University of Bath

PHD

Practical simplification of elementary functions using CAD

Phisanbut, Nalina

Award date:
2011

Awarding institution:
University of Bath

Link to publication

Practical Simplification of Elementary Functions using
Cylindrical Algebraic Decomposition

submitted by

Nalina Phisanbut

for the degree of Doctor of Philosophy

of the

University of Bath

Department of Computer Science

August 2011

COPYRIGHT

Signature of Author ....

Nalina Phisanbut

Acknowledgements

I would like to take this opportunity to convey my gratitude to many people who have helped and supported me through my entire PhD study. First of all, sincere thanks and deepest appreciation to my supervisors, Professor James H. Davenport and Dr. Russell J. Bradford, for their time, guidance, support, encouragement, the opportunity and many interesting and challenging discussions over the years. Without them, this thesis would not be possible. Thanks also due to EPSRC for the grant on the "Accurate Simplification of Elementary Functions" project of which this thesis is part, and Dr James C. Beaumont, a research officer on the project for many useful discussions; it has been an enjoyable few years collaborating with him.

I would like to extend my gratitude to Professor Marc Moreno Maza and Changbo Chen for their assistance on Maple, and Professor Christopher W. Brown for his assistance on QEPCAD. I am grateful to my examiners, Professor John Shackell and Professor Nicolai N. Vorobjov for their comments, which improved the exposition of the thesis. I would also like to thank Dr. John Power for sparing his valuable time to proofread my thesis.

Many thanks to everyone at the Department of Computer Science for their support and friendship. There are of course many other people who have commented on my work, offered words of encouragement or helped me in any way, however small it is, it meant a lot to me and gave me confidence and strength that kept me going.

Last but not least, special thanks to my family: my father, even though he may no longer be here to share this with me, and most importantly, my mother and sister for their endless love and support throughout my life, for listening to my various ramblings in the last few years, and for not laughing at my work. Thank you.

Abstract

‘Simplification’ is a key concept in computer algebra. But many simplification rules, such as $\sqrt{x}\sqrt{y} \rightarrow \sqrt{xy}$, are not universally valid, due to the fact that many elementary functions are multi-valued. Hence a key question is “Is this simplification correct?”, which involves algorithmic analysis of the branch cuts involved. The problem can, in principle, be reduced to connectedness questions and can be solved via Cylindrical Algebraic Decomposition (CAD).

In practice, while CAD is a powerful technique in real algebraic geometry, its application is far from straightforward. This thesis discusses how CAD can be applied to this problem, notably

- initial problem formulation;
- choice of variable ordering;
- problem pre-conditioning;
- decomposition post-conditioning;

for two different CAD algorithms — projection and lifting and triangular decomposition.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The problem concerning the simplification of elementary functions has long been recognised as one of the fundamental problem in computer algebra [Mos71], and the subject has been intensively researched for many years. However, even for the class of elementary functions, it has not been resolved in a satisfactory way. One of the major obstacles encountered when trying to simplify such functions, either by hand or by using a computer algebra system, is that many of the elementary functions are, in principle, multi-valued functions, while most users and computer programs tend to work with single-valued functions. The consequence is that many well-known identities may no longer be true everywhere in the complex plane when working with their single-valued counterparts: generally they only hold on specific regions of the complex plane (or higher-dimensional space for multivariate identities) defined by the branch cuts of the functions. However, we cannot ignore these identities, since in some contexts they may be valid.

This thesis is concerned with the accurate simplification of multi-valued elementary functions, and concentrates on the use of Cylindrical Algebraic Decomposition (CAD) as a tool for the analysis and decomposition of branch cuts in complex (or real) space.

1.1 Motivation

Simplification rules are often stated in terms of single-valued functions, for example $\sqrt{z^2} \to z$. These are not universally true, try $z = -1$ in this example and we have an old paradox,

$$-1 = \sqrt{(-1)^2} = \sqrt{1} = 1. \tag{1.1}$$

Less obvious are that

$$\sqrt{1-z}\sqrt{1+z} \to \sqrt{1-z^2}$$

is true for all $z \in \mathbf{C}$, but the apparently similar

$$\sqrt{z-1}\sqrt{z+1} \to \sqrt{z^2-1}$$

is not for all $z \in \mathbf{C}$ ($z = -2$ is a counterexample).

Many people are unaware of these difficulties: assuming the identities are true everywhere. This leads to incorrect simplifications. Not only human beings make this mistake, computer algebra systems which "ought to" be correct, also suffer from it. Computer algebra developers have struggled for years to find the best way to handle possible simplifications. They are torn between:

- being cautious and refusing to do simplification when there is uncertainty, as in Maple's `simplify`(...) and therefore failing to perform obvious simplifications, leaving the user with an unpleasant expression when a simpler one should have been given such as $\sqrt{1-z}\sqrt{1+z} \to \sqrt{1-z^2}$;

- adopting a lax attitude and using all possible algebraic rules regardless of the exceptional domain, as in Maple's `simplify`(..., `symbolic`), and therefore making incorrect simplifications such as $\sqrt{z-1}\sqrt{z+1} \to \sqrt{z^2-1}$.

1.2 Notation and Conventions

The operation of taking square root is fundamentally ambiguous. The solutions of $z^2 = 4$ are $\{-2, +2\}$, and the solutions of $z^2 = 2$ are real numbers $\alpha \in (-2, -1) | \alpha^2 = 2$ and $\beta \in (1, 2) | \beta^2 = 2$, conventionally denoted $\alpha = -\sqrt{2}$ and $\beta = \sqrt{2}$. When it comes to $z^2 = -1$, we cannot use bounding intervals on the real line, and trying to define bounding boxes in the complex plane requires us to know what i is: a vicious circle. Hence we fix, arbitrarily but consistently throughout this thesis, i to be one of the two complex numbers such that $i^2 = -1$.

Notation 1.2.1. *$\mathbf{Z}$, $\mathbf{N}$, $\mathbf{R}$ and $\mathbf{C}$ denote the integers, natural numbers, real numbers and complex numbers respectively.*

Notation 1.2.2. *The letters x, y, u, v and their decorated varieties denote real variables and z, w and their decorated varieties are complex variables, where $z = x + iy, w = u + iv$.*

Notation 1.2.3. *$\Re(z)$ and $\Im(z)$ denote the real and imaginary parts of z respectively, where $z = \Re(z) + i\Im(z)$.*

Notation 1.2.4. *Lower case variants, such as* log, arctan *and* $\sqrt{\ }$ *denote single-valued functions from $\mathbf{C} \to \mathbf{C}$. Capitalised variants, such as* Log, Arctan *and* Sqrt *denote multi-valued functions, regarded as mapping $\mathbf{C}$ into sets of values, and defined via their inverses. For example, $\mathrm{Log}(z) = \{w : \exp(w) = z\} = \{\log(z) + 2n\pi i \mid n \in \mathbf{Z}\}$ and $\mathrm{Sqrt}(z) = \{w : w^2 = z\} = \{\pm\sqrt{z}\}$.*

Our branch cut for log is defined in Section 4.1, and other elementary functions in Appendix A. Note that our $\sqrt{z}$ takes the value of the square root of z with non-negative real part, i.e. $\sqrt{z} = ((w : w^2 = z) \wedge (\mathrm{csgn}(w) = 1))$, where csgn is as defined in Appendix C.

Notation 1.2.5. *The arithmetic operations are assumed to act on these sets element-wise, so that $A + B = \{a + b \mid a \in A, b \in B\}$.*

1.3 Elementary Functions

Elementary functions, as defined by Liouville [Ros68], are the members of a field of functions that is generated by arithmetic operations and composition of exponentials, logarithms, and algebraic functions, and constant functions. Thus, polynomials, exponential, trigonometric and hyperbolic functions, are elementary functions in the sense of Liouville. They may be many-to-one as functions $\mathbf{C} \to \mathbf{C}$. Their inverses, i.e. algebraic roots, logarithmic, inverse trigonometric, and hyperbolic functions, are also elementary functions in the sense of Liouville, but are naturally one-to-many as functions $\mathbf{C} \to \mathbf{C}$, and hence require branch cuts. To distinguish these, the latter are sometimes referred to as the *inverse* elementary functions. While fractional powers are roots of algebraic equations, non-rational powers are obtained via exponential and logarithm, and therefore bring in the constant problem [Ric68], which we will not treat further in this thesis.

1.4 Genesis and Evolution of the Thesis

This thesis contributes towards a larger EPSRC[1] project on the simplification of elementary functions, supervised by Professor James H. Davenport and Dr. Russell J. Bradford. The main objective of the project is to develop a tool to verify if a proposed simplification of elementary functions is correct: providing either a proof of the simplification, or a counterexample. In [Kah87], Kahan gave an informal description of a process: how a mathematician would attack the problem. Our CAD based simplification system (described formally in Chapter 3) can be thought of as a formalisation of this process, making each step of the process algorithmic.

The main body of the thesis is on the CAD algorithm which drives the decomposition step of our verification system. At the start of the thesis, there was essentially only one way for computing the CAD, that is Collins' projection and lifting approach [Col75]. We investigate the feasibility of employing this algorithm within

[1] Under grant number GR/R84139/01.

our system. This is important since, although the algorithm has been around and has been an active research topic since the mid-1970s, most developments have been with respect to its original use in quantifier elimination.

During the course of the thesis, a different approach which is based on triangular decomposition [CMMXY09], has surfaced. Its implementation became available soon afterwards, allowing us to study its applicability to our system, for the later part of the project, in parallel with the original Collins's method.

1.5 Thesis Outline and Contributions

In Chapter 2 we provide an overview of the possible simplification problems. Also in this chapter, we examine the existing approaches to handle multi-valued functions and evaluate the limitations and difficulties in applying these approaches.

Chapter 3 introduces an algorithm to decide whether a proposed simplification of the elementary functions is correct. The algorithm uses multi-valued function simplification, followed by the decomposition method. The latter step can be seen as a generalisation of the process outlined in [Kah87]. Also in this step we propose to use *Cylindrical Algebraic Decomposition* as a tool for the analysis and decomposition of branch cuts in the complex plane.

In Chapter 4 we describe and develop machinery to translate the set of branch cuts of the proposed simplification into a semi-algebraic representation. The process involves transforming a problem in $\mathbf{C}^n$ into a problem in $\mathbf{R}^{2n}$ [Dav03], and de-nesting square roots present in the expression where necessary. In this chapter, we also suggest possible shortcut and ways in which it can be used to reduce the size of the set of branch cuts.

Chapter 5 gives the exposition of the Cylindrical Algebraic Decomposition technique. In this chapter we review the two existing approaches for constructing Cylindrical Algebraic Decomposition:

1. projection and lifting approach, i.e. Collins's original method, and its variant namely *Partial Cylindrical Algebraic Decomposition*;

2. triangular decomposition approach, a more recently developed approach due to Chen *et al.*

In chapter 6 we demonstrate the limitations of Cylindrical Algebraic Decomposition as a building block in our technology. We then propose an algorithm to manipulate the branch cut formulae such that more information is preserved when handing them to the Cylindrical Algebraic Decomposition algorithm. The algorithm can be used as a post-processing step to the method of Chapter 4. The idea and some of the results were presented in [PBD10].

Chapter 7 focuses on the last step of our system, i.e. the identity evaluation phase of the algorithm proposed in Chapter 3. We introduce the method of *adherence* to tackle the problems inherent in numerical evaluation on non-full dimensional regions. In this chapter we also suggest how to make this step more efficient by avoiding unnecessary testing of cells.

In Chapter 8 we outline our implementations and investigate the practical applicability of CAD within our framework, using the existing implementations: QEPCAD and Maple's `CylindricalAlgebraicDecompose` command. The experiment is based on a set of well-known elementary function identities.

Chapter 9 contains a summary and suggestions for potential future work.

Chapter 2

Simplification

Simplification is a key concept in computer algebra. It is not unusual in algebraic manipulation for expressions to be several lines, or pages, long, and a wrong choice of representation can easily swell the size of the problem. Algebraic simplification plays an important role in helping to keep the expressions comprehensible and make the system efficient and effective. However, simplification, even when dealing with the class of elementary functions, is more difficult than one usually realizes and has been one of the fundamental problems in computer algebra [Mos71].

In this chapter we first describe the simplification problems together with their causes. The latter part of the chapter concentrates on a particular simplification problem generated by elementary functions. We examine a number of techniques to tackle the problem and what they are lacking.

2.1 Simplification Problems

An ideal computer algebra system should simplify the results as much as practicable but only when it is correct to do so, i.e. never changing the values of the results. Despite the straightforwardness of the statement, the aim is difficult to formalize for all potential users. A perfect simplification system may not even exist due to the fact that simplification itself is not totally well-defined. To obtain and define precisely the definition of a simplification is an old and difficult problem [Mos71] that has not been fully resolved yet. How to determine what is the best simplification is a major question. Given two expressions A and B, what is meant by saying that "expression A is simpler than expession B"? Is simplification decided on the compactness of the formula, or on the number of primitives? For example

- Is $1 + x + x^2 + ... + x^{1000}$ "simpler" than $\frac{x^{1000}-1}{x-1}$?
- Is $\arctan(z)$ "simpler" than $\frac{1}{2i}(\log(1 + iz) - \log(1 - iz))$?

The answers to these questions will probably depend on the requirements of the situation. It is often the case that one of the equivalent expressions is more useful than another. However, it is not necessary that an expression which is considered simpler than another equivalent expression in one context would always be considered simpler in every context. For example,

$$\frac{x^5}{x^6 + 1}$$

is a more compact representation of an expression than the equivalent

$$\frac{\frac{1}{6}(6x^5)}{x^6 + 1}$$

and is often easier to perform algebraic manipulation. However, the latter expression suggests the substitution $y = x^6$ which, when integrating, yields

$$\int \frac{\frac{1}{6}}{y + 1} dy$$

a much simpler expression in comparison to the first expression.

An attempt to address this issue was given in [Car04]; we will not analysis this issue further. In this thesis, we will focus solely on another important ingredient for a good simplification system, that is how to correctly manipulate functional expressions involving elementary functions. The procedure is not entirely trivial. On one hand $\sqrt{x}\sqrt{y} = \sqrt{xy}$ is false in general ($x = y = -1$ is a counterexample), but on the other hand $\sqrt{1-z}\sqrt{1+z} = \sqrt{1-z^2}$ is true everywhere. How to translate such knowledge into usable code is a challenging and unresolved task. Current computer algebra systems provide no assistance when working with simplification rules which are "partially" or "almost" true. They would either make or not make simplifications, without giving information as to what the simplifications might have been, nor what the exceptional domains are. Faced with these problems, the user of a computer algebra system cannot rely on the system to perform simplification with full confidence.

2.2 Multi-Valued Functions

The fundamental problem underlying the complication is that the inverse elementary functions are what are commonly called "multi-valued functions", at least as $\mathbf{C} \to \mathbf{C}$. One sees phrases like

> A multi-valued function (or multiple-valued function [Mar65]) is a"function" that assumes two or more distinct values in its range for at least one point in its domain.

Of course, this is not a function $\mathbf{C} \to \mathbf{C}$ in the sense of Bourbaki [Bou70, page E.II.13], or indeed computer programming (the "table-makers point of view" [Dav10]).

The usual definition of functions $\mathbf{R}^+ \to \mathbf{R}$ such as log may be as integrals, $\log(z) = \int_1^z \frac{1}{x}$, series, $\log(z) = (z-1) - \frac{1}{2}(z-1)^2 + \cdots$, or inverse functions, $\log(z) = \exp^{-1}(z)$ (with $\log(1) = 0$). Any of these definitions of $f(z)$ may be extended to $\mathbf{C}$ by analytic continuation ([Mar67, page 257]) along paths P, but it is this continuation that gives rise to ambiguity: we have no longer defined $f(z)$, but rather $f_P(z)$ where P is a path from the starting value z_0 to z.

Definition 2.2.1. *[Mar65, page 217]*
Given a multi-valued function $w = f(z)$ with continuous single-valued branches defined on a domain G with closure $\overline{G}$, we say that the point $\zeta \in \overline{G}$ is a branch point *of $f(z)$ if there exists a neighbourhood $N(\zeta)$ such that one complete circuit around an arbitrary closed Jordan curve [Mar65, page 60] $\gamma \subset N(\zeta)$ with $\zeta \in I(\gamma)$, carries every branch [Mar65, page 213] of $f(z)$ into another branch of $f(z)$.*

Definition 2.2.2. Branch cuts *are lines drawn (semi-arbitrarily [Kah87, CDKS11]) between branch points of a locally-analytic function such that the function can be uniquely defined by analytic continuation along paths that do not cross the branch cuts.*

The placement of the branch cuts is not standardised and can differ from one computer algebra system (or mathematical textbook) to another. For example, one may rule that $-\pi < \arg(z) \leq \pi$, while it is also completely correct to rule that $0 \leq \arg(z) < 2\pi$. However, the fundamental problem is caused by the fact that branch cuts are necessary, and the problems exist regardless of the choice of the branch cuts.

Suppose, here, we follow the modern convention, i.e. $-\pi < \arg(z) \leq \pi$. This places the branch cut of logarithm along the negative real axis, and it is counter-clockwise continuous (CCC), i.e. continuous with the upper half-plane of the cut. Therefore, for $x < 0$

$$\lim_{y \to 0^+} \log(x + iy) = \log(x), \tag{2.1}$$

but

$$\lim_{y \to 0^-} \log(x + iy) \neq \log(x). \tag{2.2}$$

Consequently, we lose several mathematical identities that we are accustomed to. For example,

$$\log(\overline{z}) \overset{?}{=} \overline{\log(z)}, \tag{2.3}$$

is not always true, instead $\log(\overline{z}) = \overline{\log(z)} + 2\pi i$ on the cut, and

$$\log\left(\frac{1}{z}\right) \overset{?}{=} -\log(z), \tag{2.4}$$

is not always true, instead $\log\left(\frac{1}{z}\right) = -\log(z) + 2\pi i$ on the cut.

The problems are not limited to complex numbers as is often believed; algebra with the reals (as opposed to the entirety of $\mathbf{C}$) also encounters these problems due to the facts that:

- complex numbers can be introduced via operations on reals, as in putative equation (1.1) or $\arccos(2) \approx 1.317i$, and

- the well-known "equations" may be invalid even on the reals, as in

$$\arctan(x) + \arctan(y) \overset{?}{=} \arctan\left(\frac{x+y}{1-xy}\right), \tag{2.5}$$

the two sides differ by $\pm\pi$ if $xy \geq 1$.

However, the problems are more manifest on $\mathbf{C}$.

2.3 Handling Multi-Valued Functions

This section describes a number of approaches which have been proposed over the years to tackle the problems caused by the multi-valued nature of the functions.

2.3.1 Signed Zeros

Kahan [Kah87] suggested using the concept of *signed zero* [IEE85], the IEEE standards for floating-point arithmetic, which distinguishes positive zero from negative zero. The reason for zero to have two values is because all the necessary branch cuts which we have to take into consideration are on either the real or imaginary axes, therefore the side to which the branch cut adheres depends on the the sign of the real or imaginary part, including the sign of zero. For clarity, we write 0^+ for positive zero and 0^- for negative zero. By stating that $\log(x+0^+i) = \log(|x|) + \pi i$ and $\log(x + 0^-i) = \log(|x|) - \pi i$ for $x < 0$, the "signed zero" can be used to resolved the above problems; equation (2.2)[1], (2.3)[2] and (2.4) become equalities throughout. Unfortunately, the concept does not offer any help in the situation where the putative equation is on a set of measure greater than zero, such as (1.1) and (2.5).

[1] interpreting the x on the right as $x + 0^-i$

[2] where $\overline{0^+i} = 0^-i$

2.3.2 Auxiliary Functions

[CJ96] pointed out that the errors, which occured due to the existence of branch cuts, can be prevented by introducing a special function, which they named the *unwinding number* $\mathcal{K}$.

Definition 2.3.1. *Define the* unwinding number $\mathcal{K}(z)$ *by*

$$\mathcal{K}(z) = \frac{z - \log(\exp(z))}{2\pi i} = \left\lceil \frac{\Im(z) - \pi}{2\pi} \right\rceil \in \mathbf{Z}.$$

Note that the sign convention here is as defined in [CDJ$^+$01, BCD$^+$02], which is the opposite to that of [CJ96][3]

Remark 2.3.1. *The apparently equivalent definition* $\left\lfloor \frac{\Im(z)+\pi}{2\pi} \right\rfloor$ *differs precisely on the branch cut for* $\log(z)$ *as applied to* $\exp(z)$*.*

The putative equation

$$\log(z_1 z_2) \stackrel{?}{=} \log(z_1) + \log(z_2)$$

can be rescued as

$$\log(z_1 z_2) = \log(z_1) + \log(z_2) - 2\pi i \mathcal{K}(\log(z_1) + \log(z_2)),$$

and (2.3) can be rescued as

$$\log(\overline{z}) = \overline{\log(z)} - 2\pi i \mathcal{K}\left(\overline{\log(z)}\right).$$

Note that, as part of the algebra of $\mathcal{K}$,

$$\mathcal{K}\left(\overline{\log(z)}\right) = \mathcal{K}(-\log(z)) \neq \mathcal{K}\left(\log \frac{1}{z}\right).$$

[3] $\mathcal{K}$ is defined in [CJ96] by $\mathcal{K}(z) = \left\lfloor \frac{\pi - \Im(z)}{2\pi} \right\rfloor$: the authors of [CJ96] recanted later to keep the number of -1s occurring in the formulae to a minimum.

$\mathcal{K}(z)$ depends only on the imaginary part of z, $\mathcal{K}(\log(z)) = 0$ for all z, and

$$\mathcal{K}(-\log(z)) = \begin{cases} -1 & z \text{ real negative} \\ 0 & \text{elsewhere.} \end{cases}$$

This is therefore a "fingerprint" for the branch cut.

Some correct identities for elementary functions using $\mathcal{K}$ are given in Table 2.1.

$$\begin{aligned}
z &= \log(\exp(z)) + 2\pi i\mathcal{K} \\
\mathcal{K}(a\log(z)) &= 0\ \forall z \in \mathbf{C} \quad \text{if and only if } -1 < a \leq 1 \\
\log(z_1) + \log(z_2) &= \log(z_1 z_2) + 2\pi i\mathcal{K}(\log(z_1) + \log(z_2)) \\
a\log(z) &= \log(za) + 2\pi i\mathcal{K}(a\log(z)) \\
z^{ab} &= (z^a)^b \exp(2\pi i\mathcal{K}(a\log(z)))
\end{aligned}$$

Table 2.1: Some correct identities for logarithms and powers using $\mathcal{K}$

The unwinding number approach is a two-step approach. First the unwinding number is inserted to preserve the mathematical correctness while performing algebraic manipulation and cancellation of the elementary functions. Generally the insertion of the unwinding number is done using one of the following mechanisms:

- $\log\left(\frac{z_1}{z_2}\right) = \log(z_1) - \log(z_2) - 2\pi\mathcal{K}(\log(z_1) - \log(z_2))$, or
- $\log\left(\frac{1}{z}\right) = -\log(z) - 2\pi\mathcal{K}(-\log(z))$, or
- $\log(\exp(z)) = z - 2\pi i\mathcal{K}(z)$.

The formulae for other elementary functions are given in Appendix C.

The result is then simplified by removing the unwinding number, where this is possible. This is harder than inserting the unwinding number, which can be done algorithmically. The values of the unwinding number might reduce to any of the following possibilities:

- an unwinding number may be identically zero, or
- an unwinding number may be zero everywhere except on certain branch cuts in the complex plane, or
- an unwinding number may divide the complex plan into two regions, one where it is non-zero and one where it is zero, or
- an unwinding number may correspond to the usual $+n\pi, n \in \mathbf{Z}$ of many trigonometric identities.

The unwinding number has several attractive features, namely:

- an unwinding number is single-valued, and
- an unwinding number is integer-valued: little accuracy is necessary in practice to evaluate it, and
- an unwinding number is familiar in the sense that "everyone knows" that the multi-valued logarithm can be written as

  $$\text{the principal branch} + 2\pi i k,\ k \in \mathbf{Z},$$

  and
- an unwinding number can be computed by a formula not involving logarithms.

However, using the unwinding number in computer algebra systems does have a number of disadvantages including:

- inserting the unwinding number essentially doubles the size of the printed output, giving the answer in the form $Y + 2\pi i \mathcal{K}(Y)$;
- eliminating the unwinding number cannot be done algorithmically, the rules are essentially geometric and require decisions to be made on the grounds of where its arguments are in $\mathbf{C}$.

Other mathematical functions similar to the unwinding number have been defined several times, including

- the adjustment, adj [Bra93]

$$\mathrm{adj}(z) = -2\pi i \left\lfloor \frac{\Im(z)}{2\pi} \right\rfloor,$$

- the imaginary quotient, Imq(z) [Asl96]

$$\mathrm{Imq}(z) = \left\lceil \frac{\Im(z) + \pi}{2\pi} \right\rceil \in \mathbf{Z},$$

- the unln [Pat96]

$$\mathrm{unln}(z) = \ln(\exp(z)) - z.$$

2.3.3 Multi-Valued Functions as Set Valued Functions

This approach is to accept the multi-valued nature of the elementary functions.

Notations 1.2.4 and 1.2.5 validate the algebraic rules of simplification, which otherwise might not be true in single-valued cases. For example,

$$\mathrm{Sqrt}(x)\,\mathrm{Sqrt}(y) = \mathrm{Sqrt}(xy),$$

whereas

$$\sqrt{x}\sqrt{y} \overset{?}{=} \sqrt{xy}$$

is not true: $x = y = -1$ is a counterexample.

Some multi-valued identities are given in Table 2.2, along with a counterexample to the single-valued counterpart, if it exists, and the relative dimensionality of the counterexample space.

Multi-valued rule	**Single-valued counterexample**	**Counterexample's dimensionality**
$\mathrm{Log}(z_1) + \mathrm{Log}(z_2) = \mathrm{Log}(z_1 z_2)$	$z_1 = z_2 = -1$	lower (over $\mathbf{C}$)
$\mathrm{Log}(z_1) - \mathrm{Log}(z_2) = \mathrm{Log}\left(\frac{z_1}{z_2}\right)$	$z_1 = 1, z_2 = -1$	lower (over $\mathbf{C}$)
$-\mathrm{Log}(z) = \mathrm{Log}\left(\frac{1}{z}\right)$	$z = -1$	lower (over $\mathbf{C}$)
$\overline{\mathrm{Log}(z)} = \mathrm{Log}(\overline{z})$	$z = -1$	lower (over $\mathbf{C}$)
$\mathrm{Sqrt}(z_1)\,\mathrm{Sqrt}(z_2) = \mathrm{Sqrt}(z_1 z_2)$	$z_1 = z_2 = -1$	lower (over $\mathbf{C}$)
$\mathrm{Sqrt}(z)^2 = \{z\}$	no counterexample	
$\mathrm{Sqrt}(z^2) = \{\pm z\}$	not single-valued	
$\mathrm{Arctan}(x) + \mathrm{Arctan}(y) = \mathrm{Arctan}\left(\frac{x+y}{1-xy}\right)$	$x = y = 2$	full (over $\mathbf{R}$)
$\tan(\mathrm{Arctan}(x)) = \{x\}$	no counterexample	
$\mathrm{Arctan}(\tan(x)) = \{x + n\pi \mid n \in \mathbf{Z}\}$	$x = \pi$	full (over $\mathbf{R}$)
$\sin(\mathrm{Arcsin}(x)) = \{x\}$	no counterexample	
$\mathrm{Arcsin}(\sin(x)) = \{x + 2n\pi \mid n \in \mathbf{Z}\} \cup$ $\{-x + (2n+1)\pi \mid n \in \mathbf{Z}\}$	$x = \pi$	full (over $\mathbf{R}$)

Table 2.2: Multi-valued simplification rules and single-valued counterexamples

There are, however, a few caveats which must be mentioned.

- Not all multi-valued functions have simple expressions. For example,

  $$\mathrm{Arcsin}(x) = \{\arcsin(x) + 2n\pi \mid n \in \mathbf{Z}\} \cup \{\pi - \arcsin(x) + 2n\pi \mid n \in \mathbf{Z}\}.$$

- Cancellation is no longer trivial, since in principle, rather than being zero,

  $$\mathrm{Arctan}(x) - \mathrm{Arctan}(x) = \{n\pi \mid n \in \mathbf{Z}\},$$

  or more confusingly,

  $$\begin{aligned}\mathrm{Arcsin}(x) - \mathrm{Arcsin}(x) &= \{2n\pi \mid n \in \mathbf{Z}\} \cup \{2\arcsin(x) - \pi + 2n\pi \mid n \in \mathbf{Z}\} \\ &\qquad \cup \{\pi - 2\arcsin(x) + 2n\pi \mid n \in \mathbf{Z}\} \\ &= \{2n\pi \ n \in \mathbf{Z}\} + \{0, 2\arcsin(x) - \pi, \pi - 2\arcsin(x)\}.\end{aligned}$$

  Note that $\mathrm{Arcsin}(x) - \mathrm{Arcsin}(x)$ still depends on x, unlike the case of $\mathrm{Arctan}(x) - \mathrm{Arctan}(x)$.

- Some identities take on somewhat unexpected forms in this context. For example, the multi-valued representation of

  $$\arcsin(z) \overset{?}{=} \arctan\left(\frac{z}{\sqrt{1-z^2}}\right),$$

  (which is valid except on the branch cuts), is either

  $$\mathrm{Arcsin}(z) \subset \mathrm{Arctan}\left(\frac{z}{\mathrm{Sqrt}(1-z^2)}\right),$$

  or

  $$\mathrm{Arcsin}(z) \cup \mathrm{Arcsin}(-z) = \mathrm{Arctan}\left(\frac{z}{\mathrm{Sqrt}(1-z^2)}\right).$$

- There is an aliasing problem. For example,

  $$\mathrm{Log}(z^2) \neq 2\,\mathrm{Log}(z), \tag{2.6}$$

if $2\,\mathrm{Log}(z)$ is interpreted as $\{2w \mid \exp(w) = z\}$, since $2\,\mathrm{Log}(z) = \{2\log(z) + 4k\pi i \mid k \in \mathbf{Z}\}$, but $\mathrm{Log}(z^2) = \{2\log(z) + 2k\pi i \mid k \in \mathbf{Z}\}$. For equation (2.6) to be valid, $2\,\mathrm{Log}(z)$ needs to be interpreted as $\mathrm{Log}(z) + \mathrm{Log}(z)$:

$$\mathrm{Log}(z^2) = \mathrm{Log}(zz) = \mathrm{Log}(z) + \mathrm{Log}(z) = 2\,\mathrm{Log}(z).$$

- Putative equations in which ambiguity disappears, such as $\sqrt{z^2} \overset{?}{=} z$, cannot be encoded as $\mathrm{Sqrt}(z^2) = z$, since this is trying to equate a set and a number, but rather have to be encoded as $z \in \mathrm{Sqrt}(z^2)$.

- It is unclear of what will happen when multi-valued functions are replaced by the corresponding single-valued functions of numerical programming languages.

2.3.4 Riemann Surfaces

Riemann surfaces are often used to represent multi-valued functions. They allow us to visualize the multi-valueness graphically [AS60, Tro97, CJ98]. The idea of the Riemann surface is to transform the one-to-many mapping to a one-to-one mapping by changing the domain of definition from **C** to a more complicated object, say D, and thus the multi-valued functions are made single-valued.

Consider the simple case of the Riemann surface of the square root as an example. Unlike the other approaches described above, where the attention is on the behaviour of $w = \sqrt{z}$ in the w-plane, the Riemann surface studies the behaviour of the mapping $z = w^2$ in the z-plane.

Let $w = r\exp(i\theta), -\pi < \theta \leq \pi$, then we have $z = r^2\exp(2i\theta)$. If we take two copies of the z-plane and cut along the negative real axis (Figure 2-1), then glue together the edges of these cuts so that B_1 is joined to A_2 and B_2 to A_1, we get the intuitive Riemann surface of $w = \sqrt{z}$. Starting at the point $w = r$ in the w-plane and travelling anti-clockwise along a circular path to the point $w = ir$, the image points run from the point r^2 to the point $-r^2$ on the first copy of the z-plane. Next if we continue along the circular path on the w-plane from the point ir to the point $-ir$, the image points are now on the second copy of the

z-plane going one full circle from the point $-r^2$ through the point r^2 back to the point $-r^2$. Finally, if we travel on the w-plane from the point $-ir$ to the point r, the image points are now back on the first copy of the z-plane and run from the point $-r^2$ to the point r^2. This makes the multi-valued function $w = \sqrt{z}$ single-valued on the union of the two copies of the z-plane.

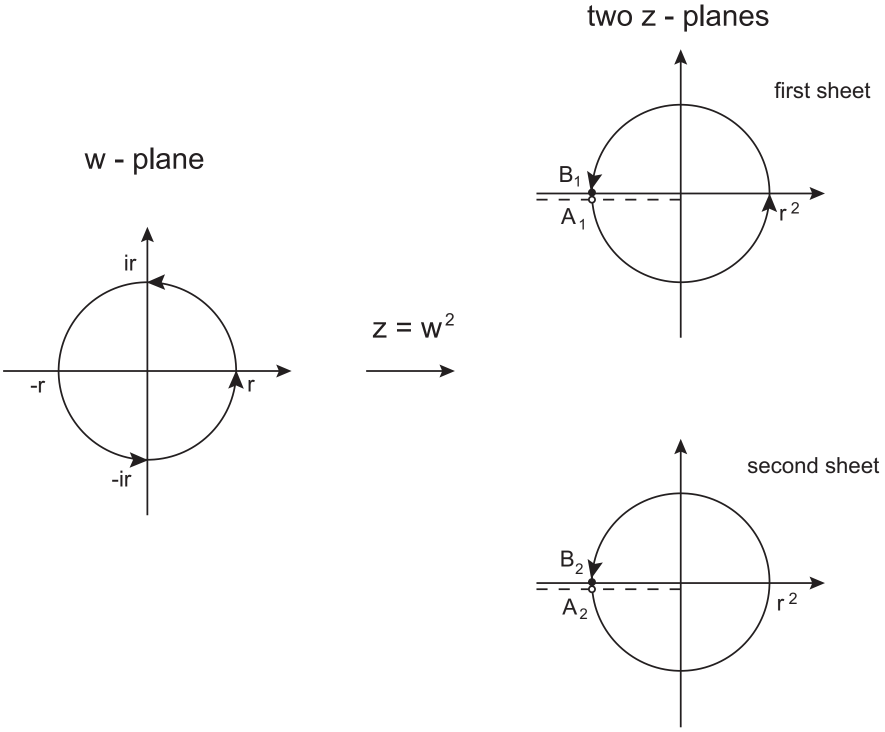


Figure 2-1: The Riemann surface for $z = w^2$

Similarly, we can study the logarithm, $w = \ln(z)$. In this case, for each $k \in \mathbf{Z}$, choose a copy S_k of the z-plane. For $z = R\exp(i\phi), -\pi < \phi \leq \pi, z \neq 0$, we define

$$\mathrm{Ln}(z) = \ln(R) + i\phi + 2k\pi i \text{ on } S_k, k \in \mathbf{Z}.$$

Make a cut along the negative real axis on the copy $S_k, k \in \mathbf{Z}, \forall k$. Then glue together the copy S_k with the copy S_{k+1} for all k. The resulting Riemann surface is an “infinite round staircase” (Figure 2-2).

Therefore, to use Riemann surfaces to study the elementary functions, one requires to know what Riemann sheet they are on. Furthermore, the Riemann surface depends upon the function being considered, for example the Riemann

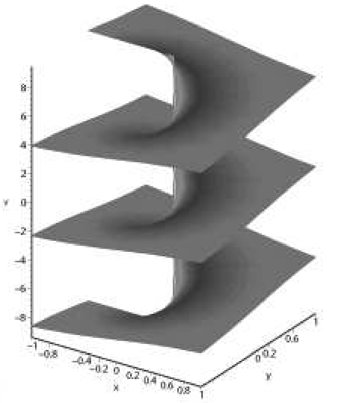


Figure 2-2: The Riemann surface for $w = \ln(z)$

sheet for $\log(z)$ is different from the Riemann sheet for $\arctan(z)$. Thus it is unclear how to use Riemann surfaces computationally when considering multi-valued functions, such as equation (2.5), since we have merely moved the problem to that of relating the Riemann surfaces for x and y to that of $\frac{x+y}{1-xy}$.

2.4 Summary

In this chapter we have introduced simplification problems. In particular, we have shown the difficulty when handling simplification of inverse elementary functions. The problem stems from the fact that these inverse elementary functions are multi-valued. Hence the well-known simplification rules may not be true everywhere when working with their single-valued counterparts.

We have given an overview of several treatments of the inherently multi-valued functions, and showed that none of these have provided a complete solution to the problem.

Chapter 3

Simplification System

The primary concern of this thesis is to develop a practical tool to analyse whether simplification in elementary functions, which is naively correct, is actually mathematically correct. This chapter outlines the algorithms that we are using to build such a system.

There have been several discussions on the best way to deal with the problem of multi-valued functions [Kah87, Bra93, Asl96, CJ96, Pat96, CDJ$^+$01, BCD$^+$02, Dav03]. However, most techniques have remained largely unimplemented. To the best of our knowledge, the first major implementation of a verification system for elementary function identities was made in [DF94]. The paper took what we shall call the *decomposition method* (Algorithm 1), which was first described informally in [Kah87].

By defining a (multi-valued) function f, starting from an element e_0 with a series expansion about z_0, we mean [Mar67, page 269 note 22], given any $z_1 \in G$, $z_1 \neq z_0$, let $L \subset G$ be a curve joining z_0 to z_1, and let e_1 be the result of continuing e_0 along L, with disk $K_1 \subset G$ and series

$$\sum_{n=0}^{\infty} a_n(z - z_1)^n \quad (z \in K_1).$$

Then $f(z_1) = a_0$.

Theorem 3.0.1. The Monodromy Theorem *[Mar67, page 269].*
Given a simply connected domain G, let e_0 be an element with disk $K \subset G$ centred at z_0. Suppose e_0 can be continued along every continuous curve [Mar65, page 59] in G, thereby defining a function $f(z)$ at every point of G. Then $f(z)$ is single-valued on G.

Algorithm 1 Decomposition method

```
1: Calculate all the branch cuts of the proposed identity.
2: Decompose $\mathbf{C}$ (or $\mathbf{C}^n$) with respect to the branch cuts into cells and find a
   sample point in each cell.
3: Evaluate the identity on each connected component using the obtained sample
   point, thereby conclude whether the identity is true or not on that entire
   region by the *Monodromy theorem.*
```

For each step of Algorithm 1, [DF94] proceeded as follow.

1. The branch cuts are represented by a triple (r_0, r_1, f) with $r_0, r_1 \in \mathbf{R} \cup \{\pm\infty\}$ and the function $f : [r_0, r_1] \mapsto \mathbf{C}$. The branch cuts of a function $E(z) = f(g(z))$ are determined by computing the inverse function $\phi(z) = g^{-1}(z)$, then applying ϕ to the interval(s) representing the cut(s) for f.

2. (a) Determining the decomposition of $\mathbf{C}$ involved eliminating the interacting branch cuts, before traversing the branch regions. The latter requires computing the derivatives to sort the branch cuts and segments which have non-infinite and infinite endpoints respectively.

   (b) Draw a line L which bisects the angle between two cuts on the boundary of region R, both with the same endpoint e on the boundary of R. Compute all intersections of L with the boundary of R. Select e' such that e' is the nearest point to e towards the interior of R.

3. Evaluate the identity numerically using the sample points.

The procedure was implemented in Mathematica and it was capable of handling many simple examples. However, the approach suffers from a few limitations.

- Only one complex variable is allowed.

- It may not be possible to compute the inverse function $\phi(z)$. For example, the function $\sqrt{p(z)}$ where $p(z)$ is a polynomial of degree 5 or more cannot generally be expressed in terms of radicals.
- The traversal will not locate the regions which are contained entirely inside other regions.

In [BD02], Bradford and Davenport proposed another algorithm to decide the correctness of simplification of elementary function identities. Like [DF94], the algorithm is based on the decomposition method (Algorithm 1), but they proposed using Cylindrical Algebraic Decomposition (CAD) in the second step. In addition, they introduced the use of multi-valued functions to find the candidate for a proposed simplification, before proceeding to the decomposition method. The overall approach proceeds as follow.

1. Find a possible simplification g of the candidate f.
2. Check algebraically that the simplification is correct in the multi-valued sense as defined in Notations 1.2.4 and 1.2.5.
3. Check, following the decomposition method, that the simplification is correct in the single-valued, branch cut respecting, sense.
   (a) Determine all branch cuts of f and g, representing the set of branch cuts by semi-algebraic equations in $\Re(z)$ and $\Im(z)$.
   (b) Determine a decomposition D of $\mathbf{C}$ (or $\mathbf{C}^n$), viewed as $\mathbf{R}^2$ (or $\mathbf{R}^{2n}$), according to the branch cuts by using Cylindrical Algebraic Decomposition.
   (c) Evaluate[1] the identity on each component C of the decomposition D (see Chapter 7).

Remark 3.0.1. *Analysis of the putative equation in terms of multi-valued functions, shows how small a non-zero discrepancy can be, so a "sufficiently accurate" numerical evaluation will do — see [BD02].*

[1] Purely numerical verification may not work on branch cuts, since the smallest error may result in us choosing a sample point from an adjacent cell instead.

This thesis is based upon Bradford and Davenport's approach, focusing particularly on the last step. Note that it is possible to combine steps 1 and 2 in practice, and they can be done by using a standard simplifier such as Maple's `simplify`(..., `symbolic`).

3.1 Necessary Conditions

The CAD based decomposition method works for the cases where:

1. the only nested elementary functions which appear inside other elementary functions are square roots, and
2. the inverse elementary functions only appear in the numerator.

The first restriction is to ensure that the branch cuts can be described by semi-algebraic sets so that CAD is feasible. The second restriction is necessary so that numerical sample point testing is possible [BD02].

3.2 Practical Restrictions

Most of the well-known identities, such as those found in [AS64] or as one is likely to meet in practice, are of one or two complex variables. Hence we shall focus on these cases, although occasionally we will choose to discuss expressions involving real variables as well. Unlike the complex problems where the branch cuts in complex space are always lower-dimensional, i.e. lines on the complex plane, the branch cuts of the real cases (functions on real domains) may be lower-dimensional or full-dimensional. For example, the branch cut of

$$h = \arctan(x) + \arctan(y) - \arctan\left(\frac{x+y}{1-xy}\right)$$

is a point at infinity, therefore it is a lower-dimensional problem.

On the other hand, the branch cut for logarithm is a half line, $(-\infty, 0]$, which is full-dimensional in real space. Hence, the set of branch cuts in the real case of

$$h = \log(x) + \log(y) - \log(xy),$$

includes the branch cut of $\log(x)$ which is full-dimensional in x-space, the branch cut of $\log(y)$ which is full-dimensional in y-space, and the product of the two, which is full-dimensional in real (x, y)-space.

In this thesis, our primary focus will be on the cases involving lower-dimensional branch cuts. The same methodology as described here can be applied to the full-dimensional problems to compute the set of branch cuts of the proposed identity and construct the decomposition, but different interpretations may be required to analyse the decomposition. Since the main interest of this thesis is the decomposition, we will not discuss further how to use the decomposition in these cases.

3.3 Summary

We have presented an algorithm for deciding whether a proposed simplification of elementary functions is correct in the presence of branch cuts. The algorithm uses multi-valued function simplification followed by verification that the branches are consistent, by means of the decomposition method using CAD. The details of each step of the algorithm will be described in later chapters.

Chapter 4

Branch Cuts

In this chapter, we describe the method for computing the set of branch cuts for the class of inverse elementary functions in complex variables. We represent the branch cuts in the form of semi-algebraic sets, which opens the door to powerful tools from real algebraic geometry [BCD$^+$02]. We begin the chapter with the branch cut conventions and some restrictions which will be used throughout the thesis.

4.1 Convention

The *existence* of branch cuts is forced on us by the mathematics of analytic functions. Their precise *positioning* is a matter on convention [Kah87, CDKS11]; however, it is important to make a consistent choice. In this thesis, the definitions and the choice of branch cuts of the inverse elementary functions are as given in [CDJW00], which repeats, with more justification, the definitions given in [AS64]. If one made a different choice, some of the *examples* in this thesis would be different, but the theorems and arguments would remain valid.

Definition 4.1.1. *Define the branch cut of* $\log(z)$ *to be* $(-\infty, 0]$ *and rule that*

$$-\pi < \Im \log(z) \leq \pi. \tag{4.1}$$

This specifies the principal value of the logarithm function to be

$$\log(z) = \log|z| + i\arg(z)$$

with $-\pi < \arg(z) \leq \pi$. Thus the branch cut of logarithm is

$$\Re(z) < 0 \wedge \Im(z) = 0.$$

The definitions and the branch cuts of other inverse elementary functions are derived from the definition and the branch cut for the logarithm function, and are summarized in Appendices A and B respectively.

Note that the precise choice of $<$ and $\leq$ in (4.1) specifies the *adherence rule* of what log actually does *on* the branch cut: it adheres to the upper half-plane (see Figure 7-1).

4.2 Semi-Algebraic Representation

The branch cuts of square roots, logarithm, and the inverse of trigonometric and hyperbolic functions, which are described in Appendix B, can be represented in terms of sets of equality and inequality (a special case of a semi-algebraic set — see [BR90], [BCR98, Chapter 2])

$$\{(s\ \sigma_1\ c_1) \wedge (t\ \sigma_2\ c_2)\},$$

where $s, t \in \mathbf{R}$ and represent the real and imaginary parts respectively, $\sigma_i \in \{=, <, \leq, >, \geq\}$, and $c_i \in \{0, \pm 1\}$ for $i = 1, 2$. For example, the branch cut for $\arctan(z) : (-i\infty, -i] \cup [i, i\infty)$ is the set

$$\{(s = 0) \wedge (t \geq 1)\} \vee \{(s = 0) \wedge (t \leq -1)\}.$$

Moreover, for every function, either $s = 0$ or $t = 0$ in each branch cut formula.

Remark 4.2.1. *In real cases, $t = 0$ always.*

4.3 Branch Cut Computation

4.3.1 Rational Functions

For simplicity, suppose that the function $f(g(z))$ where f is an inverse elementary function and $g \in \mathbf{C}(z)$, with $g(z) \neq 0$, contains only one complex variable and no square roots inside f.

We proceed as follows:

- re-writing g, if necessary, as $g(z) = \frac{p(z)}{q(z)}$, where $p, q \in \mathbf{C}[z]$;
- substituting $z \to x + iy$, where $x, y \in \mathbf{R}$;
- multiplying $g(z)$ by the expression $\frac{\Re(q)-i\Im(q)}{\Re(q)-i\Im(q)}$, with $q \notin \mathbf{R}[z]$, to obtain the equation:

$$g(z) = \frac{\Re(p)\Re(q) + \Im(p)\Im(q) + i(\Re(q)\Im(p) - \Re(p)\Im(q))}{(\Re(q))^2 + (\Im(q))^2}. \tag{4.2}$$

For each branch cut of function f:

- Setting up a system of the following form:

$$\begin{aligned} g(z) &= s + it && (4.3) \\ s \quad &\sigma_1 \quad c_1 && (4.4) \\ t \quad &\sigma_2 \quad c_2. && (4.5) \end{aligned}$$

  The σ_i and c_i depend on the function f.

- Without loss of generality, suppose the function f requires that $s = 0$. Then combining (4.3) and (4.4) gives

$$\begin{aligned} g(z) &= it && (4.6) \\ t \quad &\sigma_2 \quad c_2. && (4.7) \end{aligned}$$

Substituting (4.2) into (4.6) and equating the real and imaginary parts give

$$\begin{aligned}\frac{\Re(p)\Re(q)+\Im(p)\Im(q)}{(\Re(q))^2+(\Im(q))^2} &= 0 \qquad (4.8)\\ \frac{\Re(q)\Im(p)-\Re(p)\Im(q))}{(\Re(q))^2+(\Im(q))^2} &= t \qquad (4.9)\end{aligned}$$

$$t \quad \sigma_2 \quad c_2.$$

- Combining (4.5) and (4.9) gives

$$\begin{aligned}\Re(p)\Re(q)+\Im(p)\Im(q) &= 0 \qquad (4.10)\\ \frac{\Re(q)\Im(p)-\Re(p)\Im(q))}{(\Re(q))^2+(\Im(q))^2} &\quad \sigma_2 \quad c_2. \qquad (4.11)\end{aligned}$$

- Since the denominator of (4.2) is always strictly positive, we can multiply both sides by the denominator without changing the sign of σ_2,

$$\begin{aligned}\Re(p)\Re(q)+\Im(p)\Im(q) &= 0 \qquad (4.12)\\ \Re(q)\Im(p)-\Re(p)\Im(q) &\quad \sigma_2 \quad c_2(\Re(q))^2+(\Im(q))^2. \qquad (4.13)\end{aligned}$$

In addition, we also need to consider the set $\{z \mid q(z)=0\}$.

The case where the function f requires $t=0$ is analogous to the above.

The s and t in the original set up system can always be eliminated. The output is a semi-algebraic set with two equations in two real variables.

For the more general case of $h(f_1(g_1(z)),\ldots,f_n(g_n(z)))$, with h is a rational function in $f_i(g_i(z)), i \in 1,\ldots,n$, the above procedure is applied to each $f_i(g_i(z))$, then the union of the branch cuts is taken. This process can give rise to removable branch cuts — see Section 4.4. The procedure can be further extended to handle an arbitrary number of complex variables.

4.3.2 Nested Roots

The method of Section 4.3.1 can be generalised to cover all functions allowed by our restrictions, that is for $f(g(z))$ where f is an inverse elementary function and $g(z)$ contains square roots of arbitrary depth but no other elementary or inverse elementary functions. One can build the set of branch cuts for $f(g(z))$ by working, for convenience although not necessity, starting with g and working outwards. Thus, we first calculate the set of branch cuts for each square root of g, and then take the union. The final step requires computation of the set of branch cuts for the outermost f and adjoining this to the set of branch cuts for all g. The procedure here involves more work than that of Section 4.3.1, since once the system corresponding to equations (4.3), (4.4), and (4.5) has been set up, the square roots in equation (4.3) need to be removed. This may be achieved using one of the methods which we will describe next.

Notation 4.3.1. *Suppose $f(g(z))$ is as described earlier in this section. Without loss of generality, suppose further that the function f requires that $t = 0$. Thus,*

$$g(z) = s. \tag{4.14}$$

- **Squaring**

  A root-free expression may be derived by repeatedly squaring up the equation (4.14), rearranging if necessary. Unfortunately, for a general $g(z)$ where $g(z)$ contains 5 or more root monomials, the above procedure may be infeasible.

- **Resultant**

  Let g is a polynomial expression of the terms $\sqrt[n_1]{p_1}$, ..., $\sqrt[n_m]{p_n}$ with $p_i \in \mathbf{C}(z)$ and $m \in \mathbf{N}$.

  In line 4 of Algorithm 2, resultant$(a, b)_\phi$ denotes calculating the resultant of polynomials a and b with respect to variable ϕ.

- **Gröbner basis**

  This procedure eliminates the intermediate "square root" variables as in the resultant method [BBDP04].

Algorithm 2 Nested roots elimination: Resultant

```
Input: $g - s = 0$
Output: Root-free $g'$.
Clear denominators from $g - s = 0$ to obtain $g' = 0$.
Substitute $\omega_i = \sqrt[n_i]{p_i}$ for each distinct $\sqrt[n_i]{p_i}$ in $g'$.
for $i = 0$ to $m$ do
    $g' \leftarrow \text{resultant}(g', \omega_i^{n_i} - p_i)_{\omega_i}$
end for
```

Once we arrive at a root-free expression, we can apply the method of Section 4.3.1. However, given g in n variables, the above procedures produce g' in $n+1$ variables: that is we introduce an *extraneous* variable, i.e. s above. For simple cases, we may be able to eliminate this extraneous variable after equating the real and imaginary parts as described in Section 4.3.1. For more complicated cases, quantifier elimination such as CAD (see Section 5.1) can be used to produce an equivalent formula without the extraneous variable. However, the downside of this is that extra computational time will be added to the overall algorithm.

Once again, as in Section 4.3.1, the method of this section can be extended to include the case of $h(f_i(g_i(z)))$ where h is a rational function in $f_i(g_i(z))$, $i = 1, \ldots, n$, and to the case of arbitrarily many complex variables.

4.3.3 Spurious Branch Cuts

A side effect of removing the square roots is that it introduces spurious branch cuts.

Suppose $f(g(z))$ is as in Section 4.3.2, and

$$g(z) = \sqrt{p(z)}. \tag{4.15}$$

If the squaring method is used and we naively square (4.15). We derive

$$g^2(z) = p(z).$$

This, however, represents the solution set of $g(z) = \pm\sqrt{p(z)}$, that is the solutions for both branches of the square root, instead of just the desired principal branch.

For example,

$$h(z) \overset{?}{=} h_1(z) - h_2(z) = \operatorname{arccosh}(\sqrt{z}) - 2\log\left(\sqrt{\frac{\sqrt{z}+1}{2}} + \sqrt{\frac{\sqrt{z}-1}{2}}\right),$$

which has branch cuts for both $h_1(z)$ and $h_2(z)$. To illustrate the problem, we will look at just the branch cut for $h_1(z)$ here.

After reduction, we must consider the system:

$$\begin{aligned} \sqrt{z} &= s \\ s &< 1. \end{aligned} \tag{4.16}$$

Naively squaring (4.16) and equating real and imaginary parts produces the set

$$\{x = s^2,\ y = 0,\ s < 1\},$$

which would lead to the conclusion that the set of branch cuts for $h_1(z)$ is

$$\{(x > 0) \wedge (y = 0)\},$$

when the desired result is only the set

$$\{(0 \leq x < 1) \wedge (y = 0)\}.$$

Definition 4.3.1. *Define*

$$\operatorname{sgn}(z) = \begin{cases} 1, & (\Re(z) > 0) \vee ((\Re(z) = 0) \wedge (\Im(z) \geq 0)), \\ -1, & \textit{otherwise.} \end{cases}$$

To avoid including the negative branch in the solution set, $p(z)$ should only be solved on the region where $sgn(g(z)) = 1$. To achieve this, some polynomial constraints need to be added to the system. Thus the following pair of systems should be considered, once again assuming $f(g(z))$ has a branch cut of the form

$(s\ \sigma\ c) \wedge (t = 0)$:

$$\begin{aligned} g^2(z) &= p(z) \\ \Re(f(s,z)) &> 0 \qquad (4.17) \\ s &\ \sigma\ c, \end{aligned}$$

and

$$\begin{aligned} g^2(z) &= p(z) \\ \Re(g(z)) &= 0 \qquad (4.18) \\ \Im(g(z)) &\geq 0 \qquad (4.19) \end{aligned}$$

where equations (4.17), (4.18) and (4.19) are the additional constraints.

In the case where $g^2(z)$ does not contain any square roots, this reduces to the case of Section 4.3.1. If $g^2(z)$ does contain square roots, then the procedure has to be recursively applied and additional new variables may be needed. For example, to convert the system containing (4.17) above, the following system would be set up:

$$\begin{aligned} g^2(z) &= p(z) \qquad (4.20) \\ g^2(z) &= s_1 \qquad (4.21) \\ s_1 &> 0 \qquad (4.22) \\ s &\ \sigma\ c, \qquad (4.23) \end{aligned}$$

and the procedure proceeds from here. Notice that the original constraint on s is still required.

A disadvantage of these systems is that it may not be possible to eliminate s_i (or t_i depending on the function f) from the system after equating the real and imaginary parts as described in Section 4.3.1. These extra variables will, clearly, reduce the efficiency of the decomposition[1].

[1] The CAD method is intrinsically doubly-exponential in the number of variables (see Section 5.5).

If the resultant or Gröbner basis method is used, then we face a problem of identifying the most appropriate side-conditions as in (4.20), (4.21) and (4.22). This is because of the lack of information as to what manipulations of the original equation (4.15) have been used at each step.

While clearly it is good to remove spurious branch cuts, leaving them merely harms the efficiency of the process, not the fundamental correctness of the algorithm. Using any of the methods of Section 4.3.2, we will not lose any "true" branch cuts, but we will simply add more. Hence we will derive a decomposition with larger number of cells: the spurious cut itself might only be a single cell, but it will have divided other cells that "ought not" to have been divided.

4.4 Removable Branch Cuts

The methods of Sections 4.3.1 and 4.3.2 would return a set of all possible branch cuts. This may include some removable[2] branch cuts, which are the regions where the function is not actually discontinuous. It may not be apparent that these branch cuts exist. To illustrate this, let us consider the function

$$f(z) = \log(z+1) - \log(z-1).$$

The method above will calculate the branch cut of $\log(z+1)$, which is the set $\{(x \leq -1) \wedge (y = 0)\}$, the branch cut of $\log(z-1)$, which is the set $\{(x \leq 1) \wedge (y = 0)\}$, and conclude that the branch cut of $f(z)$ is the union of the two, i.e. $\{(x \leq 1) \wedge (y = 0)\}$, when the actual branch cut is only $\{(-1 \leq x \leq 1) \wedge (y = 0)\}$. The extra branch cut arises from the fact that the procedure fails to detect that $f(z)$ is in fact continuous across $\{(x \leq -1) \wedge (y = 0)\}$ and thus has a removable branch cut across this region.

The inclusion of the removable branch cuts will lead to an overcautious decomposition, and consequently add more work to the identity testing step. Therefore, while we will still obtain correct results, it would be nice to detect these removable

[2] This is similar to the idea of "removable singularity".

branch cuts.

4.5 Special Case

Since $\sqrt{}$ denotes the single-valued function from $\mathbf{C} \to \mathbf{C}$, and $\sqrt{z} = \{w : w^2 = z\} = \{+\,\mathrm{Sqrt}(z)\}$, we may by-pass computing some branch cuts of a function $f(g(z))$ where f is any elementary function and $g(z)$ is any function allowed by our restriction. If $g(z) = c_1\sqrt{g_1(z)} + \cdots + c_n\sqrt{g_n(z)}$ with real constants $c_i \geq 0$, then $\Re(g(z)) \geq 0$, $\forall z$. Hence, when calculating the top-level branch cuts for $f(g(z))$, the parts of the cuts which lie on the negative real or negative imaginary axis can be ignored. This is also true for arbitrarily many complex variables.

4.6 Summary

In this chapter we have showed how a branch cut expression can be mathematically computed. The methods require specification of branch cuts, which are purely conventional. The methods described here will work with any choice of branch cuts.

Chapter 5

Cylindrical Algebraic Decomposition

Decomposition of $\mathbf{C}$ (or $\mathbf{C}^n$), viewed as $\mathbf{R}^2$ (or $\mathbf{R}^{2n}$), as defined by branch cuts can be achieved via Cylindrical Algebraic Decomposition (CAD[1]). The initial application of CAD, when it was first invented in 1970s, was to solve quantifier elimination in real closed fields (as [Col75], we refer to [VDW53] for a description of real closed fields). However, it has over the years been shown that its data structure representing semi-algebraic sets is also useful in other applications requiring a decomposition of $\mathbf{R}^n$ according to certain polynomial equations and inequalities, including robot motion planning and our simplification problem. To date, there have been two different approaches to compute CAD. The first was due to Collins [Col75]. The approach is based on a *projection and lifting* technique. A more recent approach by Chen *et al.* [CMMXY09] computes *CAD via Triangular Decomposition.*

[1] Not to be confused with Computer Aided Design. In this thesis, CAD will always refer to Cylindrical Algebraic Decomposition.

5.1 Quantifier Elimination

Quantifier elimination (QE) is a procedure to remove the quantifiers, i.e. the existential quantifier (∃) and universal quantifier (∀) from the *quantified formula*

$$(Q_k x_k)(Q_{k+1} x_{k+1}) \dots (Q_r x_r) F(x_1, \dots, x_r). \tag{5.1}$$

where $(Q_i x_i)$ is either an existential quantifier $(\exists x_i)$ or an universal quantifier $(\forall x_i)$, and $F(x_1, \dots, x_r)$ is a quantifier-free formula, $1 \leq k < r$, resulting in an equivalent *quantifier-free* formula. While quantifier elimination, as an abstract problem, can be posed for any class of F, we are concerned with polynomials with integer coefficients.

The quantifier elimination method for real-closed fields was first discovered by Tarski in 1930. The work was first published in 1948 [Tar48], and in a second edition three years later [Tar51]. Although Tarski proved that quantifier elimination is possible, the complexity of the algorithm was too great for it to be practical except for trivial problems. Neither the two subsequent algorithms by Seidenberg [Sei54] in 1954 and Cohen [Coh69] in 1969 offered any advantage over Tarski's in term of complexity. The first practical approach to quantifier elimination came in 1970s with a discovery of CAD by Collins [Col75]. It was followed by a number of other methods, including the methods by Heintz, Roy and Solern*ó* [HRS90], Renegar [Ren92a, Ren92b, Ren92c] and Weispfenning [Wei98].

5.1.1 Implementations of CAD

The first implementation of CAD was made by Dennis Arnon in 1980 based on Collins' original CAD algorithm. Currently, there are four reasonably complete CAD implementations. The first three compute CAD via Projection and Lifting, while the fourth uses CAD via triangular decomposition.

1. **QEPCAD** (Quantifier Elimination by Partial Cylindrical Algebraic Decomposition) is a quantifier elimination system for computing with semi-algebraic sets using partial Cylindrical Algebraic Decomposition. It is a

stand-alone, command-line program written in C and based on the SACLIB library of computer algebra functions. The system was developed primarily by Hoon Hong, but subsequently extended by many other contributors, including Christopher W. Brown, George E. Collins, Scott McCallum, etc. The current version of QEPCAD is QEPCAD B [Bro, Bro03, Bro04]. It improves the basic implementation of CAD as well as providing additional functionality. The system is maintained by Christopher W. Brown, and is written in C++ rather than C.

2. **RLCAD** is implemented by Andreas Seidl and Thomas Sturm in REDLOG which is part of the REDUCE system.

3. **Mathematica**. Starting with version 5.0, Mathematica now provides a command `CylindricalDecomposition`$[\mathtt{ineqs}, \{\mathtt{x}_1, \mathtt{x}_2, \dots\}]$.

4. **Maple**. Starting with Maple 14, Maple now provides a command `CylindricalAlgebraicDecompose(F,R)`.

For this thesis, we will base our analysis and experimentation on the use of CAD for our verified simplification application on QEPCAD B (which will simply be referred to as QEPCAD), and Maple's `CylindricalAlgebraicDecompose` command (which will simply be referred to as Maple's CAD). We use QEPCAD for projection and lifting based CAD because it is active open source software and we have a good relationship with the maintainer.

5.2 CAD via Projection and Lifting

In this section, we look at Collins' original CAD algorithm and its ameliorations. We will refer to the algorithm as CAD-PL to distinguish it from CAD via triangular decomposition (Section 5.3). We begin with the definitions necessary to describe the algorithm. The definitions follow those given in [ACM84a] which gave somewhat different, but equivalent, ones to those in [Col75].

Definition 5.2.1. *A* region *R is a non-empty connected subset of* $\mathbf{R}^n$.

Definition 5.2.2. *For a region R, a* cylinder over R, *$Z(R)$, is the set*

$$R \times \mathbf{R}^1 = \{(\alpha, x) \mid \alpha \in R,\ x \in \mathbf{R}\}.$$

Definition 5.2.3. *Let f be a continuous, real-valued function on R. A f-*section *of $Z(R)$ is the set*

$$\{(a, f(a)) : a \in R\}.$$

Definition 5.2.4. *Let $f_1 < f_2$ be continuous, real-valued functions on R. A* (f_1, f_2)-sector *of $Z(R)$ is the set*

$$\{(\alpha, \beta) : \alpha \in R,\ f_1(\alpha) < \beta < f_2(\alpha)\}.$$

The constant functions $f_1 = -\infty$, and $f_2 = \infty$ are permitted.

Definition 5.2.5. *Let $X \subseteq \mathbf{R}^n$. A* decomposition *of X is a finite collection of disjoint regions whose union is X:*

$$X = \bigcup_{i=1}^{k} X_i,\ X_i \bigcap X_j \neq \emptyset,\ i \neq j.$$

Definition 5.2.6. *Let $f_1 < f_2 < \cdots < f_k, k \geq 0$ be continuous, real-valued functions on R. A* stack over *R is a decomposition of $Z(R)$ comprising of*

i.) the f_i-sections of $Z(R)$ for $1 \leq i \leq k$, and

ii.) the (f_i, f_{i+1})-sectors of $Z(R)$ for $0 \leq i \leq k$, where $f_0 = -\infty$ and $f_{k+1} = +\infty$.

Definition 5.2.7. *A decomposition D of $\mathbf{R}^n$ is* cylindrical *if either*

- *$n = 1$ and D is a stack over $\mathbf{R}^0$, or*
- *$n > 1$ and there exists D' a cylindrical decomposition of $\mathbf{R}^{n-1}$ such that for each region R_i of D', there is a stack D_i over R_i with $D = \bigcup D_i$.*

Definition 5.2.8. *A subset of $\mathbf{R}^n$ is said to be a* semi-algebraic *set if it can be*

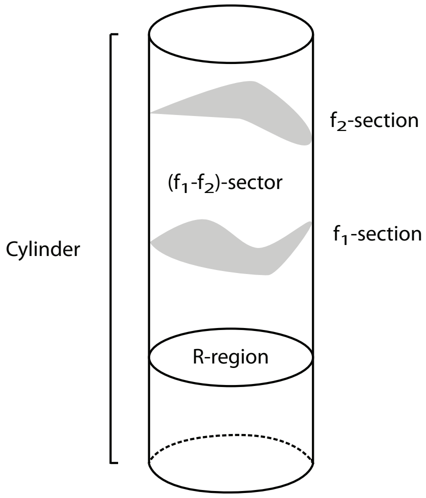


Figure 5-1: A geometrical interpretation of the definitions of region, cylinder, sections, sectors and stack

expressed in the form

$$\bigcup_{i=1}^{m}\bigcap_{j=1}^{l_i}\{x \in \mathbf{R}^n \mid f_{i,j}(x)\ \sigma_{i,j}\ 0\},$$

where $f_{i,j} \in \mathbf{R}[x_1,\ldots,x_n]$ *and* $\sigma_{i,j} \in \{=,\neq,<,\leq,>,\geq\}$ *for* $i = 1\ldots,m$, $j = 1,\ldots,l_i$.

Definition 5.2.9. *A decomposition is said to be* algebraic *if each of its regions is a semi-algebraic set.*

Definition 5.2.10. *A* Cylindrical Algebraic Decomposition *of* $\mathbf{R}^n$ *is a decomposition which is both cylindrical and algebraic.*

Definition 5.2.11. *Let* $X \subseteq \mathbf{R}^n$ *and* $F \in \mathbf{Z}[x_1,\ldots,x_n]$. *F is said to be* sign-invariant *on X if one of the following conditions holds:*

i.) $F(x) > 0 \quad \forall x \in X$;

ii.) $F(x) = 0 \quad \forall x \in X$;

iii.) $F(x) < 0 \quad \forall x \in X$.

Definition 5.2.12. *The order of F at the point x is the number of consecutive* *derivatives of F (starting with F itself), which are zero at x, i.e.*

$$order = \begin{cases} 0 & \text{if}\, F(x) \neq 0; \\ 1 & \text{if}\, F(x) = 0 \text{ but } F'(x) \neq 0; \\ k & \text{if}\, F(x) = F'(x) = \cdots = F^{(k)}(x) = 0 \text{ but } F^{(k+1)}(x) \neq 0. \end{cases}$$

F is order-invariant *on X if it has the same order at x for all $x \in X$.*

Definition 5.2.13. *A* sign-invariant CAD *of $\mathbf{R}^k$ for F is a CAD of $\mathbf{R}^k$ such that F is sign-invariant on each region.*

Remark 5.2.1. *Every order-invariant F is sign-invariant, but not vice versa.*

5.2.1 The Algorithm

CAD-PL algorithm takes a set of polynomials, A, in, say n, variables, $A \subseteq \mathbf{R}[x_1, \ldots, x_n]$ as an input, and outputs a CAD of $\mathbf{R}^n$, sign-invariant for A. The algorithm consists of three phases: projection, base and lifting phases. This section outlines the algorithm; more detailed descriptions can be found in [Col75, ACM84a].

1. **Projection Phase**
   In the projection phase, a projection operation recursively computes successive sets of polynomials, eliminating one variable at each projection until a set of univariate polynomials is obtained.

$$\begin{tikzcd}
P_n = A \subseteq \mathbf{R}[x_1, \ldots, x_n] \arrow[d] \\
P_{n-1} \subseteq \mathbf{R}[x_1, \ldots, x_{n-1}] \arrow[d] \\
\vdots \arrow[d] \\
P_1 \subseteq \mathbf{R}[x_1]
\end{tikzcd}$$

The projection from P_k to P_{k-1} is such that a decomposition D_k sign-invariant for P_k can be constructed from a decomposition D_{k-1} sign-invariant for P_{k-1}. That is the zero set of the resulting polynomials of each set is the projection of the "significant points" (e.g. self-crossings, isolated points, vertical tangent points) of the zeros of the preceding polynomials. In practice, we make each P_i be a set of square-free relatively prime polynomials, and call it a *projection factor set*.

2. **Base Phase**

   A sign-invariant decomposition D_1 of $\mathbf{R}^1$ can be derived from the univariate polynomials produced at the end of the projection phase [EGT10]. The decomposition D_1 consists of sections, which are the real zeros of these polynomials, and sectors, which are the open intervals between consecutive zeros, including the open intervals prior to and after all zeros. The sample points of cells which are sectors can be chosen to be any points belonging to the cells, but for the cells which are sections, the sample points may need to be irrational algebraic.

   These univariate polynomials and isolating intervals, in which the polynomial has only one root, decompose $\mathbf{R}^1$. For example $x^2 - 2$ induces two sections:

   > α: that root of $x^2 - 2$ with $\alpha \in (1, 2)$;
   > β: that root of $x^2 - 2$ with $\beta \in (-2, 1)$; and
   > three sectors $x < \beta$ (typical sample point -2), $\beta < x < \alpha$ (typical sample point 0) and $x > \alpha$ (typical sample point +2).

   Note that $x^2 - 3$ has a section:

   > γ: that root of $x^2 - 3$ with $\gamma \in (1, 2)$.

   However, if both $x^2 - 2$ and $x^2 - 3$ are in P_1, to answer $\alpha < \gamma$, we need to refine these to $\alpha \in \left(1, \frac{3}{2}\right)$ and $\gamma \in \left(\frac{3}{2}, 2\right)$. Isolating and refinement are discussed in many papers, e.g. [KS11, MS11, EGT10].

3. **Lifting Phase** (often called extension)

   The decomposition D_1 of $\mathbf{R}^1$ is successively lifted to a decomposition D_2

of $\mathbf{R}^2$, this decomposition D_2 of $\mathbf{R}^2$ to a decomposition D_3 of $\mathbf{R}^3$, $\ldots$, decomposition D_{n-1} of $\mathbf{R}^{n-1}$ to decomposition D_n of $\mathbf{R}^n$, each D_i being sign-invariant for P_i and cylindrical over D_{i-1}. During each step of the lifting phase, a stack is constructed over each cell of the lower-dimensional CAD. The intersections of the cylinders whose bases are the cells of the lower-dimensional space with the zeros of the next higher-dimensional polynomials are the sections. The connected sets between consecutive sections and the sets below and above all sections are sectors. These sections and sectors form a stack and all the stacks form a CAD, sign-invariant for P_k of the higher-dimensional space. The sample points for the cells of CAD of $\mathbf{R}^2$ are obtained by substituting the sample point of the CAD of $\mathbf{R}^1$ into the bivariate polynomials and computing the real zeros of the resulting univariate polynomials. The process is repeated to obtain the sample points of the cells of the CADs of $\mathbf{R}^3, \ldots, \mathbf{R}^n$.

5.2.2 Improvements

Since the introduction of CAD-PL, there have been several improvements to the original algorithm by various authors.

- Adjacency and clustering [ACM84b, ACM85, ACM88, Arn85, Arn88, MC02].
- Improved projection [McC84, McC85, McC88, Hon90, McC98, Bro01a]. Lazard [Laz94] also suggested an improved projection which further reduce McCallum's projection, but is more widely applicable. While containing interesting ideas, Lazard's projection is unproven as it stands (though no counterexamples to the *projection* have been found). Proposition 3 of [Laz94] is wrong as stated: Schicho found the counterexample: take $n = 1$, $s = 1/x$, $a = 0$ [McC10].
- Partial CAD construction [CH91].
- Interactive implementation [Col98].
- Equational constraints [Col98, McC99, McC01, BM05].

- Improved subalgorithms [Col98].
- Simplification of truth CADs [Bro98, Bro01b].
- Choice of variable ordering [DSS04, Bro04].

5.2.3 Projection Operator

An important component of CAD-PL is the projection operator. The function defines the projection factor set during the projection phase. The original algorithm produced an overly large projection set resulting in a very refined decomposition with many stacks and cells. It has been shown that the size of the projection set can be reduced, improving the efficiency of the algorithm. Significant advancements have been made in a series of improved projection methods including [McC84, McC85, McC88, Hon90, McC98, Bro01a]. They fall into two basic models, ones which produce *sign-invariant* projection sets, i.e. the Collins' and Hong's projection, and ones which produce *order-invariant* projection sets, i.e. McCallum's and Brown-McCallum's projection. For reasons that are outside the scope of this thesis, order-invariant projections have better recursion properties [Bro04].

Although, McCallum's and Brown-McCallum's projection operators produce smaller projection factor sets, they require a more complicated lifting algorithm and they may fail to produce a CAD in rare cases where a projected polynomial is zero everywhere in a cylinder: however these failures can always be detected. Another advantage of Hong's projection is that it makes clustering easier; the clusters only need to be sign-invariant, not order-invariant.

5.2.4 Partial CAD

Collins and Hong observed that in many cases of QE, a full CAD does not need to be constructed if one makes use of the information within the input formulae rather than just extracting a set of polynomials. This led them to introduce *partial CAD* (PCAD) [CH91]. The objective of PCAD is to reduce the number of stacks in the construction during the lifting phase. We note that the projection phase of PCAD is the same as for basic CAD, and therefore still doubly exponential [DH88, BD07]. The method constructs one stack at a time and avoids building unnecessary stacks over some cells in two ways.

1. **Truth value propagation**. Suppose $(Q_k x_k)$ is an existential quantifier, then as soon as any cell in the stack above cell c in $\mathbf{R}^{k-1}$ is found to be true, cell c can be assigned the value *true* and no further stack needs to be constructed. The method can also be applied to universal quantifiers, propagating *false*.

2. **Trial evaluation**. This applies when some variables are not presented in some polynomials in the input formula. Suppose cell c has already been constructed in $\mathbf{R}^k$ and at least one polynomial occurring in the input formula does not contain any other variables than the first k. The truth value of these polynomials can be evaluated at the sample point of cell c and hence the truth value of the atomic formulae in which they are presented can be determined. If these truth values already yield the truth value of the quantifier-free part of the input formula, then the stack does not need to be constructed above cell c.

It has been showed that PCAD is more efficient than the original CAD when applied to QE problems [CH91]. Since PCAD, and hence QEPCAD, were developed for QE problems and our problems are different, a question arises about the use of QEPCAD within our application. To demonstrate the idea of PCAD, we run a small example using QEPCAD. This example, although not derived from a branch cut problem, is chosen for its simplicity. We consider the formula

$$x - 2 < 0 \wedge y^2 - 4 = 0. \tag{5.2}$$

Examining data structure shows that once the polynomials are projected down to the base case, when constructing a CAD with respect to variable ordering $y > x$ (Figure 5-2), QEPCAD decomposes the real line $\mathbf{R}^1$ into a point ($x = 2$) and two open intervals. Since the point (cell 2) and the right open interval (cell 3) do not satisfy the condition $x - 2 < 0$, no stack is constructed above these two level 1

```
Enter a variable list:
(x,y)
Enter the number of free variables:
2
Enter a prenex formula:
[x-2 < 0 /\ y^2-4 = 0].
=======================================================

                            ⋮

Before Solution >
d-pscad ()
--------------------------------------------
The Partial CAD over ()

()---(1)p1(-)---(1,1)p1(-,-) F
             ---(1,2)p1(0,-) T
             ---(1,3)p1(+,-) F
             ---(1,4)p1(+,0) T
             ---(1,5)p1(+,+) F
  ---(2)p1(0) F
  ---(3)p1(+) F
--------------------------------------------
```

Figure 5-2: Partial CAD data structure for formula (5.2) ($y > x$)

cells. As a result, we only have a CAD of that part of $\mathbf{R}^2$ which lies above the left open interval (cell 1) part of $\mathbf{R}^1$. This gives us five connected cells in that partial space as shown in Figure 5-3.

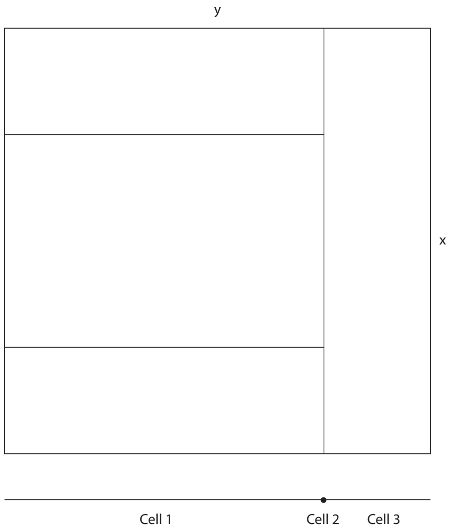


Figure 5-3: QEPCAD's CAD for formula (5.2) ($y > x$)

If a CAD is constructed with respect to variable ordering $x > y$ (Figure 5-4), the decomposition D_1 of $\mathbf{R}^1$ consists of two points ($y = \pm 2$) and three intervals. This time only the two points (cell 2 and cell 4) satisfy the condition $y^2 - 4 = 0$, and only these points are lifted to a decomposition D_2 of $\mathbf{R}^2$, giving six cells (Figure 5-5).

Clearly the smaller CAD produced by QEPCAD does not imply that the whole space is decomposed into fewer connected components, but rather that only parts of the whole space are decomposed. This poses a serious problem for us as we may not have enough sample points to perform identity testing. QEPCAD is not wrong in taking the shortcut; it just solves a different problem to ours: stressing the difficulty of using QEPCAD as a black-box in our application.

The fact that QEPCAD (at least by default) may in certain cases only give us decompositions of partial spaces has escaped unnoticed until recently because the problems we are interested in often consist of more than one branch cut and the "interesting" parts were decomposed correctly. Comparing with Maple's CAD made us realise this gap.

```
Enter a variable list:
(y,x)
Enter the number of free variables:
2
Enter a prenex formula:
[x-2 < 0 /\ y^2-4 = 0].
=======================================================
```

⋮

```
Before Solution >
d-pscad ()
--------------------------------------------
The Partial CAD over ()

()---(1)p1(-,-) F
  ---(2)p1(0,-)---(2,1)p1(-) T
               ---(2,2)p1(0) F
               ---(2,3)p1(+) F
  ---(3)p1(+,-) F
  ---(4)p1(+,0)---(4,1)p1(-) T
               ---(4,2)p1(0) F
               ---(4,3)p1(+) F
  ---(5)p1(+,+) F
--------------------------------------------
```

Figure 5-4: Partial CAD data structure for formula (5.2) ($x > y$)

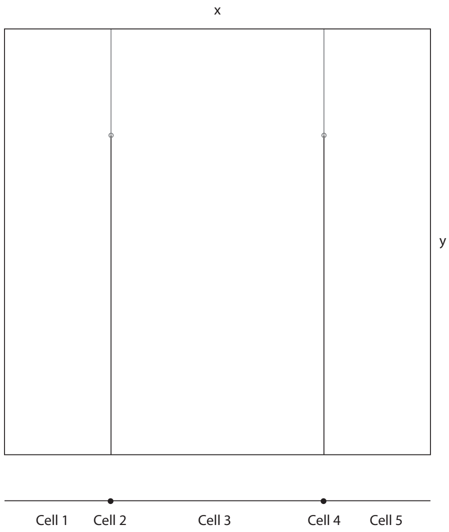


Figure 5-5: QEPCAD's CAD for formula (5.2) ($x > y$)

We see from this example that QEPCAD has in fact not decomposed the whole of $\mathbf{R}^2$ at all, but only the "interesting" portions of it. A consequence of this is that the cell counts reported in [BBDP07] are in fact a slight underestimate of the number of cells needed for a complete proof of the identity. In practice the cells omitted are the generic ones, on which the putative identity is true because it is true in a multi-valued sense, but this would need to be checked for a full proof of correctness.

5.2.5 Equational Constraints

The main idea is that if an input formula contains the constraint $f = 0$, once the decompostion of $\mathbf{R}^r$ is constructed into regions such that f is sign-invariant and other polynomials appearing in the formula are sign-invariant within the cells in which $f = 0$, then what happens when $f \neq 0$ is irrelevant.

The equational constraints idea, which was first introduced in [Col98] and partially validated in [McC99, McC01], takes advantage of a single equational constraint $f = 0$ to reduce the McCallum projection operator. A more recent development by Brown and McCallum [BM05], addressed the problem involving two equational constrains, *bi-equational constraints*, i.e. of the form $f = 0 \ \wedge \ g =$

$0 \wedge \ldots$. Unlike the earlier scheme, Brown and McCallum restricted the Brown-McCallum projection operator. Note that this development is of less direct use to us, since bi-equational constraints are not a natural representation of a branch cut.

Furthermore, equational constraints can also be used to reduce the number of stacks to be constructed during the stack construction phase. If a stack has already been constructed in which there are sections of an equational constraint polynomial, all other cells in the stack can be marked *"false"* and stacks are not required to be constructed over them.

Recall that our branch cut representations of a function in complex variables consists of one or more pairs of an equality and an inequality. Therefore, we may be able to take advantage of the equational constraints in our input formulae. Of course, there might not be an inequality in some cases, e.g. $\{xy - 1 = 0\}$ is a branch cut for equation (2.5).

5.2.6 Variable Ordering

Variable ordering is an important attribute in CAD. Generally, there is a certain degree of freedom in choosing the order in which the variables are projected for performing CAD. For most applications other than quantifier elimination, the order can be chosen completely freely. For quantifier elimination, all the quantified variables are required to be projected before all the non-quantified variables, and $\exists$ and $\forall$ are not interchangeable.

For example, consider a prenex formula γ,

$$\gamma = Q_{k+1}x_{k+1} \ldots Q_r x_r \, \phi, \; Q_i \in \{\exists, \forall\}$$

where ϕ is a semi-algebraic formula in variables $x_1, \ldots, x_r$ for $r \geq 1$.

All quantified variables $x_r, \ldots, x_{k+1}$ must be projected before unquantified variables $x_k, \ldots, x_1$, and they must be projected sequentially in order except when $Q_i = Q_{i+1}$ for $i \in \{k+1, r-1\}$, for which $Q_i x_i$ and $Q_{i+1} x_{i+1}$ are interchangeable.

Therefore, there is a certain degree of freedom in choosing variable order. This is especially true for our problems, where, in general, we do not have any quantified variables: hence we have total freedom in choosing variable order. For each variable block (a block of unquantified variables, or a block of variables with the same quantifier) of n variables, there are $n!$ possible projection orders. Since the number of cells being constructed depends greatly on the order in which the variables are projected, choosing a correct order can significantly reduce the size of the problem, for example 785 cells versus 92829 cells in Table 8.11, and with certain problems, we may only be able to construct CAD using QEPCAD in reasonable time or space with certain variable orders but not with other variable orders (see example 28, Table 8.8). Furthermore, we do significant amount of work *per cell* for identity testing (see Chapter 7), so reducing the number of cells is a good way to improve the total time of the system. However, the number of cells is not known until we have done the whole projection and lifting cycle. Therefore, it is beneficial to have a heuristic to determine a favourable variable order at the earliest possible stage.

5.2.6.1 The sotd Heuristic

Lifting is much more expensive than projection, hence, rather than doing $n!$ projection and lifting combinations and choosing the one with the fewest cells, we would be better off doing $n!$ projections, then choosing the best, or at least a good one, to lift and post-process.

Definition 5.2.14. *[DSS04] The sum of total degrees of all monomials of all polynomials in all projection sets $P_n, \ldots, P_1$ with respect to variable orderings $X = (x_n > x_{n-1} > \ldots > x_n)$ is defined as*

$$\operatorname{sotd}_{all}(P, X) = \sum_{i=1}^{n} \sum_{f \in P_i} \sigma(f),$$

where, using the convention $\boldsymbol{e} = (e_1, \ldots, e_n)$,

$$\sigma\left(\sum_{\boldsymbol{e} \in E} a_{\boldsymbol{e}} x_1^{e_1} \ldots x_n^{e_n}\right) = \sum_{\boldsymbol{e} \in E} \sum_{i=1}^{n} e_i.$$

[DSS04] observed that sotd_{all} is more correlated with the size of the output CAD than other measures. Hence we could do $n!$ projection, and pick the one with lowest sotd_{all}.

5.2.6.2 The Greedy Projection Algorithm

While cheaper than doing $n!$ liftings, computing sotd required us to do a complete projection, so [DSS04] then suggested[2] a greedy heuristic using just

$$\text{sotd}_{n-1}(P_{n-1}, x_i) = \sum_{f \in P_n - 1} \sigma(f),$$

the first contribution to sotd, as a measure.

The algorithm picks the first variable to be projected from n possible variables by first computing the first projection factor set with respect to each of the n variables. The complexity of each projection factor set is compared using $\text{sotd}_{n-1}(P_{n-1}, x_i)$ and the x_i which produces a set with lowest $\text{sotd}_{n-1}(P_{n-1}, Xx_i)$ value is chosen to be the first variable to be projected. The process is repeated using $\text{sotd}_{n-2}(P_{n-2}, (x_i, x_j))$ to choose the next variable to be projected from $n-1$ possibilities, and so on, until we have a complete order.

Since the algorithm was developed based on QE problems, we investigated the possibility of employing the algorithm within our verification system [BBP05, BBDP07]. The method proved to be effective in choosing optimal, or near optimal, projection orders, and proved to be useful in practice. However, while the algorithm is less expensive than naively examining all possible variable orders, we still needed to compute projections “one step down” at a time. Furthermore, there remains the question of which variables to choose when there is a tie in $\text{sotd}_k(P_k, (x_{i_1} \ldots, x_{i_k}))$ values. There is sometimes genuine symmetry between variables, so it does not matter which we choose, but other times it may matter how we break such ties.

[2] Confusingly [DSS04] also calls this sotd.

5.2.6.3 Brown's Heuristic

[Bro04] suggested a simple, less expensive, alternative to the greedy projection algorithm. It makes decisions based on input polynomials alone and does not require any projection computation. The heuristic suggests choosing variable order as follows:

1. descending order by degree of variable, breaking ties with
2. descending order by highest total-degree term in which the variable appears, breaking ties with
3. descending order by number of terms containing the variable.

Note that Brown's convention is that the last variable in the order is the first to be eliminated.

Although the approach has the potential to be much faster, there is, as in the greedy projection algorithm, a question of what to do if after all three steps there is still a tie. This is a major problem when applying the criterion to our problems involving real and imaginary parts of complex variables.

Definition 5.2.15. *The* coupled variables *are a pair of real variables which derive from the same complex variable.*

Suppose we have an identity, $h = 0$, containing two complex variables, z and w. Suppose further that the identity does not contain any nested root. Recall that during branch cuts computation, Chapter 4, we substituted $z := x + iy$ and $w := u + iv$ into each function h_i in our identity $h = 0$, $h_i \in \mathbf{C}[z, w]$. Applying Brown's heuristic will always result in a tie between coupled variables[3].

5.2.6.4 Combining the Approaches

Despite the fact that Brown's heuristic is unable to make a decision between coupled variables, it does not mean that the heuristic is totally inapplicable to our

[3] If additional constrains are used, or QE is required, this may not happen.

application involving complex variables. The heuristic can be used in conjunction with the greedy projection algorithm, in two ways, to reduce the cost compared to the case when the greedy projection algorithm is used alone. Suppose we have an identity containing two complex variables, $h(z, w) = 0$.

1. The greedy projection algorithm with a shortcut inspired by Brown's heuristic (abbreviated to greedy/Brown's heuristic).
   We first run the greedy projection algorithm, doing four projections to determine the first variable to be projected. Suppose, without loss of generality, x is chosen. Then the coupled variable, i.e. y, can automatically be chosen to be the next variable to be projected. This is because Brown's heuristic would put the coupled variables next to each other. We then only need to do two projections to decide between the variables u and v.

2. Brown's heuristic with the greedy projection algorithm as a tie breaker (abbreviated to Brown's/greedy heuristic).
   We first follow Brown's heuristic and then use the greedy projection algorithm to break the tie between the coupled variables. Suppose the coupled pair (x, y), without loss of generality, is favoured by Brown's heuristic. This reduces the number of possible variables to be selected as first variable to two, and we can then use the greedy projection algorithm to determine which of the tied coupled variables should be projected first. Once the first variable is decided, it is only left to determine the third variable, since the second variable is the variable coupled with the first variable and is chosen automatically along with the first variable.

Although we do not intend to carry out a full statistical analysis in the spirit of [DSS04] on these combined heuristics, the preliminary results are presented in Chapter 8.

5.3 CAD via Triangular Decomposition

In 2009, Chen *et al.* introduced an alternative approach to CAD-PL for computing Cylindrical Algebraic Decomposition via triangular decomposition [CMMXY09], which is the subject of this section. We will call this approach CAD-TD.

While CAD-PL works exclusively over $\mathbf{R}$, CAD-TD starts work over $\mathbf{C}$. Hence the concept we have previously called "cylindricity" will be referred to more precisely as $\mathbf{R}$-cylindricity in this section to avoid confusion, as CAD-TD starts by using a different kind of cylindricity, which we will refer to as $\mathbf{C}$-cylindricity[4].

Notation 5.3.1. *Throughout this section let* $\mathbf{k}$ *be a field of characteristic zero and* $\mathbf{K}$ *be the algebraic closure of* $\mathbf{k}$*. Let* $\mathbf{k}[\mathbf{y}] := \mathbf{k}[y_1, \ldots, y_n]$ *be the polynomial ring over the field* $\mathbf{k}$ *and with ordered variables* $y_1 < \cdots < y_n$[5].

Note that we are still assuming an order on the variables, even though we will not be projecting in the same sense as CAD-PL.

We first need ten standard definitions from the general theory of triangular decompositions.

Definition 5.3.1. *Let* $p \in \mathbf{k}[\mathbf{y}]$ *be a non-constant polynomial. The* main variable*,* $mvar(p)$*, is the greatest variable appearing in* p*.*

Definition 5.3.2. *The degree, the leading coefficient and the leading monomial of* p *regarded as a univariate polynomial in* $mvar(p)$ *are called the* main degree*, the* initial *and the* rank *of* p*; they are denoted by* $mdeg(p), init(p)$ *and* $rank(p)$ *respectively.*

Definition 5.3.3. *Let* $p, q \in \mathbf{k}[\mathbf{y}]$ *be two non-constant polynomials. We say that* $rank(p) < rank(q)$ *if one of the following conditions hold:*

- $mvar(p) < mvar(q)$*, or*
- $mvar(p) = mvar(q)$ *and* $mdeg(p) < mdeg(q)$*.*

Definition 5.3.4. *A* triangular set*,* $T \subset \mathbf{k}[\mathbf{y}]$*, is a set of non-constant polyno-*

[4] This is our terminology, as we found [CMMXY09] could be confusing if one did not keep track of which kind of cylindricity was in use where.

[5] This corresponds to Maple's CAD's `PolynomialRing(`$[y_n, \ldots, y_1]$`)`.

mials with pairwise distinct main variables, i.e.

$$\forall p, q \in T \subset \mathbf{k}[\mathbf{y}]\backslash\mathbf{k},\ p \neq q \Rightarrow mvar(p) \neq mvar(q).$$

Definition 5.3.5. [6] *Let $T \subset \mathbf{k}[\mathbf{y}]$ be a triangular set, $\langle T\rangle$ be the ideal it generates in $\mathbf{k}[\mathbf{y}]$ and h be a polynomial in $\mathbf{k}[\mathbf{y}]$. The* saturated ideal *of T, $Sat(T)$ is the set*

$$Sat(T) = \{q \in \mathbf{k}[\mathbf{y}] | \exists m \in \mathbf{N} \text{ s.t. } h^m q \in \langle T\rangle\},$$

where h is $\prod_{t\in T} init(t)$.

Definition 5.3.6. *The polynomial is regular modulo $\langle T\rangle$ if it is neither zero, nor a zero-divisor modulo $\langle T\rangle$.*

Definition 5.3.7. *A triangular set $T \in \mathbf{k}[\mathbf{y}]$ is a regular chain if one of the following conditions hold:*

- *$T = \emptyset$, or*
- *$T\backslash\{T_{max}\}$ is a regular chain, where T_{max} is the polynomial in T with maximum rank, and the initial of T_{max} is regular w.r.t. $Sat(T\backslash\{T_{max}\})$.*

Definition 5.3.8. *A pair $[T, h]$ is a regular system if T is a regular chain, and $h \in \mathbf{k}[\mathbf{y}]$ is regular w.r.t. $Sat(T)$.*

Definition 5.3.9. *An* algebraic variety *is a set of solutions of a system of polynomial equations.*

Definition 5.3.10. *A* constructible set *of $\mathbf{K}^n$ is any finite union*

$$(A_1\backslash B_1) \cup \cdots \cup (A_e\backslash B_e)$$

where $A_1, \ldots, A_e$ and $B_1, \ldots, B_e$ are algebraic varieties in $\mathbf{K}^n$.

We now introduce the key concept from [CMMXY09].

Definition 5.3.11. *A finite collection of constructible sets is* **C**-cylindrical *if*

[6] This is also written $Sat_h(T)$ or $\langle T\rangle : h^\infty$, where in both cases $h = \prod_{t\in T} init(t)$.

- $n = 1$. *A cylindrical decomposition of* $\mathbf{C}$ *is either* $\mathbf{C}$ *itself or of the form* $p_1 = 0, \ldots, p_r = 0, p_1 \cdots p_r \neq 0$ *where* $p_1, \ldots, p_r$ *are non-constant square-free and pairwise coprime polynomials of* $\mathbf{k}[y_1]$.
- $n > 1$. *Given a cylindrical decomposition* D' *of* $\mathbf{C}^{n-1}$, *one builds a cylindrical decomposition* D *of* $\mathbf{C}^n$. *For each cell* D_i *of* $\mathbf{C}^{n-1}$:
  - *either* $D_i \times \mathbf{C}$ *is an element of* D, *or*
  - *there exist polynomials* $p_{i,1}, \ldots, p_{i,r_i}, r_i > 0 \in \mathbf{R}[y_1, \ldots, y_n]$ *such that*
    1. *the initial of each* p_j *does not vanish on* D_i *and,*
    2. *the* p_j*'s are square-free and pairwise coprime at all* $\mathbf{u} \in D_i$,
    3. $D_i \times (p_1 = 0), \ldots, D_i \times (p_r = 0)$ *and* $D_i \times (p_1 \cdots p_r \neq 0)$ *are in* D.

Note the distinction between $\mathbf{C}$-cylindricity and $\mathbf{R}$-cylindricity, even in the case $n = 1$. If $r = 1$ and $p_1 = x^2 - 2$, then a $\mathbf{C}$-cylindrical decomposition of $\mathbf{C}^1$ is into two sets: $x^2 - 2 = 0$ and $x^2 - 2 \neq 0$ (dimensions 0 and 1 respectively), whereas an $\mathbf{R}$-cylindrical decomposition of $\mathbf{R}^1$ has two 0-dimensional cells $x = -\sqrt{2}$ (more formally $x^2 = 2 \wedge -2 \leq x \leq -1$) and $x = \sqrt{2}$ (more formally $x^2 = 2 \wedge 1 \leq x \leq 2$) and three 1-dimensional cells $x < -\sqrt{2}$, $-\sqrt{2} < x < \sqrt{2}$ and $\sqrt{2} < x$.

Definition 5.3.12. *A polynomial* p *is* delineable *on* R *if the variety, i.e. the zero set of* p, *consists of* k *disjoint sections of* $Z(R)$.

Theorem 5.3.1. *[Col75] Let* p *be a polynomial of ring* $\mathbf{R}[y_1 < \cdots < y_n]$ *and* R *be a region of* $\mathbf{R}^{n-1}$. *If* $init(p) \neq 0$ *on* R *and the number of distinct complex roots of* p *is invariant on* R, *then* p *is delineable on* R.

Corollary 5.3.1. *[CMMXY09] Let* $F = \{p_1, \ldots, p_r\}$ *be a finite set of polynomials in* $\mathbf{R}[y_1 < \cdots < y_n]$ *of level* n. *Let* R *be a region of* $\mathbf{R}^{n-1}$. *Assume that for every* $\alpha \in R$,

1. *the initial of each* p_i *does not vanish at* α;
2. *all* $p_i(\alpha, y_n), 1 \leq i \leq r$, *as polynomials of* $\mathbf{R}[y_n]$, *are squarefree and coprime.*

Then each p_i *is delineable on* R *and the sections of* $Z(R)$ *belonging to different*

p_i and p_j are disjoint.

5.3.1 The Algorithm

We now describe an algorithm, CAD-TD, for constructing a CAD.

1. **Initial Partition**
   In the initial partition phase, we compute a set of regular systems via *comprehensive triangular decomposition* [CGL$^+$07], whose set of zero sets is an intersection free basis of constructible sets $C = \{C_1, \ldots, C_e\}$ and forms a partition of $\mathbf{C}^n$ such that each $f \in F_n$ is invariant in C_i, $1 \leq i \leq e$ (f either is identically zero in C_i or vanishes at no points of C_i).

2. **Make C-cylindrical** (known as MakeCylindrical in [CMMXY09])
   During this phase the *SeparateZeros* algorithm ([CMMXY09, Section 3.1]) is recursively applied to the regular systems in the output of the Initial Partition phase. This algorithm makes further use of the regular chain-based algorithm of [CGL$^+$07]. The process produces another partition of $\mathbf{C}^n$ into disjoint constructible sets such that this second decomposition is $\mathbf{C}$-*cylindrical.*

3. **Make R-cylindrical** (known as MakeSemiAlgebraic in [CMMXY09])
   The cylindrical decomposition of $\mathbf{C}^n$ derived in the previous step is transformed into an F_n-invariant CAD of $\mathbf{R}^n$, via Corollary 5.3.1 which is obtained from Collins' Theorem 5.3.1 by means of real root isolation of *zero-dimensional regular chains.*

5.4 Comparison

In this section we give an overview comparison of the two approaches, or more precisely a comparison between QEPCAD and Maple's CAD since these are the two software packages which are used in our experiments to compute CAD via projection and lifting, and triangular decomposition respectively.

QEPCAD	Maple's CAD
- Projection and lifting approach. - Partial CAD. - Takes prenex formulae as input. - Gives a truth value for each cell.	- Triangular decomposition approach. - Full CAD. - Takes polynomials as input. - Does not give a truth value for each cell.

Table 5.1: Comparison of QEPCAD and Maple's CAD

5.5 Complexity

[DH88, BD07] show that the complexity of a CAD is at least doubly exponential in the number of variables in the input formula. In [BD07], Davenport and Brown prove that the worst case running time is

$$2^{2^{\frac{r-1}{3}}},$$

where r is the number of variables in the input formula[7].

Of course, on particular examples, particular implementations may have widely different behaviours (see Chapter 8). [BD07] has examples which are doubly exponential with respect to one variable order, but polynomial with respect to another, so that variable ordering (see Section 5.2.6) can also be seen to be important from a theoretical point of view.

5.6 Summary

Cylindrical Algebraic Decomposition is a powerful and useful tool for solving problems in real algebraic geometry and connectedness. The algorithm partitions n-dimensional real space $\mathbf{R}^n$ into regions such that

- all the regions are cylindrically arranged, that is for any k, $1 \leq k \leq n$, the projections of any two regions onto $\mathbf{R}^k$ are either identical or disjoint,
- each region is a connected subset of $\mathbf{R}^n$.

[7] [BD07] improves the bound of [DH88].

In this chapter we reviewed the two existing approaches to construct Cylindrical Algebraic Decomposition:

1. CAD via projection and lifting (CAD-PL),
2. CAD via triangular decomposition (CAD-TD).

The former has been around since 1970s and has received many improvements. However, most research has tended to focus on its original use in the QE setting, and may not be applicable to our application in simplification of elementary functions.

Variable order has significant impact on the efficiency of the CAD algorithm. We reviewed Dolzmann *et al.* heuristic and Brown's heuristic to generate an effective variable order. We then presented the combined heuristics which apply to our special circumstances: if we are working over **C**, variables are naturally paired together (real and imaginary parts).

Chapter 6

A Partial CAD for Branch Cuts

The method of branch cuts by CAD which we described so far, takes a set of semi-algebraic formulae describing the branch cuts as an input, and constructs a CAD of $\mathbf{R}^n$ such that, for each branch cut B_i, and all cells c_j, $B_i \bigcap c_j$ is either empty or c_j. While it is theoretically correct, more work is often done than is necessary due to the well-known fact that the CAD method is *too powerful*, resulting in more cells being constructed than necessary. Given a set of polynomials and a variable ordering, it produces a CAD which not only answers the problem asked but also all other potential problems involving the same polynomials and the same variable ordering, but possibly different $\sigma \in \{=, \neq, <, \leq, >, \geq\}$, different quantifiers, and different boolean formulae. For example, given a set of polynomials $y^2 = 2 \wedge \ldots$, it will naively translate this into problems involving $y^2 = 2$ $(y = \pm\sqrt{2})$, $y^2 < 2$ $(-\sqrt{2} < y < \sqrt{2})$ and $y^2 > 2$ $(y > \sqrt{2}$ and $y < -\sqrt{2})$. Thus, we will spontaneously solve problems outside $y^2 = 2$ as well. In this chapter, we present a method for manipulating the set of branch cut formulae involving boolean connective pairs of an equality and an inequality, which we will call the *pre-conditioning* method.

The idea of this pre-conditioning method is based on the observation that a smaller number of cells may be constructed if, instead of throwing a lot of useful information on the branch cut structure away when handing the problem to

CAD, more information is carried forward. We are adopting, here, a similar approach to that of Collins and Hong's Partial CAD [CH91] in the sense that we carry forward more information from the given problem to the CAD phase. However, while Collins and Hong's Partial CAD method is looking at the lifting phase, our pre-conditioning method is looking at the input polynomials prior to CAD construction. Hence our pre-conditioning method, unlike Collins and Hong's Partial CAD which is useful only for CAD via projection and lifting, is applicable and useful for both CAD via projection and lifting and CAD via triangular decomposition.

The following four figures (Figures 6-1, 6-2, 6-3 and 6-4) give an overview of CAD and Partial CAD algorithms for applications to QE problem and to the verified simplification problem.

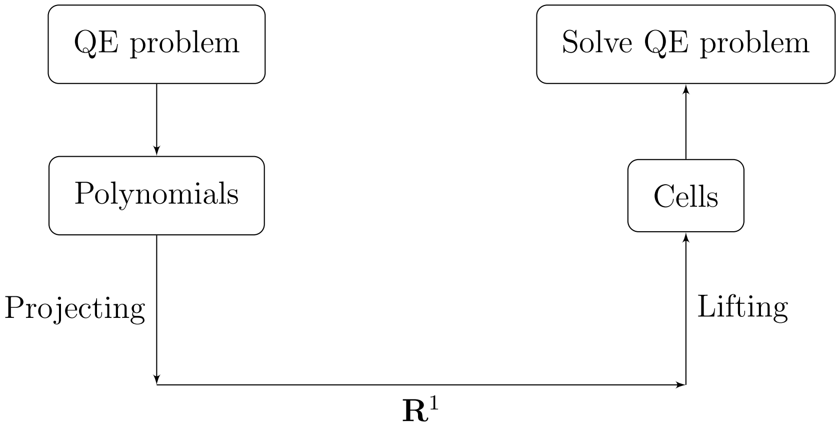


Figure 6-1: Original Collins's CAD for QE [Col75]

6.1 Motivation

Our initial idea of pre-conditioning was aimed at Maple's CAD, after realising that unlike QEPCAD which takes a boolean combination of equalities and inequalities as an input, Maple's CAD takes a set of polynomials as an input.

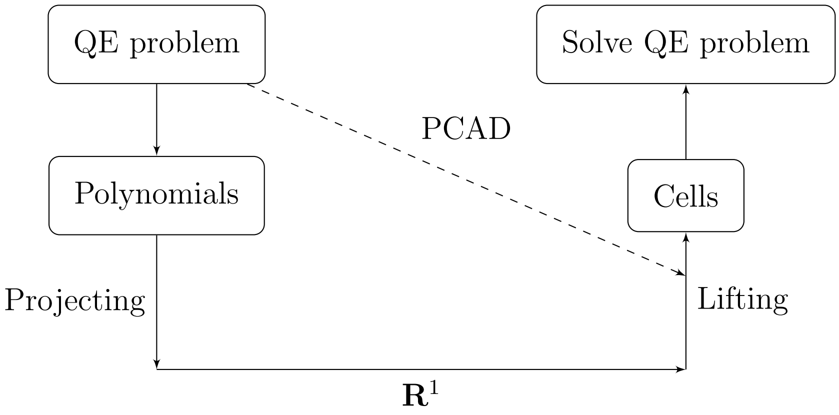


Figure 6-2: Collins and Hong's Partial CAD for QE [CH91]

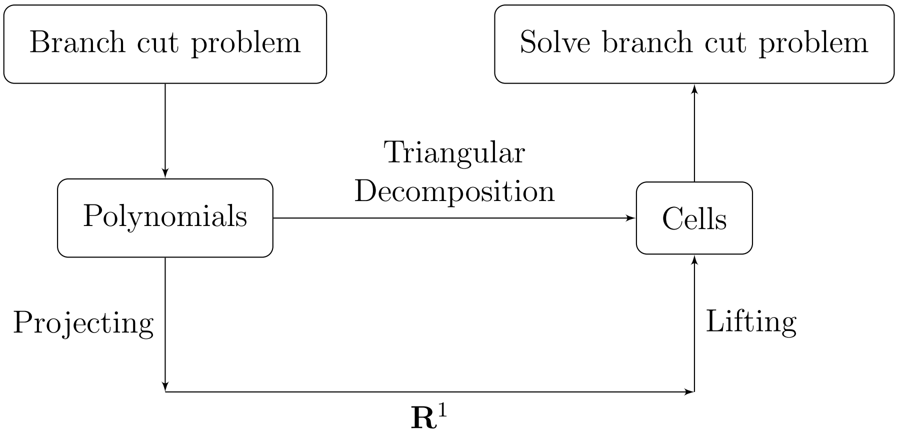


Figure 6-3: CAD for branch cuts [BBDP07]

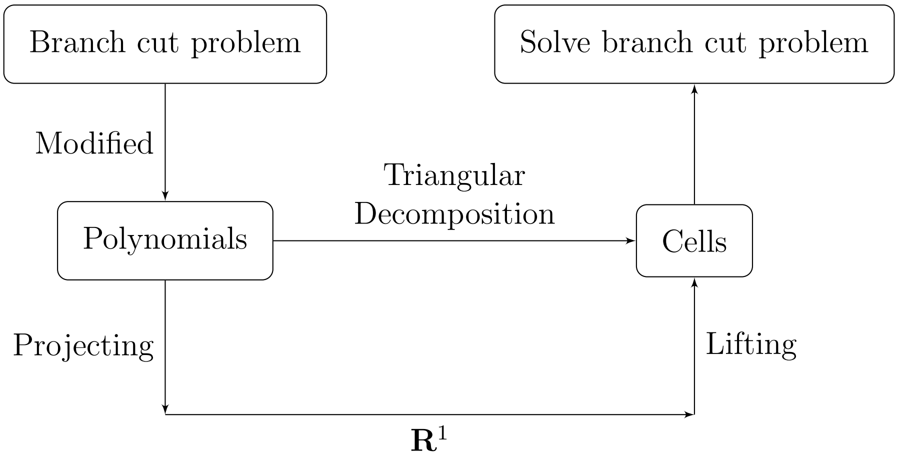


Figure 6-4: Partial CAD for branch cuts

Therefore information on pairings and relations is lost when handing information to Maple's CAD. This often leads to a larger geometric problem and hence to a greater number of cells produced by the decomposition step than is necessary. The goal of pre-conditioning is to ameliorate this by utilizing more of the available branch cut information.

6.2 An Illustrative Example

To illustrate the idea of the pre-conditioning method, let consider a simple but well-known simplification rule:

$$\sqrt{z-1}\sqrt{z+1} \overset{?}{=} \sqrt{z^2-1}. \tag{6.1}$$

Recall that the branch cut for $\sqrt{z}$ is conventionally

$$\{z \mid \Re(z) < 0 \wedge \Im(z) = 0\}.$$

Let $z = x + iy$. The set of branch cuts for (6.1) is as follows:
The branch cut for $\sqrt{z-1}$ is $\{x - 1 < 0 \wedge y = 0\}$ (Figure 6-5).
The branch cut for $\sqrt{z+1}$ is $\{x + 1 < 0 \wedge y = 0\}$ (Figure 6-6).
The branch cut for $\sqrt{z^2-1}$ is $\{x^2 - y^2 - 1 < 0 \wedge xy = 0\}$ (Figure 6-7).

These branch cuts decompose $\mathbf{C} = \mathbf{R}^2$ into three two-dimensional regions, five one-dimensional regions and three zero-dimensional regions as shown in Figure 6-8.

This optimal decomposition is not cylindrical and does not depend on variable ordering. It is unfortunate that we do not have a tool to directly produce this decomposition and have to rely on cylindrical decomposition. Cylindrical decomposition, on the other hand, depends on variable ordering and these branch cuts will produce two best cylindrical decomposition approximations to the optimal decomposition with respect to different variable orderings.

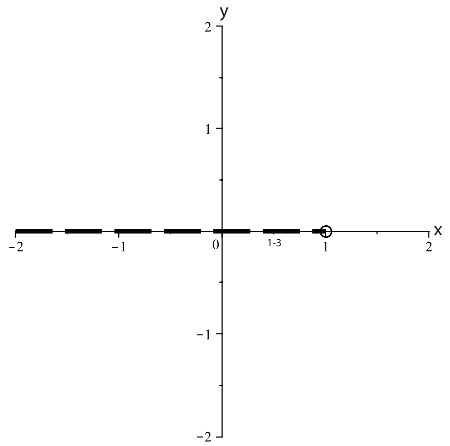


Figure 6-5: The branch cut for $\sqrt{z-1}$

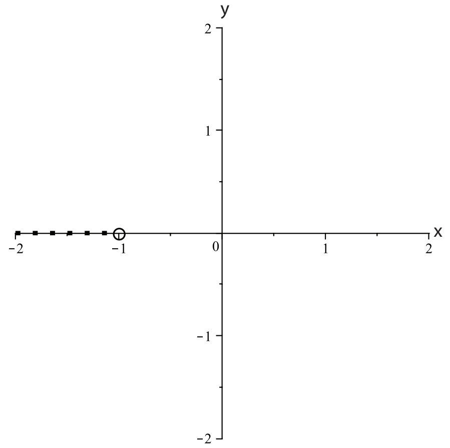


Figure 6-6: The branch cut for $\sqrt{z+1}$

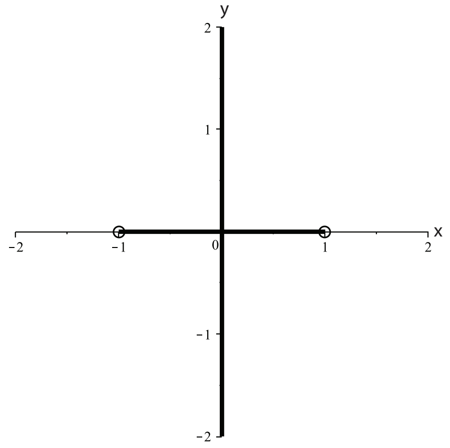


Figure 6-7: The branch cuts for $\sqrt{z^2-1}$

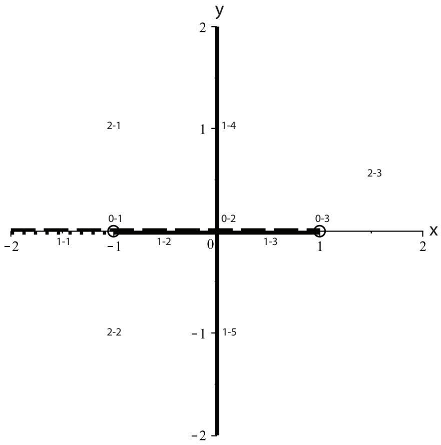


Figure 6-8: The optimal decomposition for equation (6.1)

Figure 6-9 shows the minimal cylindrical decomposition with respect to variable ordering $y > x$. $\mathbf{C} = \mathbf{R}^2$ is decomposed into seven two-dimensional regions, nine one-dimensional regions and three zero-dimensional regions.

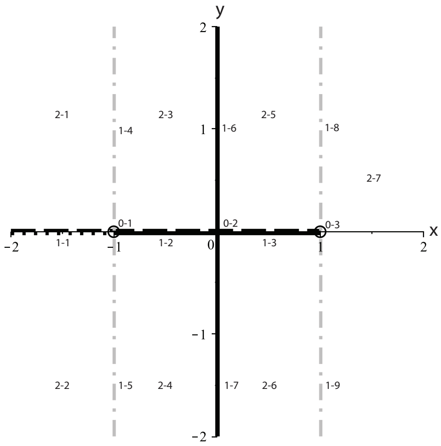


Figure 6-9: The minimal cylindrical decomposition for problem (6.1) $(y > x)$

The minimal cylindrical decomposition with respect to variable ordering $x > y$ would decompose $\mathbf{C} = \mathbf{R}^2$ into four two-dimensional regions, six one-dimensional regions and three zero-dimensional regions, see Figure 6-10.

Given that a CAD method is our only tool, the following two sections show how Maple's CAD and QEPCAD handle the problem of (6.1) when supplied with the original set of polynomials.

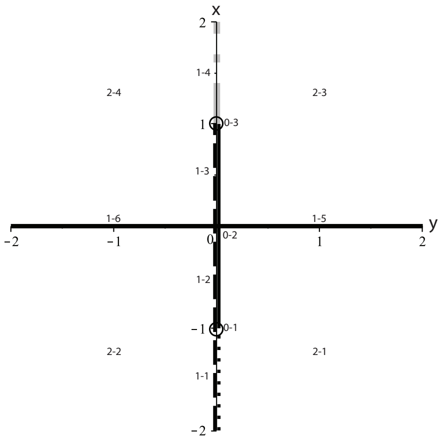


Figure 6-10: The minimal cylindrical decomposition for problem (6.1) ($x > y$)

6.2.1 Maple's CAD

Maple's CAD takes a list of polynomials:

```
> R := PolynomialRing([x, y]):
> P := [x-1, y, x+1, y, x^2-y^2-1, x*y]:
```

as an input, and therefore views the set of branch cuts as six individual polynomials. It is obvious that the redundant y in P can be removed without affecting the result.

```
> P := [x-1, y, x+1, x^2-y^2-1, x*y]:
```

Maple's CAD partitions the whole complex plane, with respect to variable ordering $y > x$, into 29 cells.

```
> R := PolynomialRing([x, y]):
> P := [x-1, y, x+1, y, x^2-y^2-1, x*y]:
> nops(CylindricalAlgebraicDecompose(P, R, output=list));
        29
```

Examining the data structure for each individual cell, we able to translate the partition into Figure 6-11.

It is apparent that the decomposition, while cylindrical, is not minimal (compare Figure 6-9: 19 cells). The two dotted line curves ($x^2 - y^2 - 1 = 0$) can be removed

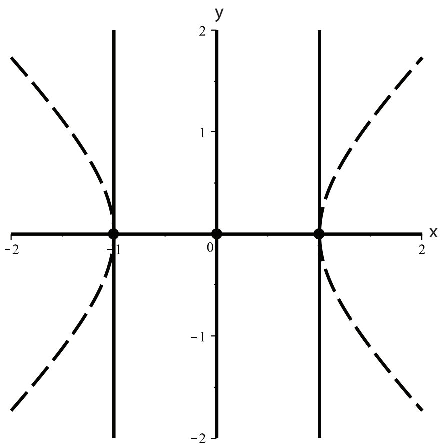


Figure 6-11: Maple's CAD for problem (6.1) ($y > x$)

without destroying the connectivity property.

6.2.2 QEPCAD

QEPCAD takes a prenex formula:

```
Enter a prenex formula:
[[x-1<0 /\ y=0]\/[x+1<0 /\ y=0]\/[x^2-y^2-1<0 /\ x y=0]].
```

as an input, and therefore, theoretically, should take into account the boolean connectives and not mix, for example, $x^2 - y^2 - 1 < 0$ and $x - 1 < 0$. In practice, however, it turns out that despite being fed in the boolean combination of equalities and inequalities, QEPCAD makes little use of this information and hence suffers from the same problem as Maple's CAD, as shown in an extract of a QEPCAD session below. Figure 6-12 gives the visualization of the corresponding CAD.

```
Enter a variable list:
(x,y)
Enter the number of free variables:
2
Enter a prenex formula:
[[x-1<0 /\ y=0]\/[x+1<0 /\ y=0]\/[x^2-y^2-1<0 /\ x y=0]].
```

$\vdots$

```
-----------------------------------------
Level                  1          2     Total
-----------------------------------------
Cells                  7         29        36
```

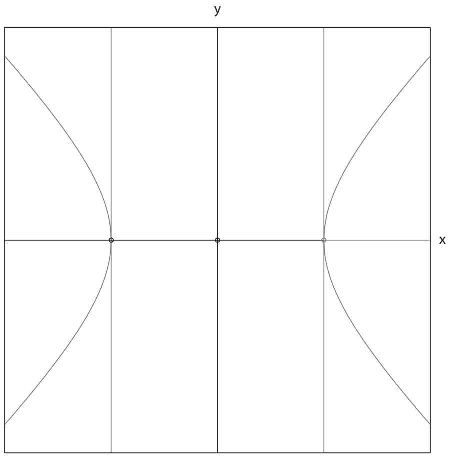


Figure 6-12: QEPCAD's CAD for problem (6.1) ($y > x$)

6.3 Pre-conditioning the Branch Cut

6.3.1 The Approach

We realised that the same branch cut can be defined by different sets of polynomials. The goal is to produce an alternative[1], hopefully simpler, but equivalent representation of the branch cut problem which should produce a smaller CAD than the initial polynomials. The concept of using a different, but equivalent for our purposes, input formula to construct a CAD, is similar to Brown and Strzebonski's Black-Box/White-Box [BS10] approach to CAD, but the underlying algorithm is different as we are working with branch cut problems while Brown and Strzebonski worked with Tarski formulae.

[1] Branch cut formulae may be unique and so such an alternative does not exist.

The current CAD algorithm partitions $\mathbf{R}^n$ into cylindrically arranged regions in which each polynomial is either identically zero or non-zero. Here, we are seeking to produce instead a decomposition in which the branch cut formula is either identically true or identically false, i.e. each polynomial is no longer considered independently, by taking advantage of the presence of the boolean connective "AND ($\wedge$)" in our branch cut formula. In this section, we propose a method which takes a pair of polynomials (an equality and an inequality) as an input and produces a polynomial that contains information of both input polynomials. Note that the method is only applicable to the complex cases where there are a pair of an equality and an inequality; the case of trivial f, for example $\{1-xy=0\}$ for $\arctan(x)+\arctan(y) \overset{?}{=} \arctan\left(\frac{x+y}{1-xy}\right)$ has no benefit here.

6.3.1.1 pprecond

For ease of exposition and without loss of generality, for the remainder of this chapter, it will be assumed that a branch cut is described by a pair of polynomials

$$f < 0 \wedge g = 0.$$

Definition 6.3.1. *Let p be a polynomial in, possibly, several variables.* $\deg(p,x)$ *and* $\operatorname{lcoeff}(p,x)$ *are the degree and the leading coefficient of p regarded as a polynomial in x respectively.*

Notation 6.3.1. *Throughout this chapter, let $\mathbf{k}[x]$ be a polynomial ring, f and g be two polynomials in $\mathbf{k}[x]$, and $d = \deg(f,x), e = \deg(g,x)$ and $c = \operatorname{lcoeff}(g,x)$.*

Definition 6.3.2. *The* pseudo-remainder*, r, of f by g with respect to the variable x is defined by*

$$mf = qg + r$$

where $\deg(r,x) < \deg(g,x)$,
$m = c^{\max(d-e+1,0)}$ *is the multiplier,*
q *is the pseudo-quotient.*

Notation 6.3.2. $[r,m] = \operatorname{prem}(f,g,x)$ *is the pseudo-remainder of f by g with respect to variable x.*

Proposition 6.3.1. *Let ϕ be an evaluation homomorphism $k[x] \to \mathbf{R}$ such that $\phi(g) = 0$, then clearly $\phi(f) = 0$ if and only if $\phi(r) = 0$. In general $\phi(r) = \phi(m)\phi(f) = \phi(c)^{\max(d-e+1,0)}\phi(f)$.*

Proposition 6.3.2. *If $\max(d - e + 1, 0)$ is even, then $\phi(r)$ and $\phi(f)$ have the same sign for any ϕ with $\phi(g) = 0$.*

Definition 6.3.3. *The method of* pre-conditioning *is to produce an alternative polynomial which is possibly smaller*[2] *than f but with a sign equivalent to f when $g = 0$.*

Algorithm 3 gives an algorithm for pre-conditioning based on the pseudo-remainder.

Algorithm 3 Pre-conditioning: `pprecond`

```
Input: A branch cut represented by $f < 0 \wedge g = 0$.
Output: Polynomial $h$ such that $h$ has lower (or the same) $x$ degree than $f$ but
    with a sign equivalent to $f$ when $g = 0$.
$d = \deg(f, x)$
$e = \deg(g, x)$
$c = \mathrm{lcoeff}(g, x)$
if $d < e$ then
  return $f$
else
  $[r, m] = \mathrm{prem}(f, g, x)$
  if $d - e + 1$ is even then
    $h = r$
  else
    $h = cr$
  end if
  return $h$
end if
```

6.3.1.2 sprecond

It may be possible to produce a smaller polynomial, h, with the same sign as f when $g = 0$ than that described above if the sparse pseudo-remainder is used instead.

[2] Smaller in the main variable but may be larger in other variables.

Definition 6.3.4. *The* sparse pseudo-remainder*, r', of f by g with respect to the variable x is defined by*

$$m'f = q'g + r'$$

where $\deg(r', x) < \deg(g, x)$,
$m' = c^k$ *is the multiplier with k is as small as possible,*
q' *is the pseudo-quotient.*

Notation 6.3.3. $[r', m'] = \text{sprem}(f, g, x)$ *is the sparse pseudo-remainder of f by g with respect to variable x.*

The sparse pseudo-remainder has the same functionality as the pseudo-remainder except that with the pseudo-remainder, the expression of the multiplier in terms of c can be calculated in advance, whereas with the sparse pseudo-remainder, the expression of the multiplier in terms of c can only be determined after computing the sparse pseudo-remainder.

Proposition 6.3.3. *If k is even, then $\phi(r')$ and $\phi(f)$ have the same sign for any ϕ with $\phi(g) = 0$.*

We now give the algorithm for pre-conditioning based on the sparse pseudo-remainder (Algorithm 4).

Remark 6.3.1. *Generally the output of `sprecond` is often the same as that of `pprecond`; it may be better (smaller) but never worse.*

6.3.2 Example

Let us revisit problem (6.1):

$$\{\{x - 1 < 0 \wedge y = 0\} \vee \{x + 1 < 0 \wedge y = 0\} \vee \{x^2 - y^2 - 1 < 0 \wedge xy = 0\}\}.$$

We have shown in Sections 6.2.2 and 6.2.1, the CADs produced by QEPCAD and Maple's CAD with variable ordering $y > x$. Figure 6-13 shows that, similar to the CAD construction with variable ordering $y > x$, the CAD produced with variable ordering $x > y$ by both QEPCAD (Figure 6-13(a)) and Maple's CAD

Algorithm 4 Sparse pre-conditioning: `sprecond`

```
Input: A branch cut represented by $f < 0 \wedge g = 0$
Output: Polynomial $h'$ such that $h'$ has lower (or the same) $x$ degree than $f$
    but with a sign equivalent to $f$ when $g = 0$.
$[r', m'] = \mathrm{sprem}(f, g, x)$
$c = \mathrm{lcoeff}(g, x)$
$k = 0$
while $m' \neq 1$ do
    $m' \leftarrow m'/c$
    $k \leftarrow k + 1$
end while
if $k$ is even then
    $h' = r$
else
    $h' = cr$
end if
return $h'$
```

(Figure 6-13(b)), is also not minimal (29 cells compared to 13 cells in Figure 6-10).

Clearly, the unwanted partitions, when using either variable ordering, come from the branch cut for $\sqrt{z^2 - 1}$. Applying the pseudo-remainder pre-conditioning to this set of branch cuts, with respect to variables x and y

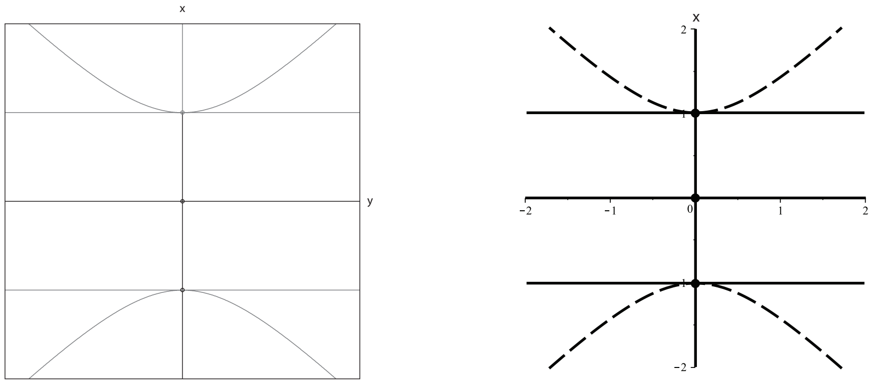
```
> f := x^2-y^2-1:
> g := x*y:
> pprecond(f, g, x);
```

$$y(-y^3 - y)$$

```
> pprecond(f, g, y);
```

$$x(x^3 - x)$$

yields two equivalent but different branch cut formulations, depending on variable



(a) QEPCAD

(b) Maple's CAD

Figure 6-13: Un-pre-conditioned CADs for problem (6.1) ($x > y$)

orderings used, for problem (6.1), i.e.

$$\{\{x-1<0 \wedge y=0\} \vee \{x+1<0 \wedge y=0\} \vee \{y(-y^3-y)<0 \wedge xy=0\}\},$$

or

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

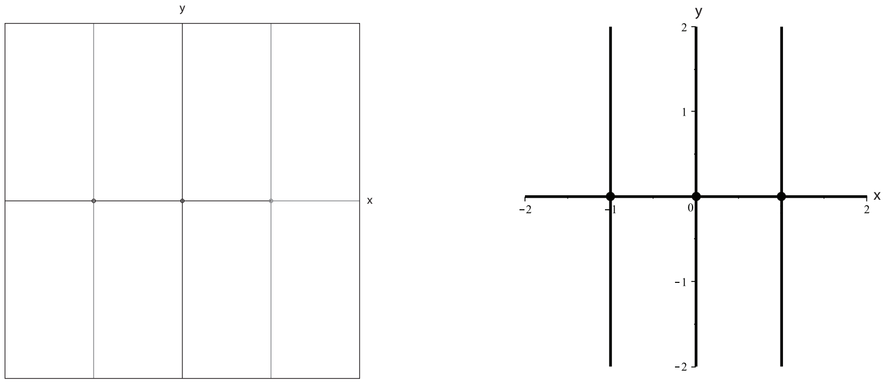
Note that for this particular example, `pprecond` and `sprecond` produce the same polynomials.

Table 6.1 shows the difference in the number of cells in the CAD constructed with different branch cut representations.

	QEPCAD		**Maple's CAD**	
	$y > x$	$x > y$	$y > x$	$x > y$
Without pre-conditioning	29	29	29	29
x-pre-conditioning	21	21	21	21
y-pre-conditioning	15	21	21	21

Table 6.1: Number of constructed cells for problem (6.1)

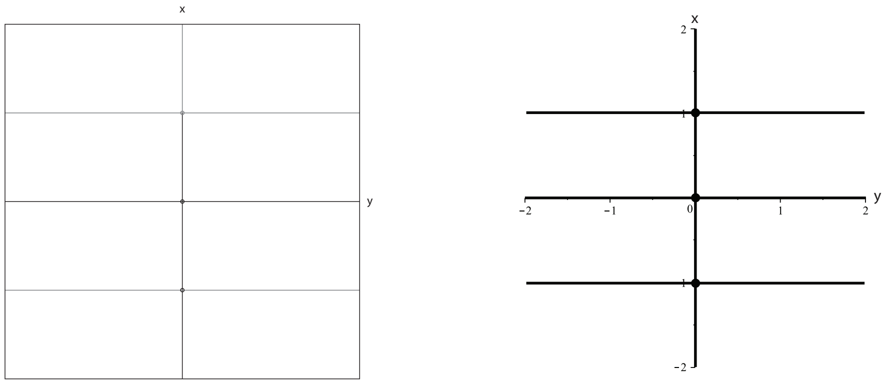
Both QEPCAD and Maple's CAD produce the same results for the original problem, and the same (but smaller CADs) for the x-pre-conditioned problem. But for the y-pre-conditioned problem, with one of the projection orders (i.e. $y > x$),



(a) QEPCAD

(b) Maple's CAD

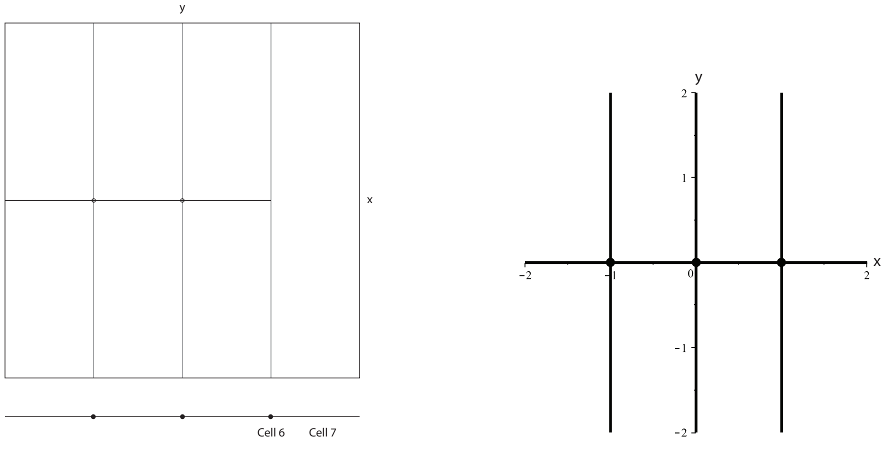
Figure 6-14: x-pre-conditioning CADs for problem (6.1) ($y > x$)



(a) QEPCAD

(b) Maple's CAD

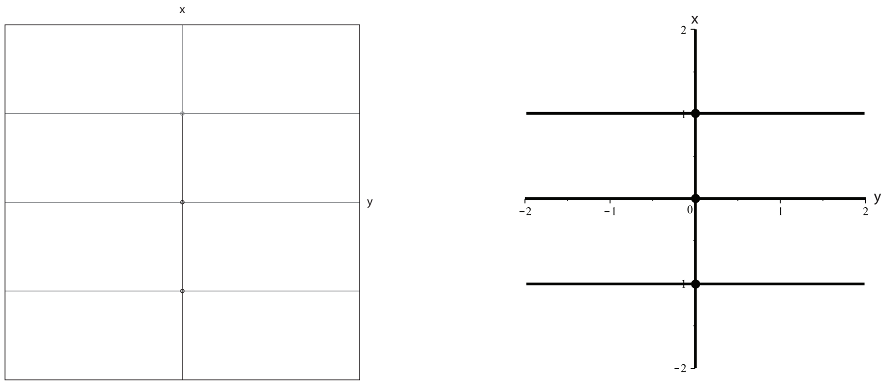
Figure 6-15: x-pre-conditioning CADs for problem (6.1) ($x > y$)



(a) QEPCAD

(b) Maple's CAD

Figure 6-16: y-pre-conditioning CADs for problem (6.1) ($y > x$)



(a) QEPCAD

(b) Maple's CAD

Figure 6-17: y-pre-conditioning CADs for problem (6.1) $(x > y)$

QEPCAD finds an additional simplification (recall that QEPCAD performs partial CAD). Analysing the data structure of the constructed CAD shows that when extending the decomposition D_1 of the real line R^1 to the decomposition D_2 of $\mathbf{R}^2$, the rightmost point (cell 6) and the right open interval (cell 7) are not lifted (see Figure 6-16(a)), resulting in a CAD of only part of $\mathbf{R}^2$. This illustrates the subtle interactions that are going on. Fortunately, for this example, QEPCAD provides enough sample points for our purpose to test the identity. However, this requires knowledge of CAD to perform the analysis, something we cannot expect every general user of our system to have.

6.4 Summary

In this chapter we have seen that neither of the software tools at our disposal makes good use of the information we have:

Maple's CAD which is obvious as it only takes polynomials as input;

QEPCAD which is more surprising as it is meant to take advantage of the logical structure of formulae.

In connection with this, we have presented algorithms to allow us to make better use of the information we have available. Essentially, the algorithms convert information we have into a potentially better choice of polynomials. It has been empirically demonstrated (see Section 8.2.2.3) that pre-conditioning the branch cuts often helps in reducing the number of cells produced by CAD. QEPCAD, despite taking in as input a boolean combination of equalities and inequalities, and therefore being aware of all information available without pre-conditioning, also benefits from it. It does not, however, guarantee that the algorithms will produce polynomials that will produce a CAD with fewer cells (example 17 in Tables 8.13 and 8.14 is an example when it does not), but for most simple cases they do (see Tables 8.13 and 8.14).

The different representation of branch cuts which is obtained through such a procedure will in turn construct different sign-invariant CADs and may make adherence (Section 7.2) harder, but this has yet to be empirically tested.

Chapter 7

Identity Evaluation

In this chapter we consider the final phase (step 3c on page 23): the component testing, of the algebraic simplification system. The validity of the proposed identity is evaluated on each relevant[1] component of that decomposition, using the sample point of each cell constructed by CAD. This can be done either numerically or symbolically. The procedure is surprisingly non-trivial and both the numerical and symbolic approaches pose problems which we discuss in the first section. The second section introduces the concept of *adherence* of branch cuts to tackle the problem of evaluation on non-full-dimensional regions. In the last section we describe *post-conditioning* the CAD as a possible way to facilitate this last phase of the system.

7.1 Problematic Issues

Definition 7.1.1. *A* sample point *for a cell is any point in the cell.*

Definition 7.1.2. *An identity, which is generally false in a cell, may nevertheless be accidentally true at a sample point. We call this the* accidental truth *problem.*

[1] Assumptions [WG93] made by the user, or an explicit lack of interest in lower-dimensional components, may mean that not all components need to be analysed. Also, if we have found a full-dimensional component on which the "identity" is false, there is little point in testing lower-dimensional components.

For example $\overline{(z+1)\log(z)} = (\overline{z}+1)\log(\overline{z})$ is generally false on the branch cut $(y = 0 \wedge x < 0)$, with error $2\pi i(x+1)$, but happens to be true at $z = -1$, which might well be chosen as a sample point.

- **Numerical**
  Numerical evaluation [BD02] suffers from the accidental truth problem. Furthermore, it requires the sample point, say $p = (x, y)$, to have finite floating point representation. Hence when evaluating the identity on branch cuts, or more generally, on the cells of dimension less than the full-space of the decomposition it may give completely incorrect results. Error bounds cannot be used here as the smallest error may result in choosing a sample point from an adjacent cell instead. For example $\log\left(\frac{1}{2+i\pi}\right) = -\log(2+i\pi)$ is false on $y = -i\pi, x < 0$, but $\log\left(\frac{1}{z}\right) = -\log(z)$ is true for every floating point z. This can be described as "falling off branch cuts".

- **Symbolic**
  There are two possible symbolic computations: one is direct symbolic evaluation, which has the same accidental truth problem as numeric evaluation. Another symbolic computation [BBD03, BBDP04] is based on the method of [vdH02], which checks enough of the series expansion of the claimed identity to distinguish accidental truth from genuine truth. The method is, however, computationally intensive. Additionally, both methods require the sample points to be explicit. Thus they cannot be used when the sample points cannot be expressed in terms of radicals. Even when the sample points are expressible in terms of radicals, there remains the *constant problem* [Ric97] and we would need to resort to algorithms which rely on the truth of number-theoretic conjectures.

7.2 Adherence

The adherence method [BBP05] provides a way to evaluate the identity on the non-full-dimensional cells. This is advantageous for two reasons:

1. the non-full-dimensional cells are more difficult ("falling off branch cut" is-

sue, and the sample points may be algebraic which makes evaluation trickier),

2. there are generally more of them, so reducing the problem to full-dimensional cells is a real efficiency.

The method is based on the concept of adjacency of the cells in the decomposition, which has been used to simplify CADs [ACM84b, ACM85, MC02].

Definition 7.2.1. *Two disjoint cells are* adjacent *if their union is connected.*

Notation 7.2.1. *$R_S(f)$ denotes the Riemann surface for function f.*

Recall from Section 2.3.4 that $R_S(f)$ is a path-connected domain having either n or an infinite number of sheets depending on the function f.

Definition 7.2.2. *Let D be a CAD with respect to the branch cuts of f. Suppose $c \subset D$ is a branch cut (section) cell and $s \subset D$ is an adjacent sector cell to c. Then c* adheres *to s if c belongs to the same sheet of $R_S(f)$ as s.*

Recall that for the examples in this thesis, we follow the modern convention, i.e. using counter-clockwise closure (CCC): any other convention would work in the same way. Thus $\arg(z) \in (-\pi, \pi]$ and the branch cut $c \in \mathbf{C}$ adheres to the side where $\arg(z) > 0$ (Figure 7-1), or in CAD terminology, the branch cut cell $(1, 2)$ adheres to the cell $(1, 3)$ rather than to the cell $(1, 1)$.

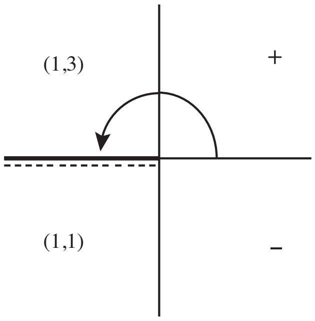


Figure 7-1 Counter-clockwise closure

In [Kah87], Kahan introduced the concept of signed zeros (see Section 2.3.1): having both 0^+ and 0^- allows branch cuts to adhere to either side. The scheme is useful when doing numerical computation. However, it is not natural mathematically for zero to have two values, and less than ideal for our situation since in the sample point testing phase, we want to be able to evaluate the identity at a point on a branch cut, and obtain a single answer from which we can conclude the accuracy of the simplification on the cut.

7.2.1 The Idea

For the rest of this chapter, we will only consider problems comprising of one complex variable or two real variables, resulting in two-dimensional CADs. This ensures that the adjacent cell to a branch cut (which is therefore one-dimensional) is always of full-dimension. For higher dimensional cases, such as four-dimensional CADs arising from two complex variables or four real variables, complications, such as singularities, on which the adherence cannot be calculated, and the possibility that several components of a branch cut having different dimensions from another, may arise, and this requires further study. Furthermore, we note that even pure adjacency in CAD is only solved up to dimension 3 [ACM85, ACM88]. In addition, we limit our discussion in this thesis to the case where the branch cut cell c derives from a single $h = f(g(z))$ only. The treatment for the case where $h \overset{?}{=} 0$ contains several building blocks $h_i = f_i(g_i)$ and c derives from more than one h_i can be found in [BBDP07].

Let c be a branch cut of $h = f(g(z))$ where f is a logarithm or n^{th} root, and g is a rational function in a complex variable. Let $\beta : [0, 1] \to \mathbf{C}$ be a path with $\beta(0) = p$ and $\beta(1) = q$, with $q \in c$, and let $g(q) = s$ where $s \in \mathbf{R}^-$ ($\mathbf{R}^-$ is a branch cut of f), see Figure 7-2.

Then if $g(\beta(t)) \to s + 0^+i$ as $t \to 1$ then c adheres to s_1, where s_1 is the cell that contains p (Figure 7-3), otherwise c adheres to another cell, s_2 say (Figure 7-4).

Once we find which adjacent cell the branch cut adheres to, we can replace a sample point on the cut with a point well off the cut in the adherence cell. In fact, we can regard the branch cut cell to be part of the adherence cell, and do not need to test this cut cell at all.

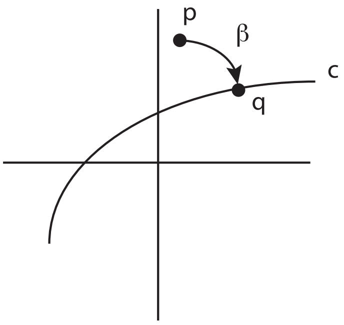


Figure 7-2 Adherence: general principle

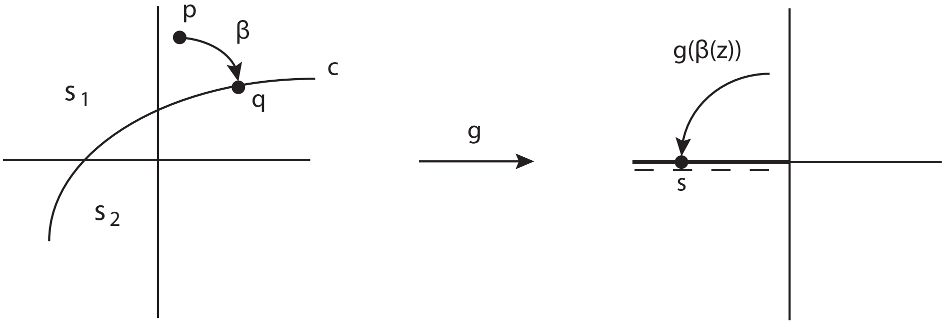


Figure 7-3: Adherence: $g(\beta(t)) \to s + 0^{+}i$ as $t \to 1$

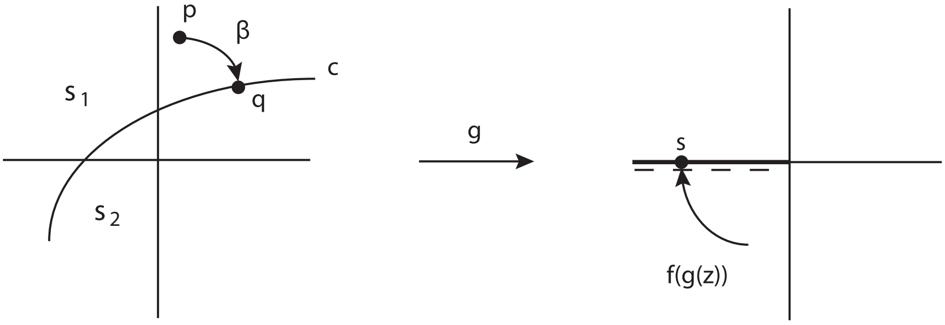


Figure 7-4: Adherence: $g(\beta(t)) \to s + 0^{-}i$ as $t \to 1$

7.2.2 The Algorithm

Definition 7.2.3. *Let f be a logarithm or n^{th} root. The* Riemann index*, $R_I(f)|_c$, of f on a cell c of the CAD induced by the branch cuts is*

$$R_I(f)|_c = \begin{cases} 2k\pi i, & \textit{if } f \textit{ is a logarithm } (\textit{where } k \in \mathbf{Z}); \\ \exp\left(\frac{2k\pi i}{n}\right), & \textit{if } f \textit{ is } n^{th} \textit{ root } (\textit{where } k \in \mathbf{Z}_n). \end{cases}$$

Definition 7.2.4. *Define*

$$\operatorname{sign}(x) = \begin{cases} 1, & \textit{if } x > 0; \\ -1, & \textit{if } x < 0. \end{cases}$$

Definition 7.2.5. *The* truth value *of cell c,* tv(c)*, with respect to a branch cut formula ϕ is a boolean value of true or false. It is equal to true if $\phi(p)$ is satisfied at any $p \in c$ and it is false otherwise.*

Algorithm 5 gives an algorithmic description of the idea described in Section 7.2.1.

Algorithm 5 Adherence: Non-nested roots

```
Input: A CAD, $D$, of the set of branch cuts of $h = f(g(z))$.
Output: Adherence cells determined.
for each $c \in D$ do
    if $\mathrm{tv}(c) = True$ then
        $sgn := \mathrm{sign}(\Im(g(p))), p \in s_1$, with $s_1, c$ adjacent.
        if $sgn = 1$ then
            $R_I(h)|_c = R_I(h)|_{s_1}$
        else
            $R_I(h)|_c = R_I(h)|_{s_2}$ where $s_2$ is adjacent to $c$ and not to $s_1$.
        end if
    end if
end for
```

7.2.3 Non-Nested Roots

There are two possibilities for c.

1. If c is not a vertical line. Then cells s_1 and s_2 are the two sector cells that are adjacent to c in the same stack (Figure 7-5). In this case we do not need to calculate adjacency of the CAD since intra-stack adjacency alone is sufficient.

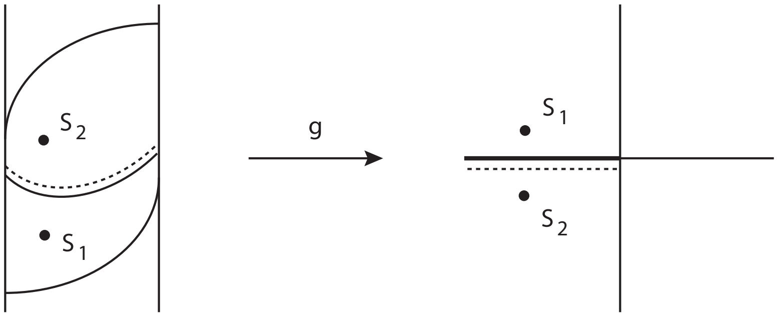


Figure 7-5: Adherence: c is not a vertical line

2. If c is a vertical line. Then cell s_1 is an adjacent sector cell to c which lies in either of the two adjacent stacks to the one in which c resides, and cell s_2 is another adjacent sector cell to c but in a different stack to the one in which s_1 resides (Figure 7-6). In this case, we need to calculate adjacency of the CAD.

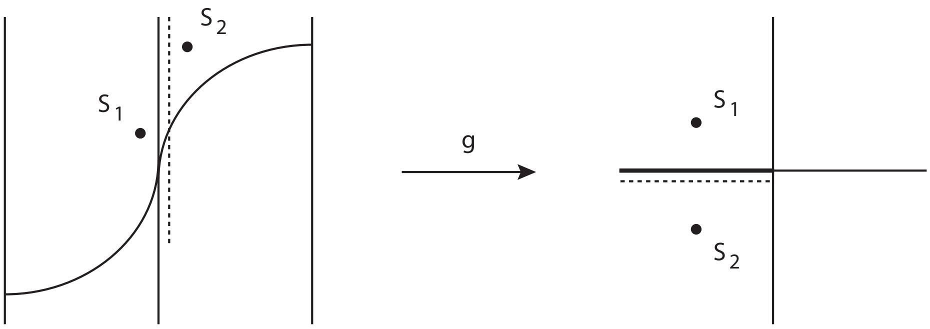


Figure 7-6: Adherence: c is a vertical line

Note that there will be a finite number of adjacent cells s_1 (or s_2) to c, and we only need to choose one cell for each branch cut.

7.2.4 Nested Roots

Each branch cut cell c falls into one of the following three categories.

1. A line which is really a branch cut. Of course the branch cut formula is *true* on this cell.
2. A line which we think it is a branch cut. The branch cut formula is *true*, but it is actually a spurious branch cut which is a result of the de-nesting process (see Section 4.3.3).
3. A lower-dimensional cell on which the branch cut formula is $false$. This is an artefact of building the CAD.

We only need to evaluate the cells which belong to the first two categories (Section 7.3). In this case, when computing the adherence of c with respect to h, we need to compute the adherence of c with respect to g first, before computing the adherence with respect to f. This is to ensure that g is continuous onto c.

It is clear that if c is a spurious branch cut then c, s_1 and s_2 will belong to the same sheet of $R_s(h)$, and therefore c adheres to both s_1 and s_2. Although the correctness of the result is guaranteed, it is wasteful in that we must compute $g(p),\ p \in s_i$. Unfortunately, we currently do not have an algorithm to detect such spurious cells.

7.2.5 Branch Cuts "at Infinity"

Consider (this analysis is from [BBDP05, Section 3, example 1])

$$h = \arctan(x) + \arctan(y) - \arctan\left(\frac{x+y}{1-xy}\right) \overset{?}{=} 0, \tag{7.1}$$

where $x, y \in \mathbf{R}$ are finite.

$\arctan(x)$ is a continuous, bijective and differentiable function $(-\infty, \infty) \to \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$. Therefore, although $\arctan(x)$ has no branch cuts over the reals, it has a branch cut at infinity since $\lim_{x\to+\infty} \arctan(x) = \frac{\pi}{2}$ whereas $\lim_{x\to-\infty} \arctan(x) = -\frac{\pi}{2}$.

Furthermore, it is possible for $\frac{x+y}{1-xy}$ to pass through ∞ even when both x and y are finite, that is when $1 - xy = 0$.

Recall that our branch cut representation is only for finite branch cuts, therefore the only branch cut for h is the branch cut at infinity deriving from $\arctan\left(\frac{x+y}{1-xy}\right)$. The branch cuts decompose the (x, y)-plane into three two-dimensional regions and two one-dimensional regions[2] as shown in Figure 7-7.

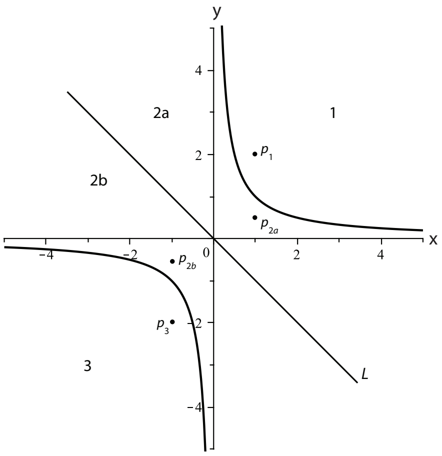


Figure 7-7: Decomposition for equation (7.1)

Evaluating the identity on each of the two-dimensional regions shows that the identity is true on region 2 and false on regions 1 and 3 and requires correction factor π and $-\pi$ respectively (Figure 7-8). Now it is left to evaluate the identity on the cuts.

We define $\arctan(\infty) = \frac{\pi}{2}$, so that $\arctan(x)$ is a function of the form $(-\infty, \infty] \to \left(-\frac{\pi}{2}, \frac{\pi}{2}\right]$. Thus, we are working with the one-point completion domain $(-\infty, \infty] = \mathbf{R} \cup \{\infty\}$, and the point $\left(\infty, \frac{\pi}{2}\right)$ belongs to the principal branch of $\arctan(x)$ whilst the point $\left(-\infty, -\frac{\pi}{2}\right)$ belongs to the branch $\arctan(x) - \pi$. Therefore, here, we work with *positive infinity* only, that is we pass onto the point ∞ continuously if we let $x \to \infty^+$, but not when we let $x \to \infty^-$.

Let $c_1 = \{(1 - xy = 0) \wedge (x > 0)\}$ and $c_2 = \{(1 - xy = 0) \wedge (x < 0)\}$. Suppose $g = \frac{x+y}{1-xy}$. Adherence can be used to evaluate the identity on each branch cut c_i by determining whether g tends to $+\infty$ or $-\infty$. In this case c_i will adhere to

[2] CAD will in fact decompose the plane into seven regions.

the cell where g is *positive*, since $\arctan(\infty) = \arctan(+\infty) = \frac{\pi}{2}$. We proceed as follow.

1. Construct a CAD which is sign-invariant with respect to g. This can be done by adding the line $L = \{(x, y) | \ x + y = 0\}$ to the CAD.

2. Evaluate the identity on each branch cut c_i.

   (a) $\mathbf{c_1}$. The adjacent cells are region 1 and region 2a. Suppose our sample points are $p_1 = (1, 2)$ and $p_{2a} = \left(1, \frac{1}{2}\right)$. Then $g(p_1) = -3$ and $g(p_{2a}) = 3$. Hence c_1 adheres to region 2, and therefore the identity is true on this branch cut.

   (b) $\mathbf{c_2}$. The adjacent cells are region 3 and region 2b. Suppose our sample points are $p_3 = (-1, -2)$ and $p_{2b} = \left(-1, -\frac{1}{2}\right)$. Then $g(p_3) = 3$ and $g(p_{2b}) = -3$. Hence c_2 adheres to region 3, and therefore the identity is false on this branch cut and requires a correction factor of $-\pi$.

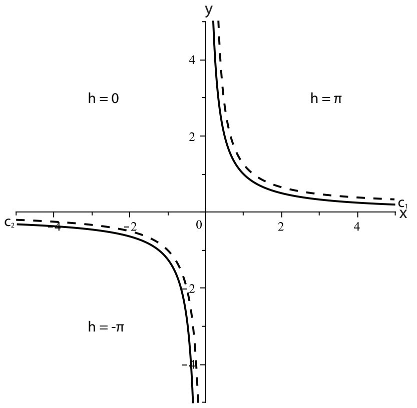


Figure 7-8: Adherence for equation (7.1)

7.3 Post-Conditioning the CAD

We reiterate that, as in previous section, we will only consider two-dimensional CADs, arising from one complex variable or two real variables.

It was demonstrated in Chapter 6 that CAD is too powerful and generally produces more cells than the actual number of connected components defined by the branch cuts. Evaluating every single constructed cell means more cells are tested than necessary. This obviously affects the efficiency of the algorithm. This section presents a method to improve the efficiency of the identity evaluation phase. The method is based on two observations.

Notation 7.3.1. *Let* $\mathrm{cf}(c_{i,j})$ *denotes the correctness factor, true/$false$*[3]*, of cell* $c_{i,j}$.

1. **Observation 1.** Many unwanted cells in the CAD construction arise from section cells where the truth values of the branch cut formulae in the cells are false. Some of these section cells can be removed with the pre-conditioning method, such as the dotted parabola lines $x^2 - y^2 = 1$ in Figure 6-11. Others which are features of cylindricity, such as the vertical lines $x = \pm 1$ in Figure 6-11, cannot be removed with the pre-conditioning method. These cells also give rise to unwanted sector cells. Thus, we can avoid testing these unwanted cells by simply examining the truth values of the branch cut formula of these cells from the CAD data structure.

2. **Observation 2.** Suppose we have determined the correctness factor for cell $c_{i,j}$, $\mathrm{cf}(c_{i,j})$. Then for an appropriate n_i, we can assign the value $\mathrm{cf}(c_{i,j})$ to each cell in the set $S = \{c_{i,t} | \, \mathrm{tv}(c_{i,t}) = False, t = j+1, \ldots, n_i\}$ without further work.

7.3.1 The Algorithms

For simplicity, suppose $h = f(g(z))$ where f is either a logarithm or n^{th}-root function, contains only one complex variable. Let $c_{i,j}$ denote the cell with index (i, j). There are 2 cases to consider, 1-d and 2-d stacks. All cells in each of the 2-d stacks must be dealt with first (Algorithm 6), before considering the remaining 1-d stacks (Algorithm 7).

[3] In practice, for simple examples, we can associate an explicit correction such as π and $-\pi$, as in problem (7.1).

Algorithm 6 Sample point testing with post-conditioning: 2-d stacks

Input: A CAD of M stacks, each with N_i cells.

Output: A CAD where each cell $c_{i,j}$ in 2-d stacks has the correctness factor $\mathrm{cf}(c_{i,j})$ for $h = 0$ assigned.

```
for $i = 1$ to $M$ by 2 do
    Compute $\mathrm{cf}(c_{i,1})$.
    for $j = 2$ to $N_i$ by 2 do
        if $c_{i,j}$ is a branch cut then
            if $c_{i,j}$ adheres to $c_{i,j-1}$ then
                $\mathrm{cf}(c_{i,j}) \leftarrow \mathrm{cf}(c_{i,j-1})$
                Compute $\mathrm{cf}(c_{i,j+1})$.
            else
                Compute $\mathrm{cf}(c_{i,j+1})$.
                $\mathrm{cf}(c_{i,j}) \leftarrow \mathrm{cf}(c_{i,j+1})$
            end if
        else
            $\mathrm{cf}(c_{i,j}) \leftarrow \mathrm{cf}(c_{i,j-1})$
            $\mathrm{cf}(c_{i,j+1}) \leftarrow \mathrm{cf}(c_{i,j-1})$
        end if
    end for
end for
```

Note that, in Line 3 of Algorithm 6, the "by 2" is a consequence of the *sector/section/sector/*... structure. Here we step through full-dimensional cells, i.e. sector cells.

In the case of a 1-d stack, we must perform a local adjacency algorithm to determine an adjacent cell to c which lies in one of the other two possible stacks, say S_1, S_2.

Algorithm 7 Sample point testing with post-conditioning: 1-d stacks

Input: A CAD with the correctness factors, $\mathrm{cf}(c)$ for all cells of 2-d stacks already determined.

Output: A CAD with each cell c is assigned a correctness factor, $\mathrm{cf}(c)$.

```
for each 1-d stack do
  for each cell $c$ do
    if $\mathrm{tv}(c) = True$ then
      Determine adjacent cell $s_1$ to $c$ from $S_1$.
      if $c$ adheres to $s_1$ then
        $\mathrm{cf}(c) = \mathrm{cf}(s_1)$
      else
        Determine adjacent cell $s_2$ to $c$ from $S_2$.
        $\mathrm{cf}(c) = \mathrm{cf}(s_2)$
      end if
    end if
  end for
end for
```

7.4 Summary

This chapter introduced the notion of the adherence of a branch cut. The method helps facilitate the identity evaluation phase. It provides a way to test the identity on a non-full-dimensional branch cut which can be problematic if using numerical or direct symbolic evaluations alone. The second part of the chapter presented the idea of post-conditioning the CAD, an attempt to improve the efficiency of the identity evaluating phase by reducing the number of examined regions.

Chapter 8

Implementations and Evaluations

In this chapter we investigate the practical aspects of the methods discussed in Chapters 4–6, focusing particularly on the feasibility of CAD in our framework. All implementations and experiments are run in QEPCAD version 1.58[1] which is compiled with SACLIB library version 2.2.5, and Maple version 14[2]. The first section gives an overview of our implementations. The second section, which forms the main part of this chapter, summarises our findings for the CAD construction phase.

8.1 Implementations

The implementations include routines to extract the relevant branch cuts of the proposed identities allowed by our restrictions, with optional commands for flexibility in displaying the computed set of branch cuts, to pre-condition the set of

[1] Some earlier work was done in earlier versions of QEPCAD. The most recent version of QEPCAD when this writing up was concluded, was version 1.65, but we stuck with QEPCAD 1.58 for consistency.

[2] Some earlier work was done in Maple 13. As this writing up was being concluded, Maple 15 came out, but we stuck with Maple 14 for consistency.

branch cut polynomials, and to compute the greedy projection algorithm. All the code, except for the greedy projection algorithm implementation, is done in Maple. While we have endeavoured to make the code efficient, we are not at this stage aiming to develop a complete and sophisticated simplification system. The primary goal is to produce necessary results for our analysis.

branchcuts

The procedure `branchcuts` takes as an input a function f and returns a set of branch cuts of f.

```
> f := sqrt(z^2-1) - sqrt(z-1)*sqrt(z+1):
> bc := branchcuts(f);
bc := {{y = 0, x < -1}, {y = 0, x < 1}, {2*x*y = 0, x^2-y^2 < 1}}.
```

Figure 8-1: Maple: `branchcuts`

Recall that the set of branch cuts of f comprises the union of the set of branch cuts of f_i where f is built up from the f_i. Furthermore, the branch cut of f_i is defined by semi-algebraic equations in real and imaginary parts. Therefore the `bc` in Figure 8-1 should be read as:

$$\{\{y = 0 \wedge x < -1\} \vee \{y = 0 \wedge x < 1\} \vee \{2xy = 0 \wedge x^2 - y^2 < 1\}\}.$$

qepcadbc and maplebc

We want to pass the set of branch cuts to either QEPCAD or Maple's CAD to do the decomposition. So representing `bc` in forms which are acceptable by QEPCAD and Maple's CAD would be convenient. Hence we provide the commands `qepcadbc` and `maplebc` which translate `bc` in Figure 8-1 to a set of branch cuts in QEPCAD and Maple's CAD input formats respectively.

```
> qepcadbc(bc);
[[y = 0 /\ x < 1] \/ [y = 0 /\ x < -1] \/ [2 x y = 0 /\ x^2-y^2 < 1]].
```

Figure 8-2: Maple: `qepcadbc`

```
> maplebc(bc);
[y, -1+x, y, 1+x, 2*x*y, -1+x^2-y^2]
```

Figure 8-3: Maple: `maplebc`

For convenience, the qepcadbc and maplebc procedures have been integrated into the branchcuts procedure and can be called using the following commands to directly produce 8-2 and 8-3 respectively:

```
> branchcuts(f, output=qepcad):
> branchcuts(f, output=maple):
```

Figure 8-4: Maple: branchcuts(f,‘output’)

pprecond and sprecond

The procedures pprecond and sprecond are based on the *pre-conditioning* method described in Chapter 6. The difference is that the former uses the pseudo-remainder method, while the latter uses the sparse pseudo-remainder method. In the Maple session in Figure 8-5, f and g are two related polynomials regarded as inequality and equation respectively. The third argument is a main variable. They return a polynomial equivalent to f.

```
> f := x^2-y^2-1:
> g := x*y:
> pprecond(f, g, x);
```

$$y(-y^3 - y)$$

```
> sprecond(f, g, y);
```

$$x(x^3 - x)$$

Figure 8-5: Maple: pprecond and sprecond

Greedy projection algorithm

It is natural to use the same projection operator[3] as the tool we are using. We have, therefore, added our code to QEPCAD to extract the projection factor sets after each projection step in order to compute $\mathrm{sotd}_k(P_k, (x_{i_1} \ldots, x_{i_k}))$ as used by the greedy projection algorithm (Section 5.2.6.1).

[3] Different projection operators may produce different projection factor sets which in turn yield different $\mathrm{sotd}_k(P_k, (x_{i_1} \ldots, x_{i_k}))$ values.

8.2 Results and Evaluation

These experiments are run using well-known simplification rules from Abramowitz and Stegun [AS64]. Since all inverse elementary functions can be defined in terms of the logarithm and square root, most of our examples are comprised of these functions. However, we will show a few examples of other inverse elementary functions as well.

8.2.1 Branch Cuts Phase

All branch cuts are computed on a 1.86 GHz Intel Core 2 Duo, and all timings are given in milliseconds. Tables 8.1 and 8.2 show the times Maple took to calculate the branch cuts for the problems listed in Appendix D. Two different timings corresponding to two nested root removal methods are listed for each problem in Table 8.2. The result shows an obvious gain in efficiency when we use resultants instead of Gröbner bases. These timings in Table 8.2 exclude the times to remove the extraneous variable by QE, if required. These times are recorded separately in Table 8.3, which shows the time taken after initialisation. Note that the variable ordering used when solving the QE problem in example 29 is $v > u > y > x$. The ordering was chosen purely at random.

Sample sets of un-pre-conditioned branch cuts for examples 1, 3, 4 and 11 are listed in Table 8.4.

8.2.2 Decomposition Phase

This section looks at the performance of QEPCAD and Maple's CAD as tools to decompose the complex plane. Cell count is used as our evaluating measure. This is because it drives the cost of the next phase (component testing phase) of our simplification system, which we have not gone into in great detail here, and more pragmatically it is the only consistent measure in our experiments.

Example	**Dimension**	**Time (10^{-3} s)**
1	1**C**	3
2	1**C**	3
3	1**C**	9
4	1**C**	9
5	1**C**	4
6	1**C**	3
7	1**C**	6
8	1**C**	6
9	1**C**	5
10	1**C**	6
11	2**R**	3
12	2**R**	3
13	2**R**	2
14	2**R**	3
15	1**C**	26
16	1**C**	13
17	1**C**	13
21	2**C**	11
22	2**C**	11
23	2**C**	26
24	2**C**	13
25	2**C**	11
26	2**C**	14
27	2**C**	16
28	2**C**	22

Table 8.1: Time taken to compute branch cuts (non-nested roots)

Example	**Dimension**	**Time (10^{-3} s)**	
		Resultant	**Gröbner basis**
18(a)	1**C**	44	64
18(b)	1**C**	76	100
19(a)	2**R**	24	44
19(b)	2**R**	20	48
20	2**R**	28	108
29	2**C**	36	69

Table 8.2: Time taken to compute branch cuts (nested roots)

Example	**Dimension**	**Time** (10^{-3} **s**)
18(a)	1$\mathbf{C}$	44
18(b)	1$\mathbf{C}$	76
19(a)	2$\mathbf{R}$	16
19(b)	2$\mathbf{R}$	20
29	2$\mathbf{C}$	134472

Table 8.3: Time taken to remove extraneous variable

1.	$\{x^2 - y^2 < 0 \vee 2xy = 0\}$
3.	$\{1 - x < 0 \wedge y = 0\} \vee \{1 + x < 0 \wedge y = 0\} \vee \{1 - x^2 + y^2 < 0 \wedge xy = 0\}$
4.	$\{x - 1 < 0 \wedge y = 0\} \vee \{x + 1 < 0 \wedge y = 0\} \vee \{x^2 - y^2 - 1 < 0 \wedge xy = 0\}$
11.	$\{xy - 1 = 0\}$

Table 8.4: Branch cut formulae for examples 1, 3, 4 and 11

8.2.2.1 Two-Dimensions (1C/2R)

In this section, we consider the problem arising from formulae containing one complex variable or two real variables. Table 8.5 reports the number of cells produced by QEPCAD and Maple's CAD in the decomposition step. The results demonstrate that in most examples, QEPCAD and Maple's CAD produce exactly the same number of cells. The only exception to this was when QEPCAD only constructed the decomposition of partial spaces. As argued in Section 5.2.4, the smaller number of cells produced by QEPCAD does not necessarily mean that it is superior to Maple's CAD. This is because in these cases we may not have enough sample points for our purpose, and therefore are required to do further examination to determine if we have sufficient sample points.

It is not surprising that in examples 1–14 both variable orderings produce the same number of cells as there is a symmetry between the two variables, x and y, in the sets of polynomials representing the branch cuts. It is less apparent in the cases where there is no symmetry between the variables how the variables should be ordered.

Examples 18, 19 and 20 represent examples with nested square roots, which need

Table 8.5: Number of constructed cells for examples 1-20

Example	**Dimension**	**QEPCAD**		**Maple's CAD**	
		$y > x$	$x > y$	$y > x$	$x > y$
1	1**C**	17	17	17	17
2	1**C**	17	17	17	17
3	1**C**	29	29	29	29
4	1**C**	29	29	29	29
5	1**C**	22†	25	25	25
6	1**C**	33	33	33	33
7	1**C**	9	9	9	9
8	1**C**	9	9	9	9
9	1**C**	17	17	17	17
10	1**C**	25	25	25	25
11	2**R**	7	7	7	7
12	2**R**	14†	14†	43	43
13	2**R**	6†	6†	9	9
14	2**R**	6†	6†	9	9
15	1**C**	391	373	391	373
16	1**C**	663	615	663	615
17	1**C**	1209	1627	1209	1627
18(a)	1**C**	47	99	47	99
18(b)	1**C**	69	141	69	141
19(a)	2**R**	10†	10†	25	25
19(b)	2**R**	10†	10†	25	25
20	2**R**	13	13	13	13

† QEPCAD only computed a partial decomposition, so the cell count is an underestimate.

to be removed using resultants, or Gröbner bases, or repeated squaring methods. For all three examples, the root free polynomials produced by the different methods happen to have exactly the same representation. In general, they may not be identical, but they will be equivalent. Examples 18 and 19 require us to solve QE problems to remove the extraneous variables, before constructing the CADs. For each example, the process produces two equivalent but different branch cut formulations, depending on the variable orders used. The result of example 18 indicates that different representations of the same set of branch cuts may result in different numbers of constructed cells. If CAD is involved, then the branch cut construction step will also depend on variable order.

We now look at the performances of the greedy projection algorithm and Brown's heuristic on each example. It must be noted that both methods were developed based on CAD-PL which has a totally different underlying algorithm to that of Maple's CAD. Moreover, if we are to run Maple's CAD in isolation, we will not be able to choose a variable order using the greedy projection algorithm since theoretically we do not have projection factor sets to compute $\mathrm{sotd}_k(P_k, (x_{i_1} \ldots, x_{i_k}))$. We also note that in the two-dimensional case, the greedy projection algorithm computes both of the possible projections. For most examples, except examples 15–18, the greedy projection algorithm was unable to make a choice between the two possible variable orderings (and the two branch cut representations in example 19). This could mean that it thinks both variable orderings are equally good, which they are. The experiments show that the resulting CADs for these examples have the same cell counts regardless of which variable ordering or which branch cut formulae are used. For examples 15–17, the greedy projection algorithm suggests variable ordering $y > x$. For example 18, where there are two branch cut formulae, each with two possible variable orderings, the greedy projection algorithm chooses variable ordering $y > x$ for 18(a) over the three other possibilities. Brown's heuristic, on the other hand, gives a tie between all examples except example 18 where it agrees with the greedy projection algorithm and chooses variable ordering $y > x$ for 18(a).

There is more than one branch cut representation of the same abstract problem for examples 18 and 19, as a result of the de-nesting process (or more precisely its use of QE). On the basis of the Dolzmann *el al.* assertion [DSS04] that “sotd is well-correlated with the final number of cells in the CAD and other “cost” measures”, we suggest using $\text{sotd}_n(P_n)$ to determine a good branch cut formula

$$\text{sotd}_n(P_n) = \sum_{f\in P_n} \sigma(f),$$

where σ is as in (5.2.14), i.e. using the convention $\boldsymbol{e} = (e_1, \ldots, e_n)$,

$$\sigma\left(\sum_{\boldsymbol{e}\in E} a_{\boldsymbol{e}} x_1^{e_1} \ldots x_n^{e_n}\right) = \sum_{\boldsymbol{e}\in E} \sum_{i=1}^{n} e_i.$$

We note that P_n in terms of QEPCAD is different from the actual input polynomials. This is because QEPCAD simplifies the input by means of factorising, square-free relatively prime decomposition and eliminating inconsistent sets of polynomials. On one hand, factoring and square-free decomposition are good since, for example, x^{10} is no worse than x^2 and both will result in the same number of cells being constructed in the CAD, but on the other hand eliminating the inconsistent sets is bad for us since, unlike QEPCAD, we do have interest in them. The measure we are going to use is the $\text{sotd}_n(P_n)$ of the reduced polynomials because this is an input as far as QEPCAD is concerned and this is what QEPCAD is going to be manipulating. The criterion selects a good formula for example 18, but no suggestion for example 19. Nevertheless, both formulae of example 19 are equally good. The heuristic can be used in conjunction with the greedy projection algorithm to reduce the number of projections considered. That is, if we first use $\text{sotd}_n(P_n)$ to select a formula, say P_1, from, in the cases of examples 18 and 19, the two possible formulae[4], then we only need to perform two projections and compare $\text{sotd}_1(P_{1_1}, x)$ and $\text{sotd}_1(P_{1_1}, y)$. Whereas, without $\text{sotd}_n(P_n)$, we would have to do four projections in order to calculate $\text{sotd}_1(P_{1_1}, x)$, $\text{sotd}_1(P_{1_1}, y)$, $\text{sotd}_1(P_{2_1}, x)$ and $\text{sotd}_1(P_{2_1}, y)$.

[4] Other problems may have different numbers of branch cut representations.

8.2.2.2 Four-Dimensions (2C)

Tables 8.6, 8.7, and 8.8 show the number of cells constructed by QEPCAD and Maple's CAD for the examples containing two complex variables, resulting in four-dimensional problems. We deemed that 30 minutes or less is a "reasonable" time to perform the computation, and terminated the program thereafter. The dashes indicate that the program either returned failed by itself, or we terminated it.

Once again, the results demonstrate that when there is a symmetry between different variables, there is no difference in cell counts between different variable orderings. When the input polynomials do not exhibit a symmetry between different variables, the number of cells being constructed in the decomposition step depends heavily on the variable ordering, and a wrong choice of variable ordering may swell up the cell count enormously: a factor of 118 more cells are constructed by QEPCAD in example 27 when using the worst projection order (92829 cells produced with respect to variable order $x > v > u > y$ or $u > y > x > v$) as opposed to the best projection order (785 produced by the variable order $y > v > x > u$ or $v > y > u > x$).

It is interesting to see that in more complicated cases, QEPCAD is shown to have more variation than Maple's CAD. With good variable orderings, they are often in the same general area. However, in the case where the variable orderings are bad, QEPCAD is often significant worse than Maple's CAD, even when it constructs only the partial CADs in examples 26, 27 and 29, and hence, if anything, the cell counts in these examples are underestimated. Considering example 27, for example, Maple's CAD constructs 557 cells with the best order and 1869 cells with the worst order, whereas QEPCAD constructs 785 cells with the best order and 92829 with the worst order. Furthermore, there are 6 variable orderings where Maple's CAD produces 1782 cells, but with these variable orderings sometimes QEPCAD produces 2049, sometimes 92829, showing a great variation in QEPCAD.

Variable order	**Example 21**		**Example 22**		**Example 23**	
	QEPCAD	**Maple's CAD**	**QEPCAD**	**Maple's CAD**	**QEPCAD**	**Maple's CAD**
$x > y > u > v$	104	113	95	113	-	-
$x > y > v > u$	104	113	95	113	-	-
$x > u > y > v$	104	113	95	113	-	-
$x > u > v > y$	104	113	95	113	-	-
$x > v > y > u$	104	113	95	113	-	-
$x > v > u > y$	104	113	95	113	-	-
$y > x > u > v$	104	113	95	113	-	-
$y > x > v > u$	104	113	95	113	-	-
$y > u > x > v$	104	113	95	113	-	-
$y > u > v > x$	104	113	95	113	-	-
$y > v > u > x$	104	113	95	113	-	-
$y > v > x > u$	104	113	95	113	-	-
$u > v > x > y$	104	113	95	113	-	-
$u > v > y > x$	104	113	95	113	-	-
$u > x > v > y$	104	113	95	113	-	-
$u > x > y > v$	104	113	95	113	-	-
$u > y > v > x$	104	113	95	113	-	-
$u > y > x > v$	104	113	95	113	-	-
$v > u > y > x$	104	113	95	113	-	-
$v > u > x > y$	104	113	95	113	-	-
$v > y > u > x$	104	113	95	113	-	-
$v > y > x > u$	104	113	95	113	-	-
$v > x > u > y$	104	113	95	113	-	-
$v > x > y > u$	104	113	95	113	-	-

Table 8.6: Number of constructed cells for examples 21-23

Variable order	**Example 24**		**Example 25**		**Example 26**	
	QEPCAD	**Maple's CAD**	**QEPCAD**	**Maple's CAD**	**QEPCAD**	**Maple's CAD**
$x > y > u > v$	-	-	1379	1437	2758	2961
$x > y > v > u$	-	-	1379	1437	7926	8277
$x > u > y > v$	-	-	5371	1085	14798	2929
$x > u > v > y$	-	-	3735	1121	12862	3013
$x > v > y > u$	-	-	1039	1085	18094	9077
$x > v > u > y$	-	-	1039	1121	23862	7509
$y > x > u > v$	-	-	507	549	1014	1161
$y > x > v > u$	-	-	507	549	2790	3021
$y > u > x > v$	-	-	591	253	5494	1217
$y > u > v > x$	-	-	211	253	1606	1217
$y > v > u > x$	-	-	211	253	1366	1893
$y > v > x > u$	-	-	211	253	2254	1513
$u > v > x > y$	-	-	801	389	2937	3021
$u > v > y > x$	-	-	441	253	1077	1161
$u > x > v > y$	-	-	2997	389	2513	1909
$u > x > y > v$	-	-	4053	389	1373	1529
$u > y > v > x$	-	-	441	253	6061	1209
$u > y > x > v$	-	-	1221	253	1781	1217
$v > u > y > x$	-	-	217	253	2845	2961
$v > u > x > y$	-	-	353	389	8129	8277
$v > y > u > x$	-	-	217	253	15697	3069
$v > y > x > u$	-	-	217	253	13897	3033
$v > x > u > y$	-	-	353	389	19125	9213
$v > x > y > u$	-	-	353	389	24465	7605

Table 8.7: Number of constructed cells for examples 24-26

Variable order	Example 27		Example 28		Example 29	
	QEPCAD	Maple's CAD	QEPCAD	Maple's CAD	QEPCAD	Maple's CAD
$x > y > u > v$	37957	989	-	-	4162	4205
$x > y > v > u$	9101	989	-	-	4162	4205
$x > u > y > v$	5985	557	-	-	3026	3069
$x > u > v > y$	6597	673	-	-	3026	3069
$x > v > y > u$	28821	1781	-	-	2878	2949
$x > v > u > y$	92829	1781	-	-	2878	2949
$y > x > u > v$	2049	989	-	-	4136	4185
$y > x > v > u$	2049	989	-	-	4136	4169
$y > u > x > v$	2049	1869	-	-	4576	4625
$y > u > v > x$	2049	1781	-	-	4576	4609
$y > v > u > x$	901	557	-	-	4008	4081
$y > v > x > u$	785	673	-	-	4008	4073
$u > v > x > y$	37957	989	- (B)	- (B)	3298	3341
$u > v > y > x$	9101	989	- (B)	- (B)	3298	3341
$u > x > v > y$	5985	557	-	-	2786	2829
$u > x > y > v$	6597	673	-	-	2786	2821
$u > y > v > x$	28821	1781	-	-	8650	7445
$u > y > x > v$	92829	1781	-	-	8650	7437
$v > u > y > x$	2049	989	51763 (G), (B)	2065 (G), (B)	642	669
$v > u > x > y$	2049	989	- (B)	6129 (B)	462	669
$v > y > u > x$	785	673	86809	-	776	817
$v > y > x > u$	901	557	-	-	776	817
$v > x > u > y$	2049	1869	-	-	1026	1053
$v > x > y > u$	2049	1781	-	-	1026	1053

Table 8.8: Number of constructed cells for examples 27-29

We now consider examples 25–27 and 29, where there are obvious variations in cell counts between different variable orderings. We will also discuss the use of the greedy projection algorithm, Brown's heuristic, greedy/Brown's heuristic and Brown's/greedy heuristic to determine a good variable ordering for these examples and example 28, where we could only complete a CAD construction for two out of 24 variable orderings by both QEPCAD and Maple's CAD. The suggestions made by the greedy projection algorithm and Brown's heuristic are marked with (G) and (B) in Tables 8.8–8.12 respectively. For brevity, the suggestions made by greedy/Brown's and Brown's/greedy heuristics are not included in these tables.

As we mentioned in Section 5.2.6.1, the greedy projection algorithm leaves an open question as to what one should do when there is a tie in $\mathrm{sotd}_k(P_k, (x_{i_1} \ldots, x_{i_k}))$ with respect to different variables. With respect to this, we chose to carry all tied variables forward as the possible candidates. For example, in example 28, after the first projection, the projection factor sets with respect to variables u and v have a joint lowest $\mathrm{sotd}_3(P_3, x_i)$ value. Both variables u and v are kept as possible candidates. There are now 6 possible candidates (instead of 3 if there is no tie) to consider. This time projection factor set with respect to variable order $v > u$ has the lowest $\mathrm{sotd}_2(P_2, (x_i, x_j))$. Therefore, we can now discard variable u from our possible first variable list, and we are left with only two remaining possibilities to consider. The obvious drawback is that we have to compute more projection steps. Using this heuristic, we are able to derive a single variable ordering, i.e. $v > u > y > x$, which is one of only two variable orderings with which QEPCAD and Maple's CAD can manage to finish building a CAD. The same suggestion is also given by greedy/Brown's and Brown's/greedy heuristics.

For example 28, Brown's heuristic, as predicted (see Section 5.2.6.3), cannot distinguish between the coupled variables. The only advice it is able to give is that variables u and v should be ordered before variables x and y.

Rank according to QEPCAD			**Rank according to Maple's CAD**		
Variable order	**QEPCAD**	**Maple's CAD**	**Variable order**	**QEPCAD**	**Maple's CAD**
$y > v > u > x$	211	253	$v > u > y > x$ (B)	217	253
$y > u > v > x$	211	253	$u > v > y > x$ (B)	441	253
$y > v > x > u$	211	253	$v > y > u > x$	217	253
$v > u > y > x$ (B)	217	253	$y > v > u > x$	211	253
$v > y > u > x$	217	253	$u > y > v > x$	441	253
$v > y > x > u$ (G)	217	253	$y > u > v > x$	211	253
$v > u > x > y$ (B)	353	389	$y > v > x > u$	211	253
$v > x > u > y$	353	389	$v > y > x > u$ (G)	217	253
$v > x > y > u$ (G)	353	389	$y > u > x > v$	591	253
$u > v > y > x$ (B)	441	253	$u > y > x > v$	1221	253
$u > y > v > x$	441	253	$v > u > x > y$ (B)	353	389
$y > x > v > u$	507	549	$u > v > x > y$ (B)	801	389
$y > x > u > v$	507	549	$v > x > u > y$	353	389
$y > u > x > v$	591	253	$u > x > v > y$	2997	389
$u > v > x > y$ (B)	801	389	$v > x > y > u$ (G)	353	389
$x > v > u > y$	1039	1121	$u > x > y > v$	4053	389
$x > v > y > u$	1039	1085	$y > x > v > u$	507	549
$u > y > x > v$	1221	253	$y > x > u > v$	507	549
$x > y > v > u$	1379	1437	$x > v > y > u$	1039	1085
$x > y > u > v$	1379	1437	$x > u > y > v$	5371	1085
$u > x > v > y$	2997	389	$x > v > u > y$	1039	1121
$x > u > v > y$	3735	1121	$x > u > v > y$	3735	1121
$u > x > y > v$	4053	389	$x > y > v > u$	1379	1437
$x > u > y > v$	5371	1085	$x > y > u > v$	1379	1437

Table 8.9: Ordered number of constructed cells for example 25

In example 25, it appears that the number of cells produced by Maple's CAD depends most on the position of the variable x in the variable ordering. This is possibly because x appears in more terms and to higher degree in the input polynomials than variables u and v, therefore it ought to be ordered last. However, it is unclear why the same thing cannot be said regarding variable y, even though it appears in the same number of terms and to the same degree as variable x. In fact 4 out of 6 possible orders which ordered y first are among those which produced the lowest number of cells. Furthermore the relationship between the number of cells constructed and the variable order reflects the symmetry between variables u and v.

Less pattern emerges from example 25 when constructing the CAD using QEPCAD. The only observation we make is that all 6 possible variable orders which project variable v first have low cell counts, even though not the lowest, and 4 out of 6 possible orderings which project v last are among the worst variable orders.

The greedy projection algorithm performs well here, recommending variable orderings $v > y > x > u$ and $v > x > y > u$, which are the best and second best orderings respectively for both QEPCAD and Maple's CAD. Brown's heuristic suggests ordering the coupled variables u and v before the coupled variables x and y, resulting in four possible orders. The prediction is on the whole a good one, particularly for Maple's CAD, where the suggestions are among the best and second best variable orders.

The number of required projection steps can be further reduced with our combined heuristics. Both heuristics choose the same orders, $v > u > y > x$ and $v > u > x > y$. While these orders differ from what the greedy projection algorithm picks, they are equally good choices.

Rank according to QEPCAD			Rank according to Maple's CAD		
Variable order	**QEPCAD**	**Maple's CAD**	**Variable order**	**QEPCAD**	**Maple's CAD**
$y > x > u > v$	1014	1161	$u > v > y > x$	1077	1161
$u > v > y > x$	1077	1161	$y > x > u > v$	1014	1161
$y > v > u > x$ (G)	1366	1893	$u > y > v > x$	6061	1209
$u > x > y > v$ (G)	1373	1529	$y > u > v > x$ (G)	1606	1217
$y > u > v > x$ (G)	1606	1217	$y > u > x > v$	5494	1217
$u > y > x > v$ (G)	1781	1217	$u > y > x > v$ (G)	1781	1217
$y > v > x > u$	2254	1513	$y > v > x > u$	2254	1513
$u > x > v > y$	2513	1909	$u > x > y > v$ (G)	1373	1529
$x > y > u > v$	2758	2961	$y > v > u > x$ (G)	1366	1893
$y > x > v > u$	2790	3021	$u > x > v > y$	2513	1909
$v > u > y > x$	2845	2961	$x > u > y > v$	14798	2929
$u > v > x > y$	2937	3021	$v > u > y > x$	2845	2961
$y > u > x > v$	5494	1217	$x > y > u > v$	2758	2961
$u > y > v > x$	6061	1209	$x > u > v > y$	12862	3013
$x > y > v > u$	7926	8277	$u > v > x > y$	2937	3021
$v > u > x > y$	8129	8277	$y > x > v > u$	2790	3021
$x > u > v > y$	12862	3013	$v > y > x > u$	13897	3033
$v > y > x > u$	13897	3033	$v > y > u > x$	15697	3069
$x > u > y > v$	14798	2929	$x > v > u > y$	23862	7509
$v > y > u > x$	15697	3069	$v > x > y > u$	24465	7605
$x > v > y > u$	18094	9077	$v > u > x > y$	8129	8277
$v > x > u > y$	19125	9213	$x > y > v > u$	7926	8277
$x > v > u > y$	23862	7509	$x > v > y > u$	18094	9077
$v > x > y > u$	24465	7605	$v > x > u > y$	19125	9213

Table 8.10: Ordered number of constructed cells for example 26

For example 26, despite the fact that all four variables appear in exactly the same number of terms and to the same degree in the input polynomials, the order in which they are ordered has a huge impact on the number of cells being constructed by both QEPCAD and Maple's CAD, and with no obvious pattern. The only observation we make is that variable orderings with the two least cell counts are those which ordered the pair of coupled variables sequentially.

Nothing can be said here regarding Brown's heuristic since all four variables resulted in a tie. There is also a tie between all four $\mathrm{sotd}_3(P_3, x_i)$ after the first projection step. Only after the second projection step can we narrow the choice down to four groups, $y > u > \ldots$, $y > v > \ldots$, $u > x > \ldots$ and $u > y > \ldots$, and arrive at four suggestions after completing the third projection. These variable orderings clearly are not the best, but they are certainly not anything like the worst. At least, they are only a factor of 1.1–1.7 wrong as opposed to being a factor of 24 (in term of QEPCAD) or 8 (in term of Maple) wrong.

A tie between all four variables when determining the first variable in the ordering either by the greedy projection algorithm or Brown's heuristic means neither greedy/Brown's nor Brown's/greedy heuristics can help to reduce the number of projection steps.

Rank according to QEPCAD			Rank according to Maple's CAD		
Variable order	**QEPCAD**	**Maple's CAD**	**Variable order**	**QEPCAD**	**Maple's CAD**
$v > y > u > x$ (G)	785	673	$y > v > u > x$ (G)	901	557
$y > v > x > u$ (G)	785	673	$u > x > v > y$	5985	557
$y > v > u > x$ (G)	901	557	$v > y > x > u$ (G)	901	557
$v > y > x > u$ (G)	901	557	$x > u > y > v$	5985	557
$v > u > y > x$	2049	989	$v > y > u > x$ (G)	785	673
$y > u > v > x$	2049	1781	$x > u > v > y$	6597	673
$v > u > x > y$	2049	989	$y > v > x > u$ (G)	785	673
$v > x > u > y$	2049	1869	$u > x > y > v$	6597	673
$y > x > v > u$	2049	989	$v > u > y > x$	2049	989
$v > x > y > u$	2049	1781	$u > v > y > x$	9101	989
$y > x > u > v$	2049	989	$v > u > x > y$	2049	989
$y > u > x > v$	2049	1869	$u > v > x > y$	37957	989
$u > x > v > y$	5985	557	$y > x > v > u$	2049	989
$x > u > y > v$	5985	557	$x > y > v > u$	9101	989
$x > u > v > y$	6597	673	$x > y > u > v$	37957	989
$u > x > y > v$	6597	673	$y > x > u > v$	2049	989
$u > v > y > x$	9101	989	$u > y > v > x$	28821	1781
$x > y > v > u$	9101	989	$y > u > v > x$	2049	1781
$u > y > v > x$	28821	1781	$x > v > u > y$	92829	1781
$x > v > y > u$	28821	1781	$x > v > y > u$	28821	1781
$u > v > x > y$	37957	989	$v > x > y > u$	2049	1781
$x > y > u > v$	37957	989	$u > y > x > v$	92829	1781
$x > v > u > y$	92829	1781	$v > x > u > y$	2049	1869
$u > y > x > v$	92829	1781	$y > u > x > v$	2049	1869

Table 8.11: Ordered number of constructed cells for example 27

In example 27, we observed that the cell counts are the same if one swaps the complex variables z and w in the variable ordering. That is the pair (x, y) can be swapped with the pair (u, v) (i.e. both u for x and v for y) without affecting the number of cells being constructed by both systems.

The greedy projection algorithm, once again, gives a good suggestion here. Brown's heuristic, on another hand, does not offer any suggestion since all four variables resulted in a tie.

As in the previous example, a tie between all four variables when following Brown's heuristic means that it does not narrow down the greedy projection algorithm search space. Hence Brown's/greedy heuristic will require exactly the same computation as the greedy projection algorithm. However, unlike the last example, this time we can apply greedy/Brown's heuristic. Although Brown's heuristic does not favour one coupled variable over another, if we follow the criterion and choose the coupled variables as adjacent in the list, the choices made by greedy/Brown's heuristic, which are $y > x > u > v$, $y > x > v > u$, $v > u > x > y$ and $v > u > y > x$, disagree with the greedy projection algorithm's choices. The number of cells produced with these orders are more than those produced by the greedy projection algorithm's suggestions. Nevertheless, in term of QEPCAD, they only construct a CAD with a factor of 2.6 more than the best, rather than a factor of 118 more.

Rank according to QEPCAD			**Rank according to Maple's CAD**		
Variable order	**QEPCAD**	**Maple's CAD**	**Variable order**	**QEPCAD**	**Maple's CAD**
$v > u > x > y$ (G)	462	669	$v > u > x > y$ (G)	462	669
$v > u > y > x$ (G)	642	669	$v > u > y > x$ (G)	642	669
$v > y > u > x$ (G)	776	817	$v > y > u > x$ (G)	776	817
$v > y > x > u$ (G)	776	817	$v > y > x > u$ (G)	776	817
$v > x > u > y$	1026	1053	$v > x > u > y$	1026	1053
$v > x > y > u$	1026	1053	$v > x > y > u$	1026	1053
$u > x > v > y$ (B)	2786	2829	$u > x > y > v$	2786	2821
$u > x > y > v$	2786	2821	$u > x > v > y$ (B)	2786	2829
$x > v > y > u$	2878	2949	$x > v > y > u$	2878	2949
$x > v > u > y$	2878	2949	$x > v > u > y$	2878	2949
$x > u > y > v$	3026	3069	$x > u > y > v$	3026	3069
$x > u > v > y$ (B)	3026	3069	$x > u > v > y$ (B)	3026	3069
$u > v > x > y$	3298	3341	$u > v > x > y$	3298	3341
$u > v > y > x$	3298	3341	$u > v > y > x$	3298	3341
$y > v > u > x$	4008	4081	$y > v > x > u$	4008	4073
$y > v > x > u$	4008	4073	$y > v > u > x$	4008	4081
$y > x > u > v$	4136	4185	$y > x > v > u$	4136	4169
$y > x > v > u$	4136	4169	$y > x > u > v$	4136	4185
$x > y > u > v$	4162	4205	$x > y > u > v$	4162	4205
$x > y > v > u$	4162	4205	$x > y > v > u$	4162	4205
$y > u > x > v$	4576	4625	$y > u > v > x$	4576	4609
$y > u > v > x$	4576	4609	$y > u > x > v$	4576	4625
$u > y > v > x$	8650	7445	$u > y > x > v$	8650	7437
$u > y > x > v$	8650	7437	$u > y > v > x$	8650	7445

Table 8.12: Ordered number of constructed cells for example 29

QEPCAD and Maple's CAD agree on variable ordering in example 29. Although not producing exactly the same number of cells, they rank the variable orders in the same order. They also favour projecting variable v first for a non-obvious reason, in spite of the fact that variable v, together with variable y, has the highest degree (to power of 4) in the polynomials.

Example 29 is an example with nested square roots and QE is required to remove an extraneous variable introduced through the nested square root solving mechanism. Therefore, we hoped Brown's heuristic would perform better here since there is no tie between the coupled variables. It turns out that following Brown's heuristic still resulted in a tie between variables x and u as first variable in the ordering. It does, however, manage to order the last two variables. Hence we have two possible suggestions from Brown's heuristic. Brown's/greedy heuristic will break the tie here to give just a single suggestion, $x > u > v > y$. However, it turns out that both suggestions are only average.

The greedy projection algorithm, in contrast, disagrees with Brown's heuristic on the first variable to be ordered and goes for variable v, which is a better choice. A tie between variables u and y after projecting P_3 to P_2, and again between the remaining variables after projecting P_2 to P_1 means we end with 4 suggestions, which are the top three and top two variable orderings in terms of QEPCAD and Maple's CAD respectively. Greedy/Brown's heuristic narrows the suggestions down to two, $v > u > x > y$ and $v > u > y > x$, which are the top two and the top variable orderings in terms of QEPCAD and Maple's CAD respectively.

8.2.2.3 Pre-Conditioning

This section gives insight into the performance of the pre-conditioning method described in Chapter 6. For direct comparison, the number of cells produced with and without pre-conditioning of the set of branch cuts polynomials prior to handing the set to QEPCAD or Maple's CAD are listed in Tables 8.13 and 8.14. Note that for all our examples, `pprecond` and `sprecond` produce the same polynomials. Theoretically, when pre-conditioning the input polynomials either by means of pseudo-remainder or sparse pseudo-remainder, the method reduces the degree of the main variable in a polynomial. This is offset by an increase in the degree of another variable in the same polynomial. We observed, based on the above two-dimensional results, that pre-conditioning is often beneficial if the resulting polynomial, h where `h=pprecond(f,g,x)` or `h=sprecond(f,g,x)`, contains fewer variables or fewer terms than the initial polynomial, f. Conversely, if the pre-conditioning method produces a polynomial with many terms and containing high degree variables, then this is not the occasion to use the pre-conditioning method as in example 17.

Table 8.13: Number of QEPCAD's constructed cells for pre-conditioned examples

Example	**No elimination**		**Eliminating x**		**Eliminating y**	
	$y > x$	$x > y$	$y > x$	$x > y$	$y > x$	$x > y$
1	17	17	9	6†	0†	0†
2	17	17	9	6†	0†	0†
3	29	29	21	7†	6†	21
4	29	29	21	21	15†	21
5	22†	25	7†	14†	22†,‡	25‡
6	33	33	25	22†	22†	25
9	17	17	6†	6†	3†	9
10	25	25	17	14†	25‡	25‡
15	391	373	211	142†	131†	403
16	663	615	599	306†	580†	1307
17	1209	1627	1931	2050†	1367	2793

† QEPCAD only computed a partial decomposition, so the cell count is an underestimate.

‡ y-pre-conditioning does not have any effect on input polynomials.

It is evident that the cell count depends not only on the variable ordering but also on the formulation of the input problem. With pre-conditioning, we have, in addition to $n!$ variable orderings, various formulations in n variables which are mathematically the same geometric problem but different algebraically. Hence, apart from the greedy projection algorithm and Brown's heuristic, we can also consider the extended greedy projection heuristic. By extended, we refer to the idea suggested at the end of Section 8.2.2.1 on page 99, which uses $\mathrm{sotd}_n(P_n)$ as a formula predictor, followed by the greedy projection algorithm in the usual manner. Despite the fact that the three heuristics give different predictions, they all perform reasonably well: even though the suggestions are not always the best,

Table 8.14: Number of Maple's CAD constructed cells for pre-conditioned examples

Example	**No elimination**		**Eliminating x**		**Eliminating y**	
	$y > x$	$x > y$	$y > x$	$x > y$	$y > x$	$x > y$
1	17	17	9	9	9	9
2	17	17	9	9	9	9
3	29	29	21	21	21	21
4	29	29	21	21	21	21
5	25	25	17	17	25‡	25‡
6	33	33	25	25	25	25
9	17	17	9	9	9	9
10	25	25	17	17	25‡	25‡
15	391	373	211	157	137	403
16	663	615	599	323	587	1307
17	1209	1627	1931	2011	1375	2793

‡ y-pre-conditioning does not have any effect on input polynomials.

they are not in the range of the worst. Table 8.15 summarises the suggestions made by the three heuristics.

Table 8.15: Operation of heuristics for pre-conditioned examples

Example	**No elimination**		**Eliminating x**		**Eliminating y**	
	$y > x$	$x > y$	$y > x$	$x > y$	$y > x$	$x > y$
1				(B)	(G), (B), (EG)	(G), (EG)
2				(B)	(G), (B), (EG)	(G), (EG)
3				(B)	(B),	(G), (EG)
4				(B)	(B),	(G), (EG)
5			(EG)	(B), (EG)		
6			(EG)	(B), (EG)	(B), (EG)	(EG)
9			(EG)	(B), (EG)	(B), (EG)	(EG)
10			(EG)	(B), (EG)		
15			(EG)	(B)	(G)	
16	(G)		(EG)	(B)	(G)	
17	(G)		(EG)	(B)		

(G) The greedy projection algorithm

(B) Brown's heuristic

(EG) The extended greedy projection heuristic

Computing the greedy projection algorithm for examples 5, 6, 9 and 10 resulted in a tie between all 6 choices.

8.3 Summary

This chapter presents the results and the evaluations of the performance of the branch cuts computation and decomposition phases of the decomposition method. The experimental results demonstrated the efficiency of the branch cuts computation phase, except perhaps when QE is required. The effectiveness of the decomposition phase is still limited by the complexity of the CAD algorithm. However, we now have a better understanding of the feasibility of CAD in practice, and more importantly the capabilities of QEPCAD and Maple's CAD. We have made the following observations.

- A Cylindrical Algebraic Decomposition is not unique. There may be more than one CAD even for a fixed variable ordering and fixed polynomials.
- Input problem formulation is important when using any CAD algorithm.
- Variable ordering is crucial in any CAD algorithm.
- The greedy projection algorithm, generally, gives good suggestions for variable ordering. It should be pointed out that although the experiments indicate a correlation between the greedy projection algorithm's prediction and the number of cells constructed by Maple's CAD, the algorithm cannot be applied without performing the projection phase of CAD-PL.
- Despite the obvious drawback of ties of Brown's heuristic, our effort to use the idea as a way of reducing the size of search space by the greedy projection algorithm has been shown to be effective. Greedy/Brown's heuristic tends to pick orders which are optimal or close to optimal (up to a factor of 2.6 more cells in the same example), whereas the truly bad orders are *much* worse than these (up to a factor of 118 more cells).
- For a given problem, when the variable produces a small number of cells for one method, it tends to do so for the other method too, and the two are comparable. However, the worst case for QEPCAD tends to be significantly worse than the worst case for Maple's CAD.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

The simplification problem is not new in computer algebra. It was recognised as far back as 1971 [Mos71]. However, only hesitant attempts at the problem have appeared. One problem among its many facets is that many elementary functions are essentially multi-valued functions. Branch cuts are a well-known device introduced to define segments on which these functions become discontinuous.

The primary concern of this thesis has been to develop a tool to determine the truth (or falsity) of the proposed identity based on analysing, using the technique of Cylindrical Algebraic Decomposition (CAD), the geometry of $\mathbf{C}$ (or $\mathbf{C}^n$) induced by branch cuts. Our algorithm took the following approach:

1. verifying that the proposed simplification is correct as a simplification of multi-valued functions, using a standard simplifier such as Maple's `simplify`$(\ldots,$`symbolic`$)$ command;

2. decomposing $\mathbf{C}$ (or $\mathbf{C}^n$), viewed as $\mathbf{R}^2$ (or $\mathbf{R}^{2n}$), with respect to the branch cuts of the relevant functions, using the technique of CAD, and finding a sample point in each region in $\mathbf{R}^2$ (or $\mathbf{R}^{2n}$) defined by the branch cuts;

3. evaluating the identity on each connected component using the obtained

sample point, thereby concluding whether the identity is true or false on that entire region by the Monodromy theorem.

In this thesis, we looked at the last two stages of the system, paying particular attention to phase 2, which turned out to be more difficult than we had expected. Most of the work, in this thesis was done in the context of QEPCAD, since for majority of the time we had to rely solely on it: CAD via triangular decomposition did not exist until 2009, and its implementation was integrated into Maple version 14 a year later.

We showed that a problem in $\mathbf{C}$ (or $\mathbf{C}^n$) can be treated as a problem in $\mathbf{R}^2$ (or $\mathbf{R}^{2n}$) and how branch cuts of a function can be expressed in terms of real semi-algebraic sets (Chapter 4), so that they are within the remit of the CAD algorithm (Chapter 5), which forms a major part of our system. It is evident that the CAD construction is a significant obstacle. Using any existing CAD implementations as a black-box is problematic, even for some seemingly small problems. QEPCAD, as its name suggests, is built based on Partial CAD for QE applications, and has limitations when applied to our problems (see Section 5.2.4). Maple's CAD, on another hand, starts with a weaker branch cut formulation, hence loses some useful information. Nevertheless, examples in Section 8.2 showed that it is still very competitive with QEPCAD, sometimes even producing smaller CADs.

The pre-conditioning method was an attempt to allow more boolean connective information to be taken into consideration when building CAD. It was intended to address Maple's CAD weakness, but subsequently found to be equally useful under the QEPCAD configuration, see Table 8.13.

We have seen that the CAD construction is sensitive to variable ordering, both in elimination and in projection (QEPCAD) and triangularization (Maple's CAD), and the interaction is significant and subtle. We considered the greedy projection algorithm in Section 5.2.6.1 and Brown's heuristics in Section 5.2.6.3. These ideas prompted us to investigate further since we are employing the CAD under special circumstances, i.e. complex variables are naturally paired together (real and imaginary parts). Using this information led us to refinements of the greedy projection algorithm and Brown's heuristic (Section 5.2.6.4). The empirical results are promising, see Section 8.2.2. Nevertheless the problem is not totally solved.

The last phase of the system relies on numerical or symbolical evaluations of sample points. When the branch cuts have full-dimension, there is no problem. But problem may occur when sampling on non-full-dimension cuts (a line in complex plane or a point in real plane), since even the slightest error may drift us off the branch cut into an adjacent cell instead, thus probably, and indeed certainly in the case where the identity is true except on these lower-dimensional components. An example of this phenomenon is $\overline{\log(z+i\sqrt{2})} = \log(\overline{z}-i\sqrt{2})$, which is false for z of the form $x - i\sqrt{2}$, but true for all floating-point z. Section 7.2 made evaluating on such cuts possible with the notion of adherence. It is worth noting that an alternative method, using power series, exists, see [BBD03]. The last phase of the system was also made more efficient with the method of post-conditioning the CAD. However, we only explored that in two-dimensional cases.

9.2 Future Work

- **Branch cut formulation**
  Branch cut computation has been made practical, except perhaps when QE is required. In order to maximize the efficiency of this step, there is a need for a finer heuristic to remove the nested roots.

  We have demonstrated that branch cut formula may not be unique, and different phrasing of the input may affect the efficiency of the decomposition step, prompting a call for a heuristic to pick the “best” representative.

- **CAD**
  There are still limitations in the use of CAD in the decomposition phase of the decomposition method, since the CAD algorithm is inherently doubly-exponential in the number of variables.

  Customized CAD algorithm

  A possible way forward is to develop a CAD algorithm for our particular application. In this regard, it is worth remembering that, if we restrict ourselves to problems involving complex variables, our semi-algebraic representation for the set of branch cuts of an identity in complex variables is of the form:

$$\{B_1 \vee \ldots \vee B_n\},$$

for some n where

$$B_i = \{p_{i1} = 0 \wedge \; p_{i2} \; \sigma_i \; c_i\},$$

with $\sigma_i \in \{<, \leq, >, \geq\}$, and $c_i \in \{0, \pm 1\}$.

There are a number of aspects which can be considered here.

1. The CAD-PL algorithm in the sense of [Col75] does not take advantage of the logical structure of the formula during the construction of a CAD. The distinction between equalities and inequalities only matters in the solution phase of a QE problem. Likewise, the CAD-TD which also aimed to construct a CAD in the sense of [Col75], also does not make use of the logical structure information in the input formula. While PCAD utilised this useful information, it is geared towards QE problems and its use in our application is still questionable (Section 5.2.4). The *pre-conditioning* method presented in this thesis (Chapter 6) successfully utilises the information in the input formula to some extent. However, it is still limited to some special cases.

2. The method for computing the set of branch cuts as described in Section 4.3.1 converts a system of polynomials in $\mathbf{C}$ (or $\mathbf{C}^n$) to a system of polynomials in $\mathbf{R}^2$ (or $\mathbf{R}^{2n}$) by taking real and imaginary parts of variables in the polynomials. This makes our application of CAD somewhat specialised, as coupled variables are dependent and may exhibit geometric properties.

   Lemma 9.2.1. *Let $f(z) = u(x, y) + iv(x, y)$ where $z = x + iy$. Then u and v meet orthogonally at point p if p is a non-singular point.*

Proof. From the Cauchy-Riemann equations, we have a pair of equations:

$$u_x = v_y \quad \text{and} \quad u_y = -v_x.$$

Hence,

$$\begin{pmatrix} u_x \\ u_y \end{pmatrix} = M \begin{pmatrix} v_x \\ v_y \end{pmatrix}$$

where

$$M = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix} \qquad \text{and} \qquad \theta = \frac{\pi}{2},$$

i.e. the 2-d rotation matrix. □

3. The presence of equational constraints $p_{i1} = 0$ in our branch cuts expressions. It would be well-worth investigating how the methods of [Col98, McC99, McC01, BM05] can be adapted to our kind of problem.

Improvement to CAD via triangular decomposition

The algorithm of CAD-TD relies heavily on triangular decomposition [ALMM99]. [CDM$^+$10] introduced a *lazy triangular decomposition*, for which, under suitable assumptions, the decomposition can be computed in singly-exponential time over the complexes. Exploring the possibility of employing this within CAD-TD would be a useful piece of future research. Furthermore, Maple 15 provides a new `SamplePoints` command which returns a sample point for each connected component of the semi-algebraic system. While it is relevant, as is QEPCAD, it is not a direct answer to our problem as it is also a partial CAD. In addition, it does not provide any adjacency information, which is required if we are to use adherence (Chapter 7).

Variable order

The choice of variable order is very important in practice. Experiments show that the greedy projection algorithm and greedy/Brown's heuristic work reasonably well. Nevertheless, these heuristics are based on CAD-PL, and an efficient and effective way to determine a good variable ordering in CAD-TD term is desirable. Another interesting observation for future investigation is that greedy/Brown's heuristic positions the coupled variables beside each other in the variable ordering and the suggestions are shown to generally be good ones despite the fact that the experimental results do not favour ordering coupled variables in this way.

- **Identity evaluation**

  Computing adherence of a cut can be hard in more complex, or high dimension problems, or when pre-conditioning is used, since we have to track how the branch cuts are transformed as we are building up the formulae. Nevertheless, we are hopeful that with better understanding, the adherence method, and the post-conditioning method, can be generalised to handle more identities one is likely to meet in practice.

- **Extension**

  Our verification system, or more precisely its use of CAD, works for the subclass of the elementary functions that have algebraic branch cuts. Undoubtedly, the cases where the branch cuts are transcendental are equally important. A typical example would be $\log(\exp(z)) \overset{?}{=} z$; the branch cut for $\log(z)$ is the negative real axis, hence the branch cut for $\log(\exp(z))$ is $\{(x,y)|y = (2k+1)\pi i,\ k \in \mathbf{Z}\}$, which is not algebraic. A possible extension to our system is to include a class of Pfaffian functions, where one may be able to use the method of [GV01], although the implementation is rarer and the feasibility of the approach is still unclear.

  Furthermore, many readily encountered identities such as those found in [AS64], involve non-elementary functions. To include non-elementary functions would be a major extension of this work.

Bibliography

[ACM84a] D.S. Arnon, G.E. Collins, and S. McCallum. Cylindrical Algebraic Decomposition I: The Basic Algorithm. *SIAM J. Comp.*, 13:865–877, 1984.

[ACM84b] D.S Arnon, G.E. Collins, and S. McCallum. Cylindrical Algebraic Decomposition II: An Adjacency Algorithm for the Plane. *SIAM J. Comp.*, 13:878–889, 1984.

[ACM85] D.S. Arnon, G.E. Collins, and S. McCallum. An Adjacency Algorithm for Cylindrical Algebraic Decomposition of Three-Dimensional Space. In *Proceedings EUROCAL 85*, pages 246–261, 1985.

[ACM88] D.S. Arnon, G.E. Collins, and S. McCallum. An Adjacency Algorithm for Cylindrical Algebraic Decompositions of Three-Dimensional Space. *J. Symbolic Comp.*, 5:163–187, 1988.

[ALMM99] P. Aubry, D. Lazard, and M. Moreno Maza. On the Theories of Triangular Sets. *J. Symbolic Comp.*, 28:105–124, 1999.

[Arn85] D.S. Arnon. A Cluster-Based Cylindrical Algebraic Decomposition Algorithm. In *Proceedings EUROCAL 85*, pages 262–269, 1985.

[Arn88] D.S. Arnon. A Cluster-Based Cylindrical Algebraic Decomposition Algorithm. *J. Symbolic Comp.*, 5:189–211, 1988.

[AS60] L.V. Ahlfors and L. Sario. *Riemann Surfaces.* Princeton University Press, 1960.

[AS64] M. Abramowitz and I. Stegun. Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, 9th printing. *US Government Printing Office*, 1964.

[Asl96] H. Aslaksen. Multiple-valued complex functions and computer algebra. *SIGSAM Bulletin*, 30(2):12–20, 1996.

[BBD03] J.C. Beaumont, R.J. Bradford, and J.H. Davenport. Better Simplification of Elementary Functions Through Power Series. In J.R. Sendra, editor, *Proceedings ISSAC 2003*, pages 30–36, 2003.

[BBDP04] J.C. Beaumont, R.J. Bradford, J.H. Davenport, and N. Phisanbut. A Poly-Algorithmic Approach to Simplifying Elementary Functions. In J. Gutierrez, editor, *Proceedings ISSAC 2004*, pages 27–34, 2004.

[BBDP05] J.C. Beaumont, R.J. Bradford, J.H. Davenport, and N. Phisanbut. Adherence is Better Than Adjacency. In M. Kauers, editor, *Proceedings ISSAC 2005*, pages 37–44, 2005.

[BBDP07] J.C. Beaumont, R.J. Bradford, J.H. Davenport, and N. Phisanbut. Testing Elementary Function Identities Using CAD. *AAECC*, 18:513–543, 2007.

[BBP05] J.C. Beaumont, R.J. Bradford, and N. Phisanbut. Practical Simplification of Elementary Functions Using CAD. In *Proceedings A3L*, pages 35–40, 2005.

[BCD$^+$02] R.J. Bradford, R.M. Corless, J.H. Davenport, D.J. Jeffrey, and S.M. Watt. Reasoning about the Elementary Functions of Complex Analysis. *Annals of Mathematics and Artificial Intelligence*, 36:303–318, 2002.

[BCR98] J. Bochnak, M. Coste, and M.-F. Roy. Real algebraic geometry. *Ergebnisse der Mathematik 38 (translated from the French and revised by the authors)*, 1998.

[BD02] R.J. Bradford and J.H. Davenport. Towards Better Simplification of Elementary Functions. In T. Mora, editor, *Proceedings ISSAC 2002*, pages 15–22, 2002.

[BD07] C.W. Brown and J.H. Davenport. The Complexity of Quantifier Elimination and Cylindrical Algebraic Decomposition. In C.W. Brown, editor, *Proceedings ISSAC 2007*, pages 54–60, 2007.

[BM05] C.W. Brown and S. McCallum. On using bi-equational constraints in CAD construction. In *Proceedings ISSAC 2005*, pages 76–83, 2005.

[Bou70] N. Bourbaki. Théorie des Ensembles. *Diffusion C.C.L.S.*, 1970.

[BR90] R. Benedetti and J.-J. Risler. Real Algebraic and Semi-Algebraic Sets. *Hermann*, 1990.

[Bra93] R.J. Bradford. Algebraic Simplification of Multiple-Valued Functions. In *Proceedings DISCO '92*, pages 13–21, 1993.

[Bro] C.W. Brown. Qepcad — quantifier elimination by partial cylindrical algebraic decomposition. `http://www.cs.usna.edu/~qepcad/B/QEPCAD.html`.

[Bro98] Brown.C.W. Simplification of truth-invariant cylindrical algebraic decomposition. In O. Gloor, editor, *Proceedings ISSAC '98*, pages 295–301, 1998.

[Bro01a] C.W. Brown. Improved Projection for Cylindrical Algebraic Decomposition. *J. Symbolic Comp.*, 32:447–465, 2001.

[Bro01b] C.W. Brown. Simple CAD construction and its applications. *J. Symbolic Comp.*, 31:521–547, 2001.

[Bro03] C.W. Brown. QEPCAD B: a program for computing with semi-algebraic sets using CADs. *ACM SIGSAM Bulletin 4*, 37:97–108, 2003.

[Bro04] C.W. Brown. Companion to the Tutorial Cylindrical Algebraic Decomposition Presented at ISSAC 2004. `http://www.cs.usna.edu/~wcbrown/research/ISSAC04/handout.pdf`, 2004. The handout for ISSAC 2004 Tutorial: Cylindrical Algebraic Decomposition.

[BS10] C.W. Brown and A. Strzeboński. Black-Box/White-Box Simplification and Applications to Quantifier Elimination. In S.M. Watt, editor, *Proceedings ISSAC 2010*, pages 69–76, 2010.

[Car04] J. Carette. Understanding Expression Simplification. In J. Gutierrez, editor, *Proceedings ISSAC 2004*, pages 72–79, 2004.

[CDJ$^+$01] R.M. Corless, J.H. Davenport, D.J. Jeffrey, G. Litt, and S.M. Watt. Reasoning about the Elementary Functions of Complex Analysis. In John A. Campbell and Eugenio Roanes-Lozano, editors, *Proceedings Artificial Intelligence and Symbolic Computation*, pages 115–126, 2001.

[CDJW00] R.M. Corless, J.H. Davenport, D.J. Jeffrey, and S.M. Watt. "According to Abramowitz and Stegun or arccoth needn't be uncouth. *SIGSAM Bulletin 2*, 34:58–65, 2000.

[CDKS11] F. Chyzak, J.H. Davenport, C. Koutschan, and B. Salvy. On Kahan's Rules for Determining Branch Cuts. `http://arxiv.org/abs/1109.2809`, 2011.

[CDM$^+$10] C. Chen, J.H. Davenport, J.P. May, M. Moreno Maza, B. Xia, and R. Xiao. Triangular Decomposition of Semi-algebraic Systems. In S.M. Watt, editor, *Proceedings ISSAC 2010*, pages 187–194, 2010.

[CGL$^+$07] C. Chen, O. Golubitsky, F. Lemaire, M. Moreno Maza, and W. Pan. Comprehensive Triangular Decomposition. In *Proceedings Computer Algebra and Symbolic Computing 2007*, pages 73–101, 2007.

[CH91] G.E. Collins and H. Hong. Partial Cylindrical Algebraic Decomposition for Quantifier Elimination. *J. Symbolic Comp.*, 12:299–328, 1991.

[CJ96] R.M. Corless and D.J. Jeffrey. The Unwinding Number. *SIGSAM Bulletin 2*, 30:28–35, 1996.

[CJ98] R.M. Corless and D.J. Jeffrey. Graphing Elementary Riemann Surfaces. *SIGSAM Bulletin*, 32:11–18, 1998.

[CMMXY09] C. Chen, M. Moreno Maza, B. Xia, and L. Yang. Computing cylindrical algebraic decomposition via triangular decomposition. In J. May, editor, *Proceedings ISSAC 2009*, pages 95–102, 2009.

[Coh69] P.J. Cohen. Decision Procedures for real and p-adic fields. *Comm. Pure Appl. Math.*, 22:131–151, 1969.

[Col75] G.E. Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In *Proceedings 2nd. GI Conference Automata Theory & Formal Languages*, pages 134–183, 1975.

[Col98] G.E. Collins. Quantifier elimination by cylindrical algebraic decomposition — twenty years of progess. *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 8–23, 1998.

[Dav03] J.H. Davenport. The geometry of $\mathbb{C}^n$ is important for the algebra of elementary functions. *Algebra Geometry and software systems*, pages 207–224, 2003.

[Dav10] J.H. Davenport. The Challenges of Multivalued "Functions". In S. Autexier *et al.*, editor, *Proceedings AISC/Calculemus/MKM 2010*, pages 1–12, 2010.

[DF94] A. Dingle and R.J. Fateman. Branch cuts in computer algebra. In *Proceedings ISSAC 1994*, pages 250–257, 1994.

[DH88] J.H. Davenport and J. Heintz. Real Quantifier Elimination is Doubly Exponential. *J. Symbolic Comp.*, 5:29–35, 1988.

[DSS04] A. Dolzmann, A. Seidl, and Th. Sturm. Efficient Projection Orders for CAD. In J. Gutierrez, editor, *Proceedings ISSAC 2004*, pages 111–118, 2004.

[EGT10] I.Z. Emiris, A. Galligo, and E.P. Tsigaridas. Random Polynomials and Expected Complexity of Bisection Methods for Real Solving. In S.M. Watt, editor, *Proceedings ISSAC 2010*, pages 235–242, 2010.

[GV01] A. Gabrielov and N. Vorobjov. Complexity of cylindrical decompositions of sub-Pfaffian sets. *J. Pure Appl. Algebra*, 164:179–197, 2001.

[Hon90] H. Hong. An Improvement of the Projection Operator in Cylindrical Algebraic Decomposition. In S. Watanabe and M. Nagata, editors, *Proceedings ISSAC '90*, pages 261–264, 1990.

[HRS90] J. Heintz, M.-F. Roy, and P. Solernò. Sur la Complexité du Principe de Tarski-Seidenberg. *Bull. Soc. Math. France*, 118:101–126, 1990.

[IEE85] IEEE. IEEE Standard 754 for Binary Floating-Point Arithmetic. *IEEE*, 1985.

[Kah87] W. Kahan. Branch Cuts for Complex Elementary Functions or Much Ado About Nothing's Sign Bit. *The State of Art in Numerical Analysis*, pages 165–211, 1987.

[KS11] M. Kerber and M. Sagraloff. Root Refinement for Real Polynomials. `http://arxiv.org/abs/1104.1362`, 2011.

[Laz94] D. Lazard. An Improved Projection Operator for Cylindrical Algebraic Decomposition. In C. L. Bajaj, editor, *Algebraic Geometry and its Applications. Collections of Papers from Abhyankar's 60th Birthday Conference*, pages 467–476. Springer, 1994.

[Lit99] G. Litt. Unwinding numbers for the Logarithmic, Inverse Trigonometric and Inverse Hyperbolic Functions. M.sc. project, Department of Applied Mathematics, University of Western Ontario, 1999.

[Mar65] A.I. Markushevich. *Theory of Functions of a Complex Variable Volume I.* Prentice–Hall, Inc., 1965.

[Mar67] A.I. Markushevich. *Theory of Functions of a Complex Variable Volume III.* Prentice–Hall, Inc., 1967.

[MC02] S. McCallum and G.E. Collins. Local Box Adjacency Algorithms for Cylindrical Algebraic Decompositions. *J. Symbolic Comp.*, 33:321–342, 2002.

[McC84] S. McCallum. *An Improved Projection Operation for Cylindrical Algebraic Decomposition.* PhD thesis, University of Wisconsin-Madison, 1984.

[McC85] S. McCallum. An Improved Projection Operation for Cylindrical Algebraic Decomposition. In *Proceedings EUROCAL 85*, pages 277–278, 1985.

[McC88] S. McCallum. An Improved Projection Operation for Cylindrical Algebraic Decomposition of Three-dimensional Space. *J. Symbolic Comp.*, 5:141–161, 1988.

[McC98] S. McCallum. An Improved Projection Operation for Cylindrical Algebraic Decomposition. *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 242–268, 1998.

[McC99] S. McCallum. On Projection in CAD-Based Quantifier Elimination with Equational Constraints. In S. Dooley, editor, *Proceedings ISSAC '99*, pages 145–149, 1999.

[McC01] S. McCallum. On Propagation of Equational Constraints in CAD-Based Quantifier Elimination. In B. Mourrain, editor, *Proceedings ISSAC 2001*, pages 223–230, 2001.

[McC10] S. McCallum. Lazard Projection. *Personal Communication to James H. Davenport*, 2010.

[Mos71] J. Moses. Algebraic Simplification — A Guide for the Perplexed. *Comm. ACM*, 14:527–537, 1971.

[MS11] K. Mehlhorn and M. Sagraloff. A deterministic algorithm for isolating real roots of a real polynomial. *J. Symbolic Comp.*, 46:70–90, 2011.

[Pat96] C.M. Patton. A Representation of Branch-Cut Information. *SIGSAM Bulletin 2*, 30:21–24, 1996.

[PBD10] N. Phisanbut, R.J. Bradford, and J.H. Davenport. Geometry of Branch Cuts. *Communications in Computer Algebra*, 44:132–135, 2010.

[Ren92a] J. Renegar. On the Computational-Complexity and Geometry of the 1st-Order Theory of the Reals.1. Introduction - Preliminaries - The Geometry of Semi-Algebraic Sets - The Decision Problem for the Existential Theory of the Reals. *J. Symbolic Comp.*, 13:255–299, 1992.

[Ren92b] J. Renegar. On the Computational-Complexity and Geometry of the 1st-Order Theory of the Reals.2. The General Decision Problem - Preliminaries for Quantifier Elimination. *J. Symbolic Comp.*, 13:301–327, 1992.

[Ren92c] J. Renegar. On the Computational-Complexity and Geometry of the 1st-Order Theory of the Reals.3. Quantifier Elimination. *J. Symbolic Comp.*, 13:329–352, 1992.

[Ric68] D. Richardson. Some Unsolvable Problems Involving Elementary Functions of a Real Variable. *Journal of Symbolic Logic*, 33:514–520, 1968.

[Ric97] D. Richardson. How to Recognize Zero. *J. Symbolic Comp.*, 24:627–645, 1997.

[Ros68] M. Rosenlicht. Liouville's Theorem on Functions with Elementary Integrals. *Pacific J. Math*, 24:153–161, 1968.

[Sei54] A. Seidenberg. A new decision method for elementary algebra. *Ann. Math.*, 60:365–374, 1954.

[Tar48] A. Tarski. A Decision Method for Elementary Algebra and Geometry. Technical report, The Rand Corporation, Santa Monica CA, 1948.

[Tar51] A. Tarski. A Decision Method for Elementary Algebra and Geometry, 2nd ed. *Univ. Cal. Press*, 1951.

[Tro97] Michael Trott. Visualization of Riemann Surfaces of Algebraic Functions. *Mathematica in Education and Research*, 6(4):15–36, 1997.

[vdH02] J. van der Hoeven. A new Zero-test for Formal Power Series. In T. Mora, editor, *Proceedings ISSAC 2002*, pages 117–122, 2002.

[VDW53] B.L. Van Der Waerden. *Modern Algebra, Volume I.* F. Ungar, New York, 1953.

[Wei98] V. Weispfenning. A new Approach to Quantifier Elimination for Real Algebra. *Quantifier Elimination and Cylindrical Algebraic Decomposition*, 1998.

[WG93] T. Weibel and G.H. Gonnet. An Assume Facility for CAS with a Sample Implementation for Maple. In *Proceedings DISCO '92*, pages 95–103, 1993.

Appendix A

Definition of the Inverse Elementary Function

These definitions of inverse elementary functions are taken from [CDJW00], which repeats, with more justification, the definitions given in [AS64]. They also agree with Maple (with the exception of arccot, for the reasons explained in [CDJW00]). These definitions are in terms of log and therefore specify the branch cuts of these functions in terms of the branch cut of log — Section 4.1.

$$\begin{aligned}
\arcsin(z) &= -i \log\left(\sqrt{1-z^2} + iz\right) \\
\arccos(z) &= \frac{\pi}{2} - \arcsin(z) = \frac{2}{i} \log\left(\sqrt{\frac{1+z}{2}} + i\sqrt{\frac{1-z}{2}}\right) \\
\arctan(z) &= \frac{1}{2i}(\log(1+iz) - \log(1-iz)) \\
\operatorname{arccot}(z) &= \frac{1}{2i} \log\left(\frac{z+i}{z-i}\right) = \arctan\left(\frac{1}{z}\right)
\end{aligned}$$

$$\begin{aligned}
\operatorname{arcsec}(z) &= \arccos\left(\frac{1}{z}\right) \\
&= -i\log\left(\frac{1}{z} + i\sqrt{1-\frac{1}{z^2}}\right), \\
&\quad \text{with } \operatorname{arcsec}(0) = \frac{\pi}{2} \\
\operatorname{arccsc}(z) &= \arcsin\left(\frac{1}{z}\right) \\
&= -i\log\left(\frac{i}{z} + i\sqrt{1-\frac{1}{z^2}}\right), \\
&\quad \text{with } \operatorname{arcsec}(0) = 0
\end{aligned}$$

$$\operatorname{arcsinh}(z) = \log(z + \sqrt{1+z^2})$$

$$\operatorname{arccosh}(z) = 2\log\left(\sqrt{\frac{z+1}{2}} + \sqrt{\frac{z-1}{2}}\right)$$

$$\operatorname{arctanh}(z) = \frac{1}{2}(\log(1+z) - \log(1-z))$$

$$\operatorname{arccoth}(z) = \frac{1}{2}(\log(-1-z) - \log(1-z))$$

$$\operatorname{arcsech}(z) = 2\log\left(\sqrt{\frac{z+1}{2z}} + \sqrt{\frac{1-z}{2z}}\right)$$

$$\operatorname{arccsch}(z) = \log\left(\frac{1}{z} + \sqrt{1+\left(\frac{1}{z}\right)^2}\right)$$

Appendix B

Branch Cuts of the Elementary Functions

Table B.1 lists the branch cuts for square root, logarithm, and the inverse of trigonometric and hyperbolic functions.

Function	**Branch cuts**
square roots	$(-\infty, 0)$
logarithm	$(-\infty, 0]$
arcsin	$(-\infty, -1) \cup (1, \infty)$
arccos	$(-\infty, -1) \cup (1, \infty)$
arctan	$(-i\infty, -i] \cup [i, i\infty)$
arccsc	$(-1, 1)$
arcsec	$(-1, 1)$
arccot	$[-i, i]$
arcsinh	$(-i\infty, -i) \cup (i, \infty)$
arccosh	$(-\infty, 1)$
arctanh	$(-\infty, -1] \cup [1, \infty)$
arccsch	$(-1, 1)$
arcsech	$(-\infty, 0] \cup (1, \infty)$
arccoth	$[-1, 1]$

Table B.1: Branch cuts of inverse elementary functions

Appendix C

Formulae for the Inverse Elementary Functions

The formulae are given in [Lit99]. They use the auxiliary function, csgn:

$$\operatorname{csgn}(z) = (-1)^{\mathcal{K}(2\log(z))} = \begin{cases} +1, & \text{if } \Re(z) > 0 \text{ or } \Re(z) = 0;\ \Im(z) \geq 0; \\ -1, & \text{if } \Re(z) < 0 \text{ or } \Re(z) = 0;\ \Im(z) < 0. \end{cases}$$

The unwinding number [CJ96] $\mathcal{K}$ is defined in Definition 2.3.1.

$$\arcsin(\sin(z)) = \begin{cases} z - 2\pi\mathcal{K}(zi), & \text{if } \operatorname{csgn}(\cos(z)) = 1; \\ \pi - z - 2\pi\mathcal{K}(i(\pi - z)), & \text{if } \operatorname{csgn}(\cos(z)) = -1. \end{cases}$$

$$\arccos(\cos(z)) = \begin{cases} z - 2\pi\mathcal{K}(zi), & \text{if } \operatorname{csgn}(\sin(z)) = 1; \\ -z - 2\pi\mathcal{K}(-zi), & \text{if } \operatorname{csgn}(\sin(z)) = -1. \end{cases}$$

$$\begin{aligned} \arctan(\tan(z)) =& z + \pi(\mathcal{K}(-zi - \log(\cos(z))) - \mathcal{K}(zi - \log(\cos(z)))), \\ & \text{provided } z \neq \frac{\pi}{2} + n\pi \mid n \in \mathbf{Z}. \end{aligned}$$

$$\operatorname{arcsinh}(\sinh(z)) = \begin{cases} z - 2\pi i\mathcal{K}(z), & \text{if } \operatorname{csgn}(\cosh(z)) = 1; \\ i\pi - z - 2\pi i\mathcal{K}(i\pi - z), & \text{if } \operatorname{csgn}(\cosh(z)) = -1. \end{cases}$$

$$\operatorname{arccosh}(\cosh(z)) = \begin{cases} z - 2\pi\mathcal{K}(z), & \text{if } \operatorname{csgn}(\sinh(z))\cos(n\pi) = 1; \\ -z - 2\pi i\mathcal{K}(-z), & \text{if } \operatorname{csgn}(\sinh(z))\cos(n\pi) = -1. \end{cases}$$

$$\text{where } n = \mathcal{K}(\log(\cosh(z) - 1) + \log(\cosh(z) + 1)).$$

$$\operatorname{arctanh}(\tanh(z)) = z + i\pi(\mathcal{K}(z - \log(\cosh(z))) - \mathcal{K}(z - \log(\cosh(z)))),$$

$$\text{provided } z \neq \frac{\pi}{2}i + in\pi \mid n \in \mathbf{Z}.$$

Appendix D

Examples

List of examples used in Chapter 8.

1.$\sqrt{z^2} \overset{?}{=} z$

2.$\sqrt{z^2} \overset{?}{=} -z$

3.$\sqrt{1-z^2} \overset{?}{=} \sqrt{1-z}\sqrt{1+z}$

4.$\sqrt{z^2-1} \overset{?}{=} \sqrt{z-1}\sqrt{z+1}$

5.$\sqrt[3]{z^3} \overset{?}{=} z$

6.$\sqrt[4]{z^4} \overset{?}{=} z$

7. $\log\left(\frac{1}{z}\right) \overset{?}{=} -\log(z)$

8. $\log\left(-\frac{1}{z}\right) \overset{?}{=} -\log(-z)$

9. $\log(z^2) \overset{?}{=} 2\,\log(z)$

10. $\log(z^3) \overset{?}{=} 3\,\log(z)$

11. $\arctan(x) + \arctan(y) \overset{?}{=} \arctan\left(\frac{x+y}{1-xy}\right)$

12. $\operatorname{arctanh}(x) + \operatorname{arctanh}(y) \overset{?}{=} \operatorname{arctanh}\left(\frac{x+y}{1+xy}\right)$

13.$\sqrt{xy} \overset{?}{=} \sqrt{x}\sqrt{y}$

14. $\log(xy) \overset{?}{=} \log(x) + \log(y)$

15. $\log(pq) \overset{?}{=} \log(p) + \log(q)$ with $p = z^2 - 3z + 2, q = z^2 + 4z + 1$

16. $\log(pq) \overset{?}{=} \log(p) + \log(q)$ with $p = z^3 - 3z + 2, q = z^2 + 4z + 1$

17. $\log(pq) \overset{?}{=} \log(p) + \log(q)$ with $p = z^3 - 3z + 2, q = z^3 + z^2 + 4z + 1$

18.$\sqrt{\sqrt{p}+\sqrt{q}}\sqrt{\sqrt{p}-\sqrt{q}} \overset{?}{=} \sqrt{p-q}$ with $p = 3z + 1, q = z - 4$

19. $\operatorname{arccosh}(x) + \operatorname{arccosh}(y) \overset{?}{=} \operatorname{arccosh}\left(xy + \sqrt{(x^2-1)(y^2-1)}\right)$

20. $\operatorname{arcsinh}(x) + \operatorname{arcsinh}(y) \overset{?}{=} \operatorname{arcsinh}\left(x\sqrt{1+y^2} + y\sqrt{1+x^2}\right)$

21. $\sqrt{zw} \overset{?}{=} \sqrt{z}\sqrt{w}$

22. $\log(zw) \overset{?}{=} \log(z) + \log(w)$

23. $\arctan(z) + \arctan(w) \overset{?}{=} \arctan\left(\frac{z+w}{1-zw}\right)$

24. $\operatorname{arctanh}(z) + \operatorname{arctanh}(w) \overset{?}{=} \operatorname{arctanh}\left(\frac{z+w}{1+zw}\right)$

25. $\log(pq) \overset{?}{=} \log(p) + \log(q)$ with $p = z^2 + 1, q = w - 1$

26. $\log(pq) \overset{?}{=} \log(p) + \log(q)$ with $p = z^2 + 1, q = w^2 - 1$

27. $\log(pq) \overset{?}{=} \log(p) + \log(q)$ with $p = z - w + 4, q = w - 2z + 3$

28. $\log(pq) \overset{?}{=} \log(p) + \log(q)$ with $p = z^2 + z - w + 1, q = w - z + 2$

29. $\sqrt{\sqrt{z}+\sqrt{w}}\sqrt{\sqrt{z}-\sqrt{w}} \overset{?}{=} \sqrt{z-w}$