The present work was submitted to the LuFG Theory of Hybrid Systems

BACHELOR OF SCIENCE THESIS

USING THOM ENCODINGS FOR REAL ALGEBRAIC NUMBERS IN THE CYLINDRICAL ALGEBRAIC DECOMPOSITION

Tobias Winkler

Examiners:
Prof. Dr. Erika Ábrahám
Prof. Dr. Viktor Levandovskyy

Additional Advisor:
Gereon Kremer, M.Sc.

Aachen, 29.09.2016

Abstract

The *cylindrical algebraic decomposition* (CAD) is a method that can be employed to decide whether a conjunction of *quantifier-free non-linear real arithmetic* constraints is satisfiable. Applied in this way, the CAD needs an exact representation for *real algebraic numbers* and algorithms performing certain operations on them. These include, for example, comparison, computation of intermediate points and determination of the sign a polynomial achieves at a number given in the specific representation.

In this thesis, we explain how real algebraic numbers can be represented by so-called *Thom encodings* and give detailed descriptions of various algorithms realizing the required operations. We have also implemented the algorithms within the SMT solver SMT-RAT and evaluate them on a benchmark set provided by the SMT community. The results are compared to the ones produced by a CAD implementation which uses *isolating intervals* as a real algebraic number representation.

Erklärung

Hiermit versichere ich, dass ich die vorgelegte Arbeit selbstständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe. Alle benutzten Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate wurden als solche gekennzeichnet.

Tobias Winkler
Aachen, den 29. September 2016

Danksagung

Mein Dank gilt Prof. Dr. Erika Ábrahám, die es mir ermöglichte, meine Bachelorarbeit für die von ihr geleitete „Theory of Hybrid Systems Group“ am Lehrstuhl 2 für Informatik zu schreiben. Ich danke außerdem Prof. Dr. Viktor Levandovskyy, der sich als Zweitgutachter zur Verfügung gestellt hat. Besonderen Dank möchte ich weiterhin meinem Betreuer Gereon Kremer für den Vorschlag des interessanten Themas und die durchgehende Unterstützung bei der Implementierung aussprechen. Darüber hinaus danke ich allen anderen Entwicklern von CArL und SMT-RAT, da diese Tools meine Arbeit erst ermöglicht haben. Insbesondere möchte ich Sebastian Junges erwähnen, der mir bei einem wichtigen Problem weiterhalf.

Ich möchte mich an dieser Stelle auch bei der Hans Hermann Voss-Stiftung und dem Bildungsfons der RWTH für die Unterstützung meines Studiums bedanken.

Zu guter Letzt danke ich meiner Familie und meiner Freundin Raquel für den starken Rückhalt, den sie mir während der Zeit des Schreibens gaben.

Contents

Chapter 1

Introduction

The concept of a *cylindrical algebraic decomposition* (CAD) was originally developed by G. E. Collins in 1975 and since then has become a standard tool in algebraic geometry [Col75]. Let $\mathcal{P} \subset \mathbb{Q}[X_1,...,X_n]$ be finite. The fundamental idea of a CAD associated to $\mathcal{P}$ is to construct a partition of $\mathbb{R}^n$ into a finite number of *cells* such that each cell is $\mathcal{P}$-*sign-invariant*. That means that if $(\xi_1,...,\xi_n) \in \mathbb{R}^n$ is a point from a CAD cell, then it holds that $sgn(p(\xi_1,...,\xi_n))$ is the same for all $p \in \mathcal{P}$.[1] We do not give a formal definition of the CAD here since it would be beyond our scope. An exact definition can be found in [ACM84].

A *quantifier-free non-linear real arithmetic* constraint is an expression $p \sim 0$ where p is a polynomial in $\mathbb{Q}[X_1,...,X_n]$ and $\sim$ is one of the relations $>, \geq, =, \leq, <$ or $\neq$ (see Chapter 2 for more details). In order to decide whether a *conjunction* of constraints $p_1 \sim 0, ..., p_k \sim 0$ is satisfiable, it is enough to know only one sample point from each cell of a CAD associated to $\{p_1,...,p_k\}$. Since there are finitely many cells and thus only finitely many sample points, we obtain a decision procedure by trying out if any of them satisfies all constraints.

The aim of the *CAD algorithm* is to compute one sample point from each cell of a CAD associated to the set $\mathcal{P} \subset \mathbb{Q}[X_1,...,X_n]$ of polynomials it receives as input. This is achieved in two stages which we are going to explain now using an example.

Suppose we want to compute a CAD associated to the polynomials $p_1(X,Y) := (X-2)^2 + (Y-2)^2 - 1$ and $p_2(X,Y) := X - Y$. Their zeros in $\mathbb{R}^2$ form the circle and the line plotted in Figure 1.1.

Projection phase In this stage, we compute a set $\mathcal{P}_1$ of polynomials in one variable less than p_1 and p_2, that means we *eliminate* a variable. In the example, we may choose to eliminate Y. The polynomials in $\mathcal{P}_1$ should be such that their *real roots* correspond to the projection of a certain set of 'significant points' of the circle and the line onto the X-axis. These points will determine the boundaries of the CAD cells. In our concrete example, they include the intersections of the circle and the line as well as the points where the tangent of the circle is perpendicular to the X-axis. They are depicted in Figure 1.1a. The *projection*, that means the polynomials in $\mathcal{P}_1$, can be computed with the help of *subresultants* [ACM84]. It is a challenge to design projection operators for the CAD in a way such that they do not produce redundant polynomials having no real roots at all or polynomials with unnecessarily high degree. Since the discovery of the original method, many improvements for the

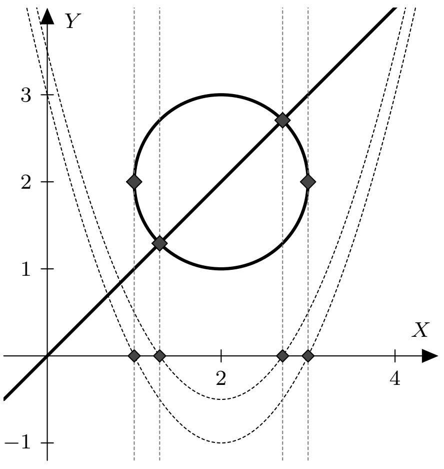
[1]For $x \in \mathbb{R}$, we let $sgn(x) = 1$ if and only if $x > 0$, $sgn(x) = -1$ if and only if $x < 0$ and $sgn(0) = 0$.

projection operator have been proposed [Hon90] [McC98]. For our example, let us suppose that the projection operator produced the result

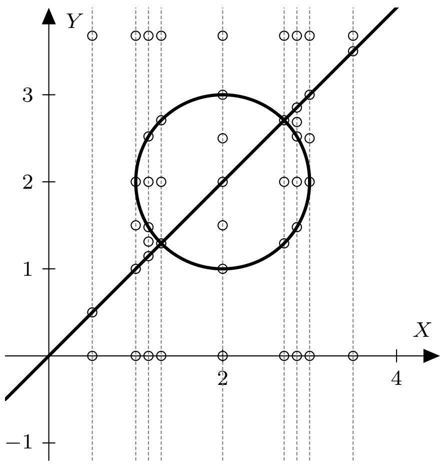
$$\mathcal{P}_1 = \{X^2 - 4X + 3,\, X^2 - 4X + \frac{2}{7}\}.$$

These polynomials are also plotted in Figure 1.1a.

Lifting phase In this stage, a sample point for each cell of the CAD is constructed. For this sake, we have to characterize the real roots of the polynomials in $\mathcal{P}_1$, a number between two consecutive roots and numbers below and above all these roots. Each of these numbers ξ_1 then has to be plugged in p_1 and p_2, which results again in univariate polynomials, but this time in Y. For each $p(\xi_1, Y)$ that we obtain in this way, we again characterize its roots as well as points between, below and above all roots. This operation if called 'lifting' the point ξ_1. For each number ξ_2 that resulted from lifting ξ_1, we get a sample point (ξ_1, ξ_2) (Figure 1.1b).



(a)



(b)

Figure 1.1: (a) The thick line and circle are the zeros of the polynomials p_1 and p_2 from our example. The indicated spots on the circle are the 'significant points'. The roots of the two parabolas, which constitute the set $\mathcal{P}_1$, correspond precisely to the points obtained by projecting the significant points onto the X-axis. (b) Sample points from all 47 cells of a CAD associated to $\{p_1,\, p_2\}$. The points on the X-axis correspond to the roots of the projection $\mathcal{P}_1$, numbers between, below and above all these roots.

The roots that have to be characterized in the lifting phase are *real algebraic numbers* which might be irrational. Thus, they cannot always be represented explicitly. As we want a formal method which is guaranteed to produce exact results, we are not satisfied with rational approximations. For this reason, we have to develop techniques and algorithms to represent the roots exactly and perform the necessary operations in this representation.

Related work The idea of *Thom encodings* as a representation technique for real algebraic numbers first showed up in [CR88]. In [BPR10], it is shown how Thom encodings can be actually used in the CAD algorithm. The *sign determination algorithm*, which constitutes the basis for the algorithms operating on numbers in the Thom representation, was proposed in [BOKR86].

Contributions We describe the algorithms from [BPR10] in detail and propose minor optimizations and adaptations to the special case of real numbers. We also implemented the algorithms within the SMT solver SMT-RAT and evaluate them on a benchmark set provided by the SMT community [BFT16].

Structure of this thesis In the next Chapter 2, we present some background and preliminary knowledge needed in order to understand both the context and the mathematical details of the subsequent chapters. In particular, we provide an overview of the real algebraic numbers operations necessary for the CAD algorithm. In Chapter 3, we discuss *Thom encodings* as the main topic of this thesis and give the algorithms performing the operations necessary for the CAD algorithm. The following Chapter 4 is dedicated to the *sign determination algorithm* and, in particular, the computation of *Tarski queries*. In Chapter 5, we describe some practical aspects of our implementation and provide experimental results. Finally, in Chapter 6, we discuss the results and list a couple of aspects that could be further improved.

Chapter 2

Background

2.1 Satisfiability checking

Propositional logic (PL) is the attempt to create a formal system which captures the intuitive meaning of phrases like 'if ... then ...', 'either ... or ...' and others that we use in our daily routine. For example, the transcription of 'If it rains, then the street is wet' into PL would be

$$A \rightarrow B,$$

where A and B are the *propositions* 'It rains' and 'The street is wet' and the symbol $\rightarrow$ denotes the *implication* 'if ... then ...'. Such an expression is called a PL *formula.* Other symbols that we frequently use to compose formulas are $\vee$ (inclusive 'or'), $\wedge$ ('and') as well as $\neg$ ('not'). An important characteristic of PL is that the propositions are completely unrelated among each other. While 'If it rains, then the street is wet' might seem a reasonable statement in natural language, within PL it looks just the same as any arbitrary nonsense like 'If I am an elephant, then $2 + 2 = 5$'. This is because the system does not 'know' that there is a relation between rain and water on the street. The actual benefit of PL is that, using the formal system, we are able to test whether the formulas we compose contain any *contradictions.* Consider, for example, the statement

'It rains and if it rains the street is wet and the street is not wet'

and its translation

$$A \wedge (A \rightarrow B) \wedge \neg B \tag{2.1}$$

into a PL formula. Clearly, the statement contains a contradiction and this is not due to any physical characteristics of rain and water but rather because the rules of 'logical reasoning' have been violated.

If a PL formula is free of contradictions, then there exist *truth values* that can be assigned to each proposition in the formula such that it is *satisfied.* In this context, we often think about propositions being *Boolean variables.* The truth values are usually denoted by $true$ and $false$ or 1 and 0, respectively. For example, in $A \rightarrow B$, we could assign A and B the values $true$ and the implication would be satisfied. It would be *unsatisfied* if we set A to $true$ and B to $false$. A detailed introduction to propositional logic can be found in [vD94].

Satisfiability problem The task to decide whether a given PL formula like 2.1 is satisfiable is called the *satisfiability problem* (SAT). It is decidable: Since every formula must be finite, it contains at most finitely many Boolean variables $A_1,...,A_n$. Thus, one can decide whether there exists a satisfying assignment by trying out all possible 2^n assignments of truth values. This method has an exponential worse case complexity and is not applicable in practice where we have to deal with huge formulas. The *Davis-Putnam-Logemann-Loveland* (DPLL) algorithm is a more sophisticated procedure for deciding SAT, which was first introduced in 1962 [DLL62]. Nowadays, many SAT solvers are still based on this algorithm, for example the open-source solver *MiniSat* [ES03]. However, it has been shown that its worst case time complexity is exponential as well and that there is not much hope for (asymptotically) faster algorithms since SAT has been proven to be an NP-complete problem [Coo71].

2.2 Satisfiability modulo theories

Satisfiability modulo theories (SMT) is an approach which tackles the drawback that the propositions in PL formulas are semantically isolated like explained above. In order to create more expressive formulas, the propositions are replaced by *constraints* defined according to some specific *theory.*

The theory of interest within this thesis is the *quantifier-free non-linear real arithmetic* (QF_NRA). Its constraints are defined by the following abstract grammar:

$$\begin{aligned} p &:= a \;\mid\; X \;\mid\; (p+p) \;\mid\; (p\cdot p) \\ c &:= p>0 \;\mid\; p\geq 0 \;\mid\; p=0 \;\mid\; p\leq 0 \;\mid\; p<0 \;\mid\; p\neq 0 \end{aligned}$$

Here, a is a rational constant and X is a real valued *variable* from an infinite pool X,Y,Z,... of variables. The p are called terms or *polynomials* and the c are the constraints. Its semantics are given by the usual meanings of $+$, $\cdot$, $>$ and so on for real numbers. For example,

$$\big((((X \;+\; -2)\cdot(X \;+\; -2)) + ((Y \;+\; -2)\cdot(Y \;+\; -2))) \;+\; -1\big) \;<\; 0$$

is a QF_NRA constraint. However, we favor the habitual notation from algebra where we leave out brackets and use exponents, such that the syntax of the above expression is simplified to $(X-2)^2+(Y-2)^2-1<0$.

A *QF_NRA formula* is obtained by substituting the propositions in a PL formula with QF_NRA constraints. For example,

$$(X-2)^2+(Y-2)^2-1<0 \wedge (\neg(X-3<0) \vee X-Y \geq 0) \tag{2.2}$$

is a QF_NRA formula. The specific SMT problem is then to decide whether there exist real values for the variables in the formula such that all constraints are satisfied.

Lazy SMT solving In the *lazy* approach [Seb07] for solving SMT problems, a *Boolean abstraction* of the input SMT formula is handed to a DPLL-based SAT solver. The abstraction is a PL formula which is obtained by replacing the theory constraints by Boolean variables again. Before we do this, we also 'push' the negations into the constraints, meaning that in the example formula 2.2, the negated constraint constraint $\neg(X-3<0)$ would become $X-3\geq 0$. A complete Boolean abstraction of formula 2.2 would then be

$$A \wedge (B \vee C). \tag{2.3}$$

The SAT solver either proves unsatisfiability of the abstraction, meaning that the SMT problem is also unsatisfiable, or it produces a satisfying variable assignment. In the example, the SAT solver might output the satisfying assignment

$$A \mapsto 1,\, B \mapsto 1,\, C \mapsto 0,$$

for the Boolean abstraction 2.3. Then it has to be checked if the theory constraints corresponding to the Boolean variables which are set to *true* under the current assignment are consistent with respect to the semantics of the theory. This functionality is implemented a so-called *theory solver*. In our example, the theory solver would have to decide whether there exist real numbers X and Y such that the inequalities (1) $(X-2)^2+(Y-2)^2-1<0$ and (2) $X-3 \geq 0$ hold *together*. All points (X,Y) that satisfy (1) must lie *within* in a circle with midpoint (2,2) and radius 1. For none of these points it holds that $X \geq 3$, so the constraints (1) and (2) are contradictory. If they were not, then we would be done and the procedure could return the result *satisfiable*. Since in our example this is not the case, the theory solver informs the SAT solver about the discovered inconsistency, providing a (preferably small) subset of constraints that are responsible for causing the conflict. Here, this *infeasible subset* would consist of the two constraints (1) and (2), but in general it can be significantly smaller than the whole input the theory module has received. As soon as the SAT solver is informed about an inconsistency, it has to find an alternative satisfying assignment of the Boolean abstraction. In our case, it might undo its decision that B is set to 1 and assign it 0 instead. Additionally, it sets C to 1. The theory solver then receives the constraints (1) $(X-2)^2+(Y-2)^2-1<0$ and (3) $X-Y \geq 0$, which are also depicted in Figure 1.1 of Chapter 1. The points satisfying (3) are the ones that lie on and below the line. Since some of these points are also located within the circle, the two constraints are not conflicting. The theory solver informs the SAT solver about this circumstance so that it can return the final result *satisfiable*.

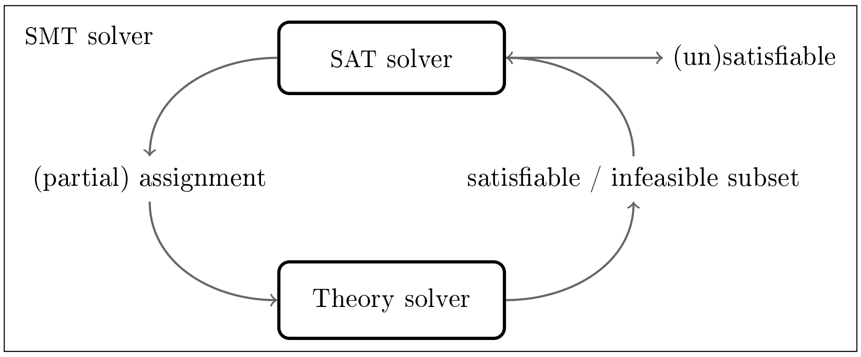


Figure 2.1: Architecture of a lazy SMT solver

The lazy approach can be further classified into *full-lazy* and *less-lazy* SMT solving. In a full-lazy framework, the theory module is only consulted when a complete satisfying assignment of the Boolean abstraction is found, just like illustrated in our example. In the less-lazy approach, the SAT solver can consult the theory solver more frequently, even with *partial* assignments. Thus, the less-lazy approach is a generalization of the full-lazy technique.

SMT-compliance A theory solver which is to be used within a lazy SMT framework should have some specific features. In the example above, we have already seen that it should have the ability to generate infeasible subsets of its input constraints if their conjunction turns out to be unsatisfiable. This has the effect that the SAT solver knows in which states it should not enter again, ensuring a terminating procedure. Each time the SAT solver is informed about an infeasible subset, it will react undoing a couple of decisions that led to the inconsistent variable assignment. The smaller an infeasible subset is, the less decisions have to be undone, meaning that the SAT solver does not always have to start from the beginning again. This results in the typical behavior that between two consecutive theory solver calls, parts of the assignment remain the same. In other words, the SAT solver *adds* and *removes* a few constraints while usually the bigger part of them remains untouched. The overall procedure greatly benefits from the theory solver being able to take advantage of this behavior.

Suppose that the theory solver is in a state S where it is able to decide whether a set $\mathcal{C}$ of constraints is consistent. For example, if the theory is QF_NRA and the solver is implemented using the CAD algorithm, this state corresponds to the situation where all sample points of a CAD associated to the polynomials of the constraints in $\mathcal{C}$ have already been computed (or at least enough sample points to prove satisfiability). Now consider a set $\tilde{\mathcal{C}} = \mathcal{C} \cup \{c\}$ of constraints for some $c \notin \mathcal{C}$. We say that the theory solver supports *incrementality* if, starting from the state S, it can 'efficiently' get into a state $\tilde{S}$ that permits deciding whether the conjunction of the constraints in $\tilde{\mathcal{C}}$ is satisfiable. By 'efficient' we mean that the cost of the computations necessary to get into $\tilde{S}$ is *strictly less* if we use S as a starting point than if we would compute $\tilde{S}$ from the initial solver state. Similarly, if the same holds when $\tilde{\mathcal{C}} = \mathcal{C} \setminus \{c\}$ for some $c \in \mathcal{C}$, then we say that the theory solver supports *backtracking*.

In summary, we call a theory solver *SMT-compliant* if it supports incrementality, backtracking and the generation of infeasible subsets.

2.3 SMT-RAT

SMT-RAT [CKJ+15] is a lazy SMT solver which supports several theories. Currently, there exist SMT-compliant solvers for linear and non-linear integer and real arithmetic, bitvectors, equality logic with uninterpreted functions and others. SMT-RAT also permits the integration of custom theory modules into the lazy SMT framework. Another core feature is that so called *strategies* can be defined in order to compose new solvers easily.

Among others, there is a non-linear real arithmetic solver available which is based on the CAD method presented in Chapter 1. The standard method is enhanced with additional features in order to make it SMT-compliant. In particular, one can add and remove polynomials from an already computed CAD efficiently. Moreover, the solving behavior is such that instead of constructing the whole set of sample points first and then checking if any of these points satisfies all constraints, the sample points are lifted one after another. Once a point is lifted completely, it is checked if it is a solution. This often avoids constructing the complete set of sample points in cases where the conjunction of the input constraints is satisfiable.

2.4 Algebraic preliminaries

2.4.1 Polynomials

One of the most essential algebraic concepts are *polynomials*. We are going to define them formally and discuss some related ideas that are relevant in our further discussion. The definitions can be found in many standard references like [BPR10], [Mis93], [GCL92] or [vzGG13], where they are usually presented in a more general context.

Throughout this section, let $R \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}\}$ be the usual integer, rational, real or complex numbers. Moreover, we let $\mathbb{N}$ be the non-negative and $\mathbb{N}_+$ be the positive integers.

Definition 2.4.1 (Polynomial). *Let $\mathbf{X} = (X_1,...,X_n)$, $n \in \mathbb{N}_+$, be a list of* variables *and let $\mathbf{e} = (e_1,...,e_n) \in \mathbb{N}^n$. An expression of the form*

$$\mathbf{X}^\mathbf{e} := \prod_{i=1}^{n} X_i^{e_i}$$

is called a monomial *in $\mathbf{X}$. An expression of the form*

$$p := \sum_{\mathbf{e} \in \mathbb{N}^n} a_\mathbf{e} \mathbf{X}^\mathbf{e} \tag{2.4}$$

where only a finite number of the coefficients *$a_\mathbf{e} \in R$ is non-zero is called a* polynomial *in the domain $R[\mathbf{X}] = R[X_1,...,X_n]$.*

Sometimes we write $p(X_1,...,X_n)$ in order to indicate which variables appear in p. If the coefficient $a_\mathbf{e}$ is non-zero, then we say that p has or contains the monomial $\mathbf{X}^\mathbf{e}$. The *degree* of a monomial $\mathbf{X}^\mathbf{e}$ is defined as $deg(\mathbf{X}^\mathbf{e}) := e_1 + ... + e_n$. We define the *degree* $deg(p)$ of a polynomial p as the maximum of the degrees of the monomials it contains, or $-\infty$ if $p = 0$. If $n = 1$, then we say that p is a *univariate polynomial*, otherwise we call it *multivariate*. For a univariate polynomial $p(X)$, we define its *leading coefficient* to be the coefficient which corresponds to the monomial $X^{deg(p)}$. A QF_NRA term containing the variables $X_1,...,X_n$ like defined in 2.2, can be viewed as a polynomial in $\mathbb{Q}[X_1,...,X_n]$ if we apply the common laws of algebra until obtaining a form like the one in 2.4.

Example 2.4.2. *Let*

$$p_1(X,Y) := 2X^2Y + X + 1.$$

Then p_1 is a polynomial in the domain $\mathbb{Z}[X,Y]$, but we can also regard it as a polynomial in the domain $\mathbb{R}[X,Y,Z]$, for example. The degree of p_1 is 3 and it is a multivariate polynomial. The polynomial

$$p_2(X) := X^5 + \frac{2}{7}X^2 - \frac{1}{2}$$

is a univariate polynomial in the domain $\mathbb{Q}[X]$ with degree 5 and leading coefficient 1.

Of course, one can also define polynomial domains with respect to other structures $R \notin \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}\}$. Usually, the only requirement on R is that it must be a commutative ring. Within the scope of this thesis however, we will only focus on the domains that are actually relevant for our problems.

Sometimes it is convenient to view a multivariate polynomial $p \in R[X_1,...,X_n,X]$, $n \in \mathbb{N}_+$, as a univariate polynomial in the variable X whose coefficients rely on the *parameters* $X_1,...,X_n$. We write $p \in R[X_1,...,X_n][X]$ whenever we apply this notion. Then we write $deg_X(p)$ in order to refer to the degree of p with respect to the variable X.

Example 2.4.3. *Consider once again $p_1(X,Y) = 2X^2Y + X + 1$ from Example 2.4.2. Viewed in the domain $\mathbb{Z}[X][Y]$, p becomes $(2X^2)Y + (X + 1)$, that means, the coefficients of p are $2X^2$ and $X + 1$ and it holds that $deg_Y(p) = 1$. Regarded as a polynomial from $\mathbb{Z}[Y][X]$ on the other hand, the coefficients are $2Y$, 1, 1 and we have $deg_X(p) = 2$.*

2.4.2 Roots

Each polynomial $p \in R[X_1,...,X_n]$ defines a function $\nu_p : R^n \to R$, called the *evaluation* function of p. Given $(\xi_1,...,\xi_n) \in R^n$, we obtain $p(\xi_1,...,\xi_n) := \nu_p(\xi_1,...,\xi_n)$ by substituting the variables $X_1,...,X_n$ for their corresponding values $\xi_1,...,\xi_n$ and evaluating the so obtained arithmetic expression in R. We often plot this function when we want to get a feeling of how a polynomial 'looks like'.

Definition 2.4.4 (Real root). *Let $p \in \mathbb{R}[X]$. Each $\xi \in \mathbb{R}$ with $p(\xi) = 0$ is called a* real root *of p.*

If ξ is a root of $p(X)$, then $p = q(X) \cdot (X - \xi)$ for some polynomial q with $deg(q) = deg(p) - 1$. This implies that p has at most $deg(p)$ different real roots.

2.4.3 Derivatives

The next definition will be particularly important in Chapter 3 where we introduce Thom encodings.

Definition 2.4.5 (Derivative). *We define derivatives for both univariate and multivariate polynomials as follows:*

- *Let $p = \sum_{i=0}^{d} a_i x^i \in R[X]$. The* derivative *$p'$ of p is defined as*

  $$p' := \sum_{i=1}^{d} i \cdot a_i x^{i-1},$$

  with the convention that the empty sum equals 0.

- *For $p \in R[X_1,...,X_n,X]$, we write p'_X for the derivative of p viewed as a polynomial in the domain $R[X_1,...,X_n][X]$ and call this the derivative of p with respect to X.*

For a non-constant univariate polynomial p, the derivative p' has degree $deg(p)-1$. We can also take the derivative of the derivative of p. We call this the second derivative and denote it by p''. In general, the k-th derivative of p is denoted by $p^{(k)}$. The derivative $p^{(deg(p))}$ is always constant and $p^{(deg(p)+l)}$ is the zero polynomial for all $l > 0$. Similar considerations hold when we regard p as a polynomial from a domain like $R[X_1,...,X_n][X]$. Then we write $p_X^{(k)}$ in order to refer to the k-th derivative of p with respect to X.

Example 2.4.6. *The derivative of the polynomial $p_2(X) = X^5 + \frac{2}{7}X^2 - \frac{1}{2}$ from Example 2.4.2 is*

$$p_2' = 5X^4 + \frac{4}{7}X,$$

and the derivative of $p_1(X,Y) = 2X^2Y + X + 1$ with respect to Y is the polynomial

$$p'_{1_Y} = 2X^2.$$

2.5 Real algebraic numbers in the CAD

Definition 2.5.1 (Real algebraic numbers)**.** *The set $\mathbb{R}_{alg}$ of real algebraic numbers is defined by*

$$\mathbb{R}_{alg} := \{\xi \in \mathbb{R} : \textit{ there exists } p \in \mathbb{Q}[X] \setminus \{0\} \textit{ with } p(\xi) = 0\}.$$

In other words, a real number is *algebraic* if it is a real root of a non-zero rational polynomial. In the literature, the real algebraic numbers are usually defined as real roots of *integer* polynomials (see for example [BPR10], Chapter 2). However, the two definitions are equivalent because for every non-zero

$$p = \frac{a_n}{b_n} X^n + ... + \frac{a_0}{b_0} \in \mathbb{Q}[X],$$

we observe that the polynomial $b_n \cdot ... \cdot b_0 \cdot p$ is in $\mathbb{Z}[X]$ and has the same roots as p. We have chosen the definition with rational polynomials because it appears more natural in our context.

Every $q \in \mathbb{Q}$ is algebraic since it is a root of the polynomial $X - q$. There are real numbers which are *not* algebraic: A famous example is the number π [Lin00]. We therefore have the relation $Q \subset \mathbb{R}_{alg} \subset \mathbb{R}$, where all inclusions are strict.

Another notable fact is that $\mathbb{R}_{alg}$ forms a *real closed field* ([BPR10], Exercise 2.11). Roughly speaking, this means that certain fundamental properties of $\mathbb{R}$ also hold in $\mathbb{R}_{alg}$. In particular, $\mathbb{R}_{alg}$ is closed under addition, multiplication and the computation of inverse elements.

2.5.1 Real algebraic number representations

Since $\mathbb{R}_{alg}$ contains irrational numbers, any finite representation of a real algebraic number must be implicit. From the definition of real algebraic numbers it follows immediately that it is a viable approach to represent a real algebraic number by a polynomial $p \in \mathbb{Q}[X]$ for which it is a root and – for the case that the p has multiple real roots – add some additional information which uniquely identifies it within the set of all real roots of p. It is intuitive that this 'additional piece of information' can be stored in a finite memory because we only need to distinguish between a finite amount of roots.

Definition 2.5.2 (Real algebraic number representation)**.** *Let $\xi \in \mathbb{R}_{alg}$ and $p \in \mathbb{Q}[X]$ such that ξ is a real root of p. A* real algebraic number representation *of ξ is a tuple (p, Ω), where Ω is a finite 'additional information' which uniquely discriminates ξ from the other real roots of p.*

Different designs for the additional information Ω yield different representation techniques. In the following, we will see two distinct approaches and in Chapter 3 we are going to study Thom encodings as a real algebraic number representation.

A straightforward (and very 'implicit') real algebraic number representation is the *order representation* ([Mis93], Section 8.5). A root of a p is uniquely identified with a positive integer describing its position in the ordered list of real roots of p. For example, $(X^2 - 2, 2)$ is an order representation of $\sqrt{2}$. At first glance, this representation might seem quite efficient: The root finding problem, for example, boils down to real root counting which is quite cheap as we will see in the next section. However, when we try to implement the comparison operation for this representation, we already run into trouble: The information about the position in the list of roots does not establish any relation between two real algebraic numbers represented by (p_1, n_1) and (p_2, n_2) with $p_1 \neq p_2$. The order representation has applications [JDM12], but in our context it does not seem well suited.

2.5.2 Required operations

The chosen real algebraic number representation greatly influences the overall complexity of the CAD algorithm because the operations performed in the lifting phase almost exclusively depend on them. The next definition gives an overview of the interfaces that any representation intended to be used in the CAD algorithm must implement.

Definition 2.5.3 (Lifting phase operations). *Let $\xi_1, ..., \xi_n \in \mathbb{R}_{alg}$ be given in terms of real algebraic number representations $(p_1, \Omega_1), ..., (p_n, \Omega_n)$.*

1. ***Root finding.*** *Find representations for all real roots of $p \in \mathbb{Q}[X]$.*
2. ***Comparison.*** *Decide whether $\xi_1 < \xi_2$.*
3. ***Intermediate points.*** *If $\xi_1 < \xi_2$, find a representation of a real algebraic number in $]\xi_1, \xi_2[$.*
4. ***Samples below and above.*** *Find representations of real algebraic numbers ξ_l and ξ_u such that $\xi_l < \xi_1$ and $\xi_u > \xi_1$.*
5. ***Evaluation.*** *For $p \in \mathbb{Q}[X_1,...,X_n]$, determine $sgn(p(\xi_1,...,\xi_n))$.*
6. ***Lifting.*** *For $p \in \mathbb{Q}[X_1,...,X_n,X]$,*
   (a) decide whether $p(\xi_1,...,\xi_n,X)$ is the zero polynomial and, if not,
   (b) find representations of all real roots of $p(\xi_1,...,\xi_n,X)$.

Operation 5 is not an actual evaluation in the sense that the output is a real number. We gave it that name because it can be used to evaluate a *constraint* $p(X_1,...,X_n) \sim 0$. For this purpose, only the *sign* of $p(\xi_1,...,\xi_n)$ is relevant.

2.6 Interval representation

Another real algebraic number representation is the interval representation where the additional information Ω is provided in terms of an *isolating interval*.

Definition 2.6.1 (Interval representation). *Let $\xi \in \mathbb{R}_{alg}$ and $p \in \mathbb{Q}[X]$ with $p(\xi) = 0$. Furthermore, let $a,b \in \mathbb{Q}, a \leq b$ be such that for any real root ξ' of p it holds that*

$$\xi' \in [a,b] \text{ implies that } \xi' = \xi.$$

Then $(p, [a,b])$ is called an interval representation *of ξ and $[a,b]$ is an* isolating interval *for ξ.*

The interval representation is a well-established real algebraic number representation that has already been proposed in the original description of the CAD algorithm [Col75].

2.6.1 Cauchy bounds and Sturm's theorem

Two basic ingredients that are needed in order to implement the operations listed in Definition 2.5.3 for the interval representation are *Cauchy bounds* and *Sturm's Theorem*. We are going to introduce them formally since they will play a role in the later chapters again.

Lemma 2.6.2 (Cauchy bound, [Mis93] Corollary 8.3.2). *Let*

$$p(X) = a_d X^d + a_{d-1} X^{d-1} + ... + a_0 \in \mathbb{R}[X] \setminus \{0\}.$$

If we let

$$C(p) := 1 + \max_{i=0,\ldots,d-1} \frac{|a_i|}{|a_d|},$$

then for any real root ξ of p it holds that $|\xi| < C(p)$.

In other words, all real roots of p lie within the interval $]-C(p),\, C(p)[$.

Example 2.6.3. *Consider $p = X^3 - 2X$. We have $C(p) = 1 + \frac{|-2|}{|1|} = 3$, thus applying Lemma 2.6.2, we conclude that for all real roots of ξ of p it holds that $-3 < \xi < 3$.*

For polynomials $p, q \in \mathbb{R}[X]$, $q \neq 0$, we let $rem(p, q)$ be the remainder of the polynomial division of p by q.

Definition 2.6.4 (Signed remainder sequence, c.f. [BPR10] Definition 1.7). *Let $p, q \in \mathbb{R}[X] \setminus \{0\}$. The* signed remainder sequence *$SRS(p, q)$ is the finite list $(p,q,r_1,...,r_k)$ defined by*

$$\begin{aligned} r_1 &:= -rem(p, q), \\ r_2 &:= -rem(q, r_1), \\ r_3 &:= -rem(r_1, r_2), \\ &\vdots \\ r_k &:= -rem(r_{k-2}, r_{k-1}) \neq 0, \\ r_{k+1} &:= -rem(r_{k-1}, r_k) = 0. \end{aligned}$$

Notice that if $deg(p) > deg(q)$, then for the remainder $r = rem(p,q)$ it holds that $deg(q) > deg(r)$. Thus, the degrees of the polynomials in a signed remainder sequence are strictly decreasing and the sequence is always finite. For a signed remainder sequence $SRS(p, q) = (p, q, r_1,...,r_k)$ we let

$$\nu_a(SRS(p, q)) := \big(p(a), q(a), r_1(a),...,r_k(a)\big)$$

be the list of values that we obtain evaluating the polynomials in the sequence at $a \in \mathbb{R}$. Moreover, we define $Var_a(SRS(p, q))$ to be the number of *sign variations* in $\nu_a(SRS(p, q))$, ignoring zeros. That is, we count a sign variation between two non-zero numbers a and b if $sgn(a) \neq sgn(b)$ and a and b are either consecutive elements in the list or there are arbitrarily many zeros between them.

Example 2.6.5. *Consider once again the polynomial $p = X^3 - 2X$ from Example 2.6.3 and the polynomial $q := 3X^2 - 2$. We compute the signed remainder sequence of p and q:*

$$SRS(p, q) = (X^3 - 2X,\ 3X^2 - 2,\ \frac{4}{3}X,\ 2)$$

The evaluated sequence at the point $a = 3$, for example, is

$$\nu_3(SRS(p,q)) = (21,\ 25,\ 4,\ 2),$$

and for $a = -3$ we obtain

$$\nu_{-3}(SRS(p,q)) = (-21,\ 25,\ -4,\ 2).$$

Thus, we have $Var_3(SRS(p, q)) = 0$ and $Var_{-3}(SRS(p, q)) = 3$.

In Example 2.6.5, the polynomial q was chosen to be the *derivative* of p. This has a reason: There is a relation between the signed remainder sequence $SRS(p, p')$ and the number of real roots of p within a specific interval.

Theorem 2.6.6 (Sturm's Theorem, [Mis93] Corollary 8.4.4, [BPR10] Theorem 2.50)**.** *Let $p \in \mathbb{R}[X]$ be non-constant and let $a,b \in \mathbb{R}$ with $a < b$. The number of different real roots of p that lie in the interval $[a,b]$ is equal to*

$$Var_a(SRS(p, p')) - Var_b(SRS(p, p')).$$

Example 2.6.7. *Using the results from Example 2.6.3 and 2.6.5, we conclude that $p = X^3 - 2X$ has 3 different real roots by Theorem 2.6.6. We verify this result noticing that in fact p has the roots $-\sqrt{2}$, 0 and $\sqrt{2}$.*

2.6.2 Operations for the interval representation

The root finding operation for the interval representation can be implemented as follows: Given a non-constant polynomial p, we first compute the Cauchy bound $C(p)$. Then we calculate the signed remainder sequence of p and p' and apply Sturm's Theorem in order to determine the number of real roots of p in the interval $[-C(p), C(p)]$, which is equal to the total number of roots of p. The interval is then split into halves and once again, Sturm's theorem is applied to count the number of roots that lie within these smaller intervals. This procedure is repeated until we finally obtain isolating intervals for each root of p.

In order to compare real algebraic numbers given as interval representations (p_1, I_1) and (p_2, I_2), a similar technique can be applied to *refine* I_1 and I_2 until they are disjoint.

More sophisticated algorithms for the interval representation can be found in [EMT08].

Chapter 3

Thom encodings of real algebraic numbers

In this chapter, we explain how real algebraic numbers can be finitely represented by so-called *Thom encodings.* The basic idea first appeared in [CR88] and since then has been studied in a number of subsequent papers. In [RS90], for example, several improvements which we shall discuss later are presented. In [CLM$^+$92], the authors give an overview of a number of algorithms performing operations such as addition, multiplication or computation of inverse elements. Nowadays, the Thom representation, which in the literature is sometimes also referred to as the *sign representation*, has found its way into standard references like [Mis93] or [BPR10].

In the latter monograph it is shown how Thom encodings can be used within the CAD algorithm to represent the real algebraic numbers arising in the lifting stage. In particular, the notion of *triangular Thom encodings* is introduced. These are a generalization of the 'standard' Thom encodings and naturally show up when lifting a sample point. The use of Thom encodings in general is motivated by the fact that in a *non-archimedean real closed field* (see [BPR10] Section 2 for a definition) the interval representation fails. According to the authors, in the case of an *archimedean* real closed field (like $\mathbb{R}$) the known algorithms for the interval representation are (in theory) superior to the ones working on the Thom representation. One of the goals of this thesis is to either confirm or refute the validity of this statement in practice and, in particular, within the context of SMT solving where we work with an SMT-compliant CAD algorithm as explained in Chapter 2.

This chapter is divided into two sections: First we are going to present the theoretical foundations necessary in order to understand how Thom encodings work. In the second part we are then going to describe the algorithms that implement the operations we listed in Definition 2.5.3.

3.1 Theoretical background

3.1.1 Thom's Lemma and applications

As we will see later, the basic idea of Thom encodings relies on a theorem called 'Thom's Lemma'. Before we can state it we have to introduce some notations.

Definition 3.1.1 (Sign condition)**.** *Let* $\mathcal{P} \subseteq \mathbb{R}[X_1,...,X_n]$ *and* $\xi \in \mathbb{R}^n$.

- *A mapping* $\sigma : \mathcal{P} \to \{-1,0,1\}$ *is called a* sign condition *on* $\mathcal{P}$.

- *We let* $sgn(\mathcal{P},\xi) : \mathcal{P} \to \{-1,0,1\}$, $p \mapsto sgn(p(\xi))$ *be the* sign condition realized by $\mathcal{P}$ on ξ.
- *We say that* $\mathcal{P}$ realizes $\sigma \in \{-1,0,1\}^{\mathcal{P}}$ *on* ξ *if and only if* $sgn(\mathcal{P},\xi)) = \sigma$.

For better readability, we will sometimes denote a sign condition σ on $\mathcal{P}$ as follows: $\sigma = p_1 \sim_1 0 \wedge \dots \wedge p_k \sim_k 0$, where for all $1 \le i \le k$ we let $\sim_i$ be the symbol

$$\begin{cases} < & \text{if } \sigma(p_i) = -1, \\ = & \text{if } \sigma(p_i) = 0, \\ > & \text{if } \sigma(p_i) = 1. \end{cases}$$

Example 3.1.2. *Let* $\mathcal{P} = \{X+1,\ X^2+Y^2-1\} \in \mathbb{R}[X,Y]$ *and* σ *the sign condition on* $\mathcal{P}$ *defined by*

$$X + 1 > 0 \wedge X^2 + Y^2 - 1 = 0.$$

Then $\mathcal{P}$ *realizes* σ *on* $\left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right) \in \mathbb{R}^2$ *because* $\frac{\sqrt{2}}{2} + 1 > 0$ *and* $\left(\frac{\sqrt{2}}{2}\right)^2 + \left(\frac{\sqrt{2}}{2}\right)^2 - 1 = 0$. *In other words,* $sgn(\mathcal{P}, \left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right)) = \sigma$.

Another notation that we will frequently use is the following: For $p \in \mathbb{R}[X]$, we let

$$Der(p) := \{p^{(i)} : 0 \le i \le deg(p)\},$$

with the convention that $p^{(0)} = p$. Note that $Der(p^{(k)})$ is the set of derivatives of p starting at the k-th derivative.

Example 3.1.3. *For* $p = 2X^2 - 1$, *it holds that* $Der(p) = \{2X^2 - 1,\ 4X,\ 4\}$, *and* $Der(p') = \{4X,\ 4\}$. *However, the expression* $Der(p''')$ *is not valid because* $deg(p) = 2 < 3$.

We now state a simplified version of Thom's Lemma as found in [CR88].

Theorem 3.1.4 (Thom's Lemma). *Let* $p \in \mathbb{R}[X] \setminus \{0\}$ *and let* σ *be a sign condition on* $Der(p)$. *The set*

$$\{\xi \in \mathbb{R} : sgn(Der(p), \xi) = \sigma\}$$

is either empty, a single point or an open interval.

The particularly interesting case of Theorem 3.1.4 is when only a single point $\xi \in \mathbb{R}$ realizes the sign condition σ on $Der(p)$. This always holds when $\sigma(p) = 0$, meaning that ξ is a *root* of p. Thus, Thom's Lemma can be used to *characterize* the real roots of a univariate polynomial. We will formulate this by the following corollary which is an immediate consequence of Thom's Lemma.

Corollary 3.1.5. *Let* $p \in \mathbb{R} \setminus \{0\}$. *If* $\xi_1, \xi_2 \in \mathbb{R}$ *are both roots of* p, *then it holds that*

$$\xi_1 = \xi_2 \iff sgn(Der(p'), \xi_1) = sgn(Der(p'), \xi_2)$$

Proof. '$\Rightarrow$' is trivial. For '$\Leftarrow$' suppose that w.l.o.g. $\xi_1 < \xi_2$ but realize the same sign condition σ on $Der(p)$. Then, by Theorem 3.1.4, all $\xi \in \mathbb{R}$ with $\xi_1 \le \xi \le \xi_2$ also realize σ on $Der(p)$. In particular, $p(\xi) = 0$ which means that p has an infinite number of roots. But this is a contradiction to the assumption that $p \neq 0$. □

The preceding corollary suggests representing a real algebraic number using a polynomial in $\mathbb{Q}[X]$ for which it is a root together with the sign condition it realizes on the derivatives of this polynomial. Using the terms defined in Chapter 2, we could also say that we obtain a real algebraic number representation letting the additional information Ω be the sign condition the derivatives realize on the root. We are going to make this more precise in Definition 3.1.9.

Example 3.1.6. *Consider the polynomial $p = X^3 - 2X \in \mathbb{Q}[X]$ and its derivatives $p' = 3X^2 - 2$, $p'' = 6X$ and $p''' = 6$. The real roots of p are $-\sqrt{2}$, 0 and $\sqrt{2}$. The tables shows the sign conditions realized by the roots on Der(p'), which are also illustrated in Figure 3.1.*

	p'	p''	p'''
$-\sqrt{2}$	1	-1	1
0	-1	0	1
$\sqrt{2}$	1	1	1

As predicted in Corollary 3.1.5, the sign conditions are all different. The real algebraic number $\sqrt{2}$, for example, can be represented by p and the additional information that $\sqrt{2}$ is positive on all derivatives of p.

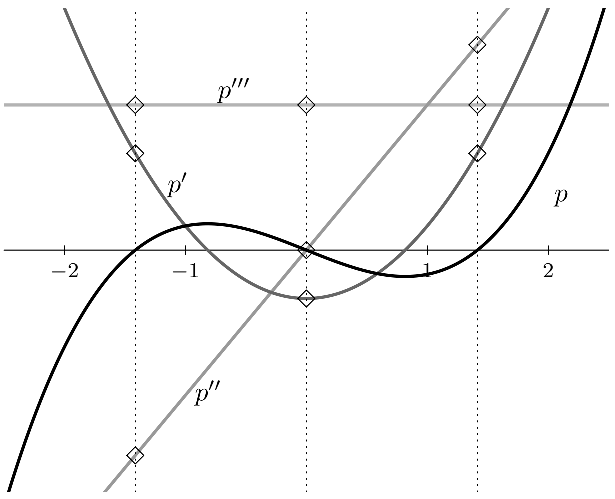


Figure 3.1: The polynomial $p = X^3 - 2X$ from Example 3.1.6 and its derivatives. The sign conditions realized by $\{p', p'', p'''\}$ on the real roots of p are all different.

In order to distinguish between the real roots of $p \in \mathbb{Q}[X]$, we do not necessarily need to know the signs realized on *all* derivatives. In particular, if $deg(p) = d$, then $p^{(d)}$ is constant and $sgn(p^{(d)}(\xi))$ is the same for all real roots ξ of p. Hence the sign conditions realized by $p^{(d)}$ on the roots ξ do not carry any information necessary to discriminate between the roots. A more general remark can be found in [RS90]: In some cases, even the signs realized on the non-constant derivatives can be discarded if we are only interested in distinguishing between the roots. Note that in Example 3.1.6, the signs realized by p'' on the roots (second column in the table) would suffice to discriminate one root from another. An extreme case is a high degree polynomial which only has a single root – since there is no need to distinguish between different roots, no additional information at all is necessary.

The next corollary first appears in [CR88] and is another consequence of Thom's Lemma 3.1.4. In particular, it enables us to *order* the real roots of a polynomial given the sign conditions realized by the derivatives. Later in Section 3.2.5, we will use it in a more general context.

Lemma 3.1.7 (Comparison lemma, [CR88]). *Let $p \in \mathbb{R}[X] \setminus \{0\}$ and let $\xi_1, \xi_2 \in \mathbb{R}$ be such that $\sigma_1 := sgn(Der(p), \xi_1) \neq sgn(Der(p), \xi_2) =: \sigma_2$. Furthermore, let $k \in \mathbb{N}$ be maximal such that $\sigma_1(p^{(k)}) \neq \sigma_2(p^{(k)})$. The following statements all hold:*

1. $\sigma_1(p^{(k+1)}) = \sigma_2(p^{(k+1)}) \neq 0$
2. $\sigma_1(p^{(k+1)}) = \sigma_2(p^{(k+1)}) = 1$ *implies that*
$$\xi_1 > \xi_2 \Longleftrightarrow \sigma_1(p^{(k)}) > \sigma_2(p^{(k)})$$
3. $\sigma_1(p^{(k+1)}) = \sigma_2(p^{(k+1)}) = -1$ *implies that*
$$\xi_1 > \xi_2 \Longleftrightarrow \sigma_1(p^{(k)}) < \sigma_2(p^{(k)})$$

Example 3.1.8. *Consider once again the polynomial $p = X^3 - 2X$ from Example 3.1.6. We can apply Corollary 3.1.7 to prove that that $-\sqrt{2} < 0$: The greatest $k \in \mathbb{N}$ such that $p^{(k)}(-\sqrt{2}) \neq p^{(k)}(0)$ is 2. Note that for both $-\sqrt{2}$ and 0, the third derivative of p is positive in accordance to item 1 of the corollary. For the same reason, we apply item 2 and conclude, since $sgn(p''(-\sqrt{2})) = -1 < 0 = sgn(p''(0))$, that $-\sqrt{2} < 0$.*

In Example 3.1.6, when we were just interested in distinguishing between the roots of p, it was not necessary to know the signs realized on p'''. The preceding example however shows that in order to *compare* these roots, this information is crucial. On the other hand, the knowledge about the signs realized on p' was also redundant for the comparison.

With these considerations in mind, we are now ready to define Thom encodings formally.

Definition 3.1.9 (Thom encoding)**.** *Let $p \in \mathbb{R}[X] \setminus \{0\}$, $k \in \mathbb{N}, k \leq deg(p)$ and σ be a sign condition on $Der(p^{(k)})$. The tuple $\tau = (p, \sigma)$ is called a* Thom encoding *of $\xi \in \mathbb{R}$ if and only if*

1. *ξ is a root of p,*
2. *$Der(p^{(k)})$ realizes σ on ξ,*
3. *for all roots ξ' of p, $\xi' \neq \xi$, it holds that $sgn(Der(p^{(k)}), \xi') \neq \sigma$.*

We refer to the number ξ as the decoding $\langle\tau\rangle$ *of τ. Furthermore, we call p the* defining polynomial *of τ.*

Our definition ensures that a Thom encoding $\tau = (p, \sigma)$ of ξ carries all information needed to compare ξ to any other root of p which is also given in terms of a Thom encoding.

In the definition we did no require p to have only rational coefficients in order to keep the definition more general. In practice however, when we implement Thom encodings as a representation for real *algebraic* numbers, only polynomials with pure rational coefficients make sense. The signs realized on the last derivative $p^{(deg(p))}$ of p can easily be deduced from the sign of the leading coefficient of p and thus would not have to be stored explicitly. In order to keep our presentation simple, we abstract from such implementational details throughout this chapter. In Section 5.1, we describe some aspects of a realistic implementation of Thom encodings.

Example 3.1.10. *Once again, let $p = X^3 - 2X$. Let $\tau = (p, \sigma)$ where σ is the sign condition on $Der(p'')$ given by*
$$\sigma = p'' > 0 \wedge p''' > 0.$$
Then τ is a Thom encoding with $\langle\tau\rangle = \sqrt{2}$.

From the definition of real algebraic numbers it is immediately clear that for each $\xi \in \mathbb{R}_{alg}$ there is a Thom encoding $\tau = (p, \sigma)$ such that $\langle \tau \rangle = \xi$ and p has only rational coefficients. However, the encoding τ is not uniquely defined by ξ because different polynomials can have ξ a as root – in fact, there are infinitely many such polynomials. It should also be noted that not all tuples (p, σ) are valid Thom encodings.

Example 3.1.11. *Let $p = X^2 - 2$ and let σ be the sign condition on $Der(p')$ defined as follows:*

$$\sigma = p' > 0 \wedge p'' > 0$$

Then $\tau = (p, \sigma)$ is also a Thom encoding of $\sqrt{2}$.

3.1.2 Triangular Thom encodings

In the lifting phase of the CAD, one must be able to substitute all but one of a polynomial's variables by real algebraic numbers represented as Thom encodings. If we would do this literally, we would need algorithms to add and multiply real algebraic numbers in the Thom representation. As explained in [CLM+92], this is possible in principal. However, the result would be a polynomial whose real algebraic coefficients again are only given in terms of Thom encodings. If we were then going to characterize the real roots of this polynomial, we would obtain Thom encodings which *depend* on the encodings defining the coefficients.

The idea of triangular Thom encodings is similar to this but it avoids the costly arithmetic operations. Before we can define them formally we need some more definitions.

Definition 3.1.12 (Zero-dimensional system, [BPR10] Section 4.5)**.** *Let $\mathcal{Z} \subset \mathbb{Q}[X_1,...,X_n]$ be finite. For a domain $D \in \{\mathbb{R}, \mathbb{C}\}$ we define the* zero set *of $\mathcal{Z}$ on D^n as*

$$Zeros(\mathcal{Z}, D^n) := \{(\xi_1,...,\xi_n) \in D^n : p(\xi_1,...,\xi_n) = 0 \text{ for all } p \in \mathcal{Z}\}.$$

The set $\mathcal{Z}$ is called a zero-dimensional system *if and only if $Zeros(\mathcal{Z}, \mathbb{C}^n)$ is finite.*

Usually, we just write $Zeros(\mathcal{Z})$ instead of $Zeros(\mathcal{Z}, \mathbb{R}^n)$, with the understanding that n is the number of different variables in the polynomials in $\mathcal{Z}$. In the univariate case, all finite subsets of $\mathbb{Q}[X]$ which do not consist only of the zero polynomial are zero-dimensional systems as the numbers of complex roots of a non-zero univariate polynomial is bounded by its degree. In the multivariate case, it is more difficult to decide whether a given set of polynomials is a zero-dimensional system. We will come back to this in Section 4.2.

Let $p \in \mathbb{R}[X_1,...,X_n]$ be a multivariate polynomial. Similar to our notation from above, we define

$$Der_{X_i}(p) := \{p^{(j)}_{X_i}, 0 \leq j \leq deg_{X_i}(p)\},$$

where $1 \leq i \leq n$ and again with the understanding that the 0-th derivative of p with respect to the variable X_i is p itself.

Definition 3.1.13 (Triangular Thom encoding, cf. [BPR10] Definition 11.5)**.** *Let $n \in \mathbb{N}_+$ and $\mathcal{P}$ be a zero-dimensional system with exactly n polynomials which have the following form:*

$$\begin{gathered} p_n(X_1,...,X_n) \in \mathbb{Q}[X_1,...,X_n] \\ \vdots \\ p_2(X_1,X_2) \in \mathbb{Q}[X_1,X_2] \\ p_1(X_1) \in \mathbb{Q}[X_1] \end{gathered}$$

For some $\mathbf{k} = (k_1,\dots,k_n) \in \mathbb{N}^n$*, we let* $Der(\mathcal{P}^{\mathbf{k}}) := \bigcup_{i=1}^n Der_{X_i}(p_i^{(k_i)})$ *and* Σ *be a sign condition on* $Der(\mathcal{P}^{\mathbf{k}})$*. The tuple* $T = (\mathcal{P}, \Sigma)$ *is called a* triangular Thom encoding of $(\xi_1,\dots,\xi_n) \in \mathbb{R}^n_{alg}$ *if and only if it holds that*

$$\begin{aligned} \tau_1 = \big(p_1(X_1), \sigma_1\big) &\quad \textit{is a Thom encoding of } \xi_1, \\ \tau_2 = \big(p_2(\xi_1,X_2), \sigma_2(\xi_1)\big) &\quad \textit{is a Thom encoding of } \xi_2, \\ &\vdots \\ \tau_n = \big(p_n(\xi_1,\dots,\xi_{n-1},X_n), \sigma_n(\xi_1,\dots,\xi_{n-1})\big) &\quad \textit{is a Thom encoding of } \xi_n, \end{aligned}$$

where

$$\sigma_1 := \Sigma\big|_{Der_{X_1}(p_1^{(k_1)})}$$

and for $1 < i \leq n$*,* $\sigma_i(\xi_1,\dots,\xi_{i-1})$ *is obtained by taking*

$$\sigma_i := \Sigma\big|_{Der_{X_i}(p_i^{(k_i)})}$$

and substituting all appearances of $X_1,\dots,X_{i-1}$ *in the polynomials of the domain of* σ_i *by their corresponding values* $\xi_1,\dots,\xi_{i-1}$*.*

We denote the n-dimensional point $(\xi_1,..,\xi_n) \in \mathbb{R}^n_{alg}$ by $\langle T \rangle$. The order of the numbers in $(\xi_1,..,\xi_n)$ is uniquely determined by the order on the variables. Given a triangular Thom encoding T, we refer to the set $Der(\mathcal{P}^{\mathbf{k}})$ as $Der(T)$.

Also note that for $n = 1$, a triangular Thom encoding is just a usual Thom encoding. The reason why we require $\mathcal{P}$ to be a zero-dimensional system is that the algorithms described in Chapter 4 only work if this conditions holds.

Example 3.1.14. *Let*

$$q := X^2 + Y^2 - 2 \in \mathbb{Q}[X,Y]$$

and consider the Thom encoding $\tau = (p, \sigma)$ *of* $\sqrt{2}$ *from Example 3.1.10. Substituting* X *for* $\sqrt{2}$ *in* q *yields* $\tilde{q} = -Y^2$*. Since the only real root of* $\tilde{q}$ *is* 0*,* $(\tilde{q},\ \tilde{q}'' > 0)$ *is a Thom encoding of* 0*. Now let* $\mathcal{P} := \{p,q\}$*, and* $\Sigma := \sigma \wedge q''_Y > 0$*. Then* $T = (\mathcal{P}, \Sigma)$ *is a triangular Thom encoding with* $\langle T \rangle = (\sqrt{2}, 0)$*, which is also illustrated in Figure 3.2. In this case,* $Der(T)$ *contains the derivatives* p'', p''' *and* q''_Y*.*

3.2 Algorithms

This section is dedicated to the algorithms necessary to perform the operations listed in Definition 2.5.3 on real algebraic numbers given in terms of (triangular) Thom encodings which we introduced in the preceding section. In [BPR10], Chapter 11, all required principles are outlined. However, the author's view on the lifting procedure is different than ours: For example, instead of computing an intermediate point between two numbers given by Thom encodings, the authors propose a method for computing *all* intermediate points that lie between the zeros of the two defining polynomials *at once*. This contradicts our notion of an *incremental* lifting process where we are interested in performing only 'local' operations. For this reason, we describe adapted algorithms which better fit to our needs. We also make some considerations on how the operations can be implemented more efficiently in the case of real numbers.

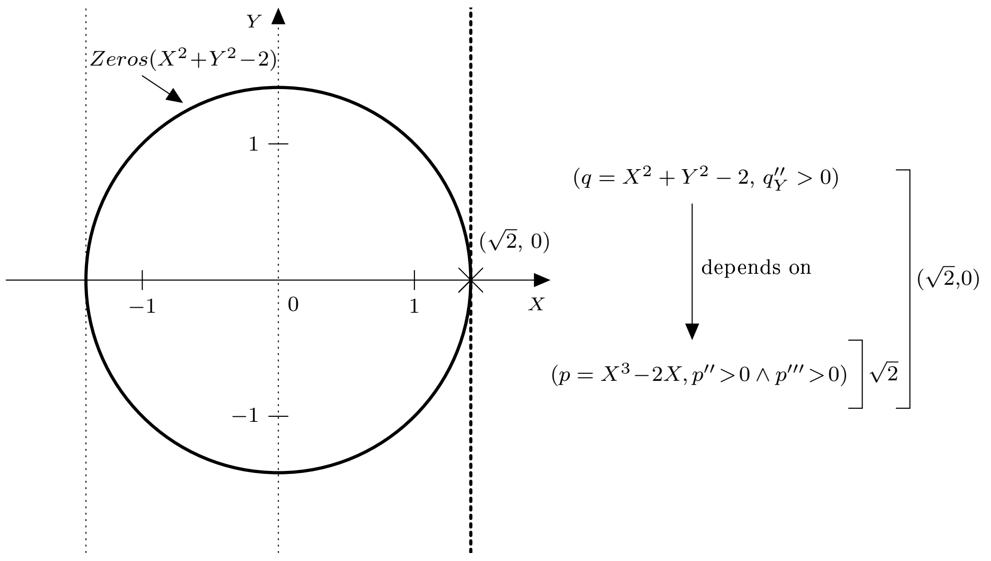


Figure 3.2: (left) Lifting the polynomial $q = X^2 + Y^2 - 2$ (whose zero set forms the solid circle) on the sample point $\sqrt{2}$ given as a Thom encoding with defining polynomial $p = X^3-2X$. The dotted lines depict the set $Zeros(p, \mathbb{R}^2)$. The thick dotted line corresponds to the root $\sqrt{2}$ of p. The cross is the point $(\sqrt{2}, 0)$. (right) Illustration of the triangular Thom encoding from Example 3.1.14 showing the dependency between the pairs of polynomials and sign conditions.

3.2.1 Sign determination black box

For the following discussion let us assume that we have an implementation of the sign determination algorithm specified as follows at hand. In Chapter 4 we describe a possible implementation more in detail. Let $sgn(\mathcal{P}, S) := \{sgn(\mathcal{P}, \xi) : \xi \in S\}$ for a finite $S \subset \mathbb{R}^n$.

Blackbox SIGNDETERMINATION

Input: A finite $\mathcal{P} \subset \mathbb{Q}[X_1,...,X_n]$ and a zero-dimensional system $\mathcal{Z} \subset \mathbb{Q}[X_1,...,X_n]$
Output: The set $\Sigma = sgn\big(\mathcal{P}, Zeros(\mathcal{Z})\big)$ of sign conditions on $\mathcal{P}$

In Chapter 4, we state some results about the complexity of sign determination. For now, let us suppose that invoking the sign determination black box is rather expensive and should therefore be used as few times as possible. At this point, the only thing that is important to understand is that the sign determination algorithm works incrementally: Given $\mathcal{P} = \{p_1,...,p_k\} \subset \mathbb{Q}[X_1,...,X_n]$ and a zero-dimensional system $\mathcal{Z} \subset \mathbb{Q}[X_1,...,X_n]$ as inputs, it computes the series

$$sgn\big(\{p_1\}, Zeros(\mathcal{Z})\big),\ ...,\ sgn\big(\{p_1,...,p_k\}, Zeros(\mathcal{Z})\big)$$

of sets of sign conditions. It can also be halted after each step in order to continue the computation later at a given time. In other words, it is not more expensive to compute first, $sgn\big(\{p_1,...,p_{i-1}\}, Zeros(\mathcal{Z})\big)$ and then $sgn\big(\{p_1,...,p_{i-1}, p_i\}, Zeros(\mathcal{Z})\big)$ instead of computing the latter quantity directly.

We are also going to assume that we can determine the number $|Zeros(\mathcal{Z})|$ for a given zero dimensional system $\mathcal{Z}$. In Section 2.6, we have already seen how to implement this in the case where $\mathcal{Z} = \{p\}$ for some $p \in \mathbb{Q}[X] \setminus \{0\}$. We will see in the next Chapter how this number can be computed in the multivariate case. We do not have to regard this as an additional effort as the sign determination algorithm will compute this number anyway.

3.2.2 Practical aspects

In our pseudo code descriptions, we assume that there is some data structure for representing rational polynomials available which is able to perform basic operations like multiplication or computing derivatives. Furthermore, we do not pay attention to numeric limits on rational coefficients. We also abstract from concrete data structures implementing containers and always use (formal) sets or mappings and the corresponding notation instead. For more details on how a real implementation might look like, see Section 5.1.

3.2.3 Root finding

With the sign determination algorithm at hand, the problem of finding Thom encodings for the real roots of a non-zero univariate polynomial becomes an easy task.

Algorithm 1 Real roots as Thom encodings

```
    Input: $p \in \mathbb{Q}[X] \setminus \{0\}$
    Output: The set $\mathcal{R}$ of all real roots of $p$ represented as Thom encodings
 1: procedure RealRoots($p$)
 2:     $\mathcal{R} \leftarrow \emptyset$
 3:     $\mathcal{D} \leftarrow \emptyset$
 4:     $k \leftarrow deg(p)$
 5:     while $|\mathcal{R}| < |Zeros(\{p\})|$ do
 6:         $\mathcal{D} \leftarrow \mathcal{D} \cup \{p^{(k)}\}$
 7:         $\Sigma \leftarrow$ SignDetermination($\mathcal{D}$, $\{p\}$)
 8:         $\mathcal{R} \leftarrow \{(p, \sigma) : \sigma \in \Sigma\}$
 9:         $k \leftarrow k - 1$
10:     end while
11:     return $\mathcal{R}$
12: end procedure
```

Algorithm 1 terminates because after at most $deg(p)$ steps, it holds that $|\mathcal{R}| = |Zeros(\{p\})|$ by Corollary 3.1.5. It is designed so that the set $Der(p^{(k)})$ is as small as possible, while still generating Thom encodings consistent with Definition 3.1.9. This is achieved by successively computing the sets

$$sgn\big(Der(p^{(deg(p))}),\ Zeros(\{p\})\big),\ sgn\big(Der(p^{(deg(p)-1)}), Zeros(\{p\})\big),\dots$$

until the number of sign conditions equals $|Zeros(\{p\})|$.

Example 3.2.1. *We illustrate the behavior of Algorithm 1 on the input polynomial $p = X^3 - 2X$ with $|Zeros(\{p\})| = 3$.*

$\mathcal{R}$	$\mathcal{D}$	Σ
$\emptyset$	$\emptyset$	-
$\{(p, p''' > 0)\}$	$\{p''' = 6\}$	$\{p''' > 0\}$
$\{(p, p'' = 0 \wedge p''' > 0),$ $(p, p'' > 0 \wedge p''' > 0),$ $(p, p'' < 0 \wedge p''' > 0)\}$	$\{p'' = 6X, p''' = 6\}$	$\{(p'' = 0 \wedge p''' > 0),$ $(p'' > 0 \wedge p''' > 0),$ $(p'' < 0 \wedge p''' > 0)\}$

The first row correspond to the state after the initialization, the second and the third row are the states achieved after the first and the second iteration, respectively.

After the second iteration we know that we are done because the number of different sign conditions in Σ equals $|Zeros(\{p\})| = 3$. Compare the result to Figure 3.1.

Next recall what happens when we want to lift a sample point: During the lifting phase of the CAD algorithm, sample points $(\xi_1,...,\xi_n) \in \mathbb{R}^n$ of level n have to be plugged into polynomials $p(X_1,...,X_n,X)$ of level $n+1$. In theory, this yields univariate polynomials $\tilde{p}(X) := p(\xi_1,...,\xi_n,X)$ whose roots then have to be specified in order to compute the sample points on level $n + 1$. In our context, a sample point $(\xi_1,...,\xi_n)$ on level n is given as a (triangular) Thom encoding and our task is to find new triangular Thom encodings representing the set of points $(\xi_1,...,\xi_n,\xi)$, where the ξ are the real roots of $\tilde{p}(X)$.

Algorithm 2 Lifting a sample point

```
    Input: $p(X_1,...,X_n,X) \in \mathbb{Q}[X_1,..,X_n,X]$ and a triangular Thom encoding $T = (\mathcal{P}, \Sigma)$, $\langle T \rangle = (\xi_1,...,\xi_n)$, such that $\mathcal{P} \cup \{p\}$ is a zero dimensional system
    Output: For each real root $\xi$ of $p(\xi_1,...,\xi_n, X)$ a triangular Thom encoding $\tilde{T}$, such that $\langle \tilde{T} \rangle = (\xi_1,...,\xi_n,\xi)$
 1: procedure RealRoots($p$, $T$)
 2:     $\mathcal{R} \leftarrow \emptyset$
 3:     $\mathcal{D}_T \leftarrow Der(T)$
 4:     $\mathcal{D} \leftarrow \emptyset$
 5:     $k \leftarrow deg_X(p)$
 6:     while $|\mathcal{R}| < |Zeros(\mathcal{P} \cup \{p\})|$ do
 7:         $\mathcal{D} \leftarrow \mathcal{D} \cup \{p_X^{(k)}\}$
 8:         $\tilde{\Sigma} \leftarrow$ SignDetermination($\mathcal{D}_T \cup \mathcal{D}, \mathcal{P} \cup \{p\}$)
 9:         $\tilde{\Sigma} \leftarrow \{\tilde{\sigma} \in \tilde{\Sigma} : \tilde{\sigma}|_{\mathcal{D}_T} = \Sigma\}$
10:         $\mathcal{R} \leftarrow \{(\mathcal{P} \cup \{p\}, \tilde{\sigma}) : \tilde{\sigma} \in \tilde{\Sigma}\}$
11:         $k \leftarrow k - 1$
12:     end while
13:     return $\mathcal{R}$
14: end procedure
```

Algorithm 2 is very similar to Algorithm 1. Actually, it can be seen as a generalization because for $n = 0$, $\mathcal{P} = \emptyset$ and Σ the empty map, Algorithm 2 becomes exactly the root finding algorithm for the univariate case. Using the sign determination black box, we characterize the points $(\xi_1,...,\xi_n, \xi)$ in $Zeros(\mathcal{P} \cup \{p\})$ by means of the sign conditions they achieve at the derivatives $\mathcal{D}_T \cup \mathcal{D}$, where $\mathcal{D} = Der_X(p^{(k)})$ and k is the greatest possible. This is achieved by the while loop, just in the same way as in Algorithm 1. Then, from the so obtained set of sign conditions, we select only those that extend the sign condition Σ defining the input point $\langle T \rangle = (\xi_1,...,\xi_n)$ (line 9).

In the input specification of Algorithm 2, we required $\mathcal{P} \cup \{p\}$ to be zero-dimensional. A necessary condition for this is that p does not vanish on the point $\langle T \rangle$. In Section 3.2.4, we describe how this can be checked. When this case occurs, we return 0 as a representative of the roots of the zero polynomial. However, there are also cases in which p does *not* vanish on $\langle T \rangle$ but $\mathcal{P} \cup \{p\}$ is *no* zero-dimensional system anyway. Whenever this happens, the sign determination procedure will notice and reject the input. We will regard this trouble as an *open problem* within the scope of this thesis. In our experiments however, we observed that is does not occur very often. We give more details about the experiments in Section 5.2.

3.2.4 Sign of a polynomial on a Thom encoding

The first algorithm we present in this section determines the sign realized by a polynomial on a point given as a triangular Thom encoding. It corresponds to the 'evaluation' operation from Definition 2.5.3.

Algorithm 3 Sign of a polynomial on a Thom encoding

```
   Input: $p(X_1,...,X_n) \in \mathbb{Q}[X_1,..,X_n]$ and a triangular Thom encoding $T = (\mathcal{P}, \Sigma)$
   with $\langle T \rangle = (\xi_1,...,\xi_n)$
   Output: $sgn(p(\xi_1,...,\xi_n))$
1: procedure SIGN($p$, $T$)
2:     $\tilde{\Sigma} \leftarrow$ SIGNDETERMINATION($Der(T) \cup \{p\}$, $\mathcal{P}$)
3:     $\tilde{\sigma} \leftarrow$ only element in $\tilde{\Sigma}$ such that $\tilde{\sigma}\big|_{Der(T)} = \Sigma$
4:     return $\tilde{\sigma}(p)$
5: end procedure
```

In line 3, there is only one sign condition $\tilde{\sigma}$ in $\tilde{\Sigma}$ with this property because otherwise there were two different $\xi_1, \xi_2 \in Zeros(\mathcal{P})$ with $sgn(Der(T), \xi_1) = sgn(Der(T), \xi_2) = \Sigma$, but this is impossible by the definition of triangular Thom encodings 3.1.13.

Using Algorithm 3, we can also implement a procedure which checks whether $p \in \mathbb{Q}[X_1,...,X_n,X]$ vanishes on $\langle T \rangle \in \mathbb{R}^n$ for a triangular Thom encoding T.

Algorithm 4 Vanish on Thom encoding

```
   Input: $p(X_1,...,X_n,X) \in \mathbb{Q}[X_1,..,X_n]$ and a triangular Thom encoding $T =$
   $(\mathcal{P}, \Sigma)$ with $\langle T \rangle = (\xi_1,...,\xi_n)$
   Output: true if $p(\xi_1,...,\xi_n, X) = 0$, otherwise false
1: procedure VANISHES($p$, $T$)
2:     $\mathcal{C} \leftarrow$ coefficients of $p$ viewed as a polynomial in the domain $\mathbb{Q}[X_1,...,X_n][X]$
3:     for all $c \in \mathcal{C}$ do
4:         if SIGN($c$, $T$) $\neq 0$ then
5:             return false
6:         end if
7:     end for
8:     return true
9: end procedure
```

If we view p in the domain $\mathbb{Q}[X_1,...,X_n][X]$, then it has the form

$$p_d(X_1,...,X_n)X^d + p_{d-1}(X_1,...,X_n)X^{d-1} + ... + p_0(X_1,...,X_n),$$

for some $d \in \mathbb{N}$ and polynomials $p_i \in \mathbb{Q}[X_1,...,X_n]$ for $0 \leq i \leq d$. They correspond to the set $\mathcal{C}$ in Algorithm 4. Thus if $p(\xi_1,...,\xi_n, X) = 0$ for a point $(\xi_1,...,\xi_n) \in \mathbb{R}^n$, then is must hold that $p_i(\xi_1,...,\xi_n) = 0$ for all $0 \leq i \leq d$.

Algorithms 4 and 2 together can be used to implement the 'lifting' operation from Definition 2.5.3 (except for the special case where we have no zero-dimensional system).

3.2.5 Comparison

The first algorithm we present here is the implementation of the comparison lemma 3.1.7.

Algorithm 5 Comparison of sign conditions with respect to the same set $Der(p)$

```
    Input: For a polynomial $p \in \mathbb{R}[X] \setminus \{0\}$ and two numbers $\xi_1, \xi_2 \in \mathbb{R}$ the sign
    conditions $\sigma_1 = sgn(Der(p), \xi_1)$ and $\sigma_2 = sgn(Der(p), \xi_2)$, such that $\sigma_1 \neq \sigma_2$
    Output: less, if $\xi_1 < \xi_2$, greater if $\xi_1 > \xi_2$
 1: procedure SignCompare($\sigma_1$, $\sigma_2$)
 2:     for $k \leftarrow deg(p) - 1, ..., 0$ do
 3:         if $\sigma_1(p^{(k)}) \neq \sigma_2(p^{(k)})$ then
 4:             if $\sigma_1(p^{(k+1)}) = 1$ then
 5:                 if $\sigma_1(p^{(k)}) < \sigma_2(p^{(k)})$ then return less
 6:                 else return greater
 7:             else
 8:                 if $\sigma_1(p^{(k)}) < \sigma_2(p^{(k)})$ then return greater
 9:                 else return less
10:             end if
11:         end if
12:     end for
13: end procedure
```

In order to find the maximal k such that $\sigma_1(p^{(k)}) \neq \sigma_2(p^{(k)})$, we iterate over the signs the derivatives of p achieve at ξ_1 and ξ_2 from back to front, starting with the second last derivative as the last one is known to be constant. In the else case in line 7, it holds that $\sigma_1(p^{(k+1)}) = -1$ because $\sigma_1(p^{(k+1)}) = 0$ is impossible by Corollary 3.1.7.

We can now apply Algorithm 5 in order to compare Thom encodings whose defining polynomials are *distinct*. We present this as the following Algorithm 6 which is an adaptation of Algorithm 10.16 in [BPR10].

The basic idea of Algorithm 6 is to compute the sign condition $\tilde{\sigma}_1$ which $\langle\tau_1\rangle$ realizes on $Der(p_2^{(k_2)})$ and then compare this to σ_2 using the comparison lemma 3.1.7. Since k_2 may be greater than 0, we sometimes have to additionally extend the sign condition σ_2 (line 11). On the other hand, it is often enough to know the sign condition realized by $Der(p_2^{(k_2')})$ on $\langle\tau_1\rangle$ for some $k_2' > k_2$. As soon as we have computed enough signs to distinguish $\langle\tau_1\rangle$ and $\langle\tau_2\rangle$, we can apply the comparison lemma (line 15). If we go through the complete for-loop and it never holds that $\tilde{\sigma}_1 = \tilde{\sigma}_2$, then we have proven that $Der(p_2)$ realizes exactly the same sign conditions on $\langle\tau_1\rangle$ and $\langle\tau_2\rangle$ and thus, the encoded numbers must be equal (line 17).

This principle can also be generalized to the multivariate case (Algorithm 10 in the appendix).

Algorithm 6 Comparison of Thom encodings

```
    Input: Thom encodings $\tau_1 = (p_1, \sigma_1)$, $\tau_2 = (p_2, \sigma_2)$ such that $\sigma_1$ is a sign condition on $Der(p_1^{(k_1)})$ and $\sigma_2$ is a sign condition on $Der(p_2^{(k_2)})$
    Output: less if $\langle\tau_1\rangle < \langle\tau_2\rangle$, equal if $\langle\tau_1\rangle = \langle\tau_2\rangle$ and greater if $\langle\tau_1\rangle > \langle\tau_2\rangle$
 1: procedure ThomCompare($\tau_1$, $\tau_2$)
 2:     if $p_1 = p_2$ then return SignCompare($\sigma_1$, $\sigma_2$)
 3:     $\mathcal{D}_1 \leftarrow Der^{(k_1)}(p_1)$
 4:     $\mathcal{D}_2 \leftarrow \emptyset$
 5:     for $i \leftarrow deg(p_2), i \geq 0, i \leftarrow i - 1$ do
 6:         $\mathcal{D}_2 \leftarrow \{p_2^{(i)}\} \cup \mathcal{D}_2$
 7:         $\Sigma \leftarrow$ SignDetermination($\mathcal{D}_1 \cup \mathcal{D}_2$, $\{p_1\}$)
 8:         $\tilde{\sigma}_1 \leftarrow$ only element in $\Sigma$ such that $\tilde{\sigma}_1|_{\mathcal{D}_1} = \sigma_1$
 9:         $\tilde{\sigma}_1 \leftarrow \tilde{\sigma}_1|_{\mathcal{D}_2}$
10:         $\tilde{\sigma}_2 \leftarrow \sigma_2|_{\mathcal{D}_2}$
11:         if $k_2 < |\mathcal{D}_2|$ then
12:             $\Sigma \leftarrow$ SignDetermination($\mathcal{D}_2$, $\{p_2\}$)
13:             $\tilde{\sigma}_2 \leftarrow$ only element in $\Sigma$ such that $\tilde{\sigma}_2|_{Der(p_2^{(k_2)})} = \sigma_2$
14:         end if
15:         if $\tilde{\sigma}_1 \neq \tilde{\sigma}_2$ then return SignCompare($\tilde{\sigma}_1$, $\tilde{\sigma}_2$)
16:     end for
17:     return equal
18: end procedure
```

Example 3.2.2. *Consider the Thom encoding*

$$\tau_1 = (p_1 = X^3 - 2X,\ p_1'' = 0 \wedge p_1''' > 0)$$

of 0 *and*

$$\tau_2 = (p_2 = X^3 - 2,\ p_2''' > 0)$$

of $\sqrt[3]{2}$*. Suppose we call* ThomCompare(τ_1, τ_2)*.* $\mathcal{D}_1$ *is initialized with* $\{p_1'', p_1'''\}$ *and* $\mathcal{D}_2$ *with* $\emptyset$*. The table illustrates the subsequent behavior of the algorithm:*

$\mathcal{D}_2$	Σ	$\tilde{\sigma}_1$	$\tilde{\sigma}_2$
$\{p_2'''\}$	$\{p_1'' = 0 \wedge p_1''' > 0 \wedge \mathbf{p_2''' > 0}$, $p_1'' > 0 \wedge p_1''' > 0 \wedge p_2''' > 0$, $p_1'' < 0 \wedge p_1''' > 0 \wedge p_2''' > 0\}$	$p_2''' > 0$	$p_2''' > 0$
$\{p_2'', p_2'''\}$	$\{p_1'' = 0 \wedge p_1''' > 0 \wedge \mathbf{p_2'' = 0 \wedge p_2''' > 0}$, $p_1'' > 0 \wedge p_1''' > 0 \wedge p_2'' > 0 \wedge p_2''' > 0$, $p_1'' < 0 \wedge p_1''' > 0 \wedge p_2'' < 0 \wedge p_2''' > 0\}$	$p_2'' = 0 \wedge p_2''' > 0$	$p_2'' > 0 \wedge p_2''' > 0$

The parts of Σ *marked in bold face correspond to the sign condition* $\tilde{\sigma}_1$*, which is extracted from* Σ *in lines 8 and 9. After the second iteration, it holds that* $\tilde{\sigma}_1 \neq \tilde{\sigma}_2$*, so we compare these sign conditions using Algorithm 5 which yields the result* less.

3.2.6 Intermediate points

The exact specification of the algorithms computing intermediate points in the univariate case is as follows:

Specification Intermediate point

Input: Thom encodings $\tau_1 = (p_1, \sigma_1)$, $\tau_2 = (p_2, \sigma_2)$ with $\langle \tau_1 \rangle < \langle \tau_2 \rangle$
Output: A Thom encoding $\tau = (p, \sigma)$ such that $\langle \tau_1 \rangle < \langle \tau \rangle < \langle \tau_2 \rangle$

There are several approaches for computing intermediate points. The first one that we are going to present is derived from Algorithm 10.18 in [BPR10] and relies on Rolle's Theorem.

Lemma 3.2.3 (Rolle's Theorem, see for example [Mis93] Theorem 8.2.10)**.** *Let $p \in \mathbb{R}[X]$ and $\xi_1, \xi_2 \in \mathbb{R}$ be both roots of p with $\xi_1 < \xi_2$. Then the derivative p' has a root $\xi \in \mathbb{R}$ with $\xi_1 < \xi < \xi_2$.*

We are going to apply Rolle's Theorem is the following way: Since the set of real roots of p_1p_2 is equal to the union of the sets of real roots of p_1 and p_2, the polynomial $(p_1p_2)'$ has at least one real root in any of the intervals $]\xi_1,\xi_2[$, where ξ_1 and ξ_2 are roots of p_1p_2. This consideration yields the following Algorithm 7.

Algorithm 7 Intermediate point using Rolle's Theorem

```
1: procedure IntermediatePoint($\tau_1$, $\tau_2$)
2:     $p \leftarrow (p_1p_2)'$
3:     $\mathcal{R} \leftarrow$ RealRoots($p$)
4:     for all $\tau$ in $\mathcal{R}$ do
5:         if ThomCompare($\tau_1, \tau$) = less and ThomCompare($\tau, \tau_2$) = less then
6:             return $\tau$
7:         end if
8:     end for
9: end procedure
```

The downsides of Algorithm 7 are that the degree of the polynomial representing the new point will be much higher than that of the input polynomials (in fact, $deg(p_1) + deg(p_2) - 1$) and that it requires many comparison operations which are quite expensive. However, it is then 'clean' algebraic approach and can also be used in the non-archimedean case.

We therefore present two other 'seminumerical' approaches for computing intermediate points which we expect to perform better in the real case. They rely on the observation that adding rational constants to real algebraic numbers already present in form of Thom encodings can be done efficiently.

Lemma 3.2.4. *Let $(p(X), \sigma)$ be a Thom encoding of $\xi \in \mathbb{R}_{alg}$ and let $\epsilon \in \mathbb{Q}$. Then*

$$\big(p(X-\epsilon), \sigma(X-\epsilon)\big) \text{ is a Thom encoding of } \xi + \epsilon,$$

with the understanding that $p(X-\epsilon)$ is obtained by substituting X in p for $X - \epsilon$ and similarly, $\sigma(X-\epsilon)$ is the sign condition which results from replacing all occurrences of X in the polynomials in the domain of σ by $X - \epsilon$.

Proof. If ξ is a root of $p(X)$ then clearly $\xi + \epsilon$ is a root of $p(X-\epsilon)$. Noticing that $(p(X-\epsilon))' = p'(X-\epsilon)$ completes the proof. □

The same technique is also used in [CLM+92]. Note however, that substituting X for $X - \epsilon$ in a 'nice' polynomial, where the number of terms is small compared to the degree, destroys this niceness.

Example 3.2.5. *Consider $p = X^5 - 2 \in \mathbb{Q}[X]$. The only real root of p is $\sqrt[5]{2}$, such that $(p, p^{(5)} > 0)$ is a Thom encoding of $\sqrt[5]{2}$. If $q = (X-1)^5 - 2 = X^5 - 5x^4 + 10x^3 - 10x^2 + 5x - 3$, then $\langle (q, q^{(5)} > 0) \rangle = \sqrt[5]{2} + 1$. The number of terms in q increased significantly compared to p.*

The next ingredient is a lower bound on the minimal distance between two real roots of a polynomial.[1]

Lemma 3.2.6 (Rumps' bound, [Rum79])**.** *Let $p \in \mathbb{Z}[X] \setminus \{0\}$ and $d = deg(p)$. By $\|p\|_1$ we denote the sum of the absolute values of the coefficients of p. Let $\xi_1, \xi_2 \in \mathbb{R}$ be different roots of p. Then*

$$|\xi_1 - \xi_2| > \frac{2\sqrt{2}}{d^{\frac{d}{2}+1}(\|p\|_1 + 1)^d}.$$

Rump's bound exhibits an exponential influence of the degree on the size of the denominator, but it only requires basic arithmetic operations and thus is easy to implement. The bound can also be applied to rational polynomials by multiplying with a multiple of the denominators of the coefficients beforehand. When using Rump's bound in a program, we have to replace $\sqrt{2}$ by a rational approximation $0 < r < \sqrt{2}$, for example 1 or 7/5. For a polynomial $p \in \mathbb{Q}[X] \setminus \{0\}$, we will refer to the bound computation as RUMP(p).

A combination of the results from Lemma 3.2.4 and Lemma 3.2.6 yields the next algorithm 8. For the substitution, we select the polynomial with smallest degree in order to keep the degree of the output as small as possible.

Algorithm 8 Intermediate point using Rump's bound

```
 1: procedure IntermediatePoint($\tau_1$, $\tau_2$)
 2:     $p \leftarrow (p_1 p_2)'$
 3:     $\epsilon \leftarrow$ Rump($p$)
 4:     if $deg(p_1) \leq deg(p_2)$ then
 5:         $q \leftarrow p_1(X - \epsilon)$
 6:         return $(q, \sigma_1(X - \epsilon))$
 7:     else
 8:         $q \leftarrow p_2(X + \epsilon)$
 9:         return $(q, \sigma_2(X + \epsilon))$
10:     end if
11: end procedure
```

Algorithm 8 cannot be generalized to the multivariate case so easily because the bound only works for univariate polynomials. We therefore present an even simpler approach which can also be used in the multivariate case (Algorithm 11 in the appendix).

[1]There exist tighter bounds, but the exponential influence of the degree on the size of the denominator seems inevitable [Col01].

Algorithm 9 Intermediate point using iterative approximation

```
1: procedure IntermediatePoint($\tau_1$, $\tau_2$)
2:     $\epsilon \leftarrow 1$
3:     if $deg(p_1) \leq deg(p_2)$ then
4:         $q \leftarrow p_1(X - \epsilon)$
5:         $\tau \leftarrow (q, \sigma_1(X - \epsilon))$
6:         while ThomCompare($\tau$, $\tau_2$) $\neq$ less do
7:             $\epsilon \leftarrow \epsilon/2$
8:             $q \leftarrow p_1(X - \epsilon)$
9:             $\tau \leftarrow (q, \sigma_1(X - \epsilon))$
10:        end while
11:        return $\tau$
12:    else
13:        $q \leftarrow p_2(X + \epsilon)$
14:        $\tau \leftarrow (q, \sigma_2(X + \epsilon)$
15:        while ThomCompare($\tau_1$, $\tau$) $\neq$ less do
16:            $\epsilon \leftarrow \epsilon/2$
17:            $q \leftarrow p_2(X + \epsilon)$
18:            $\tau \leftarrow (q, \sigma_2(X + \epsilon)$
19:        end while
20:        return $\tau$
21:    end if
22: end procedure
```

Similar to Algorithm 8, we pick the polynomial with minimal degree for the substitution. The initial value 1 for ϵ could also be another positive number. The same applies for the divisor in lines 7 and 16, but it must be greater than 1.

3.2.7 Points below and above

We can always use Lemma 3.2.4 in order to add or subtract a rational constant to a real algebraic number represented as a Thom encoding in order to obtain points below and above. In the univariate case, where $\xi = \langle\tau\rangle$ for a Thom encoding $\tau = (p, \sigma)$, we can alternatively compute the Cauchy bound $C(p)$ of p as defined in lemma 2.6.2. Then, $-C(p) < \xi < C(p)$, that means we can compute the sample points below and above as explicitly represented rational numbers.

Chapter 4

Sign determination

The aim of this chapter is to describe an implementation of the sign determination black box used in the algorithms in Chapter 3.

4.1 The sign determination algorithm

Throughout the whole section, let $\mathcal{Z} \subset \mathbb{R}[X_1,...,X_n]$ be a zero-dimensional system and let $p \in \mathbb{R}[X_1,...,X_n]$. *Tarski queries* will play a key role in the sign determination algorithm we are going to develop in this section.

4.1.1 Tarski queries

For a finite set $\mathcal{P} \subset \mathbb{R}[X_1,...,X_n]$ and a sign condition σ on $\mathcal{P}$ we define the notation

$$c(\sigma, \mathcal{Z}) := |\{\xi \in Zeros(\mathcal{Z}) : sgn(\mathcal{P}, \xi) = \sigma\}|.$$

Definition 4.1.1 (Tarski query)**.** *The* Tarski query of p on $\mathcal{Z}$ *is defined by*

$$TaQ(p,\mathcal{Z}) := c(p > 0, \mathcal{Z}) - c(p < 0, \mathcal{Z}).$$

Recall that the notations '$p > 0$', '$p < 0$' and '$p = 0$' stand for *sign conditions* like defined at the beginning of Chapter 3. In Section 4.2, we show how Tarski queries can actually be computed.

Example 4.1.2. *Let $p = X - 1$ and let $\mathcal{Z} = \{q\}$ with*

$$q = X^3 + X^2 - 4X - 4 = (X + 2)(X + 1)(X - 2).$$

We have $Zeros(\mathcal{Z}) = \{-2, \ -1, 2\}$. Then $c(p > 0, \mathcal{Z}) = 1$, because the only number $\xi \in Zeros(\mathcal{Z})$ such that $p(\xi) > 0$ is $\xi = 2$. On the other hand, $c(p < 0, \mathcal{Z}) = 2$ because there are two numbers in $Zeros(\mathcal{Z})$ on which p has negative sign. Thus, the Tarski query of p on $\mathcal{Z}$ is

$$TaQ(p,\mathcal{Z}) = c(p > 0, \mathcal{Z}) - c(p < 0, \mathcal{Z}) = 1 - 2 = -1.$$

Notice the following two properties of the Tarski query:

$$TaQ(1, \mathcal{Z}) = c(p = 0, \mathcal{Z}) + c(p > 0, \mathcal{Z}) + c(p < 0, \mathcal{Z}) \tag{4.1}$$

because the Tarski query of 1 on $\mathcal{Z}$ is just the total number of zeros in $\mathcal{Z}$. Furthermore, it holds that

$$TaQ(p^2, \mathcal{Z}) = c(p > 0, \mathcal{Z}) + c(p < 0, \mathcal{Z}) \tag{4.2}$$

because $p^2(\xi) > 0$ is true if and only if $p(\xi) > 0$ or $p(\xi) < 0$ and $p^2(\xi) < 0$ is never true. Together with Definition 4.1.1, Equations 4.1 and 4.2 lead to an interesting result.

Lemma 4.1.3. *It holds that*

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} c(p = 0, \mathcal{Z}) \\ c(p > 0, \mathcal{Z}) \\ c(p < 0, \mathcal{Z}) \end{pmatrix} = \begin{pmatrix} TaQ(1, \mathcal{Z}) \\ TaQ(p, \mathcal{Z}) \\ TaQ(p^2, \mathcal{Z}) \end{pmatrix}$$

and the matrix is invertible.

Lemma 4.1.3 can be used to implement the sign determination black box in the case where $|\mathcal{P}| = 1$.

Example 4.1.4. *Suppose that we computed the Tarski queries*

$$TaQ(1, \mathcal{Z}) = 3,\ TaQ(p, \mathcal{Z}) = 2 \text{ and } TaQ(p^2, \mathcal{Z}) = 2.$$

The inverse of the matrix in Lemma 4.1.3 is

$$\frac{1}{2} \begin{pmatrix} 2 & 0 & -2 \\ 0 & 1 & 1 \\ 0 & -1 & 1 \end{pmatrix}.$$

Thus, solving the equation system 4.1.3, we find that

$$c(p = 0, \mathcal{Z}) = 1,\ c(p > 0, \mathcal{Z}) = 2 \text{ and } c(p < 0, \mathcal{Z}) = 0.$$

The sign conditions realized by p on $Zeros(\mathcal{Z})$ are only $p = 0$ and $p > 0$, but not $p < 0$.

4.1.2 Naive sign determination

We are now going to show how the results from the previous section, especially Lemma 4.1.3, can be generalized in order to perform sign determination in the case where $|\mathcal{P}| > 1$. This will result in a 'naive' algorithm which is not applicable in practice. In the next Section 4.1.3, we will then discuss some results which improve this algorithm.

In this section, let $\mathcal{P} \subset \mathbb{R}[X_1,...,X_n]$ and let $\alpha \in \{0,1,2\}^{\mathcal{P}}$, that means α assigns each $p \in \mathcal{P}$ an integer between 0 and 2. This corresponds to the exponents of p which appear in the right-hand side of the equation in Lemma 4.1.3. Furthermore, let σ be a sign condition on $\mathcal{P}$. We introduce the following notations:

- $\mathcal{P}^\alpha := p_1^{\alpha(p_1)} \cdot ... \cdot p_k^{\alpha(p_k)} \in \mathbb{R}[X_1,...,X_n]$, and
- $\sigma^\alpha := \sigma(p_1)^{\alpha(p_1)} \cdot ... \cdot \sigma(p_k)^{\alpha(p_k)} \in \{-1,0,1\}$, with the convention that $0^0 = 1$.

The next lemma can be seen as a generalization of the Equations 4.1 and 4.2. We need it only as an intermediate result on our way to the sign determination algorithm. In [BPR10], it is not stated explicitly but we felt that it makes it easier to understand Theorem 4.1.9 which will be the main result of this section.

Lemma 4.1.5. *Let $\Sigma \subseteq \{-1,0,1\}^{\mathcal{P}}$ such that Σ contains all sign conditions realized by $\mathcal{P}$ on $Zeros(\mathcal{Z})$, that means $sgn\big(\mathcal{P}, Zeros(\mathcal{Z})\big) \subseteq \Sigma$. Then it holds that*

$$\sum_{\sigma \in \Sigma} \sigma^\alpha c(\sigma, \mathcal{Z}) = TaQ(\mathcal{P}^\alpha, \mathcal{Z}).$$

Equations 4.1 and 4.2 are simple applications of the preceding lemma. Before we give the proof, let us consider a more involved example.

Example 4.1.6. *Let $\mathcal{P} = \{p_1, p_2\}$ and let $\alpha \in \{0,1,2\}^{\mathcal{P}}$ be the mapping defined by $\alpha(p_1) = 1$ and $\alpha(p_2) = 2$. Furthermore, let us assume that $\mathcal{P}$ realizes precisely the following 4 sign conditions on $Zeros(\mathcal{Z})$:*

$$\begin{array}{ll} p_1 = 0 \wedge p_2 > 0, & p_1 > 0 \wedge p_2 = 0, \\ p_1 < 0 \wedge p_2 > 0, & p_1 < 0 \wedge p_2 < 0. \end{array}$$

Thus, applying Lemma 4.1.5, we compute

$$TaQ(p_1 p_2^2, \mathcal{Z}) = -c(p_1 < 0 \wedge p_2 > 0, \mathcal{Z}) - c(p_1 < 0 \wedge p_2 < 0, \mathcal{Z}).$$

This is not unexpected: Consider $c(p_1 p_2^2 > 0, \mathcal{Z})$. Since $p_1 p_2^2$ is positive whenever p_1 positive and p_2 is not zero, this quantity is equal to

$$c(p_1 > 0 \wedge p_2 > 0, \mathcal{Z}) + c(p_1 > 0 \wedge p_2 < 0, \mathcal{Z}).$$

The reason why these do not appear in the sum above is that we assumed that they are 0 anyway. Now let us consider $c(p_1 p_2^2 < 0, \mathcal{Z})$, which can be rewritten as

$$c(p_1 < 0 \wedge p_2 > 0, \mathcal{Z}) + c(p_1 < 0 \wedge p_2 < 0, \mathcal{Z}).$$

For the sign conditions $p_1 < 0 \wedge p_2 > 0$ and $p_1 < 0 \wedge p_2 < 0$ we have assumed that they are realized by $\mathcal{P}$ on $Zeros(\mathcal{Z})$. In the result produced by the Lemma, they appear with negative sign since $c(p_1 p_2^2 < 0, \mathcal{Z})$ also has negative sign in the Tarski query $TaQ(p_1 p_2^2, \mathcal{Z})$.

Proof of Lemma 4.1.5. From the assumption that $sgn\big(\mathcal{P}, Zeros(\mathcal{Z})\big) \subseteq \Sigma$ it follows that for all sign conditions σ' on $\mathcal{P}$ that are not in Σ it holds that $c(\sigma', \mathcal{Z}) = 0$. Thus

$$c(p_1^{\alpha(p_1)} \cdot \ldots \cdot p_k^{\alpha(p_k)} > 0, \mathcal{Z}) = \sum_{\sigma \in \Sigma} c(\sigma(p_1)^{\alpha(p_1)} \cdot \ldots \cdot \sigma(p_k)^{\alpha(p_k)} > 0, \mathcal{Z}) = \sum_{\sigma \in \Sigma, \sigma^\alpha = 1} c(\sigma, \mathcal{Z})$$

and similarly

$$c(p_1^{\alpha(p_1)} \cdot \ldots \cdot p_k^{\alpha(p_k)} < 0, \mathcal{Z}) = \sum_{\sigma \in \Sigma} c(\sigma(p_1)^{\alpha(p_1)} \cdot \ldots \cdot \sigma(p_k)^{\alpha(p_k)} < 0, \mathcal{Z}) = \sum_{\sigma \in \Sigma, \sigma^\alpha = -1} c(\sigma, \mathcal{Z}).$$

Combining these two equations we obtain

$$c(p_1^{\alpha(p_1)} \cdot \ldots \cdot p_k^{\alpha(p_k)} > 0, \mathcal{Z}) - c(p_1^{\alpha(p_1)} \cdot \ldots \cdot p_k^{\alpha(p_k)} < 0, \mathcal{Z}) = \sum_{\sigma \in \Sigma} \sigma^\alpha c(\sigma, \mathcal{Z}),$$

and the left-hand side is precisely the Tarski query $TaQ(\mathcal{P}^\alpha, \mathcal{Z})$. □

Notice that the precondition on Σ in Lemma 4.1.5 is trivially fulfilled if $\Sigma = \{-1,0,1\}^{\mathcal{P}}$. Now let $A \subseteq \{0,1,2\}^{\mathcal{P}}$ be a set of mappings from $\mathcal{P}$ to $\{0,1,2\}$ and $\Sigma \subseteq \{-1,0,1\}^{\mathcal{P}}$. In order to generalize Lemma 4.1.3, we first have to define orderings on A and Σ. For this purpose, we will suppose that there is already an ordering $<_{\mathcal{P}}$ on $\mathcal{P}$ available. In practice, when we apply the sign determination algorithm to a set $\mathcal{P} = Der(p)$, we usually take the ordering which is naturally given by $p < p' < \ldots < p^{(deg(p))}$.

Definition 4.1.7 ([BPR10], Section 10.3). *We define* lexicographic *orderings* $<_{lex}$ *on* $\{0,1,2\}^{\mathcal{P}}$ *and* $\{-1,0,1\}^{\mathcal{P}}$ *with respect to* $<_{\mathcal{P}}$ *as follows:*

1. *Let* $\alpha_1 \neq \alpha_2 \in \{0,1,2\}^{\mathcal{P}}$ *and* $p \in \mathcal{P}$ *be minimal with respect to* $<_{\mathcal{P}}$ *such that* $\alpha_1(p) \neq \alpha_2(p)$. *Then*
$$\alpha_1 <_{lex} \alpha_2 \iff \alpha_1(p) < \alpha_2(p).$$

2. *Similarly, let* $\sigma_1 \neq \sigma_2 \in \{-1,0,1\}^{\mathcal{P}}$ *and* $p \in \mathcal{P}$ *be minimal with respect to* $<_{\mathcal{P}}$ *such that* $\sigma_1(p) \neq \sigma_2(p)$. *Then*
$$\sigma_1 <_{lex} \sigma_2$$
$$\iff$$
$$(\sigma_1(p) = 0 \textit{ and } (\sigma_2(p) = 1 \textit{ or } \sigma_2(p) = -1)) \textit{ or } (\sigma_1(p) = 1 \textit{ and } \sigma_2(p) = -1).$$

In other words, the sign conditions are ordered lexicographically using the order $0 < 1 < -1$ of signs. Now consider $A = \{\alpha_1,...,\alpha_l\} \subseteq \{0,1,2\}^{\mathcal{P}}$ and $\Sigma = \{\sigma_1,...,\sigma_m\} \subseteq \{-1,0,1\}^{\mathcal{P}}$, where the elements have been ordered according to $<_{lex}$. We let $Mat(A,\Sigma)$ be the $l \times m$ matrix defined by

$$Mat(A,\Sigma) := \begin{pmatrix} \sigma_1^{\alpha_1} & \dots & \sigma_m^{\alpha_1} \\ \vdots & \ddots & \vdots \\ \sigma_1^{\alpha_l} & \dots & \sigma_m^{\alpha_l} \end{pmatrix}.$$

With this notation, if Σ fulfills the requirement of Lemma 4.1.5, it holds that

$$Mat(A,\Sigma) \cdot \begin{pmatrix} c(\sigma_1, \mathcal{Z}) \\ \vdots \\ c(\sigma_m, \mathcal{Z}) \end{pmatrix} = \begin{pmatrix} TaQ(\mathcal{P}^{\alpha_1}, \mathcal{Z}) \\ \vdots \\ TaQ(\mathcal{P}^{\alpha_l}, \mathcal{Z}) \end{pmatrix}. \tag{4.3}$$

If we let $\mathcal{P} = \{p\}$, $A = \{0,1,2\}^{\{p\}}$ and $\Sigma = \{-1,0,1\}^{\{p\}}$, Equation 4.3 is just Lemma 4.1.3. The next Theorem 4.1.9 tells us how the linear system in 4.3 can be used to perform sign determination on an arbitrary large $\mathcal{P}$. The corresponding matrix can be determined using *Kronecker products*, which we define below. This is convenient because it immediately proves that this matrix is invertible.

Definition 4.1.8 (Kronecker product). *Let* A *be an* $m \times m$ *matrix and* B *an* $n \times n$ *matrix over some structure where multiplication has been defined. The* $mn \times mn$ *matrix* $A \otimes B$ *is defined as*

$$A \otimes B := \begin{pmatrix} a_{11}B & \dots & a_{1m}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mm}B \end{pmatrix}$$

A nice property of the Kronecker products is that if A and B are invertible, then so is $A \otimes B$.

Theorem 4.1.9 ([BPR10], Proposition 2.72). *If* $A = \{0,1,2\}^{\mathcal{P}}$ *and* $\Sigma = \{-1,0,1\}^{\mathcal{P}}$, *then* $Mat(A,\Sigma)$ *is invertible. More precisely, if* $|\mathcal{P}| = k$, $Mat(A,\Sigma)$ *is the matrix* M_k *inductively defined as follows:*

$$M_1 := \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{pmatrix} \quad \textit{and} \quad M_k := M_{k-1} \otimes \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{pmatrix} \textit{ for } k > 1.$$

Example 4.1.10. *If $\mathcal{P} = \{p_1, p_2\}$, then with Theorem 4.1.9 and Equation 4.3 it follows that the sign conditions realized by $\mathcal{P}$ on $Zeros(\mathcal{Z})$ can be computed solving the following system of 9 linear equations:*

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 & -1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & -1 & -1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} c(p_1 = 0 \wedge p_2 = 0, \mathcal{Z}) \\ c(p_1 = 0 \wedge p_2 > 0, \mathcal{Z}) \\ c(p_1 = 0 \wedge p_2 < 0, \mathcal{Z}) \\ c(p_1 > 0 \wedge p_2 = 0, \mathcal{Z}) \\ c(p_1 > 0 \wedge p_2 > 0, \mathcal{Z}) \\ c(p_1 > 0 \wedge p_2 < 0, \mathcal{Z}) \\ c(p_1 < 0 \wedge p_2 = 0, \mathcal{Z}) \\ c(p_1 < 0 \wedge p_2 > 0, \mathcal{Z}) \\ c(p_1 < 0 \wedge p_2 < 0, \mathcal{Z}) \end{pmatrix} = \begin{pmatrix} TaQ(1, \mathcal{Z}) \\ TaQ(p_2, \mathcal{Z}) \\ TaQ(p_2^2, \mathcal{Z}) \\ TaQ(p_1, \mathcal{Z}) \\ TaQ(p_1 p_2, \mathcal{Z}) \\ TaQ(p_1 p_2^2, \mathcal{Z}) \\ TaQ(p_1^2, \mathcal{Z}) \\ TaQ(p_1^2 p_2, \mathcal{Z}) \\ TaQ(p_1^2 p_2^2, \mathcal{Z}) \end{pmatrix}$$

In the solution, each $c(\sigma, \mathcal{Z})$ which is not zero corresponds to a sign condition in $sgn\big(\mathcal{P}, Zeros(\mathcal{Z})\big)$.

4.1.3 Avoiding exponential complexity

The problem with the approach Theorem 4.1.9 suggests for sign determination is that the size of the matrix quickly becomes tremendously large as the number of polynomials in $\mathcal{P}$ increases. The size of the matrix is $3^k \times 3^k$, if $|\mathcal{P}| = k$. Even more disadvantageous is the fact that also a corresponding number of Tarski queries (3^k many) have to be computed.

Fortunately, there exists an ingenious algorithm discovered by M. Ben-Or, D. Kozen and J. Reif [BOKR86]. The key observation is that the number of non-zero entries in the solution vector of the $3^k \times 3^k$ sized linear system is bounded by $|Zeros(\mathcal{Z})|$, which in general may be significantly smaller than 3^k. For example, when we are in the univariate case where $\mathcal{Z} = \{q\}$ for some $q \in \mathbb{R}[X]$, then certainly the number of zeros of q is bounded by its degree.

The basic principle of the improved algorithm is to perform the computations iteratively instead of computing all realized sign conditions at once like it was the case in 4.1.10. That means, the sets

$$sgn\big(\{p_1\}, Zeros(\mathcal{Z})\big),\ ...,\ sgn\big(\{p_1,...,p_k\}, Zeros(\mathcal{Z})\big)$$

are computed one after another. In order to keep the size of the matrices small, we drop out the columns corresponding to the zero elements in the solution. This yields a rectangular matrix which is guaranteed to be of full rank. The next task is then to find a basis among the rows of this matrix and discard all the other rows, obtaining once again a square matrix. In the following iteration we proceed by taking the Kronecker product of the reduced matrix with M_1, just like in Theorem 4.1.9. The size of any matrix occurring in the procedure is then ensured to be of size at most $3 \cdot |Zeros(\mathcal{Z})| \times 3 \cdot |Zeros(\mathcal{Z})|$.

Example 4.1.11. *Consider once again Example 4.1.4. Dropping out the column that corresponds to the sign condition $p < 0$ which is not realized by p on $\mathcal{Z}$ yields the system*

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} c(p = 0, \mathcal{Z}) \\ c(p > 0, \mathcal{Z}) \end{pmatrix} = \begin{pmatrix} TaQ(1, \mathcal{Z}) \\ TaQ(p, \mathcal{Z}) \\ TaQ(p^2, \mathcal{Z}) \end{pmatrix}.$$

Now we can discard either the second or the third row of the matrix in order to obtain an invertible square matrix. Suppose we choose to keep the second row. The system *then becomes*

$$\begin{pmatrix}1 & 1\\0 & 1\end{pmatrix} \cdot \begin{pmatrix}c(p=0,\mathcal{Z})\\c(p>0,\mathcal{Z})\end{pmatrix} = \begin{pmatrix}TaQ(1,\mathcal{Z})\\TaQ(p,\mathcal{Z})\end{pmatrix}.$$

Our implementation of the improved sign determination procedure is very similar to the one proposed as Algorithm 10.11 in [BPR10]. There it is also shown in detail how the basis can be extracted from the rows of the rectangular matrices. Moreover, the authors prove that in the improved sign determination algorithm, at most $1 + 2k \cdot |Zeros(\mathcal{Z})|$ different Tarski queries have to be computed. In most cases, this is much less than the 3^k queries needed in the naive algorithm.

4.2 Computing Tarski queries

In this section we are going to explain how Tarski queries on a zero dimensional system can be determined.

4.2.1 Univariate case

In the univariate case, the Tarski queries can be computed using a generalized version of Sturm's Theorem 2.6.6.

Theorem 4.2.1 (Univariate Tarski query, [BPR10], Algorithm 9.2)**.** *Let* $p,q \in \mathbb{R}[X]$, $q \neq 0$. *Then*

$$TaQ(p,\{q\}) = Var_{-\infty}(SRS(q,q'p)) - Var_{\infty}(SRS(q,q'p)).$$

The signs achieved by the polynomials in the signed remainder sequence at $\pm\infty$ can be deduced by their degree and the signs of the leading coefficients. Alternatively, the Tarski query can also be computed only with respect to a given interval, similar to the Sturm query ([BPR10], Exercise 9.1).

Example 4.2.2. *Let* $q := X^3 - 2X$ *and* $p := q' = 3X^2 - 2$ *be the derivative of* q. *Suppose we want to compute* $TaQ(p,\{q\})$. *First we compute the signed remainder sequence*

$$SRS(q,q'p) = (X^3 - 2X,\ 9X^4 - 12X^2 + 4,\ -X^3 + 2X,\ -6X^2 - 4,\ -\frac{8}{3}X,\ 4).$$

Now consider the degrees and the signs of the leading coefficients in this sequence. For $Var_{-\infty}(SRS(q,q'p))$ *we have to count the sign variations in the list*

$$(-,\ +,\ +,\ -,\ +,\ +),$$

which are 3, *and for* $Var_{-\infty}(SRS(q,q'p))$ *we obtain the list*

$$(+,\ +,\ -,\ -,\ -,\ +),$$

where we count 2 *sign variations. Thus, applying Theorem 4.2.1, we conclude that* $TaQ(p,\{q\}) = 3 - 2 = 1$. *We can verify this using the plot in Figure 3.1.*

4.2.2 Multivariate case

The method from Theorem 4.2.1 that we use to compute Tarski queries for univariate polynomials does not generalize to the case where more than one variable is involved. The authors of [BPR10] propose a method which relies on *Hermite's quadratic form* (Algorithm 12.7 in [BPR10]). The details of the method are beyond our scope. We would only like to point out that in order to compute $TaQ(p, \mathcal{Z})$, a so-called *Gröbner basis* (see [BPR10], Section 12.1 for a definition) of $\mathcal{Z}$ has to be computed. The Gröbner basis can be used to detect if $\mathcal{Z}$ is zero-dimensional ([BPR10], Proposition 12.7).

Chapter 5

Implementation and experimental results

5.1 Implementational details

We implemented all the algorithms described in Chapter 3 and 4 within the C++ library CArL[1] which is also used by SMT-RAT. Among many other features, CArL provides data structures and basic algorithms for representing and manipulating both univariate and multivariate polynomials. There is also an implementation of an algorithm computing Gröbner basis available. CArL includes the external open source library GMP[2] which provides classes for arbitrary precision integers and rational numbers. The precision is 'arbitrary' in a sense that it is only limited by the memory available and not by a previously fixed maximum size like it is the case for the built in number types like `int` or `double`.

Once the algorithms were integrated into CArL, it was easy to use them in the CAD implementation of SMT-RAT. We have created a number of classes of which we shall discuss the most important ones here.

5.1.1 General remarks

Formally, we treated sign conditions as *mappings* from the corresponding sets of derivatives to the set $\{-1,0,1\}$. In our actual implementation, it seemed more convenient to represent a sign condition σ on $Der(p)$ as the *ordered list*

$$\big(\sigma(p),\, \sigma(p'),\, ...,\, \sigma(p^{(deg(p))})\big).$$

For the theoretical discussion however, the notation with mappings seemed more convenient.

5.1.2 Mixing Thom encodings with the explicit representation

We determine the roots of univariate polynomials of degree at most 2 using explicit formulas. For a polynomial $p = aX + b$, $a \neq 0$, the only real root is $-b/a$, which certainly is a rational number. If $p = aX^2 + bX + c$ with $a \neq 0$, then the real roots

[1]Currently available at https://github.com/smtrat/carl
[2]Source code and documentation are currently available at https://gmplib.org/

of p are given by the formula

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a},$$

which can be used to determine the roots of p explicitly. If the expression under the root is negative, then p does not have any real roots, if it is zero is has exactly one (rational) root and if it is positive it has 2 real roots. Only if the term $b^2 - 4ac$ is a square number, we can deduce explicit rational representations for the roots of p from this formula. Otherwise we characterize them as Thom encodings just as usual.

Working with the rational representation whenever it is possible of course increases the performance. On the other hand, the implementation becomes more complicated as there are numerous special cases to consider when the inputs to the operators are real algebraic numbers given in different representations. Usually this can be bypassed by converting the rational number in a Thom encoding first, which is easy because for $a \in \mathbb{Q}$, the polynomial $X - a$ has a as a root. In some cases however, the explicit rational representation can be used more efficiently.

- Computing an intermediate point between a number represented as a Thom encoding and an explicitly given rational number a: Here we can first transform a into a Thom encoding like described above and then perform Algorithm 9. The resulting Thom encoding will have a defining polynomial of the form $X - a'$, so we know that the rational number a' is the desired intermediate point. We can return it in its explicit rational representation.

- Lifting $p \in \mathbb{Q}[X_1,...,X_n,X]$ on a point $(\xi_1,...,\xi_n)$: Sometimes, the point is only partially given in terms of a triangular Thom encoding. For example, the numbers $\xi_1,...,\xi_k$ for some $k < n$ might be present in their explicit representation, while the rest of them is given in terms of a triangular Thom encoding $\langle T \rangle = (\xi_{k+1},...,\xi_n)$. Whenever we notice that we have a case like this, we substitute $X_1,...,X_k$ in p for their corresponding values, obtaining a polynomial $p' \in \mathbb{Q}[X_{k+1},...,X_n,X]$ and then call Algorithm 2 with inputs p' and T. Similar considerations hold for the evaluation operation.

- Samples below and a real algebraic number represented by a Thom encoding (p, σ) can be computed in their explicit representation using the Cauchy bound $C(p)$ of p like outlined in Section 3.2.7. This is not possible in the multivariate case though.

In summary, we try to stick to the explicit representation whenever we can.

5.1.3 The Thom encodings class

We have implemented a class that represents (triangular) Thom encodings in a similar way like discussed in Chapter 3. `ThomEncoding` is a recursive data structure as it contains a pointer to another `ThomEncoding` object. Each of the `ThomEncoding` objects in such a hierarchy corresponds to one 'level' of a triangular Thom encoding. The encoding at the lowest level is an explicit (univariate) Thom encoding and thus `point=nullptr`. For the other encodings, the member `p` is a multivariate polynomial and the member `v` is a distinguished variable contained in this polynomial. It is needed in order to know which variables are already present in the triangular Thom encoding given by `point` and which one is new on this level. The member `sigma` is a sign condition on $Der_X(p^{(l)})$, where X is the specified given by `v` and $0 \leq l \leq k$. We provide an overview of the members of `ThomEncoding` in Figure 5.1.

```
class ThomEncoding {

    // private member variables

    Polynomial p;
    Variable v;

    mutable SignCondition sigma;
    int k;

    ThomEncoding* point;

    SignDetermination sd;

    // public member functions

    public:

    ComparisonResult compare(const ThomEncoding& other) const;

    static ThomEncoding intermediatePoint
        (const ThomEncoding& lhs, const ThomEncoding& rhs);
    ...
};
```

Figure 5.1: Simplified representation of the Thom encoding class

The comparison function `compare` is implemented with a (desired) side effect: In Algorithm 6, we have seen that in order to compare Thom encodings defined with respect to *different* polynomials, we sometimes have to extend the sign condition of one of these encodings. We have implemented the comparison algorithm in a way such that this extension is stored permanently. This is also the reason why the member `sigma` is marked `mutable`. With this side effect, we avoid recomputing sign conditions over and over again when comparison is performed. In order to know which part of the sign condition stored in a `ThomEncoding` object is actually relevant for distinguishing between the other roots of the polynomial, we provide the additional member `k`. It corresponds precisely to the k in Definition 3.1.9.

`ThomEncoding` objects also have a member of type `SignDetermination`, whose purpose we are going to explain now.

5.1.4 The sign determination class and Tarski queries

The functionality discussed in Chapter 4 is mainly implemented in two classes, `Sign-Determination` and `TarskiQueryManager`. As the names suggest, the first one implements the improved sign determination algorithm and the second one implements Tarski queries.

The `TarskiQueryManager` class computes Tarski queries on the system $\mathcal{Z}$ which is passed to it at initialization. It uses the method given by Theorem 4.2.1 for computing univariate queries whenever this is possible and otherwise employs the multivariate algorithm. For this purpose, it needs to compute a Gröbner basis of the ideal generated by $\mathcal{Z}$ like pointed out in Section 4.2.2. From the basis it can read off whether $\mathcal{Z}$ is a zero-dimensional system. If this is not the case, we have encountered the *zero-*

dimensional problem described in Section 3.2.3. In our current prototype, we then abort all computations. Otherwise, the basis is used to set up the *multiplication table* which is crucial for the multivariate Tarski query algorithm. For multiplication tables we have implemented an extra class which also stores the corresponding Gröbner basis and offers interfaces to perform operations in the structure $\mathbb{Q}[X_1,...,X_n]/Ideal(\mathcal{Z})$. The most important ones are reduction of a polynomial modulo $Ideal(\mathcal{Z})$, multiplication and trace computation of elements from $\mathbb{Q}[X_1,...,X_n]/Ideal(\mathcal{Z})$. (See chapter 12 in [BPR10] for details.)

Another core feature of an object of type `TarskiQueryManager` is the ability to cache already computed Tarski queries so that they do not have to be recomputed every time they are requested. For this purpose, the manager class has a member `std::map<Polynomial, int>` which realizes the cache. The polynomials in the cache are normalized so that they have leading coefficient 1. When we want to look up a given polynomial p, we first normalize it in the same way. Let p' be the normalized polynomial. If $TaQ(p', \mathcal{Z})$ was found in the cache, we compute $TaQ(p, \mathcal{Z})$ using the relation

$$TaQ(c \cdot p,\, \mathcal{Z}) = sgn(c) \cdot TaQ(p,\, \mathcal{Z}),$$

which holds for all rational constants c. (This follows immediately from Definition 4.1.1.)

An object of the class `SignDetermination` is initialized with a zero-dimensional system $\mathcal{Z}$. It then sets up a `TarskiQueryManager` on $\mathcal{Z}$ like explained above, which is one of its members. One of its main interfaces is the function

```
getSignsAndAdd(Polynomial p),
```

which computes and returns the set $sgn(\{p\}, Zeros(\mathcal{Z}))$ of sign conditions realized by p on the common zeros of $\mathcal{Z}$. Moreover, this function updates the internal state of the sign determination object in a way such that a subsequent call to the same function with argument q would return $sgn(\{p,q\}, Zeros(\mathcal{Z}))$. This incremental behavior is crucial for the root finding algorithms 1 and 2 in Section 3.2. `SignDetermination` also offers a method `getSigns(Polynomial p)`, which is similar but does not update the internal state. This is convenient for Algorithm 3.

5.2 Experimental results

We have evaluated our implementation on a benchmark set provided by the SMT-LIB initiative [BFT16]. The problems in the particular benchmark we have used were generated by the automated theorem prover *Meti-Tarski* [AP10]. It contains a total number of 7713 examples, which involve problems with an average number of 3 different variables and average polynomial degree 6. However, it should be mentioned that a considerable amount of instances within this set can be solved trivially such that no real algebraic number representation, besides the explicit one, is ever needed. We observed that during the solving procedure of only 3909 of the 7713 examples in this set we had to characterize a root as a Thom encoding.

The table summarizes the results of the evaluation. The row labeled 'Thom' contains the results corresponding to the CAD algorithm which uses our Thom representation for real algebraic numbers. The other row, which is labeled with 'Interval', shows the results of the CAD algorithm employing the interval representation described in 2.6. Each of the 7713 test cases appears in the table under one of the following four categories: SAT/UNSAT means that the example was proven (un-)satisfiable by the solver. Here we additionally provide the average solving time. Examples for which we experienced a timeout fixed to 30 seconds are counted in the column labeled 'time-out'. Occurrences of the zero-dimensional problem are documented in the last column 'problem'. Of course, this only applies to the Thom version.

	SAT		UNSAT		timeout	problem
	solved	avg. time	solved	avg. time		
Thom	4276	425 ms	1944	426 ms	1467	26
Interval	4653	154 ms	2185	384 ms	875	-

Figure 5.2: Results of the evaluation of the two different CAD variants on the benchmark set 'Meti-Tarski'

With the interval representation, we were able to solve a total number of 6838 instances before the timeout, which corresponds to an 88.7%. With Thom encodings, only 6220 or 80.6% of all problems could be solved. Among the instances that could not be solved with the Thom approach, solely 26 caused the zero-dimensional problem. The average solving time for examples that were satisfiable increased by about 175% compared to the interval representation. Inputs that were unsatisfiable consumed around 11% more time.

Chapter 6

Conclusion

6.1 Summary

We have explained how *Thom encodings* can be used to represent *real algebraic numbers* implicitly. This was motivated by the needs of the *cylindrical algebraic decomposition* method which we use in the context of *lazy SMT-solving* for the theory *quantifier-free non-linear real arithmetic.* For our specific case, we have introduced a slightly improved variant of Thom encodings like they are usually described in the literature. Moreover, we have worked out the operations the CAD method requires for real algebraic numbers and presented the corresponding algorithms. When it was possible, we proposed adaptations for the special case of real numbers. In order to implement the 'lifting' operation, we presented *triangular Thom encodings* as a generalization of the standard Thom encodings. In this context we also stumbled upon the *zero-dimensional problem* for which we have not yet encountered a solution.

The main ingredient in our algorithms is the *sign determination* procedure of which we discussed the basic principles. We have seen that it relies on a symbiosis of linear algebra and *Tarski queries.* For the univariate case, we have demonstrated how Tarski queries can be computed. There is also a method for the multivariate case available, but the mathematical details were beyond our scope.

Apart from the theoretical discussion, we have implemented the algorithms and a data structure which represents our notion of (triangular) Thom encodings. We integrated it within the SMT solver SMT-RAT which provides an implementation of the CAD that we could easily adapt. Finally, we compared the Thom version of the CAD method with the one where the *interval representation* is employed as a real algebraic number representation.

6.2 Discussion

Our experiments demonstrate that – in the current stage of development – our Thom method is not competitive to the interval representation. Furthermore, we sometimes have to abort the procedure because of the zero-dimensional problem. Fortunately, it does not seem to occur too often: In the experiment, we observed it only in 1 out of about 150 cases where Thom encodings were actually used. We conjecture that the high amount of time consumed by the version of the CAD algorithm which uses Thom encodings is mainly due to the multivariate Tarski query algorithm which involves costly Gröbner basis computations. Moreover, some of the algorithms for Thom encodings are not well suited to the case where we want to perform the operations locally. For example, comparing two numbers given as Thom encodings with defining polynomials p_1 and p_2 is not much less expensive than ordering *all* roots of p_1 and p_2 directly. This is because the calls to SignDetermination that would have to be performed are nearly the same. However, the interfaces of the CAD implementation that we have used could not so easily be adapted to take advantage of this.

Although the interval representation seems to be the most adequate representation technique for real algebraic numbers in the CAD method, we have yet shown that other approaches are imaginable. Thom encodings are an elegant algebraic concept which also generalizes to cases where we want to compute a CAD over less common fields. We are convinced that further investigations in this direction would be worthwhile.

6.3 Future work

A more refined investigation of the zero-dimensional problem is still pending. In particular, it should be clarified if the problem is avoidable at all. If not, then it would be beneficial if there was a way to take advantage of the results that have been computed up to the point where the problem occurred instead of falling back to the interval representation and restarting the whole procedure.

Moreover, apart from general improvements of our implementation which up to now is only a prototype, the algorithms and the Thom data structure still offer some room for enhancement:

Relevant sign conditions In Chapter 3, we have already made the observation that in many cases not the whole sign condition realized by $Der(p)$ on a root of p is necessary to distinguish between the roots, but only a small part of it. We have partly taken advantage of this by equipping the Thom encoding objects with an additional parameter k, indicating that the sign condition realized by $Der(p^k)$ already contains enough information to characterize the corresponding root. Future work could go towards making the set $Der(p^{(k)})$ even smaller by keeping only the derivatives that are actually needed to make the distinction. For instance, in Example 3.1.6 we have seen that the sign condition realized by p'' was already sufficient to distinguish between the roots of p. Let us consider an even more extreme example: Let

$$p(X) := (X+1)^2(X-2)(X-3) = X^6 - 5X^5 + 8X^4 - 10X^3 + 13X^2 - 5X + 6.$$

p is of degree 6 but the only real roots are 2 and 3. The table gives an overview of the signs realized by the derivatives of p on its roots:

Root	p'	p''	p'''	$p^{(4)}$	$p^{(5)}$
2	-1	-1	1	1	1
3	1	1	1	1	1

With our currently implemented approach, we would use the sign condition realized by $Der(p'')$ on the roots for their Thom encodings. However, we can read off the table that the sign p'' (or p') achieves at the roots would already be sufficient.

Providing a smaller set of necessary derivatives means that the input size of SignDetermination when applied in RealRoots (triangular case) or ThomCompare also decreases. This in turn decreases the number of calls to the Tarski query algorithm which makes up a big part of the overall computation time.

Experimenting with intermediate points In Section 3.2.6, we presented different approaches for computing intermediate points of which we have only used the iterative variant (Algorithms 9 and 11) in our experiments so far. Here, different parameters for the initial value of ϵ and the divisor could be tested. Moreover, we do not yet have any insights on how well the other approaches for intermediate points work in practice.

Improving the sign determination procedure Finally, we would like to point out that we have only implemented a very basic version of the sign determination method because the main focus was on the algorithms operating on Thom encodings. An immediate improvement for the basic algorithm is due to D. Perrucci: He observed that the linear systems can be solved more efficiently than it is possible with the standard methods which is because of their special structure [Per09].

In [BPR10], Chapter 12, the authors propose a different and much more complicated sign determination algorithm which is adapted to the case where the zero-dimensional system has a 'triangular' form, just like the polynomials of a triangular Thom encoding.

Bibliography

[ACM84] Dennis S. Arnon, George E. Collins and Scott McCallum. Cylindrical Algebraic Decomposition I: The Basic Algorithm. *SIAM Journal on Computing*, 13(4):865–877, 1984.

[AP10] Behzad Akbarpour and Lawrence Charles Paulson. MetiTarski: An Automatic Theorem Prover for Real-valued Special Functions. *Journal of Automated Reasoning*, 44(3):175–205, 2010.

[BFT16] Clark Barrett, Pascal Fontaine and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.

[BOKR86] Michael Ben-Or, Dexter Koze and John Reif. The Complexity of Elementary Algebra and Geometry. *Journal of Computer and System Sciences*, 32(2):251–264, 1986.

[BPR10] Saugata Basu, Richard Pollack and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry, Second Edition.* Springer, 2010.

[CKJ$^+$15] Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp and Erika Abraham. SMT-RAT: An Open Source C++ Toolbox for Strategic and Parallel SMT Solving. In *Proc. of the 18th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'15)*, volume 9340 of *LNCS*, pages 360–368. Springer, September 2015.

[CLM$^+$92] Felipe Cucker, Herve Lanneau, Bud Mishra, Paul Pedersen and Marie-Françoise Roy. NC Algorithms for Real Algebraic Numbers. *Applicable Algebra in Engineering, Communication and Computing*, 3(2):79–98, 1992.

[Col75] George E. Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decompostion. In *Automata Theory and Formal Languages 2nd GI Conference Kaiserslautern*, pages 134–183. Springer, May 1975.

[Col01] George E. Collins. Polynomial Minimum Root Separation. *Journal of Symbolic Computation*, 32(5):467–473, 2001.

[Coo71] Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of the third annual ACM symposium on theory of computing*, pages 151–158. ACM, 1971.

[CR88] Michel Coste and Marie-Françoise Roy. Thom's Lemma, the Coding of Real Algebraic Numbers and the Computation of the Topology of Semi-Algebraic Sets. *Journal of Symbolic Computation*, 5(1):121–129, 1988.

[DLL62] Martin Davis, George Logemann and Donald Loveland. A Machine Program for Theorem-Proving. *Communications of the ACM*, 5(7):394–397, 1962.

[EMT08] Ioannis Z. Emiris, Bernard Mourrain and Elias P. Tsigaridas. Real Algebraic Numbers: Complexity Analysis and Experimentation. In *Reliable Implementation of Real Number Algorithms: Theory and Practice*, pages 57–82. Springer, 2008.

[ES03] Niklas Eén and Niklas Sörensson. An Extensible SAT-Solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.

[GCL92] Keith O. Geddes, Stephen R. Czapor and George Labahn. *Algorithms for Computer Algebra*. Springer Science & Business Media, 1992.

[Hon90] Hoon Hong. An Improvement of the Projection Operator in Cylindrical Algebraic Decomposition. In *Proceedings of the international symposium on Symbolic and algebraic computation*, pages 261–264. ACM, 1990.

[JDM12] Dejan Jovanović and Leonardo De Moura. Solving Non-Linear Arithmetic. In *International Joint Conference on Automated Reasoning*, pages 339–354. Springer, 2012.

[Lin00] F. Lindemann. Über die Zahl π. In *Pi: A Source Book*, pages 194–206. Springer, 2000.

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

[Mis93] Bhubaneswar Mishra. *Algorithmic Algebra*. Springer Science & Business Media, 1993.

[Per09] Daniel Perrucci. Linear Solving for Sign Determination. *arXiv preprint arXiv:0911.5707*, 2009.

[RS90] Marie-Françoise Roy and Aviva Szpirglas. Complexity of Computation on Real Algebraic Numbers. *Journal of Symbolic computation*, 10(1):39–51, 1990.

[Rum79] Siegfried M. Rump. Polynomial Minimum Root Separation. *Mathematics of Computation*, 33(145):327–336, 1979.

[Seb07] Roberto Sebastiani. Lazy Satisfiability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation*, 3:141–224, 2007.

[vD94] Dirk van Dalen. *Logic and Structure*. Springer, 1994.

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2013.

Appendix

A Index of notation

$\mathbb{N}, \mathbb{N}_+$	non-negative and positive integers
$\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$	integer, rational, real and complex numbers
$\mathbb{R}_{alg}$	real algebraic numbers
S^n	n-fold cartesian product of a set S with itself
$[a,b]$	closed interval
$]a,b[$	open interval
$f\big\|_D$	restriction of the mapping f on the domain D
B^A	set of all possible mappings from A to B
$deg(p)$	degree of a univariate polynomial p
$deg_X(p)$	degree of a polynomial p with respect to X
p', p'', p'''	first, second and third derivative of a univariate polynomial p
$p^{(k)}$	k-th derivative of a univariate polynomial p
$p_X^{(k)}$	k-th derivative of p with respect to X
$rem(p, q)$	remainder of p divided by q
$SRS(p, q)$	signed remainder sequence of p and q
$sgn(\xi)$	sign of the real number ξ
$sgn(\mathcal{P}, \xi)$	sign condition realized by $\mathcal{P}$ on ξ
$sgn(\mathcal{P}, S)$	set of sign conditions realized by $\mathcal{P}$ on the points in S
$Der(p)$	set of derivatives of p including p itself
$Der_X(p)$	set of derivatives of p with respect to X, including p itself
$Zeros(\mathcal{Z})$	set of common zeros of the polynomials in $\mathcal{Z}$
$\langle\tau\rangle$	the real algebraic number ξ represented by the Thom encoding τ
$\langle T\rangle$	the real algebraic point $(\xi_1,...,\xi_n)$ represented by the triangular Thom encoding T
$Der(T)$	set of derivatives defining a triangular Thom encoding T
$c(\sigma, \mathcal{Z})$	number of points in $Zeros(\mathcal{Z})$ that realize the same sign condition σ on $\mathcal{P}$
$TaQ(p, \mathcal{Z})$	Tarski query of p on the zero-dimensional system $\mathcal{Z}$

B Generalized algorithms

Comparison

Let σ_1 and σ_2 be sign conditions on $\mathcal{P}_1$ and $\mathcal{P}_1$, respectively, with $\mathcal{P}_1 \cap \mathcal{P}_2 = \emptyset$. We let $\sigma_1 \times \sigma_2 : \mathcal{P}_1 \cup \mathcal{P}_2 \to \{-1,0,1\}$ be the sign condition defined by

$$(\sigma_1 \times \sigma_2)(p) \mapsto \begin{cases} \sigma_1(p), & \text{if } p \in \sigma_1, \\ \sigma_2(p), & \text{if } p \in \sigma_2. \end{cases}$$

Algorithm 10 Comparison of Thom encodings (triangular case)

Input:

- a triangular Thom encoding $T = (\mathcal{P}, \Sigma)$ with $\langle T \rangle = \xi \in \mathbb{R}^n$,
- $p_1, p_2 \in \mathbb{Q}[X_1,...,X_n,X]$,
- sign conditions σ_1, σ_2 on $Der_X(p_1^{(k_1)})$, $Der_X(p_2^{(k_2)})$, respectively, such that $\tau_1 = (p_1(\xi, X), \sigma_1(\xi))$ and $\tau_2 = (p_2(\xi, X), \sigma_2(\xi))$ are Thom encodings

Output: *less* if $\langle \tau_1 \rangle < \langle \tau_2 \rangle$, *equal* if $\langle \tau_1 \rangle = \langle \tau_2 \rangle$ and *greater* if $\langle \tau_1 \rangle > \langle \tau_2 \rangle$

```
1:  procedure ThomCompare($p_1$, $\sigma_1$, $p_2$, $\sigma_2$, $T$)
2:      if $p_1 = p_2$ then return SignCompare($\sigma_1$, $\sigma_2$)
3:      $\mathcal{D}_1 \leftarrow Der(T) \cup Der_X(p_1^{(k_1)})$
4:      $\mathcal{D}_2 \leftarrow \emptyset$
5:      for $i \leftarrow deg_X(p_2), i \geq 0, i \leftarrow i - 1$ do
6:          $\mathcal{D}_2 \leftarrow \{p_{2_X}^{(i)}\} \cup \mathcal{D}_2$
7:          $\tilde{\Sigma} \leftarrow$ SignDetermination($\mathcal{D}_1 \cup \mathcal{D}_2$, $\{p_1\}$)
8:          $\tilde{\sigma}_1 \leftarrow$ only element in $\tilde{\Sigma}$ such that $\tilde{\sigma}_1|_{\mathcal{D}_1} = \Sigma \times \sigma_1$
9:          $\tilde{\sigma}_1 \leftarrow \tilde{\sigma}_1|_{\mathcal{D}_2}$
10:         $\tilde{\sigma}_2 \leftarrow \sigma_2|_{\mathcal{D}_2}$
11:         if $k_2 < |\mathcal{D}_2|$ then
12:             $\tilde{\Sigma} \leftarrow$ SignDetermination($Der(T) \cup \mathcal{D}_2$, $\{p_2\}$)
13:             $\tilde{\sigma}_2 \leftarrow$ only element in $\tilde{\Sigma}$ such that $\tilde{\sigma}_2|_{Der(T) \cup Der(p_2^{(k_2)})} = \Sigma \times \sigma_2$
14:             $\tilde{\sigma}_2 \leftarrow \tilde{\sigma}_2|_{\mathcal{D}_2}$
15:         end if
16:         if $\tilde{\sigma}_1 \neq \tilde{\sigma}_2$ then return SignCompare($\tilde{\sigma}_1$, $\tilde{\sigma}_2$)
17:     end for
18:     return equal
19: end procedure
```

Intermediate points

Specification INTERMEDIATEPOINT (triangular case)

Input:

- a triangular Thom encoding $T = (\mathcal{P}, \Sigma)$ with $\langle T \rangle = \xi \in \mathbb{R}^n$,
- $p_1, p_2 \in \mathbb{Q}[X_1,...,X_n,X]$,
- sign conditions σ_1, σ_2 on $Der_X(p_1^{(k_1)})$, $Der_X(p_2^{(k_2)})$, respectively, such that $\tau_1 = (p_1(\xi, X), \sigma_1(\xi))$ and $\tau_2 = (p_2(\xi, X), \sigma_2(\xi))$ are Thom encodings with $\langle \tau_1 \rangle < \langle \tau_2 \rangle$

Output:

- a polynomial $p \in \mathbb{Q}[X_1,...,X_n,X]$ and a sign condition σ on $Der_X(p^{(k)})$ such that $\tau = (p(\xi,X), \sigma(\xi))$ is a Thom encoding with $\langle \tau_1 \rangle < \langle \tau \rangle < \langle \tau_2 \rangle$

Algorithm 11 Intermediate point using iterative approximation (triangular case)

```
1: procedure IntermediatePoint($p_1$, $\sigma_1$, $p_2$, $\sigma_2$, $T$)
2:     $\epsilon \leftarrow 1$
3:     if $deg_X(p_1) \leq deg_X(p_2)$ then
4:         $p \leftarrow p_1(X_1,...,X_n,X - \epsilon)$
5:         $\sigma \leftarrow \sigma_1(X_1,...,X_n,X - \epsilon)$
6:         while ThomCompare($p$, $\sigma$, $p_2$, $\sigma_2$, $T$) $\neq$ less do
7:             $\epsilon \leftarrow \epsilon/2$
8:             $p \leftarrow p_1(X_1,...,X_n,X - \epsilon)$
9:             $\sigma \leftarrow \sigma_1(X_1,...,X_n,X - \epsilon)$
10:        end while
11:        return $p, \sigma$
12:    else
13:        $p \leftarrow p_2(X_1,...,X_n,X + \epsilon)$
14:        $\sigma \leftarrow \sigma_2(X_1,...,X_n,X + \epsilon)$
15:        while ThomCompare($p_1$, $\sigma_1$, $p$, $\sigma$, $T$) $\neq$ less do
16:            $\epsilon \leftarrow \epsilon/2$
17:            $p \leftarrow p_2(X_1,...,X_n,X + \epsilon)$
18:            $\sigma \leftarrow \sigma_2(X_1,...,X_n,X + \epsilon)$
19:        end while
20:        return $p, \sigma$
21:    end if
22: end procedure
```