54	*11635*	28.70
3	*26*	*89*	64625	*17.10*	21059	*15.59*
4	27	91	96833	29.71	20431	17.70
5	36	119	⊥	>600	74587	260.70
6	43	129	⊥	>600	⊥	>600
7	122	522	⊥	>600	⊥	>600
8	109	537	⊥	>600	⊥	>600
9	77	345	⊥	>600	⊥	>600
10	74	331	⊥	>600	⊥	>600
11	136	761	⊥	>600	⊥	>600
12	143	751	⊥	>600	⊥	>600

Similar to the previous example, there is probability of only 1/3 to finish full CAD and only a slightly higher probability to finish partial CAD for a random order. Both minimal card and sotd point at an optimal order wrt. tcad, npcad, and tqe and this order is almost optimal wrt. ncad.

6.2.3 Statistical Correlations

In the informal remarks after presenting for each of our examples the resulting table, we have collected some positive and negative observations concerning the possible correlations between our measures.

We are now going to systematically examine the correlations on a precise formal basis. To motivate this, consider the x-axis ellipse problem from Section 6.2.2. For the 12 admissible projection orders the card values (lower line) and the corresponding sotd values (upper line) differ pretty much as can be seen from the following interpolated plot:



Nevertheless, it is our impression that there is a correlation between these measures. To substantiate this, we introduce a formal notion of correspondence: Consider lists $p = (p_1, \ldots, p_l)$ and $q = (q_1, \ldots, q_l)$ in $\mathbb{R} \cup \{\bot\}$ like the card column and the sotd column for the ellipse example. The value $\bot$ is used to encode that the corresponding value is unknown. Define

$$I = \bigl\{\, \{i, j\} \bigm| 1 \leq i < j \leq l \text{ and } p_i,\, p_j,\, q_i,\, q_j \in \mathbb{R} \,\bigr\};$$

this is the set of all unordered pairs of different indices (orders) for which all values are known. On this basis, we define

$$C = \bigl\{\, \{i, j\} \in I \bigm| \operatorname{sign}(p_i - p_j) = \operatorname{sign}(q_i - q_j) \,\bigr\},$$

the subset of all pairs of indices where the corresponding values for p are ordered in exactly the same way as the ones for q. Finally, we define

$$C' = \bigl\{\, \{i, j\} \in I \bigm| |\operatorname{sign}(p_i - p_j)| \neq |\operatorname{sign}(q_i - q_j)| \,\bigr\},$$

where the corresponding orders between the values are at least not completely opposite. In these definitions, $\operatorname{sign}(x)$ is 1, 0, or -1 if x is positive, zero, or negative, respectively, and $|\cdot|$ denotes the usual absolute value.

If we observe $C = I$, then this would suggest very good correspondence. If $\{i, j\} \notin C$ for some $\{i, j\} \in I$, then we would consider it good correspondence to at least have $\{i, j\} \in C'$. This gives rise to the following definition of the *degree of correspondence* between p and q. It is defined for $|I| > 0$:

$$\operatorname{doc}(p, q) = \frac{|C| + 0.5|C'|}{|I|} \in [0, 1].$$

Note that doc is commutative.

It is not hard to see that for fixed list length l, the average over the degrees of correspondences for all possible choices of pairs of lists is 0.5. A value greater than 0.5 thus indicates an above-average correspondence. For our motivating ellipse example above, we obtain, e.g., $\mathrm{doc}(\mathrm{card}, \mathrm{sotd}) = 0.97$.

The following table collects for all our examples the doc values for the relevant combinations of card, sotd, ncad, tcad, npcad, and tqe. For convenience, the doc's are multiplied by 100 thus rescaling them to percentages:

	quartic	cls7	as6	con	pcol	ell	∅
card–ncad	67	100	89	46	67	67	72
sotd–ncad	100	100	88	74	33	67	77
card–tcad	50	81	78	30	50	100	64
sotd–tcad	83	81	85	56	33	100	73
card–npcad	58	84	67	67	67	70	68
sotd–npcad	91	84	54	93	33	70	70
card–tqe	50	82	62	33	50	100	62
sotd–tqe	83	82	47	60	33	100	67

The last column gives the averages over the corresponding lines. These averages indicate that sotd has a significantly higher doc with all measures on the complete computation than card.

We thus consider sotd in contrast to card to be a suitable indicator after projection for the time and space to be expected for the overall computation.

6.3 Guessing the Size of a Full CAD

We want to dedicate this section to a special measure "guess" that lies between the indicator measures card and sotd on the one side and the target measures on the other side.

Modern implementations construct the CAD tree in a dept-first manner, as for QE purposes there is hope that a partial construction suffices. For a full CAD, it doesn't hurt either to proceed this way. Thus it can be expected that the first leaf is found within a very small fraction of the overall computing time.

During the process of constructing the first leaf, the number n_i of cells of the stack over the corresponding cell of level $i-1$ is either known, or can be found out easily, e.g. by evaluating Sturm sequences at $-\infty$ and ∞. Then the guessed value of the CAD problem is $\prod_{i=1}^{r} n_i$. This defines the measure guess for a CAD problem.

Let us look at the verbose output of the author's implementation for the example `quartic` with projection order $y \to r \to q \to p$.

```
(0:3(1:11(2:9(3:5(4)(4)(4)_2)...)...)...)
```

This gives us the following information: Starting decomposition at level 0 there are 3 cells on level 1. We select one cell C_1 and procede to level 2 where there are 11 cells in the stack over C_1. Again we select a cell C_2 out of these 11 cells. The stack over C_2 turns out to have 9 cells. And again we select a cell C_3, which turns out to have a stack with 5 cells. Now we have reached real 4-space for the first time. We stop now and guess the number of cells of a CAD of real 4-space to be $3 \cdot 11 \cdot 9 \cdot 5 = 1485$. We note some observations:

1. The guessed and precise number of cells turned out to differ by a factor between 0.25 and 4 for a set of examples similar to the test set of Section 6.2. E.g. for the example `quartic` factors range between 1.2 and 3.3. Thus guess can be expected to reveal the order of magnitude of a problem.

2. No algebraic number arithmetic is needed to implement this algorithm. As on each level only one cell is needed, it suffices to choose one with rational sample point.

3. The resulting number highly depends on the cell selection strategy employed during decomposition. The author's implementation uses the cell which is first found by incremental root isolation. Other strategies are described in [CH91].

4. The size of a partial CAD, which occurs for a CADQE problem, can be unpredictably smaller than the size of a full CAD. Thus the guessed size is only an upper bound, if used in context of a CADQE problem.

The measure guess was considered as a possible choice for an indicator measure, but ruled out for two reasons. The main reason is that the quality of predictions of guess turned out worse than sotd (but better than card). In addition it seemed not to be too promising to apply CAD constructions to intermediate projection sets, both in time consumption and quality of results. So the decision of the last section to choose sotd as the most suitable indicator measure is not affected.

Nevertheless, guess can be put to good use. There is the additional possibility to take the time needed to find the first leaf and multiply it by the guessed number of cells of a full CAD. This would allow a computer algebra system to tell the user the order of magnitude of time it guesses to compute the problem. Giving the user an idea about the rough space and time requirements of his problem has several benefits.

1. The user gets an idea how large or small a problem is. She can interactively experiment with several alternative formulations, and use the one which promises to be the fastest. This would improve the user experience considerably.

2. CAD problems can be classified by the expected order of magnitude of amount of time: milliseconds, centiseconds, seconds, minutes, hours, days, weeks, months, years, decades, centuries, millennia, or aeons.

3. In particular for problems, for which a full decomposition is illusionary, this approach allows for an educated guess. E.g. for example `ell` with order 11 we obtain a guess of approx. $13,000,000$ cells.

6.4 Constructing Good Orders

We recall from the Section 6.2 that $\operatorname{sotd}(A, X)$ for a set A of polynomials and a projection order X is the sum of the total degrees of all monomials in all polynomials in all projection sets $A_r, \dots, A_1$. For a prenex first-order formula φ we may also speak of $\operatorname{sotd}(\varphi, X)$ referring to the set of polynomials occurring in φ.

The results of the previous section provide a good indication that sotd is a suitable measure that is correlated with a high degree of correspondence to all measures that one possibly wishes to optimize in order to save computational resources:

1. small size of the full CAD (ncad),
2. fast computation time for the full CAD (tcad),
3. small size of the partial CAD (npcad),
4. fast computation time for quantifier elimination (tqe).

On the basis of this result we can conclude from the knowledge of *all* projection sets for *all* admissible orders on the interesting time and space measures listed above without actually performing any base phase or extension phase.

The remaining problem is the following: In order to get a basis for the decision, there are still projection phases wrt. *all* admissible orders to be performed. The worst-case number of admissible orders is the factorial of the number of variables and hence exponential in the word length of the input.

In this section we are going to suggest a heuristic algorithm for finding a good projection order wrt. sotd. This algorithm will require quadratically many projection steps in the number of variables and thus in the word length.

We are going to evaluate the quality of the orders determined by our algorithm on the basis of our example set introduced in the previous section. It is going to turn out that the practical computation times are absolutely negligible, while the gain obtained from using the computed orders is immense.

6.4.1 Greedy Projection

We suggest a *greedy* algorithm for finding a good admissible projection order wrt. sotd. By greedy, we refer to roughly the following idea: We perform the first projection step wrt. to all possible variables. Then we determine the sum of total degrees for each single obtained set. We greedily take the best one, throw away all others, and repeat like that until there are no variables left to order.

For formulas φ, we have to recall our discussion of admissible orders in Section 6.2.2: For a formula

$$\varphi = Q_1 B_1 \dots Q_n B_n \psi$$

in prenex block representation, the admissible projection orders are characterized by projecting the blocks in the order

$$B_n \to \dots \to B_1 \to B_0,$$

where B_0 denotes the set of all unquantified variables. For each of these blocks, the order can be freely chosen. The first block to consider is B_n.

We give the algorithm with two subroutines that build on each other. The main work is done in PORDER2. Here the order of one variable block is decided. Bases on this, PORDER1 finds variable orders for all blocks, starting with the innermost block B_n. Using this routine, depending on the application, the variable order for a formula or a set of polynomials can be found.

6.1 Algorithm (greedy projection, 2nd subroutine)

$$(A', \omega) \longleftarrow \text{PORDER2}(A, (B_0, \dots, B_m))$$

Input: A finite set $A \subset \mathbb{R}[x_1, \dots, x_j]$ and pairwise disjoint sets of variables $B_0, \dots, B_m$ with $\bigcup_{i=0}^{m} B_i = \{x_1, \dots, x_j\}$.
Output: A projection order ω on B_m and the corresponding intermediate cumulative projection set A'.

```
1. $\omega := ()$
2. while $B_m \neq \emptyset$ do
   (a) $A' := \bot$
   (b) for each $x \in B_m$ do
        i. $A'' := \text{project}\big(\{f \in A \,|\, \text{for all } y \in \omega : \deg_y(f) \leq 0\}, x\big)$
       ii. $s'' := \sum_{f \in A''} \sigma(f)$
      iii. if $A' = \bot$ or $s'' < s'$ then
               $A' := A'' \cup A$
               $s' := s''$
               $x' := x$
   (c) $\omega := (x') \circ \omega$
   (d) $A := A'$
   (e) $B_m := B_m \setminus \{x'\}$
3. Return $(A', \omega)$.
```

There are some remarks to this algorithm.

1. In Step 2.(b).i, project(M, v) denotes one projection step on the set M wrt. the variable v. This includes computing irreducible factors. Depending on the projection operator it might not suffice to just provide x. In this case a preliminary order $\omega' \circ (x)$ has to be provided, where ω' is an arbitrary order on $(\bigcup_{i=0}^{m} B_i) \setminus (\omega \cup \{x\})$.

2. In Step 2.(b).ii, recall the definition of σ from Section 6.2.1, and note that sums over the empty set are zero.

3. Note that the first time Step 2.(b).iii is encountered, s' is undefined, but this does no harm, as $A' = \bot$ is true. Thus by means of lazy evaluation $s'' < s'$ needs not to be evaluated. Alternatively, after Step 2.(a), $s' := \infty$ could be added.

Based on Algorithm PORDER2 one can straightforwardly define

6.2 Algorithm (greedy projection, 1st subroutine)

$$(A', (\omega_0, \dots, \omega_n)) \longleftarrow \text{PORDER1}(A, (B_0, \dots, B_n))$$

Input: A finite set $A \subset \mathbb{R}[x_1, \dots, x_r]$ and pairwise disjoint sets of variables $B_0, \dots, B_n$ with $\bigcup_{i=0}^{n} B_i = \{x_1, \dots, x_r\}$.
Output: Projection orders ω_i on B_i and the corresponding cumulative projection set A'.

```
1. for $i$ from $n$ downto 0 do
   (a) $(A', \omega_i) := \text{PORDER2}(A, (B_0, \dots, B_i))$
   (b) $A_i := \{f \in A' \mid \text{exists } y \in \omega_i : \deg_y(f) \geq 1\}$
   (c) $A := A' \setminus A_i$
2. Return $(\bigcup_{i=0}^{n} A_i, (\omega_0, \dots, \omega_n))$.
```

With this it is finally very clear how to write the top-level algorithm.

6.3 Algorithm (greedy projection)

$$\omega \longleftarrow \text{PORDER}(\varphi)$$

Input: A prenex formula φ.
Output: An efficient projection order for φ.

```
1. Let $A$ denote the irreducible factors of the polynomials of $\varphi$.
2. Identify the variable blocks $(B_0, \dots, B_n)$ of $\varphi$.
3. Return $\omega$, where $(A', \omega) := \text{PORDER1}(A, (B_0, \dots, B_n))$
```

If one is not interested in quantifier elimination, but only CAD, then for a sets A of polynomials in variables B one can find an efficient projection order by the call $\mathrm{PORDER1(A, (B))}$.

Note that the algorithm, does not necessarily find a projection order yielding really minimal sotd. The main reason is that we apply a local criterion on projection levels, while sotd is a global criterion on entire projections. In other words, the projection order with smallest sotd need not necessarily have small sums of total degrees in the set obtained after the first projection step.

Nevertheless, we shall see that our greedy algorithm applied to the previous set of examples provides orders of very high quality in the following sense: First, they are as a rule close to optimal. We are going to substantiate this in the following section by means of a statistical analysis. Second, and even more important, the computed orders never exceed the time limit in any of our examples, even when there is a high probability of failing.

6.4.2 CAD Performance with Greedy Projection

The following tables provide a statistical analysis of the performance of the orders generated by our greedy algorithm. It shows that the performance of these orders is considerably above-average. The extra computing time is negligible.

	`quartic`	`cls7`	`as6`
order no. by greedy	3	1	22
time for greedy	0.01	<0.01	0.01
ncad by greedy	417	889	2365
rank w/i ncad	1 of 6	1 of 12	1 of 24
median of ncad	445.00	1230.00	4103.00
mean of ncad	⊥	1230.00	4273.00
tcad by greedy	0.54	0.09	0.27
rank w/i tcad	1 of 6	1 of 12	4 of 24
median of tcad	44.05	0.16	0.42
mean of tcad	>215.04	0.15	0.45
npcad by greedy	235	266	288
rank w/i npcad	1 of 6	1 of 12	5 of 24
median of npcad	251.00	388.00	352.50
mean of npcad	⊥	411.67	364.25
tqe by greedy	0.89	0.16	0.11
rank w/i tqe	1 of 6	2 of 12	10 of 24
median of tqe	72.61	0.25	0.11
mean of tqe	>224.78	0.24	0.11

	con	pcol	ell
order no. by greedy	3	5	3
time for greedy	<0.01	0.13	0.01
ncad by greedy	193	149925	64625
rank w/i ncad	1 of 6	3 of 12	2 of 12
median of ncad	251.00	⊥	⊥
mean of ncad	269.67	⊥	⊥
tcad by greedy	0.02	58.95	17.10
rank w/i tcad	1 of 6	2 of 12	1 of 12
median of tcad	0.02	⊥	⊥
mean of tcad	0.87	>431.62	>411.59
npcad by greedy	29	13310	21059
rank w/i npcad	1 of 6	3 of 12	3 of 12
median of npcad	45.00	⊥	⊥
mean of npcad	43.00	⊥	⊥
tqe by greedy	0.01	4.53	15.59
rank w/i tqe	2 of 6	2 of 12	1 of 12
median of tqe	0.01	⊥	⊥
mean of tqe	0.02	>401.72	>379.36

6.5 Conclusions

We have obtained strong statistical evidence that the projection order is of crucial importance for the success of CAD computations. For determining the quality of a given projection order, we have shown that there is one single measure, viz. sotd, on the projection sets that is correlated to all interesting time and space measures on the computation of a full CAD as well as on quantifier elimination by partial CAD. We have introduced a greedy algorithm for efficiently constructing a good projection order wrt. sotd. This algorithm is already utilized by others [BPB05, Phi05]. In addition, a method to guess the order of magnitude of the space and time requirements of a CAD problem is devised. This work closes a considerable gap within the CAD framework.

6.6 Appendix: Catalogue of Orders

We finally list for our example set computed in Section 6.2.2 the projection orders used there. The numbering here corresponds to the that of the table rows in Section 6.2.2:
`quartic`: 1. $x \to r \to q \to p$, 2. $x \to r \to p \to q$, 3. $x \to q \to r \to p$, 4. $x \to q \to p \to r$, 5. $x \to p \to r \to q$, 6. $x \to p \to q \to r$.
`cls7`: 1. $v \to u \to z \to y \to x$, 2. $v \to u \to z \to x \to y$, 3. $v \to u \to y \to z \to x$, 4. $v \to u \to y \to x \to z$, 5. $v \to u \to x \to z \to y$, 6. $v \to u \to x \to y \to z$, 7. $u \to v \to z \to y \to x$, 8. $u \to v \to z \to x \to y$, 9. $u \to v \to y \to z \to x$,

10. $u \to v \to y \to x \to z$, 11. $u \to v \to x \to z \to y$, 12. $u \to v \to x \to y \to z$.
`as6`: 1. $x' \to c \to b \to a \to x \to y$, 2. $x' \to c \to b \to x \to a \to y$,
3. $x' \to c \to a \to b \to x \to y$, 4. $x' \to c \to a \to x \to b \to y$,
5. $x' \to c \to x \to b \to a \to y$, 6. $x' \to c \to x \to a \to b \to y$,
7. $x' \to b \to c \to a \to x \to y$, 8. $x' \to b \to c \to x \to a \to y$,
9. $x' \to b \to a \to c \to x \to y$, 10. $x' \to b \to a \to x \to c \to y$,
11. $x' \to b \to x \to c \to a \to y$, 12. $x' \to b \to x \to a \to c \to y$,
13. $x' \to a \to c \to b \to x \to y$, 14. $x' \to a \to c \to x \to b \to y$,
15. $x' \to a \to b \to c \to x \to y$, 16. $x' \to a \to b \to x \to c \to y$,
17. $x' \to a \to x \to c \to b \to y$, 18. $x' \to a \to x \to b \to c \to y$,
19. $x' \to x \to c \to b \to a \to y$, 20. $x' \to x \to c \to a \to b \to y$,
21. $x' \to x \to b \to c \to a \to y$, 22. $x' \to x \to b \to a \to c \to y$,
23. $x' \to x \to a \to c \to b \to y$, 24. $x' \to x \to a \to b \to c \to y$.
`con`: 1. $y \to x \to z$, 2. $y \to z \to x$, 3. $x \to y \to z$, 4. $x \to z \to y$, 5. $z \to y \to x$,
6. $z \to x \to y$.
`pcol`: 1. $y \to x \to t \to v_y \to v_x$, 2. $y \to x \to t \to v_x \to v_y$, 3. $y \to t \to x \to v_y \to v_x$,
4. $y \to t \to x \to v_x \to v_y$, 5. $x \to y \to t \to v_y \to v_x$, 6. $x \to y \to t \to v_x \to v_y$,
7. $x \to t \to y \to v_y \to v_x$, 8. $x \to t \to y \to v_x \to v_y$, 9. $t \to y \to x \to v_y \to v_x$,
10. $t \to y \to x \to v_x \to v_y$, 11. $t \to x \to y \to v_y \to v_x$, 12. $t \to x \to y \to v_x \to v_y$.
`ell`: 1. $y \to x \to c \to b \to a$, 2. $y \to x \to c \to a \to b$, 3. $y \to x \to b \to c \to a$,
4. $y \to x \to b \to a \to c$, 5. $y \to x \to a \to c \to b$, 6. $y \to x \to a \to b \to c$,
7. $x \to y \to c \to b \to a$, 8. $x \to y \to c \to a \to b$, 9. $x \to y \to b \to c \to a$,
10. $x \to y \to b \to a \to c$, 11. $x \to y \to a \to c \to b$, 12. $x \to y \to a \to b \to c$.

Chapter 7

Regularization

In this chapter we study the effect of a linear transformation of variables on sets of polynomials and its application to the CAD method for quantifier elimination. The plan of this chapter is as follows:

1. We give syntactic and geometric reasons to motivate that a linear change of variables can lead to CAD with fewer cells.

2. We define the notion of regularity, give a transformation, investigate its effects and show how a set of polynomials can be transformed into a regular one by means of this transformation.

3. We show how this transformation can be applied to formulas in the preparation phase of the CAD algorithm to rewrite the input formula to a regular formula.

4. We show that it is often unnecessary to regularize wrt. all variables and show how to find an efficient subset of variables. We sketch a way to find such a set faster.

5. We look at examples. In particular, examples with few or without free variables and with large blocks of like quantifiers are applicable.

7.1 Introduction

On input of a first-order formula φ, the CAD method for real QE starts off by extracting the polynomials of the formula as set A. Then, during projection phase, the irreducible factors of A are extended to the projection set F. The algorithm we propose rewrites the input formula to an equivalent one, such that some or many polynomials of this formula have a constant leading coefficient. We can expect two benefits from this approach.

First, from a syntactical point of view, constant leading coefficients are desirable, as this leads to smaller projection sets with Collins-Hong style projection operators. By regularization we can always transform for a given variable a polynomial into a polynomial with constant leading coefficient wrt. this variable. We expect smaller projection sets, and therefore benefits for the CAD algorithm.

Second, there is a geometric motivation: As an additional effect, the intended transformations are capable of yielding a situation, which is more suitable for CAD. Consider the set $A_1 := \{4y^2 + 4x^2 - 12x + 5, 4y^2 - 12y + 5 + 4x^2\}$ of input polynomials. Independent of the choice $y \rightarrow x$ or $x \rightarrow y$ of the variable order this yields a CAD with 41 cells. If we, however, apply a sheering $x \mapsto x + y$, then we get the set $A_2 := \{8y^2 + 4x^2 + 8xy - 12x - 12y + 5, 8y^2 - 12y + 5 + 4x^2 + 8xy\}$ of transformed polynomials, and for the variable order $y \rightarrow x$ we get a CAD of only 25 cells. This is illustrated in Figure 7.1. This example indicates that this approach allows us to have *tilted* cylinders.

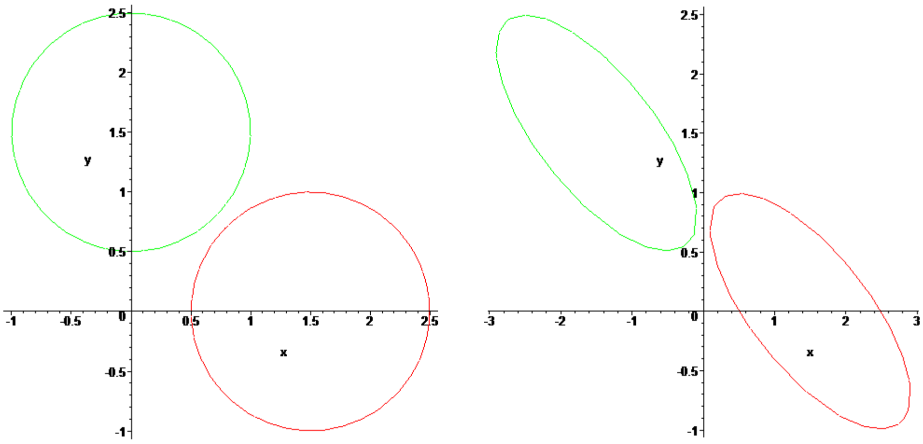


Figure 7.1: Left: The two circles yield a CAD of 1+3+5+7+9+7+5+3+1 cells. Right: After a sheering in x-direction, we end up with 1+3+5+3+1+3+5+3+1 cells.

7.2 Regularity

We first need to introduce the notion of regularity. This notion is derived from the study of power series and related to the Weierstrass Vorbereitungssatz, see [GR65, Wei86]. In this chapter, D always denotes a domain.

7.2.1 Two Flavors of Regularity

Let f be a polynomial in $D[y_1, \ldots, y_m]$. The polynomial f is called *regular in* y_m *wrt.* $y_1, \ldots, y_{m-1}$, if the leading coefficient of f wrt. y_m is a constant polynomial in

$y_1, \ldots, y_{m-1}$, i.e. if

$$\mathrm{tdeg}_{(y_1,\ldots,y_{m-1})}(\mathrm{lc}_{y_m}(f)) \leq 0.$$

It is called *strongly regular in* y_m *wrt.* $y_1, \ldots, y_{m-1}$, if it is regular in y_m wrt. $y_1, \ldots, y_{m-1}$ and in addition the total degree of f wrt. $y_1, \ldots, y_m$ equals the degree of f wrt. y_m, i.e. if in addition

$$\mathrm{tdeg}_{(y_1,\ldots,y_m)}(f) = \deg_{y_m}(f).$$

With these definitions, regularity and strong regularity are ternary predicates, depending on a polynomial, a variable and a list of variables. As a notational convention, for our convenience and with hindsight of implementation, we collect the second and third argument into one list. Thus we say more shortly f is (strongly) regular wrt. $(y_1, \ldots, y_m)$ instead of f is (strongly) regular in y_m wrt. $(y_1, \ldots, y_{m-1})$.

7.2.2 A Transformation for Regularization

In this section we define a linear transformation τ of variables.

7.1 Definition (τ)
For a variable list $(\mathbf{y}) := (y_1, \ldots, y_m)$ and a list of natural numbers $(\mathbf{c}) := (c_1, \ldots, c_{m-1})$, define τ to be the unique homomorphism of rings

$$\tau_{(\mathbf{y}),(\mathbf{c})} : D[\mathbf{y}] \to D[\mathbf{y}],$$

with $\tau_{(\mathbf{y}),(\mathbf{c})}|_D = \mathrm{id}_D$ and with

$$\tau_{(\mathbf{y}),(\mathbf{c})}(y_j) = \begin{cases} y_m, & \text{if } j = m, \\ y_j + c_j y_m, & \text{otherwise.} \end{cases}$$

As a notational convenience, we simply write $\tau_{\mathbf{y},\mathbf{c}}$ instead of $\tau_{(\mathbf{y}),(\mathbf{c})}$. This is save, as a sequence of $2m-1$ objects can always be uniquely split into the first m and the latter $m-1$ objects.

7.2 Remark

1. $\tau_{(\mathbf{y}),(\mathbf{c})}$ is an automorphism. To prove this it remains to show that $\tau_{\mathbf{y},\mathbf{c}}$ is a bijection. Define

   $$\tau_{\mathbf{y},\mathbf{c}}^{-1} := \tau_{\mathbf{y},(-c_1,\ldots,-c_{m-1})}.$$

   For all $f \in D[y_1, \ldots, y_m]$ we have $\tau^{-1}(\tau(f)) = f$, hence τ is injective. Furthermore, for $f \in D[y_1, \ldots, y_m]$ we have $\tau(\tau^{-1}(f)) = f$ as well, hence f is surjective.

2. Using the homomorphism property from 7.1, we know that for a given polynomial f we have:

   $$(\tau_{\mathbf{y},\mathbf{c}}(f))(\mathbf{y}) = f(y_1, y_2 + c_1 y_1, \ldots, y_m + c_m y_1).$$

7.2.3 Properties of this Transformation

In order to get a clearer idea what τ does and why it is suitable for our purposes we at first assume a slightly more general setting. Define τ' similar to τ, but use variables $\mathbf{x} = (x_1, \ldots, x_{m-1})$, which are distinct to $\mathbf{y} = (y_1, \ldots, y_m)$, instead of constants $c_2, \ldots, c_m$. This makes $\tau'_{\mathbf{y},\mathbf{x}}$ a map

$$D[y_1, \ldots, y_m] \to D[x_1, \ldots, x_{m-1}, y_1, \ldots, y_m].$$

7.3 Lemma (action of τ' on a monomial)
For $\mathbf{e} := (e_1, \ldots, e_m) \in \mathbb{N}^m$, variables $(\mathbf{x}) = (x_1, \ldots, x_{m-1})$ distinct from variables $(\mathbf{y}) = (y_1, \ldots, y_m)$, $d \in D \setminus \{0\}$, a monomial $t := d \prod_{j=1}^{m} y_j^{e_j}$, and $f' := \tau'_{\mathbf{y},\mathbf{x}}(t) \in D[\mathbf{x}, \mathbf{y}]$ we have:

1. *The number of monomials of f' wrt. $(\mathbf{y})$ is $\prod_{j=1}^{m-1}(1 + e_j)$.*
2. *Each monomial t' in f' has $\operatorname{tdeg}_{\mathbf{y}}(t') = \sum_{j=1}^{m} e_j = \operatorname{tdeg}_{\mathbf{y}}(t)$.*
3. *There is exactly one monomial t' in f' with $\deg_{y_m}(t') = \sum_{j=1}^{m} e_j$, i.e. f' is strongly regular wrt. $\mathbf{y}$. In addition, $lc_{y_m}(t') = c \prod_{j=1}^{m-1} x_j^{e_j}$ for a domain element c, i.e. $\operatorname{tdeg}_{\mathbf{x}}(t') = \sum_{j=1}^{m-1} e_j$.*

Proof. Note that the product sign is used in the following for both, the set-theoretical cartesian product and the arithmetical product. As the object that follows the product sign is either a set or an arithmetic expression the meaning is always clear. Note further, that the assumption $d \neq 0$ implies $t \neq 0$.

$$\begin{aligned}
\tau'(t) &= dy_m^{e_1} \prod_{j=1}^{m-1} (y_j + x_j y_m)^{e_j} \\
&= dy_m^{e_m} \prod_{j=1}^{m-1} \sum_{k=0}^{e_j} \binom{e_j}{k} y_j^k (x_j y_m)^{e_j - k} \\
&= \sum_{f \in \prod_{i=1}^{m-1}\{0,\ldots,e_i\}} dy_m^{e_m} \prod_{j=1}^{m-1} \binom{e_j}{f_j} y_j^{f_j} (x_j y_m)^{e_j - f_j}
\end{aligned}$$

Set $F := \prod_{i=1}^{m-1}\{0, \ldots, e_i\}$ and for $f \in F$ set $s_f := dy_m^{e_m} \prod_{j=1}^{m-1} \binom{e_j}{f_j} y_j^{f_j} (x_j y_m)^{e_j - f_j}$. Let $f, g \in F$ with $f \neq g$. Choose $2 \leq i \leq m$ with $f_i \neq g_i$. Then

$$\deg_{y_i}(s_f) = f_i \neq g_i = \deg_{y_i}(s_g),$$

i.e. $s_f \neq s_g$. Thus $\tau'(t)$ results in $|F| = \prod_{j=1}^{m-1}(1 + e_j)$ monomials. This shows (1). The second claim follows immediately from

$$\operatorname{tdeg}_{\mathbf{y}}(s_f) = e_m + \sum_{j=1}^{m-1} (f_j + (e_j - f_j)) = \sum_{j=1}^{m} e_j.$$

To show (3), consider the zero map $\mathbf{0} \in F$ and set

$$t' := s_{\mathbf{0}} = dy_m^{e_m} \prod_{j=1}^{m-1} \binom{e_j}{0} x_j^{e_j} y_m^{e_j}.$$

Now $\deg_{y_m}(t') = e_m + \sum_{j=1}^{m-1} e_j = \sum_{j=1}^{m} e_j$ and $\operatorname{tdeg}_{\mathbf{x}}(t') = \sum_{j=1}^{m-1} e_j$. As $\deg_{y_m}(s_f) = e_m + \sum_{j=1}^{m-1}(e_j - f_j)$ it is clear that for all $f \in F$, $f \neq \mathbf{0}$ the monomial s_f has smaller degree in y_1 than t'. The remainder of the claim is immediately clear. □

Recall that with $\operatorname{monexps}_{\mathbf{y}}(f)$ we denote the list of m-tuples of monomial exponents of a polynomial f in m variables $\mathbf{y}$.

7.4 Example
Consider variables (x, y, z), the monomial $t := xyz$ and $f' := \tau_{(x,y,z),(a,b)}(t)$. Then $f' = xyz + xbz^2 + az^2y + az^3b$. As expected from the Lemma we get 4 monomials, each with total degree of 3 wrt. (x, y, z). And f' is strongly regular wrt. (x, y, z) due to the monomial $t' := az^3b$.

7.5 Corollary (action of τ' on a polynomial)
For variables $\mathbf{x} = (x_1, \ldots, x_{m-1})$ distinct from variables $\mathbf{y} = (y_1, \ldots, y_m)$, $f \in D[\mathbf{y}]\backslash\{0\}$, and $f' := \tau'_{\mathbf{y},\mathbf{x}}(f) \in D[\mathbf{x}, \mathbf{y}]$ we have:

1. *The number of monomials of f' wrt. $\mathbf{y}$ is smaller or equal to*

$$\sum_{\mathbf{e} \in \operatorname{monexps}_{\mathbf{y}}(f)} \prod_{j=1}^{m-1} (1 + e_j).$$

2. $\operatorname{tdeg}_{\mathbf{y}}(f') = \operatorname{tdeg}_{\mathbf{y}}(f)$ *holds.*

3. *There is a monomial t' in f' with $\deg_{y_m}(t') = \operatorname{tdeg}_{\mathbf{y}}(t')$, i.e. f' is strongly regular wrt. $\mathbf{y}$ and $\operatorname{tdeg}_{\mathbf{x}}(t') \leq \operatorname{tdeg}_{\mathbf{y}}(f) - \mu$, where μ is the minimum of all y_m-degrees occurring in a monomial of f of highest total degree.*

Proof. Set $E := \operatorname{monexps}_{\mathbf{y}}(f)$. Write f as a sum of monomials $f = \sum_{\mathbf{e} \in E} t_{\mathbf{e}}$. By the homomorphism property of τ' we know $\tau'(f) = \sum_{\mathbf{e} \in E} \tau'(t_{\mathbf{e}})$. Now (1) and (2) follow immediately from Lemma 7.3,(1) and (2).

To show (3), let $H \subseteq E$ denote the set of highest monomial exponents of f, i.e. $\mathbf{e} \in H$ implies $\sum_{j=1}^{m} e_j = \operatorname{tdeg}_{\mathbf{y}}(f)$. For each $\mathbf{e} \in H$ let $t'_{\mathbf{e}}$ denote the unique monomial in $\tau'(t_{\mathbf{e}})$ that exists by Lemma 7.3,(3) with $\operatorname{lc}_{y_m}(t'_{\mathbf{e}}) = c_{\mathbf{e}} \prod_{j=1}^{m-1} x_j^{e_j}$ for a domain element $c_{\mathbf{e}}$. Let $\mathbf{e}, \mathbf{e}' \in H$ with $\mathbf{e} \neq \mathbf{e}'$. Choose $2 \leq i \leq m$ with $e_i \neq e'_i$. Then $\operatorname{lc}_{y_m}(t'_{\mathbf{e}})$ and $\operatorname{lc}_{y_m}(t'_{\mathbf{e}'})$ differ in the power of x_i. Thus $\sum_{\mathbf{e} \in H} \operatorname{lc}_{y_m}(t'_{\mathbf{e}}) \neq 0$, i.e. f' is strongly regular wrt. $\mathbf{y}$.

As with 7.3,(3) we have $\operatorname{tdeg}_{\mathbf{x}}(t'_{\mathbf{e}}) = \sum_{j=1}^{m-1} e_j = \sum_{j=1}^{m} e_j - \deg_{y_m}(t'_{\mathbf{e}})$, the second part of claim (3) is clear as well. □

7.2.4 Regularization

We have just seen that the transformation τ can, in principle, be used to map a polynomial to a regular one. In principle, as if we use concrete constants $(\mathbf{c})$ instead of generic variables $(\mathbf{x})$, there is the possibility that the coefficient of the monomial that makes the polynomial regular vanishes. This then results in a transformed polynomial that lacks the desired property.

7.6 Example
Consider the polynomial $h := -81x^3 + 35x^3z - 72y^3z + 51y^2z^2 - 86x^3y^2 + 86x^2z^3$. Regularization wrt. (x, y, z) with choice $(1, 1)$ for constants does not succeed. The slightly different choice $(2, 1)$ works, however.

We turn now to the question if there is always a choice of constants to perform regularization. The situation can be seen as follows: If the coefficients of the monomials of the transformed polynomial are polynomial expressions in the constants. The goal is to find a choice of constants that is a non-root for the polynomial expression that is the coefficient of the monomial that makes the transformed polynomial regular. Such a choice has to be found for several polynomials simultaneously. We show that this is always possible.

7.7 Lemma (existence of non-roots for a polynomial)
A non-zero real polynomial f in $m \in \mathbb{N}$ variables $(\mathbf{y}) = (y_1, \ldots, y_m)$ has a non-root in $\mathbb{Z}^m$. Moreover, for given $Z_i \subseteq \mathbb{Z}$ with $Z_i \neq \emptyset$ and $|Z_i| \geq 1 + \deg_{y_i}(f)$ for $1 \leq i \leq m$ there exists a non-root $(\mathbf{c}) \in \prod_{i=1}^{m} Z_i$.

Proof. Let f be a non-zero real polynomial. We prove the claim by induction on the number of variables. If $m = 0$ then for $\emptyset \in \mathbb{Z}^0$ we have $f(\emptyset) \neq 0$ and indeed $(c) := \emptyset \in \prod \emptyset = \{\emptyset\}$ is a non-root of f.

If $m > 0$, then, as f is non-zero, $\mathrm{lc}_{y_m}(f)$ is non-zero. By induction hypothesis, choose a non-root $(\mathbf{c}) = (c_1, \ldots, c_{m-1})$ of $\mathrm{lc}_{y_m}(f)$. For given $Z_1, \ldots, Z_m \subseteq \mathbb{Z}$ we can even assume $(\mathbf{c}) \in \prod_{i=1}^{m-1} Z_i$. Now, $f(\mathbf{c}, y_m)$ is a non-zero univariate polynomial. This polynomial can have at most $\deg_{y_m}(f(\mathbf{c}, y_m))$ real roots in $\mathbb{Z}$. As $|Z_m| > \deg_{y_m}(f) = \deg_{y_m}(f(\mathbf{c}, y_m))$ choose a non-zero c_m from Z_m such that $f(\mathbf{c}, c_m) \neq 0$. Hence we have found a non-root $(\mathbf{c}, c_m)$ of f in $\mathbb{Z}^m$. □

7.8 Corollary
A finite set A of non-zero real polynomials in $m \in \mathbb{N}$ variables $(\mathbf{y}) = (y_1, \ldots, y_m)$ has a non-root in $\mathbb{Z}^m$. Moreover, for given $Z_i \subseteq \mathbb{Z}$ with $Z_i \neq \emptyset$ and $|Z_i| \geq 1 + \sum_{f \in A} \deg_{y_i}(f)$ for $1 \leq i \leq m$ there exists a non-root $(\mathbf{c}) \in \prod_{i=1}^{m} Z_i$.

Proof. An element $(\mathbf{c}) \in \mathbb{Z}^m$ is a non-root of A iff it is a non-root of $g := \prod_{f \in A} f$. As $\deg_{y_i}(g) = \sum_{f \in A} \deg_{y_i}(f)$, the claim follows from Lemma 7.7. □

7.9 Theorem (existence of non-roots for a set of polynomials)

For a finite set A of non-zero real polynomials in $(\mathbf{y}) = (y_1, \dots y_m)$ there exist integer constants $(\mathbf{c}) = (c_1, \dots, c_{m-1})$ such that $\tau_{\mathbf{y},\mathbf{c}}(f)$ is regular wrt. $(\mathbf{y})$ for all $f \in A$.

Proof. For variables $(\mathbf{x}) = (x_1, \dots, x_{m-1})$ disjoint from $(\mathbf{y})$ and for domain elements $(\mathbf{c}) = (c_1, \dots, c_{m-1})$ let $\gamma_{\mathbf{x},\mathbf{c}}$ denote the unique map $D[\mathbf{x}, \mathbf{y}] \to D[\mathbf{y}]$ defined by mapping x_i to c_i. Clearly $\tau_{\mathbf{y},\mathbf{c}} = \gamma_{\mathbf{x},\mathbf{c}} \circ \tau'_{\mathbf{y},\mathbf{x}}$. As just seen, $\mathrm{lc}_{y_m}(\tau'_{\mathbf{y},\mathbf{x}}(f))$ is a non-trivial polynomial in $(\mathbf{x})$ for all $f \in A$. Consider the set $B := \left\{\mathrm{lc}_{y_m}(\tau'_{\mathbf{y},\mathbf{x}}(f)) \;\middle|\; f \in A\right\}$ of polynomials in $(\mathbf{x})$. Choose by means of Corollary 7.8 a non-root $(\mathbf{c})$ of B. Now $\gamma_{\mathbf{x},\mathbf{c}} \circ \tau'_{\mathbf{y},\mathbf{x}}(f)$ is regular wrt. $(\mathbf{y})$ for all $f \in A$. □

Theorem 7.9 guarantees that ther is a legal choice of $(\mathbf{c})$. Corollary 7.8 tells us how big a cuboid must be to certainly find a legal choice. Clearly, if we have a bijection from $\mathbb{N}$ to an appropriate product space of $\mathbb{Z}$, this would solve our problem.

7.10 Remark

It is not difficult to verify the following facts:

1. Define $\nu : \mathbb{N} \to \mathbb{N} : n \mapsto \left\lfloor -\frac{1}{2} + \sqrt{\frac{1}{4} + 2n} \right\rfloor$ and $\mu : \mathbb{N} \to \mathbb{N} : n \mapsto \frac{1}{2}n(n+1)$. Then $\nu \circ \mu = \mathrm{id}_{\mathbb{N}}$.

2. Define $\gamma_1 : \mathbb{N} \to \mathbb{N} : n \mapsto \nu(n) - \gamma_2(n)$ and $\gamma_2 : \mathbb{N} \to \mathbb{N} : n \mapsto n - \mu(\nu(n))$. Then the map $\gamma : \mathbb{N} \to \mathbb{N}^2 : n \mapsto (\gamma_1(n), \gamma_2(n))$ is a bijection.

3. If A, B are sets and $\alpha : \mathbb{N} \to A$ and $\beta : \mathbb{N} \to B$ bijections, then the map
$$\mathbb{N} \to A \times B : n \mapsto (\alpha \circ \gamma_1(n), \beta \circ \gamma_2(n))$$
is also a bijection.

4. Define $\varepsilon_m := \mathrm{id}_{\mathbb{N}}$, if $m = 1$, and $\varepsilon_m : \mathbb{N} \to \mathbb{N}^m : n \mapsto (\varepsilon_1 \circ \gamma_1(n), \varepsilon_{m-1} \circ \gamma_2(n))$, for $m > 1$. Then ε_m is a bijection for all $m \in \mathbb{N}_1$. As a small modification, define $\varepsilon_1^* : \mathbb{N} \to \mathbb{N}_1 : n \mapsto n + 1$, and $\varepsilon_m^* : \mathbb{N} \to \mathbb{N}_1^m : n \mapsto (\varepsilon_1^* \circ \gamma_1(n), \varepsilon_{m-1}^* \circ \gamma_2(n))$, for $m > 1$. Then ε_m^* is a bijection for all $m \in \mathbb{N}_1$.

5. Define $\zeta_1 : \mathbb{N} \to \mathbb{Z}$ by $n \mapsto \frac{n}{2}$, if n is even, and by $n \mapsto -\frac{n+1}{2}$, otherwise. Define $\zeta_m : \mathbb{N} \to \mathbb{Z}^m : n \mapsto (\zeta_1 \circ \gamma_1(n), \zeta_{m-1} \circ \gamma_2(n))$, for $m > 1$. Then ζ_m is a bijection for all $m \in \mathbb{N}_1$.

We can now define an algorithm which maps a set of polynomials to a set of regular polynomials. We use the bijection $\varepsilon_{m-1}^* : \mathbb{N} \to \mathbb{N}_1^{m-1}$.

7.11 Algorithm (regularization subroutine)

$$(\mathbf{c}) \longleftarrow \mathit{REG1}(F, (\mathbf{y}))$$

Input: Variables $(\mathbf{y}) = (y_1, \dots, y_m)$ and a set of polynomials $F \subseteq D[\mathbf{y}]$.
Output: Constants $(\mathbf{c})$ such that $\tau_{(\mathbf{y}),(\mathbf{c})}[F]$ is a set of polynomials regular wrt. $(\mathbf{y})$.

```
1. Set $i := 0$, $F' := F$ and $(\mathbf{c}) = (c_1, \ldots, c_{m-1}) := (0, \ldots, 0)$.
2. While $F'$ is not regular wrt. $(\mathbf{y})$ do:
       $(\mathbf{c}) := \varepsilon^*_{m-1}(i)$
       $F' := \tau_{(\mathbf{y}),(\mathbf{c})}[F']$
       $i := i + 1$
3. Return $(\mathbf{c})$.
```

7.12 Remark

1. This algorithm terminates, because of Theorem 7.9 eventually an i will be found such that the while loop terminates. It is correct, as after the termination of the while loop F' is regular wrt. $(\mathbf{y})$.
2. Note that in case that the input is already regular the zero tuple $(\mathbf{0})$ is returned, which makes $\tau_{\mathbf{y},\mathbf{0}}$ the identity on $D[\mathbf{y}]$. In all other cases, due to the use of ε^*, never 0 is chosen for a constant. We will examine in a later section how one can systematically introduce zeros where possible.

Based on Algorithm REG1 we straightforwardly define:

7.13 Algorithm

$$F' \longleftarrow REG(F, (\mathbf{y}))$$

Input: Variables $(\mathbf{y}) = (y_1, \ldots, y_m)$ and a set of polynomials $F \subseteq D[\mathbf{y}]$.
Output: A set F' of polynomials that is regular wrt. $(\mathbf{y})$.

```
1. Return $\tau_{(\mathbf{y}),\mathrm{REG1}(F,(\mathbf{y}))}[F]$.
```

7.14 Remark

From a mathematical point of view it would now be interesting to investigate, which kind of enumeration leads to fast regularization, i.e. to a small number of times of execution of the body of the while loop.

In practice, however, it turns out that the body of the while-loop is rarely executed more than once. To test this, we generated random polynomials in Maple, using the command

```
randpoly([z,y,x],terms=5),
```

which gives a polynomial in three variables. In general, such a polynomial has total degree five and five terms, with coefficients ranging from -99 to 99.

Regularization of 1000 such random polynomials with the enumeration ε^*, which does not choose 0 for any constant, reveals that in general only 1–3 need a second choice of constants.

Thus finding a regular polynomial *fast* is not the problem. We will see, however, that finding a *relatively sparse* one is the task of interest. The way to go will be to carefully decide if it makes sense to choose 0 for certain constants.

7.3 Regularization of Formulas

After we have developed the necessary tools for regularization of polynomials we now examine how they can be applied to the quantifier elimination process. The aim is to transform a formula to a equivalent one with regular polynomials in it. This can be done in the following way:

1. Without loss of generality the input formula is prenex and every polynomial of the formula is irreducible.
2. For every quantifier block, starting with the innermost, we can apply regularization. It is important to note that we cannot apply regularization to each polynomial with an individual choice of constants, but that we have to find a choice of constants to regularize all polynomials of the formula simultaneously and consistently. See Algorithm 7.16 and 7.18. This transformed formula is equivalent to the formula we started with.
3. QE by CAD is applied to this modified formula. Note that the polynomials of this formula are irreducible. So irreducible factors computation does not destroy the regularity property. Thus the regularized polynomials are handed to the projection operator, where we can leverage constant leading coefficients.

Recall that for a non-constant polynomial f in $(\mathbf{x}) = (x_1, \ldots, x_m)$ the level of f wrt. $(\mathbf{x})$ is the maximal index $1 \leq i < m$ such that $\deg_{x_i}(f) \geq 1$. Constant polynomials are assigned level 0.

The input A of the following algorithm will usually be the polynomials of a formula. The set A is comprised of polynomials in $(x_1, \ldots, x_r)$. The index i_1 denotes the beginning of a certain quantifier block, i_2 denotes the end of that quantifier block.

7.15 Algorithm (regularization of a block of variables subroutine)

$$L \longleftarrow \text{REGBLOCK1}(A, (\mathbf{x}), i_1, i_2)$$

Input: A list of variables $(\mathbf{x}) = (x_1, \ldots, x_m)$, a set of polynomials in $(\mathbf{x})$, and natural numbers $1 \leq i_1 \leq i_2 \leq m$.
Output: A list L for regularization.

```
1. $A' := A$
2. For $i$ from $i_2$ downto $i_1 + 1$ do:
       Let $A''$ denote the set of all $i$-level polynomials in $A'$ wrt. $(\mathbf{x})$.
       $(\mathbf{c}^{(i)}) := \text{REG1}(A'', (x_{i_1}, \ldots, x_i))$
       $A' := \tau_{(x_{i_1},\ldots,x_i),(\mathbf{c}^{(i)})}[A']$
3. Return $(((x_{i_1}, x_{i_1+1}), (\mathbf{c}^{(i_1+1)})), \ldots, ((x_{i_1}, \ldots, x_{i_2}), (\mathbf{c}^{(i_2)})))$
```

Consider a list $L = ((\mathbf{x}^{(1)}, \mathbf{c}^{(1)}), \dots, (\mathbf{x}^{(n)}, \mathbf{c}^{(n)}))$ as returned from the algorithm. Such a list contains pairs consisting of a list of variables and a list of constants of appropriate length. We can straightforwardly extend the definition of τ by:

$$\tau_L := \tau_{(\mathbf{x}^{(1)},\mathbf{c}^{(1)})} \circ \cdots \circ \tau_{(\mathbf{x}^{(n)},\mathbf{c}^{(n)})}$$

7.16 Algorithm (regularization of a block of variables)

$$A' \longleftarrow \text{REGBLOCK}(A, (\mathbf{x}), i_1, i_2)$$

Input: As in Algorithm REGBLOCK1.
Output: A set of polynomials A' such that for each $i_1 \leq i \leq i_2$ each i-level polynomial of A' is regular wrt. $(x_{i_1}, \dots, x_i)$.

1. Return $\tau_{REGBLOCK1(A,(\mathbf{x}),i_1,i_2)}[A]$.

As for the correctness of the algorithm, assume that an i-level polynomial f was made regular wrt. $(x_{i_1}, \dots, x_i)$. Then in $\mathrm{lc}_{x_i}(f)$ none of the variables $x_{i_1}, \dots, x_{i-1}$ occurs. Thus subsequent regularization steps cannot destroy this property.

7.17 Algorithm (regularization of a formula subroutine)

$$L \longleftarrow \text{REGFOF1}(\varphi, (\mathbf{x}))$$

Input: A prenex formula φ and an admissible variable order $(\mathbf{x}) = (x_1, \dots, x_r)$.
Output: A list L for regularization.

1. Write φ in quantifier block form: $\varphi(\mathbf{x}^{(0)}) := \mathsf{Q}_{i_1}\mathbf{x}^{(1)}(\dots \mathsf{Q}_{i_n}\mathbf{x}^{(n)}(\psi)\dots)$. Then $(\mathbf{x}) = (\mathbf{x}^{(0)}, \dots, \mathbf{x}^{(n)})$. Set $i_0 := 1$ and $i_{n+1} := r + 1$. Then the j-th quantifier block $(\mathbf{x}^{(j)})$ is $(x_{i_j}, \dots, x_{i_{j+1}-1})$.
2. Let A denote the polynomials of φ.
3. for $j = n$ downto 1 do:
   $(\mathbf{L}_j) := REGBLOCK1(A, (\mathbf{x}^{(j)}), i_j, i_{j+1})$
4. Return $(\mathbf{L}_1, \dots, \mathbf{L}_n)$.

We extend straightforwardly τ to formulas. Let $\tau_{\mathbf{x},\mathbf{c}}(\varphi)$ denote the result of replacing every polynomial f of φ by $\tau_{\mathbf{x},\mathbf{c}}(f)$. Similarly, define $\tau_L(\varphi)$.

7.18 Algorithm (regularization of a formula)

$$\varphi' \longrightarrow \text{REGFOF}(\varphi, (\mathbf{x}))$$

Input: As in Algorithm REGFOF1.
Output: A prenex formula φ' such that for each bound variable x_i each i-level polynomial is regular wrt. $(x_{i_1}, \dots, x_i)$, if $(x_{i_1}, \dots, x_{i_2})$, $i_1 \leq i \leq i_2$, is the variable block x_i resides in, and such that $\mathbb{R} \models \varphi \longleftrightarrow \varphi'$

1. Return $\tau_{\text{REGFOF1}(\varphi,(\mathbf{x}))}(\varphi))$

Note that it would of course be desirable to have each i-level polynomial to be regular wrt. $(x_1, \dots, x_i)$, as it then would be guaranteed to have a constant leading coefficient. Regularization is, however, limited to variable blocks, as otherwise the condition, that the transformed formula is equivalent to the original formula, could not be guaranteed. Nevertheless there is the expectation, that often this limited regularization suffices to produce constant leading coefficients.

Before we see some examples we show how to limit the growth of the number of monomials during regularization.

7.4 Efficient Regularization

So far, regularization means for us to perform a transformation wrt. every variable, except the variable the polynomial should finally be regular in. We can be more efficient in this respect: Based on an observation we show that it often suffices to consider less variables for the transformation. This yields sparser polynomials (less monomials and therefore a smaller sotd value), and is hence expected to be more efficient for CAD, as seen in Chapter 6.

7.4.1 An Observation

Consider the following randomly generated polynomials:

$$\begin{aligned}
f_0 &= 79y + 56z^2x^2 + 49zy^2x + 63y^2x^2 + 57z^3y^2 - 59z^2y^3 \\
f_1 &= 77yx^2 + 66z^3y + 54zyx^2 - 5zx^3 + 99y^2x^2 - 61zx^4 \\
f_2 &= zx - 47zyx - 91zx^3 - 47yx^3 - 61z^4x + 41zy^3x \\
f_3 &= -86z^2y + 23z^2x - 84y^3x + 19z^2x^3 - 50y^5 + 88y^2x^3 \\
f_4 &= -85z^2 - 86x^3 + 30z^3y + 80zy^2x + 72z^5 + 66z^3yx
\end{aligned}$$

We want regularity wrt. (x, y, z). If we apply the algorithm REG to f_i and (x, y, z) we are guaranteed to get regular results wrt. (x, y, z). The number of monomials of the results are given in the last line of Table 7.2. Now, what happens if we apply the algorithm to f_i and (y, z), (x, z), or (z), respectively? The results are guaranteed to be regular wrt. (y, z), (x, z), or (z), respectively, but will they be regular wrt. (x, y, z) as well? Looking at Table 7.2 again we see from the third line that, with the exception of

f_2, all polynomials are regular wrt. (x, y, z), after they were made regular wrt. (y, z). And from the second line we read that, with the exception of f_0 and f_1, all polynomials are regular wrt. (x, y, z), after they were made regular wrt. (x, z). Finally, looking at the first line, only f_4 was regular wrt. (x, y, z) from the beginning.

In addition we observe that we can expect smaller polynomials if we regularize wrt. fewer variables. E.g. for f_3 we find a regular version with 14 instead of 26 monomials.

	f_0	f_1	f_2	f_3	f_4
(z)	–	–	–	–	6
(z, x)	–	–	16	14	6
(z, y)	12	9	–	16	6
(z, y, x)	15	21	22	26	6

Figure 7.2: Number of monomials of the regularized polynomial, if regular wrt. (z, y, x).

7.4.2 Optimal Regularization wrt. Number of Monomials

If A is a set of polynomials in $D[\mathbf{y}]$ we call an sublist $(\mathbf{y}')$ of $(\mathbf{y})$ *suitable for regularization of A wrt.* $(\mathbf{y})$, if each element of $\mathrm{REG}(A, (\mathbf{y}'))$ is regular wrt. $(\mathbf{y})$. The sublist $(\mathbf{y}')$ is called *optimal for regularization of A wrt.* $(\mathbf{y})$, if it is suitable and the sum of all numbers of monomials of elements of $\mathrm{REG}(A, (\mathbf{y}'))$ is minimal among all suitable sublists.

After the observation above, it is straightforward to give an algorithm, which finds an optimal sublist for regularization wrt. number of monomials:

7.19 Algorithm (optimal regularization subroutine)

$$S \longleftarrow \mathrm{REGOPT2}(A, (\mathbf{y}))$$

Input: Variables $(\mathbf{y}) = (y_1, \ldots, y_m)$, $m \geq 1$, and a set of polynomials $A \subseteq D[\mathbf{y}]$.
Output: A non-empty list of lists of variables S, such that each list of variables $(\mathbf{y}') \in S$ is a sublist of $(\mathbf{y})$, each element of $\mathrm{REG}(A, (\mathbf{y}'))$ is regular wrt. $(\mathbf{y})$, and $(\mathbf{y}')$ is optimal for regularization of A wrt. $(\mathbf{y})$.

1. For each sublist $(\mathbf{y}')$ of $(y_1, \ldots, y_{m-1})$ collect a pair

$$\left((\mathbf{y}', y_m), \sum_{f' \in \mathrm{REG}(A, (\mathbf{y}', y_m))} nom_{\mathbf{y}}(f) \right),$$

   but only if each element of $\mathrm{REG}(A, (\mathbf{y}', y_m))$ is regular wrt. $(\mathbf{y})$. Call this list of pairs L.

2. Return $S := \{(\mathbf{y}', y_m) \mid ((\mathbf{y}'), c) \in L \text{ and } c \text{ minimal in } \{\pi_2(p) | p \in L\}\}$.

7.20 Algorithm (optimal regularization subroutine)

$$(\mathbf{c}) \longleftarrow \text{REGOPT1}(A, (\mathbf{y}))$$

Input: As in Algorithm REGOPT2.
Output: A list of constants $(\mathbf{c}) = (c_1, \ldots, c_{m-1})$, such that each element of $\tau_{\mathbf{y},\mathbf{c}}[A]$ is regular wrt. $(\mathbf{y})$.

1. Choose an element $(\mathbf{y}') \in \text{REGOPT2}(A, (\mathbf{y}))$.
2. $(\mathbf{c}') := \text{REG1}(A, (\mathbf{y}'))$
3. For $1 \leq i \leq m-1$ define $c_i := \begin{cases} 0, & \text{if } y_i \text{ does not occur in } (\mathbf{y}') \\ c'_j, & \text{if } y_i \text{ occurs on position } j \text{ in } (\mathbf{y}'). \end{cases}$
4. Return $(\mathbf{c})$.

Based on Algorithm REGOPT1 we straightforwardly define:

7.21 Algorithm (optimal regularization)

$$A' \longleftarrow \text{REGOPT}(A, (\mathbf{y}))$$

Input: As in Algorithm REGOPT1.
Output: A set A' of polynomials that is regular wrt. $(\mathbf{y})$.

1. Return $\tau_{(\mathbf{y}),\text{REGOPT1}(A,(\mathbf{y}))}[A]$.

Note that REG1 and REGOPT1, and also REG and REG1 have the same specifications. This means that the optimized versions are drop-in replacements for their non-optimized counterparts.

7.4.3 An Efficient Way to Find Good Regularization Subsets

In contrast to applications of a projection operator the transformations discussed in this chapter are rather cheap. Nevertheless, we present here an idea how to speed up regularization.

The number of possible sublists considered by REGOPT2 is exponential in the number of variables. Here, similar to the approach in Chapter 6, a greedy algorithm could be utilized. Such an algorithm would reduce the number of sublists to be considered to one polynomial in the number of variables. We cannot expect to always find an optimal result, but we would expect to find a good result.

7.5 Examples

We study the impact of regularization by looking at some examples. Some of the examples demonstrate that we can indeed achieve the desired savings with our regularization method.

Motzkin Polynomial

Consider the question whether the Motzkin polynomial is positive semidefinite:

$$\varphi_{mp} = \forall x \forall y (1 + x^2 y^2 (x^2 + y^2 - 3) \geq 0)$$

This leads to a full CAD of 19 cells. The regularized equivalent formula

$$\varphi_{mpr} = \forall x \forall y (x^4 y^2 + 4y^3 x^3 + 7x^2 y^4 + 6y^5 x + 2y^6 - 3x^2 y^2 - 6xy^3 - 3y^4 + 1 \geq 0)$$

leads to a CAD with only 15 cells.

A CAD Example with 3 Variables

Consider the polynomial $f := ux^3 + ux + vx^2 + x^3 + 2x - 1$. After projection wrt. the variable order $x \to u \to v$ we end up with 9 projection factors, and a corresponding full CAD with 429 cells. If we, however, regularize f wrt. (u, v, x) and perform projection wrt. the same order as above, then we end up with only 5 projection factors and 187 cells.

An Implicitation Example

Consider the implicitation example

$$\varphi_{cox6} := \exists u \exists v (x = uv \wedge y = u^2 \wedge z = v^2)$$

The set of input polynomials is $A = \{-uv + x, -u^2 + y, -v^2 + z\}$, and (x, y, z, u, v) is computed as an efficient projection order. This CADQE problem results in 9 projection factors, a full CAD of 863 cells, and a partial CAD of 179 cells. We can regularize the block of existential quantifiers by computing $REGOPTBLOCK(A, (x, y, z, u, v), 4, 5)$. Replacing the polynomials in φ_{cox6} by those we get the regularized formula

$$\varphi_{cox6r} := \exists u \exists v (-uv - v^2 + x = 0 \wedge -u^2 - 2uv - v^2 + y = 0 \wedge -v^2 + z = 0)$$

Note that now all polynomials are of the highest level. In particular, in the second polynomial the variable v occurs now. This equivalent formulation leads to 37 projection factors, and a full CAD guessed to consist of a 6-digit number of cells, and a partial CAD of 51017 cells. For this example, regularization does not lead to the expected gain in efficiency.

7.6 Further Work and Remarks

1. Based on this work more advanced methods regarding the particular choice of constants and strategies to detect when regularization has its benefits and when not could be developed.

2. Deciding on a projection order, as seen in Chapter 6, and regularization, as seen in this chapter, is, from a more abstract point of view, the same approach: a linear bijection (a permutation in the first case, a sheering in the second case) is applied to blocks of variables. This leads to the more general question which further kinds of transformations could be utilized.

7.7 Conclusions

In this chapter we have defined a method for transforming a set of polynomials into one which is equivalent for QE for CAD by regularization. We have made precise on which polynomials occurring within the projection phase this transformation can be used. On the one hand we expected smaller projection sets and thus savings. On the other hand, the increase of the sotd value and the level for some polynomials lead us to the *a priori* suspicion that negative effects are to be expected. Looking at examples we see that these two effects indeed occur. Thus, based on this work, more sophisticated methods of regularization could be developed to guarantee more consistently good results.

Chapter 8

Implementation

Improvements suggested in this thesis were implemented, so large examples could be computed that were not feasible by hand. In this chapter we give an overview of this implementation. More precisely:

1. We introduce the computer algebra system REDUCE in which most of the implementation took place.
2. We demonstrate that CADQE can be combined with quantifier elimination by virtual substitution (VSQE) for added benefits.
3. We talk about some implementation details and design decisions.
4. We give an overview of the available commands.

8.1 REDUCE and REDLOG

Most of the implementation work was done within the REDUCE package REDLOG.

8.1.1 REDUCE

REDUCE is a computer algebra system that dates back to the 1960s. Its name is written in capital letters, as in those times input devices had no lower case letters. The name is not an acronym, but intended as a joke, as computer algebra systems can give very large output in certain cases [Hea05]. The system is *open source*—in the literal sense, i.e. the sources are shipped, not in the modern sense of free and open source software (FOSS). Its installation base is comprised of roughly 1500[1] licenses, many of them at universities, so the user base is presumably higher.

REDUCE is based on a Lisp dialect called *Standard Lisp*. Several implementations are available, the most important ones being Portable Standard Lisp (PSL), maintained

[1] This number refers to 3.7 systems.

by Winfried Neun at the Zuse Institut Berlin (ZIB), and Cambridge Standard Lisp (CSL), maintained by Arthur Norman (Trinity College/Codemist Ltd.).

The emphasis of REDUCE and its Lisp base lies on simplicity, efficiency, and portability. The speed and efficiency that can be achieved matches and surpasses[2] implementations in C. On the downside, the programming language lacks features like type checking, which are considered basic nowadays, and the current Lisp systems impose a memory limit of 128MB on REDUCE. With the advent of 64bit architectures, however, this memory limit is lifted.

The user interface is minimalistic, but there are several front-ends available. REDFRONT provides the features of the GNU Readline and is meanwhile shipped with the system. The PSL version of REDUCE comes with a graphical interface for Windows that provides menus and some fancy printing. TeXmacs allows for sophisticated fancy printing of the output, thus it is suitable for demonstrations. For the CSL-based version a graphical interface based on the platform-independent Fox-toolkit is developed.

The current release is 3.8 from 15 April 2004. Future plans for REDUCE are to further open it up, and to possibly turn it into free software in 2010.

8.1.2 REDLOG

REDLOG, the REDUCE logic system is developed in Passau since 1992. It allows to compute with first-order logic over a fixed language and theory. For this work, only the language of ordered rings and the theory of real closed fields is of interest.

It provides quantifier elimination by several alternative methods, simplification of first-order formulas, using factorization and Gröbner basis techniques, normal form computation and numerous tools for constructing and processing formulas.

The available quantifier elimination methods for the reals are: virtual substitution of test points (VS), parametric real root counting (now called Hermitian QE), and, thanks to this work, cylindrical algebraic decomposition (CAD).

It is of mutual advantage to have a computer logic system integrated into a computer algebra system, in contrast to have a stand-alone system. This is pointed out in [DS03].

REDLOG is part of the REDUCE development system. This means that contributions to REDLOG will automatically be part of the next REDUCE release. In particular, the CAD implementation, the improvements of generic CAD and efficient projection orders are already included in REDUCE 3.8.

8.2 Combination of Methods

With CAD and VS implemented on one system we can think of combining these two methods into one QE strategy. The CAD and the VS method differ in some respects, and thus complement each other:

[2] Communication with Thomas Sturm, who compared a REDUCE/Lisp implementation with a Asir/C implementation of the VS method.

- VS imposes restrictions on the degree of the polynomials in the input formula. CAD has no restrictions.
- VS is doubly exponential in the number of quantifier changes, but for fixed quantifier type only singly exponential in the number of quantifiers; most importantly, the number of parameters does not contribute to complexity in a relevant way. CAD, on the contrary, is doubly exponential in all variables.
- VS can eliminate one quantifier at a time, staring with the innermost quantifier. CAD has to eliminate all quantifiers simultaneously.

Altogether this hints the following strategy:

1. Eliminate as many quantifiers as possible via the VS method.
2. If at some step the VS method fails due to degree restrictions, finish with the CAD method.

Note that it is not possible to do it the other way round, due to the characteristics of the methods.

Consider as an example the following implicitation problem taken from [CLO92]:

$$\exists u \exists v(x = u \cdot v \wedge y = u \cdot v^2 \wedge z = u^2)$$

Virtual substitution eliminates successfully the first quantifier:

$$\begin{gathered}\exists v(z \geq 0 \wedge ((v^4 \cdot z - y^2 = 0 \wedge v^2 \cdot z - x^2 = 0 \wedge \\ v \cdot x \leq 0 \wedge (v = 0 \vee y \leq 0)) \vee (v^4 \cdot z - y^2 = 0 \wedge \\ v^2 \cdot z - x^2 = 0 \wedge v \cdot x \geq 0 \wedge (v = 0 \vee y \geq 0))))\end{gathered}$$

But now, due to degree restrictions, VS cannot be used anymore. CAD continues and succeeds:

$$\begin{gathered}(x^4 - y^2 \cdot z = 0 \wedge x = 0 \wedge y = 0 \wedge z \geq 0) \vee \\ (x^4 - y^2 \cdot z = 0 \wedge x \neq 0 \wedge y \neq 0 \wedge z > 0)\end{gathered}$$

As for timings,[3] the combined strategy needed 110ms, CAD alone 330ms.

As a consequence, this strategy is implemented as the default behavior in REDLOG. This automatic combination of two quantifier elimination methods on one system empowers users to compute larger examples as previously possibly. An example for a successful application of this strategy is [SW03].

[3] On a Pentium 933Mhz 128Mb.

8.3 Implementation Details

For real algebraic number computation we use a reimplementation of the approach that was developed in [Sei01], with some improvements [Sei02]. With respect to real algebraic numbers, the three available implementations of CAD differ. QEPCAD uses primitive element computation, and Mathematica uses validated numerics. It is an interesting—but so far unanswered—question, which approach is the best.

In the preparation phase it is possible to utilize the degree decreasing strategy from [DS98], and to use an efficient projection order (cf. Chaper 6).

For projection, Collins' operator with Hong's improvement [Hon90] is used for levels higher than 3. For lower levels, McCallum's operator is used.

For decomposition we use a recursive algorithm that builds a CAD tree. This includes an implementation of partial CAD (with first and second improvement). For root isolation an improvement to the Sturm-Sylvester method is used, which the author calls *incremental root isolation.* The idea here is as follows: After we have found the first isolating interval for a root of a polynomial that has the property that no other polynomial has a root inside and on the border we pause root isolation. One root suffices to generate the sample points for two cells by extending the sample point of the base cell by the lower bound of the interval, and the real algebraic number itself. This process of finding one root and generating two cells is iterated. The final sample point is generated with the upper bound of the isolating interval of the largest root. In the context of PCAD this approach can yield savings. This root isolation technique is believed to be an important reason why, when looking at the following timings,[4] the author's implementation `rlcad` is not either always faster or always slower than QEPCAD.

Example	`rlcad`	QEPCAD
cox12	150ms	50ms
ter	80ms	60ms
col	0ms	70ms
davhei	280ms	80ms
colj	60ms	50ms
lwcoj	1380ms	160ms
gamma1	460ms	3280ms
css	8400ms	70ms
quartic	7720ms	140ms
aci	13920ms	220610ms

As for solution formula construction we use the method based on signatures of projection polynomials proposed by Hong; thus in rare cases, solution formula construction may fail. For simplification of the output formula we use the available routines from REDLOG. The results are often less simple than the ones of QEPCAD. The main reason for this seems to be that the tree-valued logic simplification method implemented

[4] On a Pentium 933Mhz 128Mb.

in QEPCAD (which is tailored to signature based formulas) proposed by Hong yields better results than the (more general) methods utilized by REDLOG.

8.4 Documentation

In this section we give a more detailed overview of the available commands than in the REDLOG manual for REDUCE 3.8 [DSS04]. We assume basic data structures as *formula* and other functionality to be known from the manual.

Some of this functionality may be changed or removed in future releases of REDLOG.

8.4.1 Quantifier Elimination by CAD

`rlcad` *formula [reverse_projection_order]* Function

Quantifier elimination by cylindrical algebraic decomposition. Returns a quantifier-free equivalent of *formula*, if the method succeeds, the input formula, otherwise. A projection order can be given as an optional argument. There are no degree restrictions on the polynomials in *formula*.

`rlcadproj` *formula [reverse_projection_order]* Function

Projection factors set. Returns a *projection_set*. A projection order can be given as an optional argument.

`rlcadswitches` Function

CAD switches. Prints the current settings of the most important switches. Note that this command has to be called as `rlcadswitches()`.

`rlcaddecdeg` Switch

Decrease degree. Tries to decrease the degree of the input formula during preparation phase. Turned off by default.

`rlcadpartial` Switch

Partial CAD. Turned on by default.

`rlcadte` Switch

Trial evaluation, the first improvement to partial CAD. Turned on by default.

`rlcadpbfvs` Switch

Propagation below free variable space, the second improvement to partial CAD. Turned on by default.

`rlcadisoallroots` Switch

Isolate all roots. Turned off by default.

`rlcadtrimtree` Switch

Trim CAD tree. Unneeded branches of the tree are pruned in order to save space (cf. Remark 2.51). Turned on by default.

`rlcadfulldimonly` Switch

Find full-dimensional cells only. Warning: If this switch is turned on, then the result is in general not correct. Turned off by default.

`rlcadrmwc` Switch

Remove white cells. Turned on by default.

`rlcadrawformula` Switch

Output the signature-based solution formula without simplification.

`rlcaddnfformula` Switch

Output the solution formula in DNF. If turned off, the solution formula is returned in CNF. Turned on by default.

`rlqefb` Switch

Fall back QE. This affects `rlqe`. If turned on, the CAD method is used, if QE by VS fails due to degree restrictions. Turned on by default.

`rlcadpreponly` Switch

Preparation phase only. Turned off by default.

`rlcadprojonly` Switch

Projection phase only. Turned off by default.

`rlcadextonly` Switch

Extension phase only. Turned off by default.

8.4.2 CAD with Answers

`rlcadans` Advanced Switch

Propagate answers. Returns answers in form of verbose output. The following switches need to be set: on `rlcadisoallroots`, on `rlcadpartial`, off `rlcadpbfvs`, off `rlcadte`, and off `rlcadtrimtree` Turned off by default.

8.4.3 Generic CAD

`rlgcad` *formula [reverse_projection_order]* Function

Generic quantifier elimination by CAD. Returns a pair consisting of a *theory* and a quantifier-free formula.

`rlgcadporder` *formula* Function

Efficient projection order for generic CAD. Returns a *reverse_projection_order*.

8.4.4 Efficient Projection Orders

`projection_order` Data Structure

A list of variables $(x_1, \ldots, x_r)$ that indicates that projection is performed in the order: $x_r \to \ldots \to x_1$.

`reverse_projection_order` Data Structure

A list of variables $(x_r, \ldots, x_1)$ that indicates that projection is performed in the order: $x_r \to \ldots \to x_1$.

`projection_set` Data Structure

A list of list of polynomials $(F_r, \ldots, F_1)$, such that F_i contains the level-i projection factors.

The main procedures for finding projection orders:

`rlcaddefaultorder` *formula* Function

Default order, the *reverse_projection_order*, which the system would use by default. Returns a *reverse_projection_order*.

`rlcadporder` *formula* Function

Efficient projection order. Returns a list of variables, a *reverse_projection_order*.

`rlgcadporder` *formula* Function

Efficient projection order for generic CAD. Returns a list of variables, a *reverse_projection_order*.

`rlcadporders` *formula* Function

Admissible projection orders. Returns a list of *reverse_projection_order*. Warning: If *formula* contains long blocks of quantifiers, the returned result can get quite large. Use `rlcadpordersnum` to check beforehand.

`rlcadpordersnum` *formula* Function

Number of admissible projection orders. This is very fast, as the projection orders need not to be generated. Returns a number.

`doc` *number_list number_list* Function

Degree of correspondence of two lists of numbers of same length. Returns a rational number.

8.4.5 Exact Number of Cells of a full CAD

`rlcadnum` *projection_set reverse_projection_order* Function

Compute the size of a full CAD for a given list of polynomials and a projection order. Returns a number.

`rlcadnum1` *projection_set reverse_projection_order* Function

Compute the number of cells of a full CAD and of all induces CAD's for a given list of polynomials and a projection order. Returns a list $(|D_r|, \ldots, |D_1|)$ of numbers.

`rlcadnumauto` *formula* Function

Compute the size of a full CAD for a given formula using the default projection order. Returns a number.

`rlcadnumepo` *formula* Function

Compute the size of a full CAD for a given formula using an efficient projection order as computed by `rlcadporder`. Returns a number.

8.4.6 Guessed Number of Cells of a full CAD

`rlcadguess` *projection_set reverse_projection_order* Function

Guess the size of a full CAD for a given list of polynomials and a projection order. The resulting value gives quickly an idea on how big the order of magnitude of the size of a full CAD is.

`rlcadguess1` *projection_set reverse_projection_order* Function

Guesses the size of a full CAD subroutine. Returns a list $(d_r, \dots, d_1)$ of numbers such that $\prod_{i=1}^{j}$ is the guessed number of level-j cells.

`rlcadguessauto` *formula* Function

Guess the size of a full CAD wrt. the projection order the system would actually choose.

`rlcadguessepo` *formula* Function

Guess the size of a full CAD using an efficient projection order as computed by `rlcadporder`.

8.4.7 Number of Cells in a Partial CAD

`rlcadpnum` *formula reverse_projection_order* Function

Compute the number of leaves of the resulting partial CAD tree. Note that the cells of the yield of the partial CAD tree are not necessarily on the highest level.

8.4.8 Regularization

These functions are implemented in a Maple worksheet and not available in REDUCE.

`transreg` *polynomial_list variable_list constant_list* Function

This implements the transformation τ.

`epsstar` *number number* Function

This implements ε^*.

`REG1` *polynomial_list variable_list* Function

`REG` *polynomial_list variable_list* Function

This implements algorithms 7.11 and 7.13.

`REGBLOCK1` *polynomial_list variable_list number number* Function
`REGBLOCK` *polynomial_list variable_list number number* Function

This implements algorithms 7.15 and 7.16.

`REGOPT2` *polynomial_list variable_list* Function
`REGOPT1` *polynomial_list variable_list* Function
`REGOPT` *polynomial_list variable_list* Function

This implements algorithms 7.19, 7.20 and 7.21

`REGOPTBLOCK1` *polynomial_list variable_list number number* Function
`REGOPTBLOCK` *polynomial_list variable_list number number* Function

Like REGBLOCK1 and REGBLOCK above, but with optimized drop-in replacements.

8.4.9 Graphviz Interface

`rlcadtree2dot` Switch

Export CAD tree to dot format in a file *cadtree.dot*. Usually one wants to turn off `rlcadtrimtree`. The file includes as a comment a suggested command line call to produce a postscript file. Turned off by default.

8.5 Summary

The author implemented CADQE (including PCAD) as well as modifications and improvements from this thesis within the computer logic system REDLOG. This made it possible to combine this method with VSQE on one system. The implementation differs from other implementations by several design decisions. As part of the REDUCE development system, the implementation is already available in part with REDUCE 3.8, and will be available in full the next version of REDUCE.

Chapter 9

Conclusions

This thesis presented application-oriented extensions and modifications of the cylindrical algebraic decomposition algorithm (CAD) for real quantifier elimination (QE). Due to these methods, larger problems than before can be tackled in practice and/or more interesting information is provided as output.

Chapter 1 contains no new contributions, it serves to provide some basic notions and notations in order not to clutter up later chapters.

In Chapter 2 we have presented the CAD algorithm for real QE in a consistent way that reuses as much notation from the literature as possible. The presentation is modern in that it views CADs as trees, which is more suitable than lists of cells for QE purposes. It is implementation-friendly in that it discusses some representation issues and gives concrete algorithms. It contains no new results; its main purpose is to put up a scaffolding to raise Chapter 3.

In Chapter 3 we have developed the framework of cylindrical subdecomposition (SCAD). This framework gives insight into what happens, if one does not compute a cell decomposition of full space, but only a subspace, and then uses this subdecomposition for QE purposes. In particular, this framework gives an algorithm for and semantics of subdecomposition-based real quantifier elimination (SCADQE). One original feature of this algorithm is to allow external assumptions on bounded variables in a semantically clean way. The approach is compatible with partial CAD. It provides an abstraction layer on which further applications can be realized. There are applications possible, where assumptions accompany the input, or are found at various stages during algorithm execution, or both. By using the SCAD framework one can focus on application ideas, while semantics and correctness is provided by the framework.

In Chapter 4 we have defined a generic projection operator and described how to perform generic quantifier elimination (combined with partial CAD) on the basis of this operator. The idea of generic QE is to automatically exclude certain special cases to cut down on the overall cost of computation. By means of highly non-trivial quantifier elimination examples, we have demonstrated that our generic approach significantly extends the application range of partial CAD. This particularly affects input formulas with polynomials of high degree and many parameters, thus filling a present gap in the applicability of real quantifier elimination techniques.

Furthermore, in Chapter 4 we have developed quantifier elimination by local CAD as an additional application within the SCAD framework. The idea of local elimination is to cater for cases where the user already knows sample values for some or all parameters of his formulation. The result is then locally correct in an area that contains these points. The algorithm makes use of this additional information given by the user to concentrate on the cases of interest. We demonstrate that this leads to large savings. In addition, we have shown that these approaches can be combined with each other, by giving a criterion to be checked during generic projection set computation to ensure that the overall assumptions are not getting inconsistent. Finally we sketched some more applications.

By looking at examples we see that the amount of computation needed for GCAD, LCAD, and the combined LGCAD, is greatly reduced as expected. In return, the results are correct only wrt. a relaxed semantics. While from a theoretical point of view a weaker semantics might seem inferior at first, it turns out that for both generic CAD and local CAD the generated assumptions consist of interesting information. In addition, from a practical point of view the actual user might be more happy to get an result that excludes certain degenerate or special cases, or that is locally correct, than to wait considerably longer, or indefinitely, for a general result.

Putting the results of Chapter 3 and 4 together, we have gathered strong evidence that a relaxed semantics, where the result of QE is equivalent to the input under some assumptions, is highly useful. There are three reasons why the suggested approach should be considered as a feature. First, the assumptions made automatically can provide added insight into the problem, as seen with generic CAD. Second, an algorithm can be designed to cater for a special problem, as seen with local CAD. Third, it is natural to think about a problem and conditions, so providing support on an algorithmic level to separate assumptions and not to intermingle them with the problem formulation is promising and user friendly. We have provided the theoretical background, the algorithmic means, and a variety of applications and computation examples to substantiate the usefulness of this approach.

In Chapter 5 we have seen that it is highly desirable to produce answers in addition to a solution formula as output of quantifier elimination for a formula with a leading existential block of quantifiers. We have devised an algorithm to extend the cylindrical algebraic decomposition method to produce such answers. For non-decision problems, these sample solutions are in general parametric. For the dual case, i.e. for formulas with a leading universal block of quantifiers, we get parametric counter-examples.

Altogether we have shown in Chapter 3 through 5, that the paradigms of generic QE, local QE, QE with answers, and QE with external theory, which were originally developed with the virtual substitution method (VS), can be successfully applied to CAD as well. This is a surprising and original result, as the concrete realization of these application for CAD is totally different to the VS approach. Consequently, these methods are now available for all real QE problems, without degree restrictions.

In Chapter 6 we have exploited a degree of freedom in the CAD algorithm: the order in which projection is conducted. We have obtained strong statistical evidence that the projection order is of crucial importance for the success of CAD computations. For determining the quality of a given projection order, we have shown that there is one single measure, namely the sum of total degrees of all monomials (sotd), on the projection sets that is correlated to all interesting time and space measures on the computation of a full CAD as well as on quantifier elimination by partial CAD. We have introduced a greedy algorithm for efficiently constructing a good projection order wrt. sotd. This algorithm provides consistently excellent results and is already utilized by others. In addition, a method to guess the order of magnitude of the space and time requirements of a CAD problem is devised.

In Chapter 7 we have defined a method for transforming a set of polynomials into one which is equivalent for QE for CAD by regularization. Basically we apply a sheering, so in a certain sense tilted cylinders are considered. On the one hand this method can give smaller projection sets by introducing constant leading coefficients. On the other hand, the increase of the level for some polynomials can lead to an increased projection set. Looking at examples we see that these two effects indeed occur. Thus, based on this work, more sophisticated methods of regularization could be developed to guarantee more consistently good results.

The approaches of Chapter 6 and 7 have in common that in both cases a linear bijection (a permutation in the first case, a sheering in the second case) is applied to blocks of variables. This leads to the more general (and open) question which further transformations are suitable and could be successfully utilized.

Chapter 8 shows that with the author's implementation of CAD in REDLOG, this QE method can be combined with the VS method with an overall benefit. The implementation allowed for the first time a combined elimination strategy based on two real QE methods within a single system. For each of the chapters 4 through 7 we have provided an implementation to compute large examples.

The following table finally gives an overview which application-oriented concept leads to a relaxed semantics, provides additional information, exploits a degree of freedom, is applicable for pure CAD (i.e. without a QE problem in the background), and leads to a consistent speed-up.

	SCAD	G/LCAD	answers	PORDER	REGFOF	combination
relaxed semantics	+	+				
additional information	+	+	+			
degree of freedom				+	+	
appl. for pure CAD	+			+	+	
speed-up	usually	+		+	±	+

Bibliography

[ACM84] Dennis S. Arnon, George E. Collins, and Scott McCallum. Cylindrical algebraic decomposition I: The basic algorithm. *SIAM Journal on Computing*, 13(4):865–877, November 1984.

[ADY$^+$96] Chaouki T. Abdallah, Peter Dorato, Wei Yang, Richard Liska, and Stanly Steinberg. Applications of quantifier elimination theory to control system design. In *Proceedings of the 4th IEEE Mediterranean Symposium on Control and Automation*, pages 340–345. IEEE, 1996.

[BPB05] James Beaumont, Nalina Phisanbut, and Russel Bradford. Practical simplification of elementary functions using CAD. In Andreas Dolzmann, Andreas Seidl, and Thomas Sturm, editors, *Proceedings of the 2005 Conference on Algorithmic Algebra and Logic (A3L 2005)*, pages 35–39, Passau, Germany, April 2005.

[Bro99] Christopher W. Brown. Guaranteed solution formula construction. In Sam Dooley, editor, *Proceedings of the ISSAC 99*, pages 137–144. ACM Press, New York, NY, July 1999.

[Bro01] Christopher W. Brown. Improved projection for cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 32(5):447–465, November 2001.

[CH91] George E. Collins and Hoon Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12(3):299–328, September 1991.

[Cho88] Shang-Ching Chou. *Mechanical Geometry Theorem Proving*. Mathematics and its applications. D. Reidel Publishing Company, Dordrecht, Boston, Lancaster, Tokyo, 1988.

[CJ89] G. E. Collins and J. R. Johnson. Quantifier elimination and the sign variation method for real root isolation. In Gaston H. Gonnet, editor, *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation*, pages 264–271, Portland, OR, July 1989. ACM Press.

[CLO92] David Cox, John Little, and Donald O'Shea. *Ideals, Varieties and Algorithms.* Undergraduate Texts in Mathematics. Springer-Verlag, New York, Berlin, Heidelberg, 1992.

[Col75] George E. Collins. Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In H. Brakhage, editor, *Automata Theory and Formal Languages. 2nd GI Conference*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183. Springer-Verlag, Berlin, Heidelberg, New York, 1975.

[Dol00] Andreas Dolzmann. *Algorithmic Strategies for Applicable Real Quantifier Elimination.* Doctoral dissertation, Department of Mathematics and Computer Science. University of Passau, Germany, D-94030 Passau, Germany, March 2000.

[DS97a] Andreas Dolzmann and Thomas Sturm. Guarded expressions in practice. In Wolfgang W. Küchlin, editor, *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation (ISSAC 97)*, pages 376–383, Maui, HI, July 1997. ACM, ACM Press, New York, 1997.

[DS97b] Andreas Dolzmann and Thomas Sturm. Redlog: Computer algebra meets computer logic. *ACM SIGSAM Bulletin*, 31(2):2–9, June 1997.

[DS97c] Andreas Dolzmann and Thomas Sturm. Simplification of quantifier-free formulae over ordered fields. *Journal of Symbolic Computation*, 24(2):209–231, August 1997.

[DS98] Andreas Dolzmann and Thomas Sturm. Redlog. In *ICM 1998: Abstracts of Short Communications and Poster Sessions*, page 375. Geronimo GmbH, Rosenheim, 1998.

[DS03] Andreas Dolzmann and Andreas Seidl. REDLOG – first-order logic for the masses. *Journal of Japan Society for Symbolic and Algebraic Computation*, 10(1):23–33, 2003.

[DSS03] Andreas Dolzmann, Andreas Seidl, and Thomas Sturm. Efficient projection orders for CAD. Technical Report MIP-0401, FMI, Universität Passau, D-94030 Passau, Germany, January 2003.

[DSS04] Andreas Dolzmann, Andreas Seidl, and Thomas Sturm. *REDLOG User Manual*, April 2004. Edition 3.0.

[DSW98] Andreas Dolzmann, Thomas Sturm, and Volker Weispfenning. A new approach for automatic theorem proving in real geometry. *Journal of Automated Reasoning*, 21(3):357–380, 1998.

[DW] Andreas Dolzmann and Volker Weispfenning. Multiple object semilinear motion planning. To appear.

[DW00a] Andreas Dolzmann and Volker Weispfenning. Local quantifier elimination. In Carlo Traverso, editor, *Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation (ISSAC 2000), St Andrews, Scotland*, pages 86–94. ACM Press, New York, NY, August 2000.

[DW00b] Andreas Dolzmann and Volker Weispfenning. Local quantifier elimination. Technical Report MIP-0003, FMI, Universität Passau, D-94030 Passau, Germany, February 2000.

[EFT78] H. Ebbinghaus, J. Flum, and W. Thomas. *Einführung in die mathematische Logik*. Wissenschaftliche Buchgesellschaft Darmstadt, 1978.

[GR65] Robert C. Gunning and Hugo Rossi. *Analytic functions of several complex variables*. Prentice Hall, Inc., Englewood Cliffs, N. J., 1965.

[Hea05] Anthony C. Hearn. REDUCE: The first fourty years. In Andreas Dolzmann, Andreas Seidl, and Thomas Sturm, editors, *Proceedings of the 2005 Conference on Algorithmic Algebra and Logic (A3L 2005)*, pages 19–24, Passau, Germany, April 2005.

[Hen95] Eckhard Hennig. Rechnergestützte Dimensionierung analoger Schaltungen auf der Basis symbolischer Analyseverfahren. Technical report, Zentrum für Mikroelektronik, Universität Kaiserslautern, December 1995. Annual report on a research project.

[HH98] Eckhard Hennig and Thomas Halfmann. Analog Insydes tutorial. ITWM, Kaiserslautern, Germany, 1998.

[HLS97] Hoon Hong, Richard Liska, and Stanly Steinberg. Testing stability by quantifier elimination. *Journal of Symbolic Computation*, 24(2):161–187, August 1997. Special issue on applications of quantifier elimination.

[Hon90] Hoon Hong. An improvement of the projection operator in cylindrical algebraic decomposition. In Shunro Watanabe and Morio Nagata, editors, *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC 90)*, pages 261–264. ACM Press, New York, August 1990.

[Ioa99] Nikolaos I. Ioakimidis. REDLOG-aided derivation of feasibility conditions in applied mechanics and engineering problems under simple inequality constraints. *Journal of Mechanical Engineering (Strojnícky Časopis)*, 50(1):58–69, 1999.

[Jir97] Mats Jirstrand. Nonlinear control system design by quantifier elimination. *Journal of Symbolic Computation*, 24(2):137–152, August 1997. Special issue on applications of quantifier elimination.

[Kah75] William Kahan. An ellipse problem. *ACM SIGSAM Bulletin*, 9(3):11, August 1975. SIGSAM Problem #9.

[KW00] M'hammed el Kahoui and Andreas Weber. Deciding hopf bifurcations by quantifier elimination in a software component architecture. *Journal of Symbolic Computation*, 30(2):161–179, August 2000.

[Laz88] Daniel Lazard. Quantifier elimination: Optimal solution for two classical examples. *Journal of Symbolic Computation*, 5(1&2):261–266, February 1988.

[Laz94] Daniel Lazard. An improved projection for cylindrical algebraic decomposition. In C. L. Bajaj, editor, *Algebraic Geometry and its Applications: Collections of Papers from Shreeram S. Abhyankar's 60th Birthday Conference.* Springer, Berlin, 1994.

[Loo82] Rüdiger Loos. Computing in algebraic extensions. In Bruno Buchberger, George E. Collins, Rüdiger Loos, and Rudolf Albrecht, editors, *Computer Algebra: Symbolic and Algebraic Manipulation*, pages 173–178. Springer-Verlag, Wien, New York, second edition, 1982.

[McC84] Scott McCallum. *An Improved Projection Operation for Cylindrical Algebraic Decomposition.* Ph.D. dissertation, Computer Sciences Department, University of Wisconsin, Madison, 1984.

[McC87] Scott McCallum. Solving polynomial strict inequalities using cylindrical algebraic decomposition. Technical Report 87-25.0, RISC, Johannes Kepler University, Linz, Austria, 1987.

[McC88] Scott McCallum. An improved projection operation for cylindrical algebraic decomposition of three-dimensional space. *Journal of Symbolic Computation*, 5(1–2):141–161, February–April 1988.

[Mig91] Maurice Mignotte. *Mathematics for Computer Algebra.* Springer-Verlag, 1991.

[MS99] Maurice Mignotte and Doru Stefanescu. *Polynomials.* Springer-Verlag, 1999.

[Phi05] Nalina Phisanbut. Simplification of elementary functions. Talk at the 11th International Conference on Applications of Computer Algebra, August 2005.

[Que01] Boto von Querenburg. *Mengentheoretische Topologie.* Springer, 3., neubearb. u. erw. aufl. edition, 2001.

[Sei01] Andreas Seidl. Effiziente Realisierung reeller algebraischer Zahlen. Diploma thesis, Universität Passau, D-94030 Passau, Germany, August 2001.

[Sei02] Andreas Seidl. An efficient representation of algebraic numbers and polynomials for cylindrical algebraic decomposition (extendend abstract). In *Proceedings of the 8th Rhine Workshop on Computer Algebra*, pages 167–172, Mannheim, Germany, 2002.

[Sei04] Andreas Seidl. Extending real quantifier elimination by cylindrical algebraic decomposition to get answers. In V. G. Ganzha, E. W. Mayr, and E. V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing*, pages 423–431, St. Petersburg, Russia, jul 2004.

[SS03] Andreas Seidl and Thomas Sturm. A generic projection operator for partial cylindrical algebraic decomposition. In Rafael Sendra, editor, *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation (ISSAC 03), Philadelphia, Pennsylvania*, pages 240–247. ACM Press, New York, NY, 2003.

[Stu99a] Thomas Sturm. *Real Quantifier Elimination in Geometry.* Doctoral dissertation, Department of Mathematics and Computer Science. University of Passau, Germany, D-94030 Passau, Germany, December 1999.

[Stu99b] Thomas Sturm. Reasoning over networks by symbolic methods. *Applicable Algebra in Engineering, Communication and Computing*, 10(1):79–96, September 1999.

[Stu00] Thomas Sturm. An algebraic approach to offsetting and blending of solids. In V. G. Ganzha, E. W. Mayr, and E. V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing. Proceedings of the CASC 2000*, pages 367–382. Springer, Berlin, 2000.

[SW] Thomas Sturm and Volker Weispfenning. Algebra und logik. To appear in 2006.

[SW03] Werner M. Seiler and Andreas Weber. Deciding ellipticity by quantifier elimination. In V. G. Ganzha, E. W. Mayr, and E. V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing. Proceedings of the CASC 2003*, pages 347–355. Institut für Informatik, Technische Universität München, Passau, 2003.

[Tar48] Alfred Tarski. A decision method for elementary algebra and geometry. Prepared for publication by J. C. C. McKinsey. RAND Report R109, RAND, Santa Monica, CA, 1948.

[Usp48] J. V. Uspensky. *Theory of Equations.* McGraw-Hill Book Company, Inc., first edition, 1948.

[Wei86] Karl Weierstrass. *Abhandlungen aus der Functionenlehre von Karl Weierstrass.* Verlag von Julius Springer, Berlin, 1886.

[Wei88] Volker Weispfenning. The complexity of linear problems in fields. *Journal of Symbolic Computation*, 5(1&2):3–27, February–April 1988.

[Wei94] Volker Weispfenning. Parametric linear and quadratic optimization by elimination. Technical Report MIP-9404, FMI, Universität Passau, D-94030 Passau, Germany, April 1994.

[Wei01a] Volker Weispfenning. Semilinear motion planning among moving objects in REDLOG. In V. G. Ganzha and E. W. Mayr, editors, *Computer Algebra in Scientific Computing. Proceedings of the CASC 2001*, pages 541–553. Springer, Berlin, 2001.

[Wei01b] Volker Weispfenning. Semilinear motion planning in REDLOG. *Applicable Algebra in Engineering, Communication and Computing*, 12:455–475, June 2001.