RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN
LEHRSTUHL FÜR THEORIE HYBRIDER SYSTEME

Bachelor Thesis

An Extension of the GiNaCRA Library for the Cylindrical Algebraic Decomposition

Joachim Redies
Matrikelnr. 289502

1. Examiner: Prof. Dr. Erika Ábrahám
2. Examiner: Prof. Dr. Peter Rossmanith

Supervisor: Ulrich Loup

January 19, 2012

Erklärung

Hiermit versichere ich, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe.

Aachen, den 19.01.2012

(Joachim Redies)

Contents

Chapter 1

Motivation

In the first section of this chapter, we describe the general problem. Afterwards, there will be a short section about possible applications in the context of hybrid systems.

1.1 Problem formulation

In this paper we study (in)equation systems of multivariate polynomials with real-valued variables and rational coefficients. The aim is to check such (in)equation systems for satisfiability and to compute a satisfying solution if the system is satisfiable.

Example 1. Let be given two polynomial equations $(x-1)^2+(y-1)^2-1=0$ and $x-y=0$. Geometric interpretations of these equations: The first one is a circle and the second is a line in the 2-dimensional euclidian vector space (cf. Figure 1). Common solutions are given by the intersection points of the circle and the line. In general, the satisfiablity problem can be lead back to determining the common real roots fo multivariate polynomials.

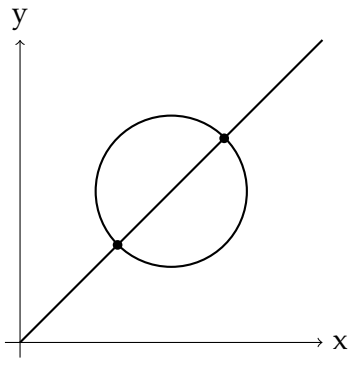

Figure 1 – The common roots of circle and a line.

For our example the solutions are $(\frac{(2-\sqrt{2})}{2}, \frac{(2-\sqrt{2})}{2})$ and $(\frac{(2+\sqrt{2})}{2}, \frac{(2+\sqrt{2})}{2})$.

In this work, we present a technique to represent and find solutions for multivariate polynomial (in)equation systems. In contrast to numerical approaches,

exact solutions are provided. The decision prodecure does not suffer from numerical instability. However, the procedures have a much greater complexity than numerical approaches solving similar problems. In fact, the basic presented algorithm has doubly exponential complexity. Nevertheless, we sketch techniques to prune the search.
Despite the complexity drawbacks, the procedure has a structure so that it can be extended not only to solve simple polynomial (in)equation systems but also arbitrary Boolean combinations of polynomial (in)equations.

Example 2. Given different polynomial constraints in a boolean structure:

$$\Phi = \exists x \exists y \; (x-1)^2 + (y-1)^2 - 1 = 0 \wedge (x - y = 0 \vee x - y - 1 = 0)$$

Then, the solutions of our example above are models of the formula. Moreover, there are two additional models $(1, 0)$ and $(2, 1)$ satisfying the formula Φ.

1.2 Applications

The techniques presented in this paper can be used for model checking in - for instance - hybrid systems. As an examplary application, we sketch the well know bouncing ball example.

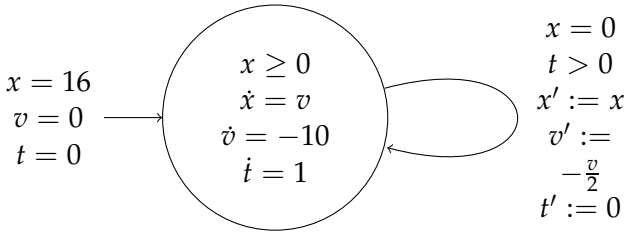

Figure 2 – Hybrid Automaton Model of a bouncing ball.

Example 3. A ball is dropped from an inital height $x_0 = 16$ at time $t_0 = 0$. At this point, its velocity is $v_0 = 0$. The derivate of v_0 is given by -10. From a physical perspective $\dot{v}$ is the accerlation ($-10 \approx 9,81$ which is the gravity of earth). The ball keeps falling and accerlating until it hits the ground. In the hybrid automaton, contact with the ground is modeled with the self loop which is guarded by $x = 0 \wedge t > 0$. The invariant $x \geq 0$ of the only location assures that the discrete transition is taken when the ball reaches the height 0. The ball bounces up in opposite direction with half of the speed. Nevertheless, the accerlation is still negative so the ball will start falling eventually. $x' = x$ denotes that a new variable x_1 is introduced with the value of x_0. The same notation is used to introduce v_1 and t_1.
For a finite number of bounces k we formalize the above presented model with the help of polynomials. The formula consists of constraints denoted by ϕ for the discret part (the self loop) and constraints denoted by ψ for the continous part (the single location).

$$\phi(i) := x_i = 0 \; \wedge \; t_i > 0 \; \wedge \; v_{i+1} = -\frac{v_i}{2} \; \wedge \; x_{i+1} = x_i \wedge t_{i+1} = 0$$

$$\psi(i) := x_{i+1} \geq 0 \wedge t_{i+1} > t_i \ \wedge \ v_{i+1} = v_i - 10t_{i+1} \ \wedge \ x_{i+1} = x_i - 5t_{i+1}^2$$

The constraints for velocity v_i and position of the ball x_i can be deduced from the model with integral calculus. The overall formula to represent k bounces is given by

$$\varphi(k) := x_0 = 16 \ \wedge \ v_0 = 0 \ \wedge \ t_0 = 0 \ \wedge \ \bigwedge_{i=0}^{k} \phi(i) \wedge \psi(i).$$

The formula is a conjunction of polynomial constraints. Consequently, we are able to check whether certain conditions hold for this hybrid system with the help of the techniques presented in this paper. For instance, whether the ball exceeds a certain height x in the i-th bounce:

$$\varphi(k) \wedge x_i \geq x.$$

1.3 Related work

Tarski showed that the first-order theory of real numbers under multiplication and addition is decidable by providing a decision procedure [26]. Nevertheless, this decision procedure has a non-elementary complexity bound. A doubly-exponential time decision procedure called cylindrical algebraic decomposition for real algebra was introduced by Collins in 1974 [8]. There are several improvements of the Collins' basic algorithm [19, 13, 9, 6].
There several implementions of the basic algorithms and its improvements available: *QEPCAD* [5] is a `C++` implementation of the cylindrical algebraic decomposition and its improvements by [13]. *Redlog* [12] is a package of the computer algebra system *Reduce* which is based on `Lisp`. There is also an implementation within the Mathematica computer algebra system [25]. All the mentoined implementations are closed-source projects. For Mathematica we did not find a distributed specification of which algorithms and improvements are implemented.
Apart from cylindrical algebraic decomposition there are other methods to solve polynomial (in)equations systems like Gröbner basis and the virtual substiution method.
There is a project called Z3 with the aim to embed different theory solvers into a SMT framework. Theory solvers for linear arithmetics and array theory are used in this project. [11]

1.4 Contribution

In this work, we present the basic theoretical foundations of the cylindrical algebraic decomposition. We describe all methods necessary to calculate a cylindrical algebraic decomposition for a given polynomial system. Independent of

the actual constraints all possible constraints can be tested on a complete cylindrical algebraic decomposition. Together with this paper, an implementation of the methods was developed within the GiNaCRA framework [17].
There are several methods to represent a solution of the polynomial (in)equation system. The interval representation is a representation of polynomial roots which can be directly transformed into an approximate numeric representation. These approximation can be refined without suffering from numeric instability. It is possible to check the satisfiability of polynomial (in)equation systems with other representations. Nevertheless, a cylindrical algebraic decomposition implementation with interval representation provides both the satisfiability and a possible sample solution which can be approximated in a numeric representation.
Furthermore, we describe and illustrate how this implementation can be used to implement a theory solver which is suited for embedding into an incremental SMT-Solver.
To our knowledge, this is the first implementation of the cylindrical algebraic decomposition with interval representations. Furthermore, GiNaCRA is an extendable open-source `C++` library unlike many other implementations. Moreover, those are often closed-source projects or part of computer algebra systems.

Chapter 2

Real Algebra

In this chapter, the algebraic foundations of the problem and of the presented approach are given. The first section is about the definition and basic notations of polynomials. Subsequently, Section 2.2 which is about basic properties of different fields of polynomials and different field extensions that are necessary to qualify the solutions. In Section 2.3, we present a general decision procedure to isolate roots of univariate polynomials with rational coefficients. This is an important foundation for the techniques presented in Chapter 3. The last two sections are about basic properties of polynomial roots which are necessary for Chapter 3.

2.1 Polynomials

Assume in the following an integral domain $(D, +, \cdot)$, i.e., a commutative ring that has no zero divisors and which is not the trivial ring $D = \{0\}$. Assume furthermore that $(F, +, \cdot)$ is the quotient field of D and that F is the smallest field in which D can be embedded.
An expression

$$f(X) = \sum_{i=0}^{n} a_i X^i$$

is a *polynomial* in X with coefficients a_i in a ring $(D, +, \cdot)$. $D[X]$ denotes the set of *univariate* polynomials in the variable X with coefficients in D. The set of univariate polynomials $D[X]$ forms a ring themself.
The set of *multivariate* polynomials in k variables is defined recursively:

$$D[X_1, X_2 \dots X_n] := D[X_1, X_2 \dots X_{n-1}][X_n]$$

That means, a polynomial in $D[X_1, X_2 \dots X_n]$ is a polynomial in X_n with coefficients in $D[X_1, X_2, \dots X_{n-1}]$. [20, p. 35] We call this variable X_n the *main variable*. We regard multivariate polynomials as parameterized univariate polynomials. Thus, we can apply algorithms and definitions for univariate polynomials to multivariate polynomials.

Let P be a polynomial in $F[X]$:

$$P = a_n X^n + a_{n-1} X^{n-1} + \cdots + a_1 X + a_0.$$

Then, the *degree* n of the polynomial P is denoted by $\deg(P)$. We denote the *leading coefficient* a_n by $\text{lcoeff}(P)$. The *i-th coefficient* a_i is denoted by $\text{coeff}_i(P)$. The *first derivate* P' of the polynomial P is given by

$$P = n a_n X^{n-1} + (n-1) a_{n-1} X^{n-2} + \cdots + a_1.$$

Given two polynomials $P, Q \in F[X]$ we call the polynomials *similar* if there exists an $a \in F$ such that $P = aQ$.
Let $Q \in F[X]$ be another non-zero polynomial. Then A is the *quotient* denoted by $\text{quo}(P, Q)$ and R is the *remainder* $\text{rem}(P, Q)$ in the equation with $\deg(P) < \deg(Q)$

$$P = AQ + R.$$

We say that $Q \in F[X]$ is a *divisor* of $P \in F[X]$ if there exists a polynomial $A \in F[X]$ such that

$$P = AQ.$$

The *greatest common divisor* $\gcd(P, Q)$ of two non-zero polynomials $P, Q \in F[X]$ is the polynomial with maximal degree which is both divisor of P and Q. If not both P and Q are zero then there exists a greatest common divisor which is unique. [3, p. 14ff] [20, p. 35f]
Let $A \in F^{n \times n}$ be a quadratic matrix, then we denote the *determinant* of the matrix by $\det(A) = |A|$. [28]

Definition 1 (Truncation)**.** Let $P = a_n X^n + \cdots + a_0 \in F[x]$. Then for $0 \leq i \leq n$ we define

$$\text{Tru}_i(P) := P = a_i X^i + \cdots + a_0.$$

The *set of truncations* of a polynomial $P \in F[x]$ denoted by $\text{Tru}(P)$ is

$$\text{Tru}(P) := \begin{cases} \{P\} & \text{if } \text{lcoeff}(P) \in F \wedge \deg(P) = 0 \\ \{P\} \cup \text{Tru}(\text{Tru}_{\deg(P)-1}(P)) & \text{otherwise.} \end{cases}$$

[3, p. 21f] [14, p. 19]

2.2 Field extensions

Definition 2. A field is an *algebraic extension* of the field F if every element of L is a root of a polynomial with coefficients in F and if $L \supset F$ is a field extension of F. A field F is called *algebraically closed* if every non-constant univariate polynomial $P \in F[X]$ has a root in F. The minimal algebraically closed extension of a field F is called the *algebraic closure* of F.

Example 4. The field of the real numbers is not algebraically closed. For instance, the polynomial $x^2 + 1$ has no roots in $\mathbb{R}$. An algebraic field extension of the real numbers is the field of the complex numbers $\mathbb{C}$. Indeed, it is the minimal algebraic extension which is algebraically closed.

The focus of this paper is the field of polynomials with coefficients in $\mathbb{Q}$. We present a decision procedure for constraints over rational multivariate polynomials.
The expressive power of the polynomial ring $\mathbb{Q}[X]$ is equivalent to $\mathbb{Z}[X]$. An arbitrary polynomial in $P \in \mathbb{Q}[X]$

$$P = \frac{p_n}{q_n}X^n + \frac{p_{n-1}}{q_{n-1}}X^{n-1} + \cdots + \frac{p_0}{q_0}$$

can be transformed into a polynomial $Q \in \mathbb{Z}[X]$

$$Q = \sum_{i=0}^{n} q_i \cdot P(X)$$

with the same roots.
The field of rational numbers is not algebraically closed. For example $x^2 - 2$ has no root in $\mathbb{Q}$.

Definition 3. The algebraic closure of $\mathbb{Q}$ is called the field of the *algebraic numbers*. The algebraic numbers are those numbers which are a root of a non-zero polynomial with integer coefficients. The set of algebraic numbers is denoted by $\mathbb{A}$. [27]

Example 5. There are both irrational numbers e.g. $\sqrt{2}$ (given by $x^2 - 2$) and complex numbers e.g. i (given by $x^2 + 1$) within the algebraic numbers. Numbers which are not algebraic are called *transcendental* (e.g. π is a real transcendental number because there is no integer polynomial which has a root that equals to π).

Definition 4 (Real field)**.** For a field, F we denote by $F^{(2)}$ the set of squares of elements of F. Furthermore, $\Sigma\, F^{(2)}$ is the set of sums of squares of elements of F. A field F is called *real* if and only if $-1 \notin \Sigma\, F^{(2)}$. A field F is called *real closed* if it is real and every polynomial in $F[X]$ of odd degree has a root in F.

We need this property later on because our basic decision procedure requires a real closed field. The field of the real numbers $\mathbb{R}$ is real closed. In the following assume F to be a real closed field. As this paper focuses on polynomials with integer coefficients it is sufficient to regard a subset of the real numbers:

Definition 5 (Real algebraic numbers)**.** The *real algebraic numbers* $\mathbb{R}_{alg} = \mathbb{A} \cap \mathbb{R}$ are the real numbers which are root of a univariate polynomial with integer coefficients.

The real algebraic numbers form a real closed field. Nevertheless, the field $\mathbb{R}_{alg}$ is not algebraically closed as it does not contain complex roots. For our purpose, the property that they are real closed is sufficient. [20, p. 297ff] [3, p. 29f]

2.3 Sturm sequences

Definition 6 (Signed remainder sequence)**.** Let $P, Q \in F[X]$ be non-zero univariate polynomials then the *signed remainder sequence* $\mathrm{SRemS}(P, Q)$ is a sequence $(\mathrm{SRemS}_0(P, Q), \mathrm{SRemS}_1(P, Q), \ldots \mathrm{SRemS}_k(P, Q))$ of polynomials in $F[X]$ defined by

$$\begin{aligned} SRemS_0(P, Q) &:= && P \\ SRemS_1(P, Q) &:= && Q \\ &\vdots \\ SRemS_k(P, Q) &:= && -\operatorname{rem}(\mathrm{SRemS}_{k-2}(P, Q), \mathrm{SRemS}_{k-1}(P, Q)) \\ 0 &= && -\operatorname{rem}(\mathrm{SRemS}_{k-1}(P, Q), \mathrm{SRemS}_k(P, Q)). \end{aligned}$$

We call $\mathrm{SRemS}(P, P')$ a *Sturm sequence* or *Sturm chain*.

Intuitively, the signed remainder sequence is similar to the Euclidean algorithm for computation of the greatest common divisor. In contrast to the Euclidean algorithm, it has the negative sign in front of the recursive expression. Thus, it differs slightly from the polynomial remainder sequence produced by the Euclidean algorithm.
The algorithm itself has a complexity of $\mathcal{O}(n^2)$ with $n = \max(\deg(P), \deg(Q))$ but it suffers from a significant growth of the coefficients which makes calculations slow.
We introduce the signum function [29] $\operatorname{sgn} : \mathbb{R} \to \{-1, 0, 1\}$ for the set of real numbers:

$$\operatorname{sgn}(x) := \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0. \end{cases}$$

Definition 7 (Number of sign variations)**.** Let $a = (a_1, a_2 \ldots a_n)$ be a sequence of non-zero real numbers. Then the *number of sign variations* $\mathrm{Var}(a)$ is given by

$$\begin{aligned} \mathrm{Var}(a_1) &:= && 0 \\ \mathrm{Var}(a_1, \ldots, a_n) &:= && \begin{cases} \mathrm{Var}(a_1, \ldots, a_{n-1}) & \text{if } \operatorname{sgn}(a_{n-1}) = \operatorname{sgn}(a_n) \\ \mathrm{Var}(a_1, \ldots, a_{n-1}) + 1 & \text{otherwise.} \end{cases} \end{aligned}$$

If $\mathcal{P} = (P_1, \ldots, P_n)$ is a sequence of polynomials in $F[X]$ and $a \in F$ then $Var(\mathcal{P}; a) := \mathrm{Var}(P_1(a), \ldots, P_n(a))$. Furthermore, we define $\mathrm{Var}(\mathcal{P}; a, b) := \mathrm{Var}(\mathcal{P}; a) - \mathrm{Var}(\mathcal{P}; b)$.

Theorem 1 (Sturm's theorem)**.** *Let F be a real closed field, $a, b \in F$, $a < b$ and $P \in F[X]$ be a non-zero polynomial with $P(a) \neq 0$ and $P(b) \neq 0$. Then*

$$\mathrm{Var}(\mathrm{SRemS}(P, P'); a, b))$$

is the number of roots of P in F in the interval (a, b).

Theorem 2 (Cauchy bound)**.** [7, p. 122ff] [20, p. 309ff] [3, p. 52] *Let F be a real closed field for every polynomial $P = a_pX^p + \cdots + a_qX^q \in F[X], p > q, a_qa_p \neq 0$ every real root z of P is bounded by*

$$|z| < \sum_{q \leq i \leq p} \left| \frac{a_i}{a_p} \right|.$$

This bound is called the Cauchy bound.

The Cauchy bound provides an upper and lower bound for the search space for real roots of polynomials in a real closed field. However, the search space consists of infinitely many real algebraic numbers. Nevertheless, we can use Sturm's theorem to divide the search space: The basic idea is to divide the interval $(-z, z)$ into different sub intervals until each interval contains exactly one root. Details on the algorithm are not part of this paper but we introduce a function isolateRoots$(P) := (a_1, b_1),\ \ldots\ (a_n, b_n)$ which gives us a sequence of disjoint intervals each containing exactly one root of the polynomial P. Furthermore, countRoots(P) denotes the number of distinct roots of P and the left (respectively right) limit of an open interval o is denoted by o. left (respectively o. right). Additionally, for a finite set $\mathcal{P} \subset F[X]$ of polynomials we define

$$\text{isolateRoots}(\mathcal{P}) := \bigcup_{P \in \mathcal{P}} \text{isolateRoots}(P)$$

and

$$\text{countRoots}(\mathcal{P}) := \left| \bigcup_{P \in \mathcal{P}} \text{isolateRoots}(P). \right|$$

To compute the above sets of roots, we must be able to test two real algebraic numbers for equality. There is an algorithm to compute the additive inverse of a real algebraic number [20, p. 331] and an algorithm to compute the sum of two real algebraic numbers [20, p. 332]. To check whether two real algebraic numbers α, β are equal, we compute the additive inverse $-\alpha$ of α and check whether $-\alpha + \beta = 0$.

Example 6. Given the polynomial $P := X^2 - 3X + 1 \in \mathbb{Q}[X]$. The Sturm sequence SRemS(P, P') is

$$\begin{aligned} &X^2 - 3X + 1 \\ &2X - 3X \\ &\frac{5}{4} \end{aligned}$$

The Cauchy bound is given by

$$\left|\frac{1}{1}\right| + \left|\frac{-3}{1}\right| + \left|\frac{1}{1}\right| = 5.$$

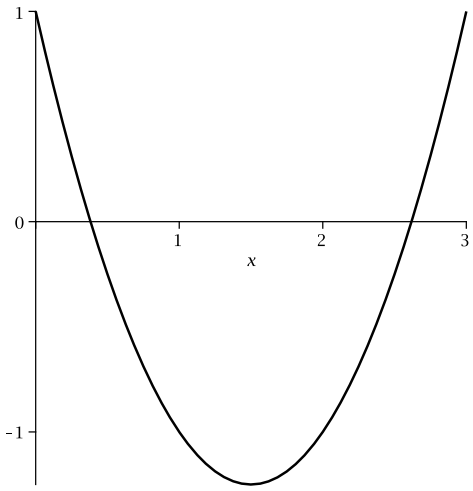

Figure 3 – The polynomial $X^2 - 3X + 1$.

Thus, $(-5, 5)$ is the maximal interval to search for real roots. We compute $\text{Var}(\text{SRemS}(P, P'), -5, 5)$ to determine the number of real roots in P:.

$$\begin{aligned}\text{Var}(\text{SRemS}(P, P'); -5, 5) &= \text{Var}(\text{SRemS}(P, P'), -5) - \text{Var}(\text{SRemS}(P, P'), 5)) \\ &= 2\end{aligned}$$

The sign variations of $(-5, 5)$ are bigger than 1. Therefor, we split $(-5, 5)$ into sub-intervals $(-5, 0)$ and $(0, 5)$, and compute the sign variations of $(-5, 0)$ and $(0, 5)$. Note that 0 is not a root of P. This procedure is repeated until all intervals have a sign variation of 1 or 0.

2.4 Representation of roots

As we have seen in the section on Sturm sequences above, our decision procedure does not compute a numeric representation of the roots. Instead, we compute an interval for each real root with bounds in $\mathbb{Q}$ containing the root. This corresponds to the representation of real algebraic numbers we use in this paper:

Definition 8 (Interval representation)**.** A pair (P, o) with $P \in F[X]$ and o an interval (a, b) with $a < b$, $a, b \in \mathbb{Q}$ and $\text{Var}(\text{SRemS}(P, P'); a, b) = 1$ is called *interval representation* and represents a real algebraic number. In addition, if $P(a) = 0$ then the real algebraic number a is represented by $(P, [a, a])$. Note that $[a, a]$ is a closed interval containing a. For convenience (and for memory efficiency), we write the value of $a \in F$ called *numeric representation*.

There are other representations of real algebraic numbers like *radical expression*. However, in general the roots of a polynomial with integer coefficients are not representable with radical expressions [2].

In addition, there are other representations which are capable to represent all real algebraic numbers like sign representation or order representation. [20, p. 327] In this work, we present all techinques and algorithms only for interval representations.
We can easily refine an interval representation of a root given by a interval and the polynomial. The presented algorithm is a combination of the commonly known *bisection algorithm* [22, p. 250ff] and a special case treat with Sturm sequences [20, p. 329ff]. We try to avoid the calculation of sign variations in the Sturm sequence and we try to avoid the calculation of the Sturm sequence itself. The complexity of the calculation of the Sturm sequence is in $\mathcal{O}(\deg(P)^4)$. However, the growth of the coefficients in the Sturm sequence is significant. [14, p. 11ff] Thus, even the evaluation of the Sturm sequence at different points can be a hard task. The problem is discussed later on in the section about subresultants.
To avoid unnecessary computations like refinements, we provide procedures to intelligently guess a numeric representation of roots (like integers or fractions with small numerator and denominators) whenever possible.

Algorithm 1 [Refine]: For an interval representation of a real algebraic number calculate a representation $(P, (a', b'))$ with $|a' - b'| < \varepsilon$.

Require: $P \in \mathbb{Z}[X], \varepsilon > 0, a < b$ and $\varepsilon, a, b \in \mathbb{R}$
Output: Interval representation $\alpha = (P, (a', b')$ with $a \leq a' \leq b' \leq b$ and $|b' - a'| \leq \varepsilon$ or $(P, [c, c])$ with $P(c) = 0$

```
sturmSequence ← 0
while |a − b| > ε do
  c ← (a+b)/2
  if P(c) = 0 then
    return (P, [c, c])
  end if
  if P(a)P(b) < 0 then
    if P(a)P(c) < 0 then
      b ← c
    else
      a ← c
    end if
  else
    if sturmSequence = 0 then
      sturmSequence ← calculateSturmSequence(P)
    end if
    if Var(sturmSequence; a, c) = 1 then
      b ← c
    else
      a ← c
    end if
  end if
end while
```

The output of the algorithm is either an exact numeric representation of the root or a refined interval representation.

2.5 Properties of polynomial roots

Lemma 1 (Common roots)**.** *Let $P, Q \in F[X]$ be two non-zero polynomials. Then the number $c \in \mathbb{N}_{\geq 0}$ of P and Q in F of common roots is given by*

$$c = \deg(\gcd(P, Q)).$$

Proof. Assume $p_1, \ldots, p_s$ are the distinct roots of P and $q_1, \ldots, q_t$ are the distinct roots of Q. If $s = 0$ and $t = 0$ then $\gcd(P, Q)$ is a real number and thus $\deg(\gcd(P, Q)) = 0$. Otherwise, P and Q can be written in the form

$$P = (X - p_1)^{a_1}(X - p_2)^{a_2} \ldots (X - p_s)^{a_s} \cdot z_p$$
$$Q = (X - q_1)^{b_1}(X - q_2)^{b_2} \ldots (X - q_t)^{b_t} \cdot z_q$$

where $a_1, \ldots, a_s$ and $b_1, \ldots, b_t$ are the multiplicities of the roots and $z_p, z_q \in F, z_p \neq 0, z_q \neq 0$. Moreover, the greatest common divisor can be written in the form

$$\gcd(P, Q) = (x - r_1)^{c_1} \ldots (x - r_u)^{c_u} \cdot z$$

with $z \in F, z \neq 0, \{r_1, \ldots, r_u\} = \{p_1, \ldots, p_s\} \cap \{q_1, \ldots q_t\}$ and for each $r_i = p_j = q_k$ and $i, j, k \in \{1, \ldots, n\}$ we have $c_i = \min(a_j, b_k)$. We see that the greatest common divisor's degree is the number of the common roots of P and Q. [14, p. 10] [3, p. 355] □

Lemma 2 (Distinct roots)**.** *Given a polynomial $P \in F[X]$, the number of distinct roots of P in the algebraic closure of F is*

$$d = \deg(P) - \deg(\gcd(P, P')).$$

Proof. If $z_1, \ldots, z_n$ are the distinct roots of the polynomial P and $\mu_1, \ldots, \mu_n$ are their multiplicities, we see that

$$P = z \cdot \prod_{i=1}^{n} (x - z_i)^{\mu_i}.$$

Moreover, we rearrange the derivate P':

$$\begin{aligned} P' &= z \cdot \Big(\sum_{i=1}^{n} (x - z_i)^{\mu_i - 1} \prod_{i=1}^{n} \prod_{j \neq i} (x - z_j)^{\mu_j}\Big) \\ &= z \cdot \Big(\sum_{i=1}^{n} \Big(\prod_{j=1}^{n} (x - z_j)^{\mu_j - 1}\Big)\Big(\prod_{j=1}^{n} \prod_{j \neq i} (x - z_j)\Big)\Big) \\ &= z \cdot \Big(\prod_{i=1}^{n} (x - z_i)^{\mu_i - 1}\Big) \cdot \Big(\sum_{i=1}^{n} \Big(\prod_{j=1}^{n} \prod_{j \neq i} (x - z_j)\Big)\Big) \end{aligned}$$

Therefore, we see

$$\gcd(P, P') = (X - z_1)^{\mu_1 - 1} \dots (X - z_n)^{\mu_n - 1} \cdot z$$

with $z \in F$. Moreover, $\deg(P) = \sum_{i=1}^{n} \mu_i$ and $\deg(\gcd(P, P')) = \sum_{i=1}^{n} (\mu_i - 1)$, thus we see that $d = n$, i.e., d is the number of distinct roots of P. [14, p. 10f] [3, p. 355] □

The two lemmas above give us a way to handle univariate polynomials. Nevertheless, they can not be directly applied to parameterized univariate polynomials.

Example 7. Let $P \in \mathbb{Z}[a, b][X]$ be the following polynomial:

$$(a - b)X^3 + aX^2 + bX + 1.$$

It is not possible to state a general proposition on the number of roots of P in $\mathbb{A}$ knowing the values of a and b. Therefor, we have to distinguish between three cases:

- $a \neq b$: $\deg(P) = 3$
- $a = b$ and $a \neq 0$: $\deg(P) = 2$
- $a = b = 0$: $\deg(P) = 0$

The total number of roots depends on the parameters.

The above lemmas are essential for the cylindrical algebraic decomposition. An efficient way to calculate the greatest common divisor of two polynomials is necessary. For two polynomials P, Q the k-th element in the signed remainder sequence $(\mathrm{SRemS}_0(P, Q), \dots \mathrm{SRemS}_k(P, Q))$ is the greatest common divisor of P and Q. In fact, the elements of the signed remainder sequence are similar up to signs to the polynomials produced by the well-known Euclidean algorithm. [3, p. 16]

Example 8. Given a polynomial $P = X^7 + X^6 - X^5 + 1$, the signed remainder sequence of P and P' is

$$\begin{aligned}
&X^7 + X^6 - X^5 + 1 \\
&7X^6 + 6X^5 - 5X^4 \\
&\frac{20}{49}X^5 - \frac{5}{49}X^4 - 1 \\
&\frac{49}{16}X^4 - \frac{343}{20}X - \frac{1519}{80} \\
&-\frac{16}{7}X^2 - \frac{96}{49}X + \frac{80}{49} \\
&\frac{799}{35}X + \frac{4429}{280} \\
&-\frac{4833409}{2553604}
\end{aligned}$$

The bit size of the coefficients in this example grows significantly. This has several drawbacks: The memory consumption of the sequence grows, the calculations necessary to compute the sequence are harder and even the evaluation of the sequence for specific numbers is harder. Thus, we need a way to handle $\deg(\gcd(P, Q))$ easier. [14] [3, p. 302ff]

Definition 9 (Signed subresultants sequence)**.** Let $P, Q \in F[X]$ be two polynomials

$$P := a_m X^m + a_{m-1} X^{m-1} + \cdots + a_0$$

$$Q := b_n X^n + b_{n-1} X^{n-1} + \cdots + b_0$$

with $m > n$. Then, we define the *signed subresultants sequence*

$$(\mathrm{sResP}_0(P, Q), \ldots \mathrm{sResP}_n(P, Q))$$

as follows:

$$\mathrm{sResP}_0(\mathrm{P}, \mathrm{Q}) := \begin{vmatrix} a_m & \ldots & a_1 & a_0 & & & X^{n-1}P \\ & \ddots & \ddots & \ddots & \ddots & & \vdots \\ & & a_m & \ldots & a_1 & a_0 & XP \\ & & & a_m & \ldots & a_1 & P \\ b_n & \ldots & b_1 & b_0 & & & X^{m-1}Q \\ & \ddots & \ddots & \ddots & \ddots & & \vdots \\ & & b_n & \ldots & b_1 & b_0 & XQ \\ & & & b_n & \ldots & b_1 & Q \end{vmatrix}.$$

For all $0 < i < n$

$$\mathrm{sResP}_\mathrm{i}(\mathrm{P}, \mathrm{Q}) := \begin{vmatrix} a_m & \ldots & a_1 & a_0 & & & X^{n-i-1}P \\ & \ddots & \ddots & \ddots & \ddots & & \vdots \\ & & a_m & \ldots & a_{i+1} & a_i & XP \\ & & & a_m & \ldots & a_{i+1} & P \\ b_n & \ldots & b_1 & b_0 & & & X^{m-i-1}Q \\ & \ddots & \ddots & \ddots & \ddots & & \vdots \\ & & b_n & \ldots & b_{i+1} & b_i & XQ \\ & & & b_n & \ldots & b_{i+1} & Q \end{vmatrix}.$$

In addition, we define $\mathrm{sResP}_n(P, Q) = P$. We call $\mathrm{sResP}_i(P, Q)$ the *i-th subresultant* for P and Q. Moreover, the *i-th principal subresultant coefficient* denoted by $\mathrm{sRes}_i(P, Q)$ is $\mathrm{lcoeff}(\mathrm{sResP}_i(P, Q))$. [20, p. 250ff] [3, p. 355ff]

We will not describe an algorithm to compute signed subresultants in this paper. Nevertheless, we provide an implementation of a fast signed subresultants computation algorithm described in [16]. The complexity is bounded by $\mathcal{O}(\deg(P) \deg(Q))$.

Example 9. Given a polynomial $P = X^7 + X^6 - X^5 + 1$ and $P' = X^6 + 6X^5 - 5X^4$ from the example above, the signed subresultants sequence for P and P' is

$$\begin{aligned}
&X^7 + X^6 - X^5 + 1\\
&7X^6 + 6X^5 - 5X^4\\
&20X^5 - 5X^4 - 49\\
&25X^4 - 140X - 155\\
&-175X^2 - 150X + 125\\
&-1225X^2 - 1050X + 875\\
&63920X + 44290\\
&-2761948.
\end{aligned}$$

Even though, the coefficients are bigger than the coefficients of the signed remainder sequence in the example above, the bit size is much smaller. In fact, the bit size of the coefficients for the signed subresultants sequence of two polynomials $P, Q \in \mathbb{Z}[X]$ is bound by $\mathcal{O}((\tau + \rho)(p + q))$ where p, q are the degrees of the polynomials, τ is the maximum bit size of coefficients in P and Q and ρ is the bit size of $p + q$. [3, p. 315]
As the signed subresultants sequence's elements are similar to the elements of the signed remainder sequences, it can be substituted for many applications. For example, if not the exact coefficients but the degree of certain elements in the sequence are of interest.
However, the calculation of the signed remainder sequence to obtain the number of sign variants in a certain interval (Sturm's theorem) cannot be avoided as the exact evaluation of the sequence's elements is necessary.

Lemma 3. *[14] Let $P, Q \in F[X]$ be two polynomials with $\deg(P) > \deg(Q)$. Then $\deg(\gcd(P, Q)) \geq j$ with $0 \leq j \leq deg(Q)$ if and only if*

$$\mathrm{sRes}_0(P, Q) = \mathrm{sRes}_1(P, Q) = \cdots = \mathrm{sRes}_{j-1}(P, Q) = 0.$$

Consequently, subresultants can express both properties from Lemma 1 and 2. This method does not suffer from the bit-size growth of the coefficients as much as the gcd-calculation with the signed remainder sequence. This is the fundamental basis for the cylindrical algebraic decomposition.

Chapter 3

Cylindrical algebraic decomposition

In this chapter, we describe the decision procedure for rational multivariate polynomials. It is an extension which uses the Sturm/Cauchy decision procedure for rational univariate polynomials as a base. The basic idea is to project the multivariate polynomials to a set of univariate polynomials in a certain way and to interpret the results to obtain multivariate solutions.
The first section is about basic notations to define structures which are necessary for the proof presented in the next section and the description of the algorithm in the section thereafter.
In the following, F is a real closed field. The provided algorithm is implemented for the field $\mathbb{Q}$ and its real closure $\mathbb{R}_{alg}$. Nevertheless, we want to describe the methods more general for real closed fields because most of the methods can be applied to other fields and their algebraic closures.

3.1 Preliminaries

Definition 10 (Semi-algebraic set)**.** Let F be a real closed field. Then, we call $S \subset F^k$ a *semi-algebraic subset* [3, p. 57ff] of F^k if it can be constructed from finitely many applications of union, intersection and complementation of sets of the form

$$\{x \in F^k \mid P(x) \geq 0\}$$

for some $P \in F[X_1, \ldots, X_k]$. Each semi-algebraic set can be constructed as a finite union of sets of the form

$$\{x \in F^k \mid P(x) \Diamond 0\}$$

with $\Diamond \in \{=, \neq, <, \leq, >, \geq\}$ and $\mathcal{P} \in F[X_1, \ldots X_k]$.

We use a canonical projection denoted by π and defined as follows:

Definition 11 (Canonical Projection)**.** Given a semi-algebraic set

$$S = \{(x_1, \ldots, x_{k+1}) \in F^{k+1} \mid P(x_1, \ldots x_{k+1}) \Diamond 0\}.$$

for some $P \in F[x_1, \ldots, x_{k+1}]$ and $\Diamond \in \{=, \neq, <, \leq, >, \geq\}$. Then the canonical projection [3, p. 57ff] of S is

$$\pi(S) = \{(x_0, x_1, \ldots, x_k) \in F^k \mid \exists x_{k+1} \in F \; P(x_1, \ldots, x_{k+1}) \Diamond 0\}.$$

Example 10. For instance the semi-algebraic set

$$C = \{(x, y) \in \mathbb{R}_{alg} \mid (x-2)^2 + (y-2)^2 - 1 \leq 0\}$$

has the canonical projection

$$\pi(C) = \{x \in \mathbb{R}_{alg} \mid 1 \leq x \leq 3\}.$$

Figure 4 illustrates the sets C and $\pi(C)$.

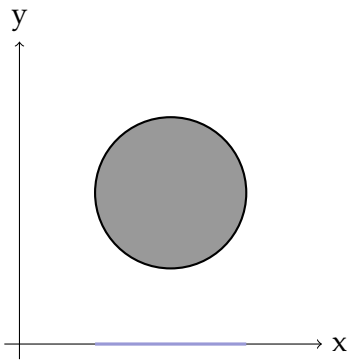

Figure 4 – The set C and its projection $\pi(C)$ (in blue).

Given a set $S \subset F^k$ for some $k \in \mathbb{N}$ we call $S \times F$ the *cylinder* of S. Figure 5 illustrates the cylinder of the set $\pi(C)$ from the previous example. Applied to the example above, the cylinder of $\pi(C)$ is not a circle but an infinite rectangular shape given by $\{(x, y) \in \mathbb{R}_{alg} \mid 1 \leq x \leq 3\}$.

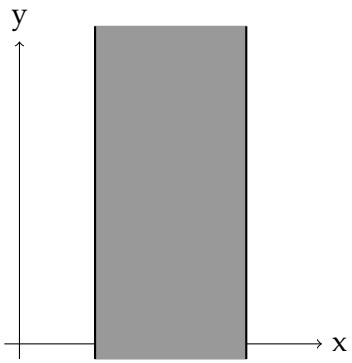

Figure 5 – The cylinder of $\pi(C)$.

Definition 12 (Decomposition)**.** Given $S \subset F^k$ then we call a finite sequence of disjoint subsets $S_1, \ldots S_n \subset S$ a *decomposition* of S if

$$S = \bigcup_{i=1}^{n} S_i \ \wedge \ \forall i, j \in \{1, \dots n\}\ S_i \cap S_j \neq \emptyset \Rightarrow i = j.$$

A decomposition is *semi-algebraic* if its components are semi-algebraic sets.

Definition 13 (Stack)**.** Assume a real closed field $F, k \in \mathbb{N}_{>0}$, a set $R \subset F^k$ and a cylinder $S = R \times F$. A *stack* over R is a finite sequence of $f_1, \dots f_n$ functions of type $R \to S$ with $f_i(x) < f_{i+1}(x)$ for all $x \in F$ for all $1 \leq i < n$. The functions $f_1, \dots, f_n$ define a decomposition of the cylinder S into $2n + 1$ subsets:

- n subsets of the form

$$\{(x, y) \in R \times F \mid y = f_i(x)\} \subset S$$

- $n - 1$ subsets of the form

$$\{(x, y) \in R \times F \mid f_i(x) < y < f_{i+1}(x)\} \subset S$$

- 1 subset of the form

$$\{(x, y) \in R \times F \mid y < f_1(x)\} \subset S$$

- 1 subset of the form

$$\{(x, y) \in R \times F \mid f_n(x) < y\} \subset S$$

Example 11. To understand the concept of stacks, we define three functions which decompose the cylinder of the open interval $(0, 1)$:

$$f_1 : (0, 1) \to \mathbb{R}_{alg}, x^4 - \frac{1}{2}$$

$$f_2 : (0, 1) \to \mathbb{R}_{alg}, -x^2 + x + \frac{1}{2}$$

$$f_3 : (0, 1) \to \mathbb{R}_{alg}, -x^4 + x + 1$$

Due to the definition of stacks, we have 7 disjoint subsets of $(0, 1) \times \mathbb{R}_{alg}$: Three subsets defined by the graphs of the functions, two subsets which are between f_1 and f_2, and between f_2 and f_3, and the two subsets of points which are above the graph of f_1 and below the graph of f_3. Even though $f_2(1) = f_1(1)$, it is a valid stack because $(0, 1)$ is an open interval. We see that the decomposition induced by the stack is semi-algebraic. Figure 6 illustrates the example.

Definition 14 (Cylindrical decomposition)**.** [14] A finite sequence $\mathcal{S} = (S_1, S_2 \dots S_k)$ is a *cylindrical decomposition* of F^k if

- S_1 is a decomposition of F which only consists of open intervals and points

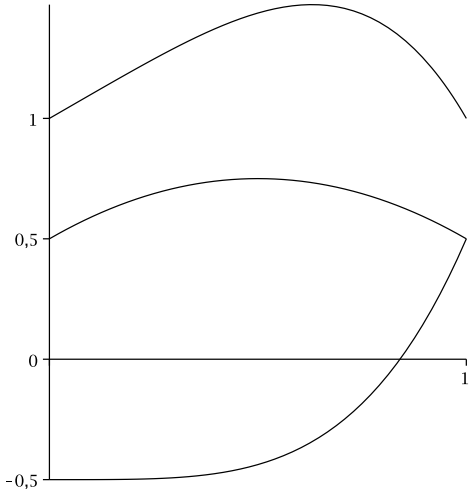

Figure 6 – The stack (f_1, f_2, f_3) over $(0, 1)$.

- For each $1 < i \leq k$, let S_{i-1} be the decomposition $S_{i-1,1}, \ldots, S_{i-1,\mu_{i-1}}$ of F^{i-1}. Then for each $i \leq j \leq \mu_{i-1}$ there is a stack defining a decomposition of $S_{i,j}$ of $S_{i-1,j} \times F$.

Cylindrical decompositions induce a tree structure into multidimensional sets which we make use of later on when we present algorithms to generate decompositions.

Example 12. We decompose the set $\mathbb{R}^2$ into subsets in the above mentioned cylindrical way. Initially, we decompose the 1-dimensional set $\mathbb{R}$:

$$\mathbb{R} = (-\infty, 0) \cup 0 \cup (0, \infty)$$

We decompose $\mathbb{R}^2$ into subsets defined by the stack only consisting of the function $f(x_1) = x_1$. The resulting decomposition is visualized in Figure 7. The root node, represents the empty decomposition of the 0-dimensional space.

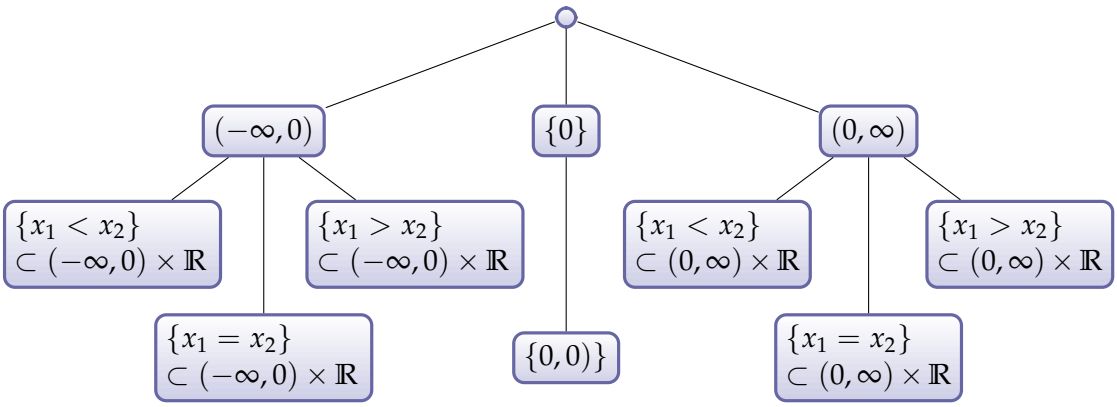

Figure 7 – Decomposition of $\mathbb{R}^2$.

All leaf sets as well as all other sets are semi-algebraic.

Definition 15 (Cylindrical algebraic decomposition)**.** A decomposition of F^k is called a *Cylindrical algebraic decomposition* (CAD) if it is cylindrical and semi-algebraic. The components of the CAD are called *cells*.

The decomposition in Example 12 is a CAD. However, it does not have a connection to a polynomial (in)equation system. The properties above are formal requirements for the solution space. In addition, we need another property to connect the formal requirements with the polynomial system:

Definition 16 ($\mathcal{P}$-invariant)**.** Assume a finite set $\mathcal{P} \subset F[X_1, X_2 \dots X_k]$. Then a set $S \subset F^k$ is *$\mathcal{P}$-invariant* if

$$\forall P \in \mathcal{P}\ \forall x, y \in S\ \operatorname{sgn}(P(x)) = \operatorname{sgn}(P(y)).$$

A CAD $\mathcal{S} = S_1, S_2 \dots S_k$ of F^k is $\mathcal{P}$-invariant if the cells of S_k are $\mathcal{P}$-invariant.

[14] [3, p. 159ff] [20, p. 348ff]

3.2 Proof of existence

The main purpose of this section is to proof the following theorem:

Theorem 3 (Existence of a CAD)**.** *For each finite set $\mathcal{P} \subset F[X_1, \dots X_k]$ of polynomials there is a CAD of F^k which is $\mathcal{P}$-invariant.*

We intend to proof the theorem by induction on the dimension k of the set to decompose. Another requirement for the proof is a construction to aid the proof of the existence of a $\mathcal{P}$-invariant CAD of F^k with the assumption that a CAD of F^{k-1} for a certain set of polynomials is given.

Definition 17 (Elimination)**.** Given a finite set $\mathcal{P} \subset F[X_1, \dots X_{k-1}][X_k]$ of polynomials we define $\operatorname{Elim}_{X_k}(\mathcal{P}) \subset F[X_1, \dots X_{k-1}]$ consisting of the following elements:

- For all $T \in \operatorname{Tru}(\mathcal{P})$, $\operatorname{lcoeff}(T)$.
- For all $P \in \mathcal{P}$ with $\deg(P) \geq 2$, for all $T \in \operatorname{Tru}(P)$, for $0 \leq i \leq \deg(T) - 2$, $\operatorname{sRes}_i(T, T')$.
- For all $P, Q \in \mathcal{P}$
  - If $\deg(P) > \deg(Q)$, for all $0 \leq i \leq \deg(Q)$, $\operatorname{sRes}_i(P, Q)$,
  - If $\deg(P) < \deg(Q)$, for all $0 \leq i \leq \deg(Q)$, $\operatorname{sRes}_i(Q, P)$,
  - If $\deg(P) = \deg(Q)$, for all $0 \leq i \leq \deg(\operatorname{lcoeff}(P)Q - \operatorname{lcoeff}(Q)P)$, $\operatorname{sRes}_i(P, \operatorname{lcoeff}(P)Q - \operatorname{lcoeff}(Q)P))$.

This construction encodes properties of the roots of the polynomials in $\mathcal{P}$ in a set of lower-dimensional polynomials. Subsequently, we can remove redundant polynomials such as constant polynomials. Furthermore, we remove polynomials which are similar to another polynomial. [3][p. 405]

Example 13. Given $\mathcal{P} := \{X^2 + Y^2 + 1;\ X + Y^3 - 1\} \subset \mathbb{Z}[Y][X]$ we calculate the set $\text{Elim}_X(\mathcal{P})$:
$\text{sRes}_0(X^2 + Y^2 + 1,\ 2X) = 4y^2 + 4$ thus the set contains the similar polynomial

$$y^2 + 1.$$

In addition, we insert the polynomial $\text{sRes}_0(X^2 + Y^2 + 1,\ X + Y^3 - 1)$ given by

$$y^6 - 2y^3 + y^2 + 2$$

and

$$-1 + y^3$$

into the set $\text{Elim}_X(\mathcal{P})$.

Lemma 4. *Let $\mathcal{P} \subset F[X_1, X_2 \dots X_{k-1}][X_k]$ be a finite set of polynomials and $S \subset F^{k-1}$ be a semi-algebraic subset which is $\text{Elim}_{X_k}(\mathcal{P})$-invariant. Then, the degree $\deg(P(x))$ and the number of distinct roots $\deg(P(x) - \deg(\gcd(P(x), P'(x)))$ is constant for all $x \in S$, $P \in \mathcal{P}$. Furthermore, the number of common roots $\deg(\gcd(P(x), Q(x)))$ is constant for all $P, Q \in \mathcal{P}$ and $x \in S$.*

Proof. The polynomials $T \in \text{Tru}(P)$ for all $P \in \mathcal{P}, \text{lcoeff}(T)$ fix the degree of each $P \in \mathcal{P}$ over S. If there is a maximal j with $\text{lcoeff}(\text{Tru}_j(P)) = 0$ then $\deg(P) = \deg(\text{Tru}_j(P))$ over S.
The polynomials $P \in \mathcal{P}$, for all $T \in \text{Tru}(P)$, $\text{sRes}_i(T, T')$ fix the number of distinct roots over S (c.f. Lemma 1 and Lemma 3). In addition, we insert all truncations T of polynomials P in case the leading coefficient of P (or of one of the truncations) is zero (cf. Example 7). Moreover, it is not necessary to fix the number of distinct roots for polynomials or truncations with degree smaller than 2 because in this case the number of distinct roots is always constant.
The polynomials $P, Q \in \text{Tru}(\mathcal{P})$, $\deg(P) \neq \deg(Q)$, $\text{sRes}_i(P, Q)$ fix the number of common roots of P and Q. Analogously to the case above, we have to encode the property for all combinations of truncations of P and Q. Regarding Lemma 1 and 3, we see that the construction is correct. There are maximal $\deg(P)$ common roots. Thus, it is sufficient to add the first $\deg(P) - 1$ subresultants coefficients to the set.
In summary, the degree and the number of distinct roots is fixed for every $P \in \mathcal{P}$ over S, and the number of common roots for polynomials $P, Q \in \mathcal{P}$ is fixed over S. □

Lemma 5. *Let $\mathcal{P} \subset F[X_1, \dots X_k]$ be a finite set of polynomials and $S \subset F^{k-1}$ be a semi-algebraic subset with constant degrees, distinct roots and common roots of polynomials in $\mathcal{P}$ over S. Then, for the cylinder $S \times F$ of S there is a semi-algebraic decomposition defined by the functions $f_1 < \dots < f_r : S \to F$. Given $x' \in S$ then the set $\{f_1(x'), \dots f_r(x')\}$ is exactly the set of roots of polynomials $P \in \mathcal{P}$. The sets*

- $\{(x', f_i(x')) \in S \times F\}$
- $\{(x', y) \in S \times F \mid f_i(x') < y < f_{i+1}(x')\}$

- $\{(x', y) \in S \times F \mid y < f_1(x')\}$
- $\{(x', y) \in S \times F \mid f_n(x') < y\}$

are semi-algebraic and $\mathcal{P}$-invariant.

Proof. Wlog we distinguish between two cases:

- Given a polynomial $P \in \mathcal{P}$ with distinct roots $z_1 \dots z_r$ for a fixed $x' \in S$. We write $P(x', x)$ as

$$P = (x - z_1)^{\mu_1} \dots (x - z_t)^{\mu_t}.$$

  Thus, $z_1 \dots z_t$ are the roots with multiplicities $\mu_1 \dots \mu_t$ of P for a fixed $x' \in S$. We define sets $D_i := \{w \in S \mid |w - z_i| < r_i\} \subset S$ which are commonly known as the *open disk* centered at z_i of radius r_i for each $1 \leq i \leq t$. For every z_i we choose r_i such that there is exactly one root of multiplicity μ_i of $P(x', x)$ in D_i. In addition, the radiuses r_i are chosen such that the disks D_i are disjoint for all $1 \leq i \leq t$. Moreover, there are continuous functions $f_i : S \to F$ such that for $x' \in S$ $f_i(x')$ is a root of P for every $1 \leq i \leq t$. Furthermore, the sets of the form $\{(x', f_i(x')) \in S \times F\}$ are P-invariant. For every $1 \leq i < t$ the sign of $P(a)$ with $a \in (z_i, z_{i+1})$ is constant. Subsequently, the other sets are also P-invariant.

  As S is a semi-algebraic set, it can be expressed with a FO-formula $\Phi(x')$ over the domain of real algebraic constraints. Furthermore, we can express the set $\{(x', f_i(x')) \in S \times F\}$ an FO-formula Ψ:

$$\begin{aligned}\Psi(X_k) =& \Phi(X') \wedge \exists\, Y_1\, \exists\, Y_2\, \dots \exists\, Y_t\, (\, Y_1 < Y_2 < \dots Y_t \wedge \\ & \forall\, Y\, (P(X', Y) = 0 \Rightarrow (Y = Y_1 \vee Y = Y_2 \vee \dots Y = Y_t \wedge \\ & X_k = Y_i))).\end{aligned}$$

  There is an equivalent quantifier-free formula Ψ' [3][p. 70] [20][p. 361] [23]. Thus, the set expressed by Ψ' (and Ψ) is semi-algebraic. A similar construction works for the other sets. Hence, we deduce that the above mentioned sets are semi-algebraic and defined by continuous functions.

- Given two polynomials $P, Q \in \mathcal{P}$. Then we denote the distinct roots of $PQ(x', x)$ by $z_1 \dots z_t$ for a fixed $x' \in S$.

  The multiplicities of $z_1 \dots z_t$ for $P(x', x)$ are $\mu_1 \dots \mu_t$ respectively $\nu_1 \dots \nu_r$ for $Q(x', x)$. Note that if z_i is not a common root of P and Q with $1 \leq i \leq t$ then either $\mu_i = 0$ or $\nu_i = 0$.

  We define the sets $D_i := \{w \in S \mid |w - z_i| < r_i\}$ and choose r such that there is exactly one root of PQ with multiplicity $\mu_i + \nu_i$ in D_i and such that all disks are disjoint for $1 \leq i \leq t$.

  Subsequently, there are $s \leq t$ continuous functions f_i defining the roots with $\min(\mu_i, \nu_i) > 0$ for $1 \leq i \leq t$ of PQ in S. For all other $s - t$ functions

the construction above can be applied. The resulting sets $\{(x', f_i(x')) \in S \times F\}$ are both P- and Q-invariant. We can use an analog construction to encode the sets $\{(x', f_i(x')) \in S \times F\}$ with FO-formulas over the domain of real algebraic constraints. Consequently, we can use the same argument to proof that the sets are semi-algebraic.

The proof can be extended to finite sets of polynomials of arbitrary size by induction on the number of elements. □

Proof of Theorem 3. [3, p. 164ff] The proof is by induction of the dimension k on the set to decompose. The induction basis is the case $k = 1$: We know that there is a decomposition of F into subsets

$$F = F_1 \cup F_2 \cup \cdots \cup F_m$$

such that

$$\forall 1 \leq i \leq m \, \forall P \in \mathcal{P} \, \forall x, y \in F_i \; \text{sgn}(P(x)) = \text{sgn}(P(y)).$$

The decomposition consists of the open intervals (sgn $\neq$ 0) and points (sgn $= 0$).
The induction hypothesis is that there is a $\text{Elim}_{X_k}(\mathcal{P})$-invariant CAD $\mathcal{S}'$ of F^{k-1}. For every leaf cell C in $\mathcal{S}'$ there is a stack of functions which decomposes C into $\mathcal{P}$-invariant cells (c.f. Lemma 4 and 5). Thus, there is a $\mathcal{P}$-invariant CAD of F^k. □

3.3 Algorithm

In this section, an algorithm [14] which calculates a $\mathcal{P}$-invariant CAD for a given set of multivariate polynomials $\mathcal{P}$ is presented. First, we sketch the algorithm abstractly so that the reader gets an overview of the techniques used in the algorithm.
The algorithm consists of three phases. We first give a brief description of the purpose of the phases and afterwards we see how they interact to calculate the CAD.

- Elimination phase

  Given a finite set of polynomials $\mathcal{P} \subset F[X_1, \ldots X_k]$. Then the elimination operator Elim is called recursively for $k - 1$ variables. The result is a set of polynomials $\mathcal{P}_1 \subset F[X_1]$. We denote $\mathcal{P}$ by $\mathcal{P}_k$. We obtain k sets of polynomials including $\mathcal{P}_k$. The index indicates the main variable of the set, e.g., for $\mathcal{P}_1$ the main variable is X_1 and so on (c.f. Figure 8).

- Base phase

  For the set $\mathcal{P}_1$ we use the results on Sturm sequences and Cauchy bounds to decompose the space F into finitely many $\mathcal{P}_1$- invariant cells. We represent these cells with algebraic numbers. These representations are called *sample points* and the result forms a set $\mathcal{A}_1 \subset F$ (c.f. Figure 8).

- Lifting phase

  Starting with the set $\mathcal{P}_2 \subset F[X_1, X_2]$ the variable X_1 is substituted with each sample points from the base phase. The result is a new polynomial for each sample point from the base phase and each polynomial in $\mathcal{P}_2$ (c.f. Figure 8). These polynomials are evaluated with the methods used in the base phase. We obtain several samples for each base phase sample point and polynomial in $\mathcal{P}_2$. Moreover, we combine the new sample points and the old samples points to 2-dimensional sample vectors. The sample vectors form a set $\mathcal{A}_2 \subset F^2$. The set $\mathcal{A}_2$ is applied to the set $\mathcal{P}_3$ and so on. This is repeated recursively until a set $\mathcal{A}_{k-1}$ is applied to $\mathcal{P}_k$. Furthermore, this cylindrical decomposition of F^k is the result and is in fact a $\mathcal{P}$-invariant CAD.

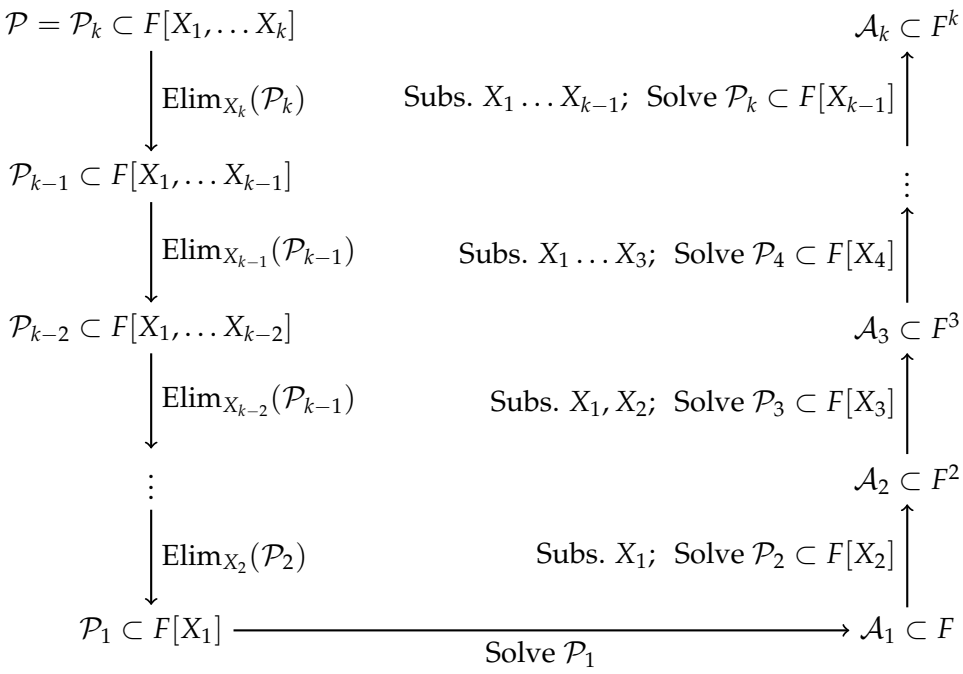

Figure 8 – The phases of the algorithm

3.3.1 Elimination phase

In our context, elimination is a way to project a set of polynomials in n variables to a set of polynomials in $n-1$ variables without loss of information about $n-1$-dimensions of roots of the original polynomial system. Applying this procedure recursively, we can project the problem to a set of univariate polynomials and solve it with the approaches presented in Section 2.3.
First, we introduce the non-recursive version of the elimination algorithm namely Algorithm 2 which is based on the Elim-operator from Definition 17:

Algorithm 2 [Elimination]: Calculate a set of polynomials $\mathcal{P}_{k-1}$ which satisfies Lemma 4

Require: $\mathcal{P} \subset F[X_1 \dots X_{k-1}][X_k]$
Output: $\mathcal{P}_{k-1} \subset F[X_1 \dots X_{k-1}]$

return $\text{Elim}_{X_k}(\mathcal{P})$

Let $m := \max_{P \in \mathcal{P}}(\deg(P))$. Moreover, there are $|\mathcal{P}|^2$ polynomial pairs to consider and for each polynomial there are at most m truncations. Furthermore, the encoding of the distinct roots and the degree depends only linear on s. In contrast, the encoding of the number of common roots depends quadratic on the number of polynomials. Subsequently, the number of signed subresultants calculations is bounded by $\mathcal{O}(m^2 s^2)$. A subresultants calculation with the algorithm we use requires at most $\mathcal{O}(m^2)$ operations. In summary, the overall complexity of Algorithm 2 is $\mathcal{O}(m^4 s^2)$.
The output set of Algorithm 2 consists of at most $\mathcal{O}(m^3 s^2)$ polynomials because in each step which treats a polynomial pair at most $m - 1$ polynomials are added to the output set. The degree of the subresultants of two polynomials $P, Q \in F[X_1 \dots X_k]$ is bound by $\mathcal{O}(\max(\deg(P), \deg(Q))^2)$. Moreover, the maximal degree of polynomials in the output set is $\mathcal{O}(m^2)$.

Algorithm 3 [Recursive Elimination]: Calculate the sequence of recursive elimination sets

Require: $\mathcal{P} \subset F[X_1 \dots X_{k-1}][X_k]$
Output: A sequence $C = (\mathcal{P}_1 \subset F[X_1] \dots \mathcal{P}_{k-1} \subset F[X_1 \dots X_{k-1}], \mathcal{P}_k \subset F[X_1 \dots X_k])$

$i \leftarrow k$
$C[k] \leftarrow \mathcal{P}$
while $i > 1$ **do**
  $C[i-1] \leftarrow \text{Elim}_{X_i}(C[i])$
  $i \leftarrow i - 1$
end while
return C

To study the complexity of the recursive Algorithm 3, we assume a less strict bound for Algorithm 2 which is $\mathcal{O}((ms)^3)$. If for instance $k = 3$ then the first step is to compute a set $\mathcal{P}_2$ with $\mathcal{O}((ms)^3)$ elements. Then the set $\mathcal{P}_1$ has $\mathcal{O}((ms)^3 \cdot (ms)^3) = \mathcal{O}(ms)^{3^2}$ elements and so on. This leads to the general formula

$$\prod_{i=1}^{k-1} (ms)^3 \in \mathcal{O}((ms)^{3^{k-1}})$$

for the number of elements in $\mathcal{P}_1$ for arbitrary k. Subsequently, the number of elements in $\mathcal{P}_1$ is doubly exponential in k. [3][p. 407]

3.3.2 Base phase

The base phase is based on the results on Sturm sequences and Cauchy bounds in Section 2.3. Despite the representation of roots, we need to extend the results to represent the non-root subsets of F with real algebraic numbers.

Algorithm 4 [Base phase]: Calculate the set of sample points $\mathcal{A}$

Require: $\mathcal{P} \subset F[X]$
Output: A set of real algebraic numbers $\mathcal{A} \subset F$

roots $\leftarrow$ isolateRoots($\mathcal{P}$)
count $\leftarrow$ countRoots($\mathcal{P}$)
for $i = 1$ **to** count-1 **do**
  $\mathcal{A} = \mathcal{A} \cup \{\frac{\text{roots}[i].\,\text{right} + \text{roots}[i+1].\,\text{left}}{2}\} \cup \{\text{roots}[i]\}$
end for
$A = A \cup \{\text{roots}[1] - 1\}$
$A = A \cup \{\text{roots}[\text{count}] + 1\} \cup \{\text{roots}[\text{count}]\}$
return $\mathcal{A}$

The base phase consist of two parts: The calculation of the isolating intervals and the calculation of additional sample points. The computational complexity and the drawbacks of the methods on how to isolate roots have been studied in Chapter 2. The effort necessary to compute the additional sample points depends on the number of roots of polynomials in $\mathcal{P}$. Thus, it is bound by $\mathcal{O}(m \cdot |\mathcal{P}|)$.

Definition 18 (Sample point)**.** A *sample point* is a representation of a cell in a CAD. If the represented cell has a zero-sign with respect to the polynomial which defines the cell, then, an interval representation is necessary to represent the cells as a sample point. Otherwise, it is sufficient to choose an arbitrary numeric within the cell.

Example 14. Given the polynomial $P = X^2 - 3X + 1 \in \mathbb{Z}[X]$ (cf. Example 6). We continue the procedure in Example 6 until each interval consists of exactly one root:

$$\text{isolateRoots}(P) = ((0,2);(2,4)).$$

Moreover, we apply Algorithm 4 to $\{P\}$:

$$(-1;(0,2);2;(2,4);5).$$

Each of these five real algebraic numbers represent one connected semi-algebraic subset of $\mathbb{R}_{alg}$. The two intervals represent the cells which have a zero-sign with respect to P, -1 and 5 represent cells with a positive sign as numerics and 2 represents the cells with a negative sign.

3.3.3 Lifting phase

Given the elimination sets $C = (\mathcal{P}_1, \ldots, \mathcal{P}_k)$ we assume to have a $\mathcal{P}_1$-invariant decomposition of F (cf. Algorithm 4). The next thing to do is to extend the decomposition of F to a CAD of F^2. Therefor, it is necessary to regard all polynomials in $\mathcal{P}_2$ substituted with all real algebraic numbers representing the $\mathcal{P}_1$-invariant decomposition. The substitution process is referred to as *lifting*.
To lift a multivariate polynomial with a real algebraic number, we have to distinguish between the cases: Lifting a polynomial with a real algebraic number represented by a numeric representation and lifting a polynomial with an real algebraic number represented by an interval representation.

Algorithm 5 [NumericLift]: Lifts the first $k-1$ variables of a polynomial with a real algebraic point

Require: $P \in F[X_1 \ldots X_k]$ and $r \in \mathbb{Q}^{k-1}$
Output: $P \in F[X_k]$

for $i = 1$ **to** $k-1$ **do**
  Substitute X_i with $r[i]$ in P
end for
return P

The number of substitutions done by Algorithm 5 is exactly $k-1$. So the complexity is bounded by $\mathcal{O}(k)$ univariate polynomial evaluations.
Without loss of generality, we assume that a representation of a real algebraic number is either a numeric representation in each dimension or an interval representation in each dimension.
To substitute a variable with a real algebraic number in interval representation form, we need a method to evaluate the sign of a real algebraic number for a univariate polynomial:

Lemma 6 (Sign determination for real algebraic numbers)**.** *Given a real algebraic number $\alpha = (P, o) \in \mathbb{R}_{alg}$ and a univariate polynomial with integer coefficients $Q \in \mathbb{Z}[X]$. Then the sign of Q evaluated at α is determined by*

$$\mathrm{Var}(\mathrm{SRemS}(P, P' \cdot Q); o.\mathrm{left}, o.\mathrm{right}).$$

We denote this expression by $\mathrm{sgn}(\alpha, Q)$.

Example 15. Given a polynomial $2X^2 + 4Y^2 + Z^3 \in \mathbb{Z}[X, Y, Z]$ and two algebraic numbers $\alpha = (X^2, (-\frac{1}{10}, \frac{1}{10}))$ and $\beta = (X^3 - 2X + 1, (\frac{9}{10}, 1\frac{1}{10}))$.
We lift Y with α and Z with β.
The sign of $4Y^2$ evaluated at α can be calculated by means of Lemma 6:

$$\mathrm{sgn}(4Y^2, \alpha) = 0.$$

If the sign of $4Y^2$ at α is 0 then this coefficient vanishes for α. Thus, the resulting polynomial is

$$2X^2 + Z^3.$$

Furthermore,

$$\operatorname{sgn}(Z^3, \beta) = 1.$$

Unfortunately, the coefficients necessary for the sign check are not that homogeneous in general. To lift z in the following polynomial

$$(Z^2 + YZ^3)X^2 + ZX$$

we have to "collect" the coefficients containing Z first. For ZX this is simply $\operatorname{coeff}(ZX)$ but the first expression $(Z^2 + YZ^3)X^2$ is more complicated. It is necessary to regard $\operatorname{coeff}((Z^2 + YZ^3)X^2) = Z^2 + YZ^3$. This approach is a projection of the 3-dimensional problem into to several two dimensional problems. This concept is implemented as recursive calls in the following algorithm.

Algorithm 6 [IntervalLift]: Lifts the first variable of a $k - i$ dimensional polynomial with a real algebraic number

Require: $P \in F[X_i \dots X_k]$, interval representation (P, o) and $i \geq k$
Output: $P \in F[X_{i-1} \dots X_k]$ or $F[X_k]$ if $i - 1 = k$

if i = k+1 **then**
  for $j = 0$ **to** $\deg(P)$ **do**
    $C \leftarrow \operatorname{coeff}_j^{X_{i-1}}(P)$
    $P \leftarrow P - C \cdot X_i^j$
    if $sgn((P, o), C) \neq 0$ **then**
      $P \leftarrow P + C(\frac{o.\text{left} + o.\text{right}}{2}) \cdot X_{i-1}^j$
    else
      for $j = 0$ **to** $\deg(P)$ **do**
        $C \leftarrow \operatorname{coeff}_j^{X_{i-1}}(P)$
        $P \leftarrow P - C \cdot X_{i-1}^j$
        $P \leftarrow P + \text{IntervalLift}(C, (P, o))$
      end for
    end if
  end for
end if

In contrast to Algorithm 5, this algorithm does not lift $k - 1$ variables at once but only one variable. Subsequently, it has to be applied $k - 1$ times to completely lift a k-dimensional polynomial. The number of sign determinations is bound by $\mathcal{O}(\deg_{X_i}(P) \cdot \deg_{X_{i+1}}(P) \dots \deg_{X_k}(P))$.
The algorithm can easily be extended to lift the first $k - 1$ variables like Algorithm 5 but we avoid this here for the sake of of readability.

Example 16. Due to the cylindrical structure of the CAD, it is not sufficient to lift all the polynomials at once. Different stacks with potentially different

polynomials emerge and the lifting has to be done in each stack. To clarify this we introduce a simple example of CAD calculation. Let $\mathcal{P} = \{X^2 + Y^2 - 1\} \subset \mathbb{Z}[X, Y]$ be the set of polynomials to decompose. Then, the elimination sets are given by

$$\mathcal{P}_2 = \{X^2 + Y^2 - 1\}$$
$$\mathcal{P}_1 = \{X^2 - 1\}.$$

We apply the base phase algorithm to $X^2 + 1$ and obtain sample points representing the decomposition of $\mathbb{R}_{alg}$. We represent (visually and algorithmically) this decomposition with a tree with an empty root node and real algebraic numbers as siblings:

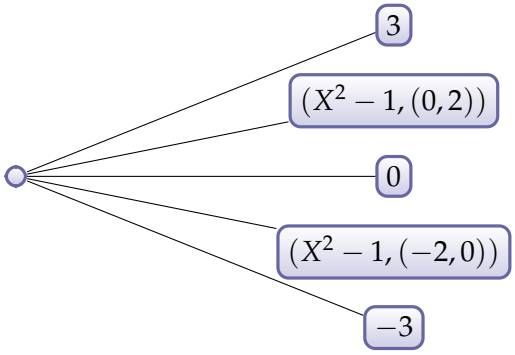

Figure 9 – Decomposition of $\mathbb{R}_{alg}$ after the base phase.

The only polynomial in $\mathcal{P}_2 = \{X^2 + Y^2 - 1\}$ has to be lifted for each sample point in $\mathbb{R}_{alg}$ visualized in Figure 9. Due to the symmetry of the example, we only extend the silblings 3 and $(X^2 - 1, (-2, 0))$:
Starting with 3, we substitute X in $X^2 + Y^2 - 1$ with 3:

$$Y^2 - 8.$$

The next step is to apply the base phase algorithm on $Y^2 - 8$ and to extend the tree by appending the resulting sample points to the 3-node. We repeat this procedure with the $(X^2 - 1, (-2, 0))$-node and obtain three additional samples to append to the node. The siblings of the nodes 3 and -3, and $(X^2 - 1, (-2, 0))$ and $(X^2 - 1, (2, 0))$ are equivalent. Furthermore, if we substitute X with 0 in $X^2 + Y^2 - 1$ the result is

$$Y^2 - 1$$

which is up to variable names similar to the first level of the tree in Figure 4. Thus, extending the tree in Figure 10 with the above mentioned additional samples results in a complete tree of height 2.

In general, this tree is called a *sample tree*. A sample tree of a finite polynomial set $\mathcal{P} \subset F[X_1, \ldots, X_k]$ is a complete tree of height k constructed in the above mentioned manner. The complete sample tree contains all the information necessary to compute the set of samples points for a $\mathcal{P}$-invariant CAD.

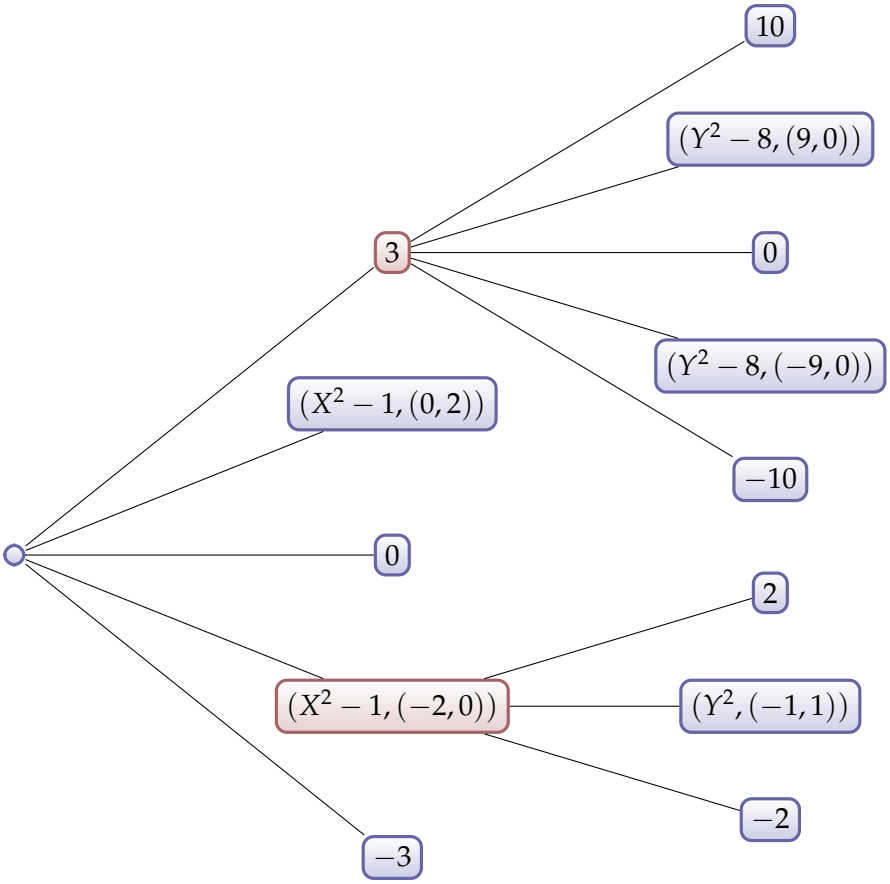

Figure 10 – Lifting and extension of $(X^2 - 1, (-2, 0))$ and 3.

Starting from the empty root node, each path $(v_0, \ldots, v_k)$ of length $k + 1$ to a leaf node represents sample point of dimension k. As the nodes of the tree are real algebraic numbers, the paths are real algebraic vectors in F.
To clarify the capabilities of the sketched decision procedure, we present a more complex example:

Example 17. The problem is a slightly modified version of the motivational Example 1. Let $\mathfrak{P}$ be a set of polynomial in-equations:

$$\mathfrak{P} = \{X + Y + 1 \leq 0, X^2 + Y^2 - 1 < 0\}$$

The according set of polynomials is denoted by $\mathcal{P}_2$. Moreover, the set $\mathcal{P}_1 = \text{Elim}_X(\mathcal{P})$ is given by

$$\{Y^2 - 1, Y^2 + Y, Y + 1\}.$$

The set of sample points $\mathcal{A}_1$ is given by

$$\{-2, -1, -\frac{1}{2}, 0, \frac{1}{2}, 1, 2\}.$$

The sample points $\mathcal{A}_1$ represent the possible values of the Y-coordinate of each sample point.

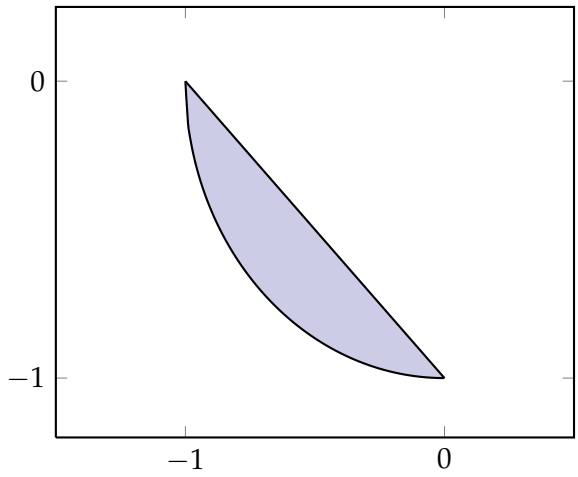

Figure 11 – Solution space of $\mathfrak{P}$.

Note, there are no interval representations because an algorithm to intelligently guess the simple solutions has been applied. In general, it is not possible to guess numeric representations for all interval representation.

In each polynomial in $\mathcal{P}_2$ the variable Y is substituted with each sample point in $\mathcal{A}_1$:

-2	X^2+3	$X-1$
-1	X^2	X
$-\frac{1}{2}$	$X^2-\frac{3}{4}$	$X+\frac{1}{2}$
0	X^2-1	$X+1$
$\frac{1}{2}$	$X^2-\frac{3}{4}$	$X+\frac{3}{2}$
-1	X^2	$X+2$
-2	X^2+3	$X+3$

Table 1 – Substituted Y with each sample point for each polynomial in $\mathcal{P}_2$.

By applying the base algorithm to the polynomials and combining them with the according sample points in $\mathcal{A}_1$, we obtain $\mathcal{A}_2$:

$$
\begin{aligned}
&(0,-2);(1,-2);(2,-2);\\
&(0,-2);\\
&(-1,-1);(0,-1);(1,-1);\\
&(-1,-1);(0,-1);(1,-1);\\
&(-\frac{5}{2},-\frac{1}{2});(]-\frac{9}{16},-\frac{15}{32}[,-\frac{1}{2});(1,-\frac{1}{2});\\
&(-\frac{11}{4},-\frac{1}{2});(]-\frac{7}{8},-\frac{105}{128}[,-\frac{1}{2});(0,-\frac{1}{2});(]\frac{105}{128},\frac{7}{8}[,-\frac{1}{2});(\frac{11}{4},-\frac{1}{2});\\
&(-2,0);(-1,0);(0,0);\\
&(-2,0);(-1,0);(\frac{1}{2},0);(1,0);(2,0);\\
&(-\frac{7}{2},\frac{1}{2});(]-\frac{25}{16},-\frac{95}{64}[,\frac{1}{2});(1,\frac{1}{2});\\
&(-\frac{11}{4},\frac{1}{2});(]-\frac{7}{8},-\frac{105}{128}[,\frac{1}{2});(0,\frac{1}{2});(]\frac{105}{128},\frac{7}{8}[,\frac{1}{2});(\frac{11}{4},\frac{1}{2});\\
&(-4,1);(]-\frac{33}{16},-\frac{63}{32}[,1);(1,1);\\
&(-1,1);(0,1);(1,1);\\
&(-5,2);(]-\frac{49}{16},-3[,2);(1,2);\\
&(0,2)
\end{aligned}
$$

In matter of clarity, we only state the intervals to represent the interval representations according to the polynomials above (each line corresponds to a polynomial from left to right and line wise cf. Table 1). To check whether the constraints are satisfiable, we have to check whether at least one sample point $\alpha \in \mathcal{A}_2$ models $\mathfrak{P}$. Therefor, we search for a sample point α with $\operatorname{sgn}(P,\alpha) = 1$ for each $P \in \mathcal{P}_1$ (cf. Lemma 6).
The sample points $(]-\frac{9}{16},-\frac{15}{32}[,-\frac{1}{2})$ (marked in red) which is actually given by $((X-\frac{1}{2},(-\frac{9}{16},-\frac{15}{32})),-\frac{1}{2}) = (-\frac{1}{2},-\frac{1}{2})$ is a solution of $\mathfrak{P}$. Hence, $\mathfrak{P}$ is satisfiable.

Chapter 4

Satisfiability Modulo Theories

4.1 Satisfiability

Satisfiability (SAT) is the problem to decide whether there is a satisfying assignment for a given boolean formula. The set of boolean formulas is denoted by propositional logic. Propositional logic formulas are built from of atomic propositions TRUE and FALSE and a finite number of variables Var. An assignment is a function $\alpha : \text{Var} \to \{\text{TRUE}, \text{FALSE}\}$. Furthermore, if φ and ψ are propositional formulas then

- $\varphi \wedge \psi$

- $\neg\varphi$

are propositional logic formulas. Moreover, operators like $\vee$, $\leftarrow$ or $\leftrightarrow$ are defined as combination of the operators above. We say that α models a propositional formula if α assigns all variables to an atomic proposition and the formula is valid.

The SAT problem is NP-complete. If we assume that $P \neq NP$ then there exists no algorithm with polynomial complexity in the number of variables which solves this problem. *SAT-Checkers* or *SAT-Solvers* are programs that decide whether a propositional formula is satisfiable or not. They have been studied intensively in the past years. If a SAT-Checker is applied to a certain propositional logic formula it either returns a satisfying assignment or it returns UNSAT. Furthermore, it returns a minimal unsatisfiable core for unsatisfiable formulas. A *minimal unsatisfiable core* is a minimal unsatisfiable sub formula of the original propositional formula. Even though the problem is NP-complete it is successfully applied in the industry for instance for circuit planning. [10] A successful approach which uses conflict-clause learning and conflict-driven backtracking is called DPLL. This approach is still widespread in the list of top SAT-Checkers. [15, p. 28ff]

4.2 Satisfiability Modulo Theories

The Satisfiability modulo theories (SMT) is an approach combing the SAT problem and first order (FO) formulas. The idea is to divide the tasks into solving the underlying boolean structure with a fast SAT-Solver and to study the first order constraints with another tool called theory solver.
A *theory* is a set of FO-formulas over a certain signature Σ. Let T be a theory over Σ. Then we say that a FO-Formula φ is a *satisfiable modulo* of T if $\{T\} \cup \varphi$ is satisfiable. We say that a theory is decidable if there exists a decision procedure which checks whether a quantifier free formula over Σ is satisfiable. The theory of real algebra which is studied in this paper is decidable. [26]
A *theory* solver $\mathcal{T}$ is a decision procedure to decide whether a conjunction of atomic propositions for a certain formula is satisfiable. If $\mathcal{T}$ is satisfiable the theory solver returns a satisfying assignment otherwise it returns UNSAT.
There are theory solvers for various theories like the Simplex algorithm for the theory of linear arithmetic or theory solvers for uninterpreted functions. In fact, the algorithm described in Chapter 3 can be used as a theory solver for the theory of real algebra. For a certain set of polynomials, we obtain a set of sample points. To decide whether the set of polynomial (in)equalities is satisfiable, we have to check if the set of sample points contains a sample point with proper signs for all the polynomials (in)equations in the set (cf. Example 17). [1]

Example 18. Given a formula

$$\varphi := (x = y \vee x > 0) \wedge x + y = 0 \wedge 2y \geq 0$$

from the theory of linear arithmetic. We calculate a propositional logic formula called *boolean abstraction* of φ. The boolean abstraction introduces a new variable for every new theory constraint and keeps the boolean structure:

$$(a \vee b) \wedge c \wedge d$$

The boolean abstraction is denoted by boolAbs(ϕ). We assume, the first satisfying assignment the SAT-Checker returns for the boolean abstraction is the assignment b, c, d to TRUE and a to FALSE. Subsequently, we have to apply the theory solver for linear arithmetic (e.g. Simplex) to the following set of linear constraints:

$$\{x > 0, x + y = 0, 2y \geq 0\}$$

It is easy to see that this set of constraints it not satisfiable. Hence, the SAT-Checkers learns a conflict clause resulting in an extended boolean abstraction:

$$(a \vee b) \wedge c \wedge d \wedge \neg(b \wedge c \wedge d)$$

Moreover, we assume that the next assignment is setting a, c, d to TRUE and b to false. The resulting set

$$\{x = y, x + y = 0, 2y \geq 0\}$$

is satisfiable. The solution for these linear constraint is $x = y = 0$.

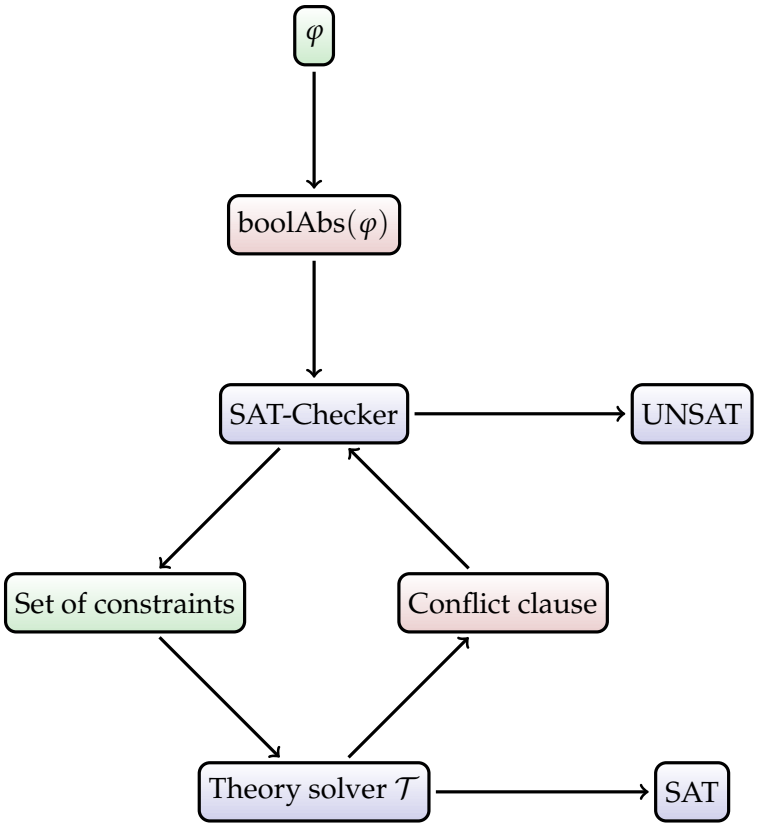

Figure 12 – Fully-lazy SMT.

For a unsatisfiable formula, we add conflict clauses to the formula until the SAT-Checker indicates the unsatisfiability of the formula. Assuming we have a fitting theory solver, we can apply this approach to arbitrary theories. This approach is called *full-lazy SMT* (cf. Figure 12). The solvers and their results are marked in blue, the boolean extension of FO-properties and the set of FO-Properties are marked in green and the boolean abstractions of the original formula and the unsatisfiable set of constraints are marked in red.

It is likely that calculations in the theory solver are done multiple times. Results from former calculations are not re-used to improve the speed of new calculations.

Another approach is called *less-lazy SMT*. The theory solver is not just called whenever the SAT-Solver has found a full assignment but also earlier to check if the current partial assignment is already unsatisfiable. The theory solver must be able to include constraints incrementally so that the calculation does not start over once a new constraint is added. Thus, the theory solver has to make use of older results. [1] [15]

4.3 Preparations for less-lazy solving of real algebra constraints

We can already provide a theory solver for full-lazy SMT. The algorithm presented in Chapter 3 is a decision procedure for a set of multivariate polynomial (in)equality constraints.
We extend the constructions in Chapter 3 so that we can provide techniques used by the less-lazy approach. We have to enable incremental addition of polynomials constraints to elimination sets without recomputing the hole set. We see in the definition of the elimination operator (cf. Definition 17) that the first two conditions concern one polynomial only and that the other polynomials do not effect the result. We separate these two conditions and define new elimination operators:

Definition 19 (Incremental elimination)**.** Given a polynomial $P \in F[X_1 \dots X_{k-1}][X_k]$ we define $\text{IncElim}^a_{X_k}(P) \subset F[X_1, \dots X_{k-1}]$ containing the following elements:

- For all $T \in \text{Tru}(P)$, $\text{lcoeff}(T)$.
- For all $T \in \text{Tru}(P)$, for $0 \leq i \leq \deg(T) - 2$, $\text{sRes}_i(T, T')$.

Given two polynomials $P, Q \in F[X_1, \dots X_{k-1}][X_k]$ we define $\text{IncElim}^b_{X_k}(P, Q) \subset F[X_1, \dots X_{k-1}]$ containing the following elements:

- If $\deg(P) > \deg(Q)$, for all $0 \leq i \leq \deg(Q)$, $\text{sRes}_i(P, Q)$,
- If $\deg(P) < \deg(Q)$, for all $0 \leq i \leq \deg(Q)$, $\text{sRes}_i(Q, P)$,
- If $\deg(P) = \deg(Q)$, for all $0 \leq i \leq \deg(\text{lcoeff}(P)Q - \text{lcoeff}(Q)P)$, $\text{sRes}_i(P, \text{lcoeff}(P)Q - \text{lcoeff}(Q)P))$.

A combination of the two new elimination operators forms a complete elimination set according to Definition 17:

$$\text{Elim}_{X_k}(\mathcal{P}) = \bigcup_{P \in \mathcal{P}} \text{IncElim}^a_{X_k}(P) \cup \bigcup_{P,Q \in \mathcal{P}} \text{IncElim}^b_{X_k}(P, Q)$$

To add a polynomial to existing elimination set, the following property can be used:

$$\text{Elim}_{X_k}(\mathcal{P} \cup \{Q\}) = \text{Elim}_{X_k}(\mathcal{P}) \cup \text{IncElim}^a_{X_k}(Q) \cup \bigcup_{P \in \mathcal{P}} \text{Elim}^b_{X_k}(P, Q)$$

Compared to recalculating the entire elimination set, this approach decreases the number signed subresultant calculations to add a new constraint significantly.
Another way to increase the performance of the theory solver is not build up the sample tree level-wise but build it in a depth-first-search- manner (DFS). A possible sample point satisfying the set of constraints can be found before

the whole sample tree is built. These satisfying sample points can be evaluated for other polynomial constraints without calculating an elimination set and without changing the sample tree. If the sample point does not satisfy the new polynomial constraint the old sample tree can be extended to avoid recomputing old calculations.

Example 19. We assume that the constraints given to the theory solver are

$$X^2 + Y^2 - 1 > 0,$$
$$X^2 + Y^2 - 109 > 0.$$

These are simple constraints as all points in $\mathbb{R}_{alg}$ which are not in a circle of radius 200 around the point $(0, 0)$ are valid solutions.

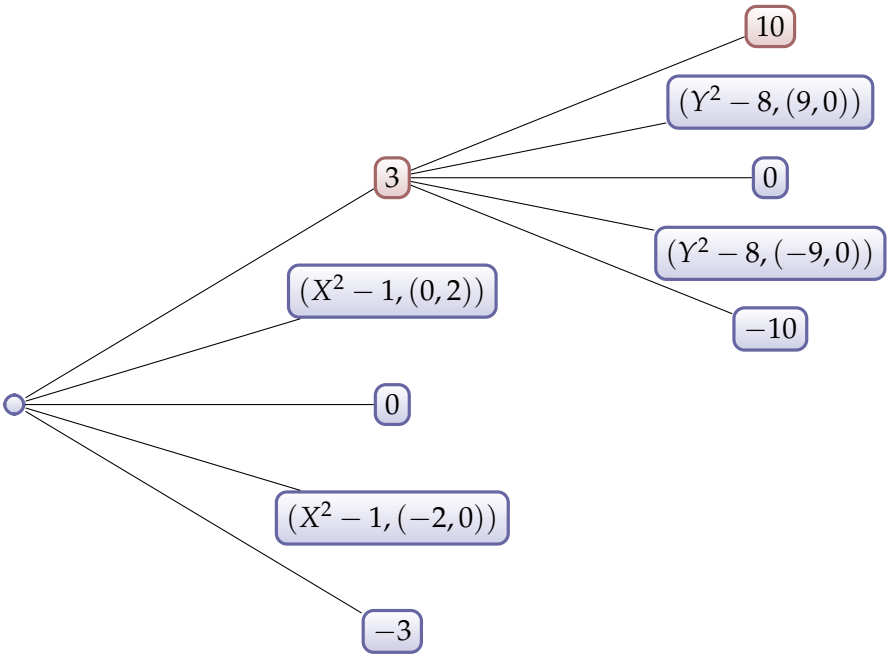

Figure 13 – Lifting and extension of $(X^2 - 1, (-2, 0))$ and 3.

Building a sample tree in the DFS-manner, we find the solution $(3, 10) \in \mathbb{R}_{alg}$ to be a solution before building up the whole sample tree.
Moreover, we add the second polynomial constraint to the theory solver. We check whether $(3, 10)$ is a valid solution but as $3^2 + 10^2 = 109$ the sign of the real algebraic number on the second constraint is 0. Instead of building a new sample tree, we extend the first sample by appending new silblings to 3. The elimination set of $X^2 + Y^2 - 109$ is similiar to the elimination set of $X^2 + Y^2 - 1$:

$$C = (X^2 - 109, X^2 + Y^2 - 109)$$

Furthermore, we substitute X in $X^2 + Y^2 - 109$ with 3 and obtain

$$Y^2 - 100.$$

The result of the base phase algorithm applied to this polynomial is:

$$(-\frac{109}{8}; (Y^2 - 100, (\frac{101}{8})), -\frac{101}{16})); 0; (Y^2 - 100, (0, 101)); 102)$$

Hence, we extend our sample tree and see that $(3, 102)$ is a solution.

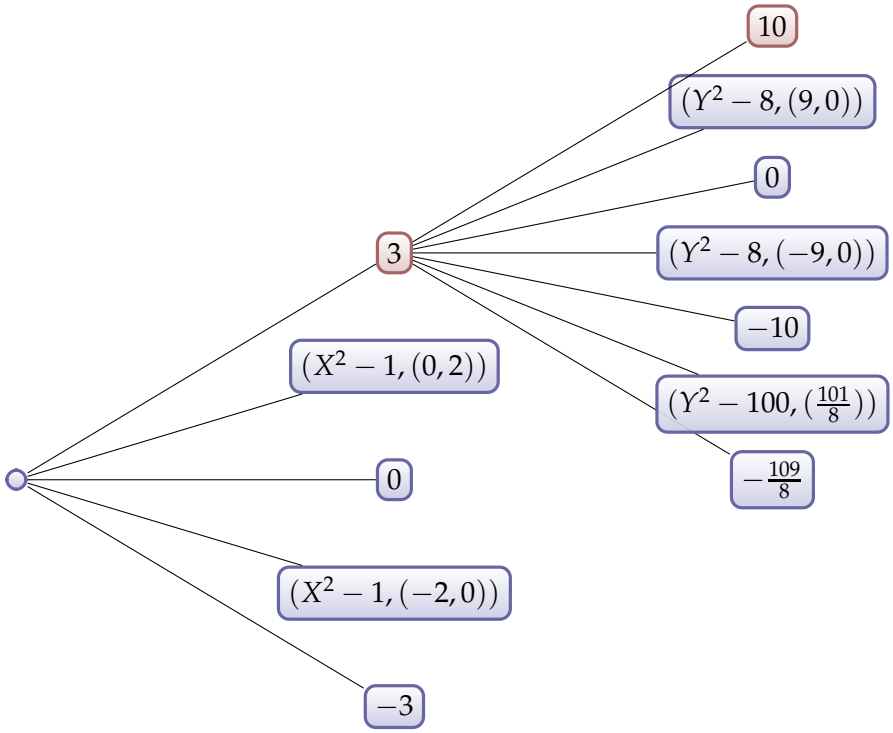

Figure 14 – Lifting and extension of $(X^2 - 1, (-2, 0))$ and 3.

Chapter 5

Implementation

In this section, we present the already completed implementations concerning the techniques presented in this paper.

5.1 GiNaC

GiNaC is a C++-library for symbolic computations and especially for the efficient handling of multivariate polynomials. It was initially intended to solve problems in theoretical quantum field theory. Nevertheless, it is applied in different fields and the project is recently in development. In contrast to computer algebra systems like Maple or Mathematica, it is entirely written in C++. Subsequently we show, that it can be directly integrated in fast object oriented C++-applications or library extensions.
Moreover, it is distributed under terms of the GNU general public license. Most of the above mentioned and commonly known computer algebra systems are commercial software products. [4]

5.2 GiNaCRA

GiNaCRA is a C++-library for efficient decision procedures within the field of real algebra. It is currently developed by Ulrich Loup, Sebastian Junges and Joachim Redies. It is the framework and library for the implementation of the algorithms in this paper. GiNaCRA is based on the GiNaC library and distributed under GNU general public license as well. [17]

5.3 Implemented methods

We sketch the general structure of GiNaCRA and how the algorithms presented in this paper interact with the framework (cf. Figure 15). Several abbreviations are used in this figure: RealAlgebraicNumber (RAN), IntervalRepresentation (IR), NumericRepresentation(NR) and UnivariatePolynomial(UniPol). In the actual implementation within the GiNaCRA framework, we avoid abbreviations (despite a few exceptions). The figure is a simplification of the

GiNaCRA framework. For instance the data structures do not contain the objects themself but pointers due to C++ issues. Furthermore, this is just a small subset of the classes within the GiNaCRA framework. For example, the inheritance chain of UnivariatePolynomial is longer. Nevertheless, it displays the most important classes and methods necessary to compute CADs.

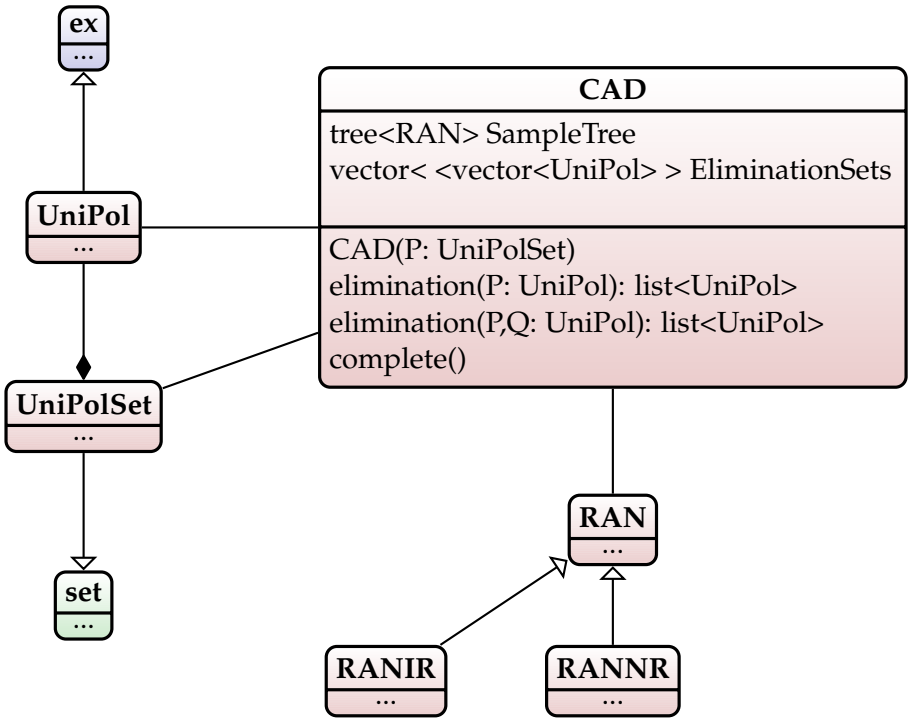

Figure 15 – Class relation of classes of STL(green), GiNaC(blue) and GiNaCRA (red).

The class ex is originated in the GiNaC framework. It represents a general mathematical expression. In the GiNaCRA framework we inherit its properties and extend it with the restriction to be an univariate polynomial. However, an univariate polynomial can be parameterized. Thus, it is a multivariate polynomial in an univariate representation (cf. p. 11). We use a container class UnivariatePolynomialSet inherited from the Standard Template Library (STL) class set to avoid redundancies in the different sets within the CAD. [24]
The class CAD is both an instance of a cylindrical algebraic decomposition and a collection of methods to compute it. The methods elimination(P) and elimination(P, Q) correspond to Definition 19 for the incremental computation of elimination sets. These methods can be used statically without a CAD object. The classical elimination operator (cf. Algorithm 2) is implemented in the constructor CAD of the class and it creates a vector of univariate polynomials for each elimination set. To generate the sample points and store them in the sample tree of the CAD instance, we use the method complete which generates all possible sample points. The CAD as an instance of a cylindrical algebraic decomposition contains a representation of the sample tree. For this tree we used an STL-like implementation namely tree.h which is distributed under

GNU general public license [21]. The classes RealAlgebraicNumber and its specialization refer to the different representations of real algebraic numbers presented in this paper (cf. Definition 8). Notice that the description above is a simplification of both the structure and the actual usage of the methods. Nevertheless, there is an extensive documentation of the framework available [18].
To explain the usage of the above mentoid classes and methods, we introduce a minimal code example:

```
1  #include <ginacra/ginacra.h>
2
3  using namespace GiNaCRA;
4
5  int main ()
6  {
7      symbol x("x"), y("y");
8      UnivariatePolynomial p(pow(x,2)+pow(y,2)-1, x);
9      vector<symbol> v;
10     v.push_back(x);
11     v.push_back(y);
12
13     UnivariatePolynomialSet P;
14     P.insert(p);
15
16     CAD C ( P , v);
17     C.complete();
18 }
```

The examplary program calculates the elimination sets. Moreover, it generate all possible sample points.

Chapter 6

Conclusion

In this paper, we have presented the foundations and the methods necessary to calculate and evaluate a CAD. We have seen how it is implemented within the GiNaCRA framework. Furthermore, the techniques can be applied and modified to work as a theory solver for an incremental less-lazy SMT-Solver.
During the implementation, of the methods we noticed that some of the sub algorithms even for simpler tasks like for example basic calculations with real algebraic numbers have an unfortunate complexity bound.
We think there are several sub algorithms which can be improved with heuristics to decrease the overall runtime of the CAD algorithms:

- Selection of subtrees to extend in the extension phase
- Selection of extended subtrees which will be used for the generation of sample points
- Transformation of interval representations into exact numeric root representations whenever possible

Moreover, there are faster versions of the subresultant algorithm and the elimination operator which can be implemented. Further challenges arise when embedding the theory solver in a SMT-Solver.

Bibliography

[1] E. Ábrahám, U. Loup, B. Becker, V. Bertacoo, R. Drechsler, and M. Fujita. Smt-solving for the first-order theory of the reals. *Algorithms and Applications for Next Generation SAT Solvers, Dagstuhl Seminar*, 2010.

[2] H. Anai, K. Yokoyama, and Fujitsu Laboratories. International Institute for Advanced Study of Social Information Science. *Radical representation of polynomial roots*. IIAS-RR-. International Institute for Advanced Study of Social Information Science, Fujitsu Ltd., 1994.

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

[4] C. Bauer, A. Frink, and R. Kreckel. Introduction to the ginac framework for symbolic computation within the C++ programming language. *Journal of Symbolic Computation*, 33(1):1–12, 2002.

[5] Christopher W. Brown. Qepcad b: A program for computing with semi-algebraic sets using cads. *SIGSAM BULLETIN*, 37:97–108, 2003.

[6] C.W. Brown. Improved projection for cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 32(5):447–465, 2001.

[7] A.M. Cohen, H. Cuypers, and H. Sterk. *Some tapas of computer algebra*. Springer Verlag, 1999.

[8] George E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition–preliminary report. *SIGSAM Bull.*, 8:80–90, August 1974.

[9] George E. Collins and Hoon Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *J. Symb. Comput.*, 12:299–328, September 1991.

[10] S.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.

[11] L. De Moura and N. Bjørner. Z3: An efficient smt solver. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, 2008.

[12] Andreas Dolzmann and Thomas Sturm. Redlog computer algebra meets computer logic. *ACM SIGSAM Bulletin*, 31:2–9, 1996.

[13] H. 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.

[14] M. Jirstrand. *Cylindrical algebraic decomposition: An introduction*. LiTH-ISY-R. Linköpings university, 1995.

[15] D. Kroening and O. Strichman. *Decision Procedures: An Algorithmic Point of View*. Springer-Verlag, 2008.

[16] Henri Lombardi, Marie-Françoise Roy, and Mohad Safey El Din. New structure theorem for subresultants. *Journal of Symbolic Computation*, 29:663–690, 2000.

[17] U. Loup and E. Ábrahám. Ginacra: A C++ library for real algebraic computations. *NASA Formal Methods*, pages 512–517, 2011.

[18] Ulrich Loup. Ginacra - ginac real algebra package. Website, 2012. Available online at `http://ginacra.sourceforge.net/doc/index.html`.

[19] S. McCallum. An improved projection operation for cylindrical algebraic decomposition. In *EUROCAL'85*, pages 277–278. Springer, 1985.

[20] B. Mishra. *Algorithmic algebra*. Texts and monographs in computer science. Springer-Verlag, 1993.

[21] Kasper Peeters. tree.hh documentation. Available online at `http://tree.phi-sci.com/tree.pdf`.

[22] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical mathematics*. Texts in Applied Mathematics. Springer, 2000.

[23] A. Seidenberg. A new decision method for elementary algebra. *The Annals of Mathematics*, 60(2):365–374, 1954.

[24] B. Stroustrup. *C++*. John Wiley and Sons Ltd., 2003.

[25] Adam Strzebonski. Cylindrical algebraic decomposition. Website, 2012. Available online at `http://mathworld.wolfram.com/CylindricalAlgebraicDecomposition.html`.

[26] A. Tarski. *A decision method for elementary algebra and geometry*. Rand report. Rand Corp., 1948.

[27] Eric W. Weisstein. Algebraic number. Website, 2011. Available online at `http://mathworld.wolfram.com/AlgebraicNumber.html`.

[28] Eric W. Weisstein. Determinant. Website, 2011. Available online at `http://mathworld.wolfram.com/Determinant.html`.

[29] Eric W. Weisstein. Sign. Website, 2011. Available online at `http://mathworld.wolfram.com/Sign.html`.