Theory of Hybrid Systems Informatik 2
RWTH AACHEN UNIVERSITY

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

BACHELOR OF SCIENCE THESIS

EXPLOITING STRICT CONSTRAINTS IN THE COMPUTATION OF CYLINDRICAL ALGEBRAIC COVERINGS

Philipp Bär

Examiners:
Prof. Dr. Erika Ábrahám
Prof. Dr. Jürgen Giesl

Additional Advisor:
Jasper Nalbach

Aachen, 23 August 2022

Numquam coepe desinere, numquam desine coepere.
Fange nie an aufzuhören, höre nie auf anzufangen.

Marcus Tullius Cicero

Abstract

Satisfiability Modulo Theories (SMT) solving is a technique used to determine the satisfiability of *Quantifier-free First-order Logic* formulae over a fixed theory. The *Cylindrical Algebraic Decomposition* (CAD) method is a commonly used strategy for solving problems of *Non-linear Real Arithmetic*. Recently, the *Cylindrical Algebraic Covering* (CAlC) method has been developed, which is a variant of the CAD method that incrementally tries to construct a satisfying solution for a conjunction of polynomial constraints or to detect its unsatisfiability. On the basis of a given variable and a partial assignment to some other variables, the main principle of the CAlC method is to generate single real values and open intervals of real numbers which can be omitted when searching for a satisfying solution.

We present two adaptions of the CAlC method that exploit strict constraints in the input conjunction. Both aim at extending the generated open intervals to include some of their endpoints in order to speed up the computation.

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Dr. Erika Ábrahám[a] and my advisor Jasper Nalbach[a]. Both have strongly supported me in various meetings and through discussions as well as valuable suggestions and feedback. In particular, I thank them for always having a sympathetic ear for my concerns. This also applies to the weekly meetings in which I flooded Jasper with questions.

I would also like to thank Prof. Dr. Christopher W. Brown[b], Prof. Dr. Matthew England[c], and Prof. Dr. James H. Davenport[d]. They explained complex topics in a simple way and guided me through the mathematical foundations and proofs.

Furthermore, I would like to thank Prof. Dr. Jürgen Giesl[a] for his readiness to supervise this work as a second examiner.

Finally, I like to thank my parents Ivonne and Manfred Bär as well as my grandfather Fritz Bär for their support during my Bachelor studies.

[a] RWTH Aachen University, Germany
[b] United States Naval Academy, United States of America
[c] Coventry University, United Kingdom
[d] University of Bath, United Kingdom

Contents

1 Introduction **9**

2 Preliminaries **11**
2.1 Quantifier-free Non-linear Real Arithmetic 11
2.2 Satisfiability Checking 14
2.3 CAD Techniques 14
2.4 Cylindrical Algebraic Coverings 17

3 Closed Intervals in the CAlC Method **27**
3.1 Motivation 27
3.2 Related Work 28
3.3 Deducing Closed Interval Bounds 29
3.4 Closed Bounds Based on Intervals 30
3.5 Closed Bounds Based on Polynomials 37

4 Benchmarks **47**
4.1 Dataset and Environment 47
4.2 Experimental Results 47

5 Conclusion **51**
5.1 Summary 51
5.2 Future Work 52

Bibliography **53**

Chapter 1

Introduction

Efficiently determining the satisfiability of polynomial inequalities has become increasingly relevant in recent decades. Various properties of real-world systems can be expressed in mathematical formulae whose satisfiability can be used to draw conclusions about concrete systems, e.g. for checking the validity of circuit designs or to analyse security issues [Stu06, Sne05]. *Satisfiability Modulo Theories* (SMT) solvers are tools used to reason about such encodings of real-world problems. They aim at checking the satisfiability of formulae over some existential fragment of a *First-order Logic* theory.

In the middle of the 20th century, Alfred Tarski showed that the theory of *Non-linear Real Arithmetic* is decidable [Tar51]. That implies, it is possible to algorithmically determine whether there exists a real-valued solution for a conjunction of polynomial equations and inequations, so-called polynomial constraints which are built using addition and multiplication of variables and rational coefficients. In SMT solving, special *Theory Solvers* are employed to check the consistency of sets of theory constraints.

A possible decision procedure for a theory solver for Non-linear Real Arithmetic in SMT solving is the *Cylindrical Algebraic Decomposition* (CAD) method [Col75]. Based on the techniques of the CAD, in 2021 Ábrahám et al. developed a related method: Instead of calculating a decomposition of the real space, this strategy works on *Cylindrical Algebraic Coverings* (CAlCs) [ÁDEK21]. In Chapter 2 we first introduce the necessary basics, in particular we formalise the used theory and some basic CAD tools before introducing CAlCs and the corresponding CAlC method.

This work investigates how additional information can be drawn from strict inequations, allowing us to make statements about the satisfiability of polynomial inequations more efficiently, i.e. with less computational effort. In Chapter 3, we present two ideas on how it might be possible to reduce the number of steps performed to analyse a conjunction of constraints. That is done by leaving out some particularly cost-expensive points of the real space.

The implementation of these variants in the SMT toolbox `SMT-RAT` is discussed in Chapter 4. We analyse the differences between the two alternatives in practise and evaluate their benefits before drawing conclusions in Chapter 5.

Chapter 2

Preliminaries

Before analysing strict constraints in the CAlC method, we introduce some basic terminology and techniques. First, we define *polynomials* and *polynomial constraints*, then the theory of *Quantifier-free Non-linear Real Arithmetic*, the core idea of *Satisfiability Checking*, the *Cylindrical Algebraic Decomposition*, and finally the main concept of *Cylindrical Algebraic Coverings*.

Within this chapter, we often refer to Ábrahám et al. and Nalbach et al., who introduce the main concepts of this thesis [ÁDEK21, NÁS+22]. Most of the following definitions are adaptions of the ones given there. Moreover, we assume that $0 \in \mathbb{N}$.

2.1 Quantifier-free Non-linear Real Arithmetic

In this section we formalise the syntax and semantics of Quantifier-free Non-linear Real Arithmetic and the related problem we try to solve utilising the CAlC method.

Definition 2.1.1 (Variable Ordering)**.** Let $\mathcal{X}$ be a set of variables. A *variable ordering* $\prec_{\mathcal{X}}$ is an order on $\mathcal{X}$, i.e. a transitive and antisymmetric relation. If $\mathcal{X}$ is unambiguous from the context, we omit the set and denote the ordering by $\prec$.

Within this thesis, we stick to the ordering $x_1 \prec \cdots \prec x_n$ if $\mathcal{X} = \{x_1, \ldots, x_n\}$ and $x \prec y \prec z$ if $\mathcal{X} = \{x, y, z\}$.

Definition 2.1.2 (Polynomial)**.** A *polynomial in normal form* is of the form

$$p = \sum_{i=1}^{k} \left(c_i \cdot \prod_{j=1}^{n} x_j^{e_{ij}} \right)$$

with $n, k \geq 0$ natural numbers, $\mathcal{X} = \{x_1, \ldots, x_n\}$ a set of variables, $c_1, \ldots, c_k \in \mathbb{Q}$ rational numbers, and $e_1 = (e_{11}, \ldots, e_{1n}), \ldots, e_k = (e_{k1}, \ldots, e_{kn}) \in \mathbb{N}^n$ pairwise different n-dimensional tuples.

The set of all polynomials in normal form over the variables $\mathcal{X}$ is denoted by $\mathbb{Q}[x_1, \ldots, x_n]$. The notation points out that the coefficients of such polynomials are rationals and $x_1, \ldots, x_n$ the used variables. The latter ones can be assigned to any real value. We will not always stick to the normal form but different representations, e.g. factorised polynomials.

If the number of different variables appearing with a non-zero coefficient as well as a non-zero exponent is greater than one, then p is called a *multivariate polynomial*, p is *univariate* in case the number is exactly one, and otherwise *constant*.

Example 2.1.1. The polynomial $p_1 := (x-1)^2 + (y-1)^2$ is multivariate in x and y, i.e. $p_1 \in \mathbb{Q}[x, y]$. The polynomial $p_2 := (x-1)^2 + 2$ is univariate in x.

Definition 2.1.3. Let $p \in \mathbb{Q}[x_1, \ldots, x_n]$ be a multivariate polynomial. It can be interpreted univariately as a polynomial $p \in \mathbb{Q}[x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n][x_i]$ in the variable x_i with coefficients in $\mathbb{Q}[x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n]$.

Definition 2.1.4 (Variables of a Polynomial)**.** Let p be a polynomial according to Definition 2.1.2. The set $\mathrm{var}(p)$ of variables appearing in p is inductively defined:

- If $p = x$ is a variable, then $\mathrm{var}(p) = \{x\}$.
- If $p = c$ is a constant $c \in \mathbb{Q}$, then $\mathrm{var}(p) = \emptyset$.
- If $p = \tilde{p} + \hat{p}$, then $\mathrm{var}(p) = \mathrm{var}(\tilde{p}) \cup \mathrm{var}(\hat{p})$.
- If $p = \tilde{p} \cdot \hat{p}$, then $\mathrm{var}(p) = \mathrm{var}(\tilde{p}) \cup \mathrm{var}(\hat{p})$.

By the precedences of addition and multiplication, p can be structurally decomposed. Note that exponentiation is just a shorthand notation for repeated multiplication.

Definition 2.1.5 (Degree and Level of a Polynomial)**.** Let p be a polynomial using the notation of Definition 2.1.2. The *degree* of p

$$\deg(p) := \max_{\substack{1 \le i \le k \\ c_i \neq 0}} \sum_{j=1}^{n} e_{ij}$$

is the largest sum of all exponents of variable products in p with non-zero coefficients.

The *main variable* of p is the highest variable regarding the order in Definition 2.1.1 that appears in p with a non-zero coefficient as well as a non-zero exponent (if it exists). If x_i is the main variable of p, then p is of *level* i, denoted by $\mathrm{level}(p) = i$. If no such x_i exists, we say $\mathrm{level}(p) = 0$.

Definition 2.1.6 (Polynomial Irreducibility and Square-Freeness, as in [NÁS+22])**.** Let $p \in \mathbb{Q}[x_1, \ldots, x_n] \setminus \{0\}$ be a non-zero polynomial. If $p = \tilde{p} \cdot \hat{p}$ for some polynomials $\tilde{p}, \hat{p} \in \mathbb{Q}[x_1, \ldots, x_n]$, then $\tilde{p}$ and $\hat{p}$ are called *factors* of p. The set of all factors of p is denoted by $\mathrm{factors}(p)$.

We call p *irreducible* if for all $\tilde{p}, \hat{p}$ with $p = \tilde{p} \cdot \hat{p}$ we have that either $\tilde{p} \in \mathbb{Q}$ or $\hat{p} \in \mathbb{Q}$ and p is *square-free* if for all non-constant $f \in \mathrm{factors}(p)$ we have $f^2 \notin \mathrm{factors}(p)$.

If polynomials are compared to each other, we obtain constraints.

Definition 2.1.7 (Constraint)**.** Let $p_1, p_2 \in \mathbb{Q}[x_1, \ldots, x_n]$ be polynomials. A *constraint* c over p_1 and p_2, denoted $c : p_1 \sim p_2$, compares p_1 and p_2 using a comparison operator $\sim \in \{<, \le, =, \neq, \ge, >\}$. The constraint c is in *normal form* if p_1 is in normal form and $p_2 = 0$. In that case, the polynomial p_1 is the *defining polynomial* of c.

Every constraint $c : p_1 \sim p_2$ can be transformed to normal form by subtracting p_2 from both sides and transforming the resulting polynomials to normal form.

Example 2.1.2. The polynomials p_1 and p_2 from Example 2.1.1 are defining for the constraints $c_1 : p_1 > 0$ and $c_2 : p_2 \leq 1$ or equivalently $c_2 : (x-1)^2 + 2 \leq 1$. The latter one simplifies to $x^2 - 2x + 2 \leq 0$ being in normal form.

Definition 2.1.8 (Quantifier-free Non-linear Real Arithmetic (QFNRA))**.** Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a set of variables and $\neg, \wedge, \vee$ the standard Boolean connectives. A *Quantifier-free Non-linear Real Arithmetic* formula φ over $\mathcal{X}$ is obtained using the following rules:

$$\begin{array}{lcl} \mathit{poly} & := & \mathit{const} \mid x_1 \mid \ldots \mid x_n \mid (\mathit{poly} + \mathit{poly}) \mid (\mathit{poly} \cdot \mathit{poly}) \\ \mathit{constr} & := & \mathit{poly} < 0 \mid \mathit{poly} \leq 0 \mid \mathit{poly} = 0 \mid \mathit{poly} \neq 0 \mid \mathit{poly} \geq 0 \mid \mathit{poly} > 0 \\ \varphi & := & \mathit{constr} \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid \neg\varphi \end{array}$$

Hereby, *const* represents all rational numbers $\mathit{const} \in \mathbb{Q}$. Note that constraints appearing in QFNRA formulae are normal formed.

Having defined polynomials and constraints syntactically, we now specify their semantics by means of assignments.

Definition 2.1.9 (Assignment)**.** Let $\mathcal{X} := \{x_1, \ldots, x_n\}$ be a set of variables. An *assignment* $\mathcal{V} : \mathcal{X} \to \mathbb{R}$ is a function that maps variables to the real domain, i.e. assigns them to real numbers. Instead of stating the variables and the assignment separately, we often use the notation $(x_1 \mapsto \alpha_1, \ldots, x_n \mapsto \alpha_n)$ or simply $(\alpha_1, \ldots, \alpha_n)$ for real values $\alpha_1, \ldots, \alpha_n$ as a shorthand notation for the function $\mathcal{V} : \mathcal{X} \to \mathbb{R}$, $x_i \mapsto \alpha_i$ with $1 \leq i \leq n$.

Definition 2.1.10 (Evaluation)**.** Let $\mathcal{V}$ be an assignment, p, p_1, p_2 polynomials, and $c : p_1 \sim p_2$ a polynomial constraint. The *(partial) evaluation of a polynomial* p under $\mathcal{V}$ results in another polynomial $\hat{p}$ which is obtained from p by replacing the variables in p according to $\mathcal{V}$ and transforming the result into the normal form. The evaluation is denoted by $\hat{p} = [\![p]\!]^{\mathcal{V}}$ or $\hat{p} = p(s)$ if we use the notion of a point s instead of an assignment function $\mathcal{V}$. In general, we cannot assume that $\hat{p} \in \mathbb{R}$ because $\mathcal{V}$ might not instantiate every variable in $\mathrm{var}(p)$. Moreover, $\hat{p}$ does not need to have only rational coefficients.

A constraint c can be evaluated employing an assignment $\mathcal{V}$ if it assigns real numbers to all variables in $\mathrm{var}(p_1) \cup \mathrm{var}(p_2)$. The *evaluation* $[\![c]\!]^{\mathcal{V}}$ yields a truth value from the *Boolean domain* $\mathbb{B} := \{\texttt{True}, \texttt{False}\}$ where $[\![c]\!]^{\mathcal{V}} = \texttt{True}$ if and only if $[\![p_1]\!]^{\mathcal{V}} \sim [\![p_2]\!]^{\mathcal{V}}$ holds. Hereby, we make use of the standard semantics of $\sim$. If $\mathcal{V}$ does not assign all variables in $\mathrm{var}(p_1) \cup \mathrm{var}(p_2)$, the *partial evaluation* $[\![c]\!]^{\mathcal{V}}$ again might be a constraint.

The evaluation of a QFNRA formula is performed the standard way, i.e. first the constraints are evaluated and then the Boolean truth values are combined according to the structure of the formula.

Definition 2.1.11 (Non-linear Real Arithmetic Satisfiability Problem)**.** Let $\mathcal{F}$ be a QFNRA formula over $\mathcal{X}$. The *Non-linear Real Arithmetic Satisfiability Problem* asks whether there exists an assignment $\mathcal{V}$ for $\mathcal{X}$ such that $[\![\mathcal{F}]\!]^{\mathcal{V}} = \texttt{True}$. That way, the variables in $\mathcal{F}$ are implicitly existentially quantified.

2.2 Satisfiability Checking

When solving the problem from Definition 2.1.11 for a QFNRA formula $\mathcal{F}$, the goal of *Satisfiability Checking* is, to determine whether there exists an assignment $\mathcal{V}$ that satisfies $\mathcal{F}$. If so, the answer is SAT otherwise UNSAT. Solving this question does not only relate to QFNRA formulae. When using Boolean variables instead of constraints, one obtains formulae of *Propositional Logic*. In the 1970s, Cook and Levin showed that the *Propositional Satisfiability Problem* is NP-complete [Coo71, Lev73]. Thus, it is possible to algorithmically determine the satisfiability of a Propositional Logic formula, but as we know so far, it is computational expensive as well.

When dealing with QFNRA formulae, we do not only have to consider the Boolean structure but also the truth values of the constraints when assigning the variables. Hence, an extension of pure Boolean Satisfiability Checking is needed, called *Satisfiability Modulo Theories* (SMT) solving. The key aim is to determine the satisfiability of a sentence, i.e. a logic formula without free variables from an existential fragment of a first-order theory. We do not introduce *First-order Logic* itself and instead refer to [Fit96].

In the 1950s, the logician Alfred Tarski has shown that the *Quantifier Elimination Problem* is decidable for real arithmetic [Tar51]. That implies the satisfiability of conjunctively connected polynomial constraints can be determined algorithmically, too. However, the original method presented by Tarski is very inefficient in terms of complexity, so different methods have been evolved.

2.3 CAD Techniques

There exist many approaches to determine the satisfiability of QFNRA formulae, e.g. *Interval Constraint Propagation* [FHT+06], *Virtual Substitution* [CÁ11], *Subtropical Satisfiability* [FOSV17], or the *Cylindrical Algebraic Decomposition* (CAD) [Col75] method.

We will focus on the techniques used in the latter one, which, contrary to the other methods above, is complete, but we do not give a full inside into this method as we are dealing with a related procedure below in Section 2.4.

The CAD method was introduced by Collins in 1975. When given polynomial constraints in n variables, the key idea is to partition the real space $\mathbb{R}^n$ into finitely many cells, such that every defining polynomial of a constraint maintains its sign on each cell [Col75, NÁS+22]. This is possible as every univariate polynomial has finitely many zeros. In the following, we give some basic definitions.

Definition 2.3.1 (Sign)**.** The *sign* function $\sigma : \mathbb{R} \to \{-1, 0, +1\}$ states, whether a real number is negative, zero, or positive.

Note that zero is a sign itself, because the comparison operators we are heading for might only differ in their judgement on zero. Most of the time we neglect the explicit use of the sign function and instead use comparisons to express a sign condition.

Definition 2.3.2 (Cell, as in [NÁS+22])**.** Let $n, m \in \mathbb{N}$, $S \subseteq \mathbb{R}^m$ for some $0 \leq m \leq n$, $P \subseteq \mathbb{Q}[x_1, \ldots, x_n]$ a finite, non-empty set of polynomials, and $\mathcal{C}$ a finite set of constraints built by comparing the polynomials from P to zero according to Definition 2.1.7. S is called a *cell* if it is a non-empty connected set.

If $P' = P \cap \mathbb{Q}[x_1, \ldots, x_m]$, then P' is *sign-invariant* on S if for all $p \in P'$ and $a, b \in S$ there holds $\sigma(p(a)) = \sigma(p(b))$. The set $\mathcal{C}' \subseteq \mathcal{C}$ of constraints built on P' is *truth-invariant* on S if $[\![c]\!]^a = [\![c]\!]^b$ for every $c \in \mathcal{C}'$ and $a, b \in S$.

The cell S is *semi-algebraic* if it is the solution set of a Boolean combination of polynomial constraints. It is *sampled* if a particular point $s \in S$ is associated with S.

Definition 2.3.3 (Cylindrical Algebraic Decomposition (CAD), as in [NÁS+22])**.** A set $D = \{S_1, \ldots, S_k\}$ of cells of $\mathbb{R}^n$ with $\bigcup_{1 \leq i \leq k} S_i = \mathbb{R}^n$ and $S_i \cap S_j = \emptyset$ for all $1 \leq i \neq j \leq k$ is called a *decomposition* of $\mathbb{R}^n$. It is *algebraic* if its cells are semi-algebraic. It is called *cylindrical* over some decomposition D' of $\mathbb{R}^m$ for some $m < n$ if all projections of cells $S \in D$ onto $\mathbb{R}^m$ are cells in D', i.e. the cells in $\mathbb{R}^n$ are stacks of cells filling the cylinder over the cells of D'.

A *Cylindrical Algebraic Decomposition* (CAD) D with respect to the variable ordering $\prec$ is obtained by a sequence $(D_1, \ldots, D_n)$ where $D = D_n$ and each D_i is a cylindrical algebraic decomposition of $\mathbb{R}^i$ over D_{i-1} for all $i \in \{2, \ldots, n\}$ and D_1 is an algebraic decomposition, too.

The key idea is that a decomposition inherits the invariance properties of the polynomials and, thus, of the constraints. That way, reasoning about the satisfiability of the input constraints can be broken down to lower dimensions. This happens with respect to a projection operator. Within this thesis, we make use of the McCallum projection operator, an improvement of the original projection used by Collins [McC98].

Definition 2.3.4 (McCallum Projection Operator, as in [McC98])**.** Let $x_1 \prec \ldots \prec x_n$ be the used variables, $i \in \{2, \ldots, n\}$, and $P \subseteq \mathbb{Q}[x_1, \ldots, x_i] \setminus \{0\}$ a set of irreducible polynomials. The *McCallum Projection of* P, denoted by $\text{proj}_{mc}(P)$, is defined as

$$\text{proj}_{mc}(P) := \bigcup_{\substack{p \in P \\ \text{level}(p) = i}} \text{coeff}_{x_i}(p) \cup \bigcup_{\substack{p \in P \\ \text{level}(p) = i}} \{\text{disc}_{x_i}(p)\} \cup \bigcup_{\substack{p,q \in P,\, p \neq q \\ \text{level}(p) = i \\ \text{level}(q) = i}} \{\text{res}_{x_i}(p, q)\} \cup \bigcup_{\substack{p \in P \\ \text{level}(p) < i}} \{p\}.$$

For all resulting polynomials $\hat{p} \in \text{proj}_{mc}(P)$ there holds $\text{level}(\hat{p}) < i$: If $\mathcal{X}_P$ is the set of variables appearing in P and the projection is calculated with respect to x_i, then the resulting polynomials' variables $\mathcal{X}_{\text{proj}}$ do not contain x_i and even more $\mathcal{X}_{\text{proj}} \subseteq \mathcal{X}_P \setminus \{x_i\}$. That simplifies reasoning about cells later on. In exchange, it is often the case that the resulting polynomials' degrees are increased.

We do not formally analyse this projection operator but give an intuition for the methods used in the projection steps. Therefore, we discuss the leading coefficient but not all coefficients used in proj_{mc}. Note that we will not call this projection operator in the CAlC method later on and instead state the calculations explicitly.

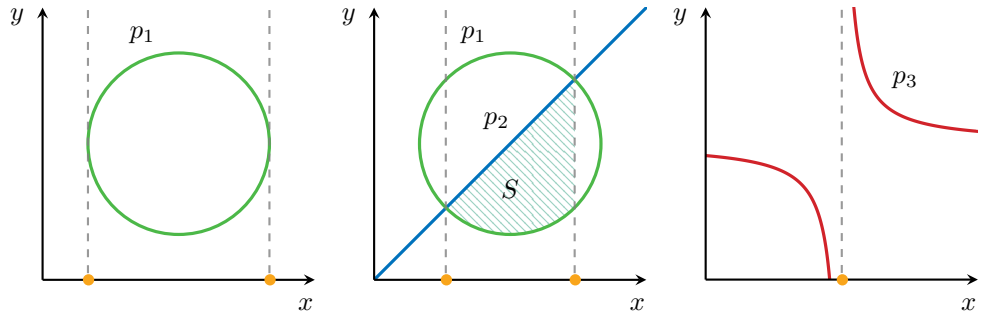
Definition 2.3.5 (Resultant, as in [Rot10, Kna08])**.** Let p_1, p_2 be univariately represented polynomials in the variable x. The *resultant* of p_1 and p_2 with respect to x is the determinant of the Sylvester matrix of p_1 and p_2:

$$\text{res}_x(p_1, p_2) = \text{Det}(\text{Sylvester}(p_1, p_2)).$$

Definition 2.3.6 (Discriminant, as in [Rot10])**.** Let p be a univariately represented polynomial in x with $\deg(p) = k$. The *discriminant* of p with respect to x is

$$\text{disc}_x(p) := \frac{1}{c_k} \cdot (-1)^{\frac{k(k-1)}{2}} \cdot \text{res}_x(p, p'),$$

whereby c_k is the leading coefficient of p and p' is the first derivative of p.



(a) Discriminant **(b)** Resultant **(c)** Leading coefficient

Figure 2.3.1: Visualisation of the three major projection steps. Depicted are the zeros of polynomials p_1, p_2, and p_3. As an abuse of notation, we name the zeros as their corresponding polynomials. When calculating the discriminant, resultant, or leading coefficient of the polynomials, one receives a polynomial of lower level whose zeros are depicted in orange. An example for a sign-invariant cell is given in Figure 2.3.1b.

Example 2.3.1. In Figure 2.3.1a the zero of the polynomial $p_1 = y^2 + x^2 - 1$ is depicted. The discriminant $\text{disc}_y(p_1) = -4(x^2 - 1)$ is zero for $x = -1$ and $x = 1$, indicating that the number of zeros of p_1 with respect to y changes there: While p_1 has no zeros for $x < -1$, p_1 has one zero at $x = -1$, two zeros for $-1 < x < 1$, one zero at $x = 1$, and no zeros for $x > 1$. Intuitively, a parallel of the y-axis can be shifted along the x-axis. If one marks the positions where the number of zeros of p_1 laying on the currently shifted y-axis changes, these are the zeros of $\text{disc}_y(p_1)$.

When calculating discriminants, resultants, or coefficients, the points of interest are mainly the zeros of the obtained polynomials. Hence, we store only square-free bases in normal form and factorise polynomials as far as possible to simplify calculations later on. To meet readability, we still use the equals sign '=' when speaking about discriminants, resultants and coefficients, e.g. in Example 2.3.1 we write $\text{disc}_y(p_1) = (x-1)(x+1)$ and store $(x-1)$ and $(x+1)$ separately.

Example 2.3.2. In Figure 2.3.1b the zeros of p_1 and another polynomial $p_2 = y - x$ are depicted. The resultant $\text{res}_y(p_1, p_2) = x^2 - \frac{1}{2}$ of p_1 and p_2 indicates where p_1 and p_2 have common zeros. When moving the y-axis again, the only positions where p_1 and p_2 have intersecting zeros are above $x = -\underline{0.71}$ and $x = \underline{0.71}$.

We use the '$\underline{\cdot}$'-notation to mark rounded numbers. In Example 2.3.2 we write $\underline{0.71}$ instead of $\frac{1}{\sqrt{2}}$.

Example 2.3.3. An example of a sign-invariant cell of the polynomials p_1 and p_2 is given in Figure 2.3.1b by the crosshatched region. For every $s \in S$ there holds $\sigma(p_1(s)) = -1$ and $\sigma(p_2(s)) = -1$. Note, however, that this cell is not maximal sign-invariant, i.e. it could be extended towards its right.

Definition 2.3.7 (Leading Coefficient)**.** Let p be a univariately represented polynomial in the variable x with $\deg(p) = k$. The *leading coefficient* $\text{lcoef}_x(p)$ of p with respect to x is the coefficient c_k of the term $c_k \cdot x^k$ in p, i.e. $\text{lcoef}_x(p) = c_k$.

Example 2.3.4. Figure 2.3.1c illustrates the importance of coefficients. The polynomial $p_3 = xy - 1$ has an asymptotic behaviour at $x = 0$. Again, this can be seen by

shifting a parallel of the y-axis to $x = 0$ or by determining the zero of $\text{lcoef}_y(p_3) = x$ which is $x = 0$.

If the univariate representation of a polynomial p is unambiguous from the context, we omit to specify it, e.g. we write $\text{lcoef}(p_3)$ instead of $\text{lcoef}_y(p_3)$. The same applies to resultants and discriminants.

2.4 Cylindrical Algebraic Coverings

The decomposition into a CAD is done dimensionwise. In the end, for every obtained full-dimensional cell, a sample point is chosen to evaluate all input constraints and to determine their satisfiability at that point, i.e. of that cell. After testing all samples, a conclusion can be drawn whether the set of constraints is SAT or UNSAT.

In this section we introduce the CAlC method, a procedure related to the CAD that draws conclusions also from partial samples. The following definitions and algorithms are slightly modified or shortened versions of the ones presented in [ÁDEK21].

In 2021, the Cylindrical Algebraic Covering method was presented by Ábrahám et al. It relaxes disjointness of the cells as long as cylindricity is maintained. Furthermore, it works incrementally to guide itself (hopefully) further away from conflicts than the pure CAD does [ÁDEK21].

Definition 2.4.1 (Cylindrical Algebraic Covering (CAlC), as in [ÁDEK21])**.** A *covering* of $\mathbb{R}^n$ is a finite set $C = \{S_1, \dots, S_k\}$ of cells with $\bigcup_{S \in C} S = \mathbb{R}^n$. The covering is *algebraic* if every cell of it is semi-algebraic, and it is *sampled* if it is associated with exactly one sample point $s \in S$ for each cell $S \in C$.

If $0 < m < n$ and C is a covering of $\mathbb{R}^n$ as well as C' a covering of $\mathbb{R}^m$, then C is called *cylindrical over* C' if its projection onto $\mathbb{R}^m$ assures that every $S \in C$ gets projected to an $S' \in C'$. The cells $S \in C$ being projected onto the same cell $S' \in C'$ form a *cylinder* over S'.

In addition, if C is a sampled covering being cylindrical over some sampled C', there has to hold that the sample point of a cell $S' \in C'$ is the projection of the samples assigned to the cells from C forming the cylinder over S'.

If for all $0 < m < n$, where C is a sampled covering of $\mathbb{R}^n$, there exists a sampled covering C' of $\mathbb{R}^m$ being its projection, then C is called *cylindrical*. A covering of $\mathbb{R}$ is always called cylindrical.

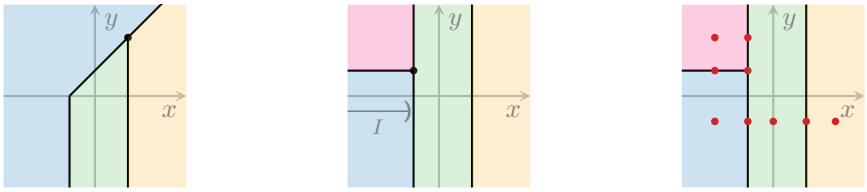
Example 2.4.1. Figure 2.4.1 illustrates three coverings of $\mathbb{R}^2$. Each coloured area, line, and black point indicates a cell.

Assuming the variable ordering $x \prec y$, the covering in Figure 2.4.1a is not cylindrical over $\mathbb{R}$ (the x-axis), but the one in Figure 2.4.1b is. Inspecting the blue cell in Figure 2.4.1a, we identify another cell, e.g. the green one, which shares its x-interval with the blue cell, but the interval is a proper subset of the blue one's.

The blue and magenta cells in Figure 2.4.1b have a common x-interval as well, but none of them is a proper subset of each other. Thus, together with the black horizontal line, they form a cylinder over the x-axis' interval I.

The last covering in Figure 2.4.1c is equal to the one in Figure 2.4.1b but is sampled by the red points. After projection, all the samples over one cylinder have the same x-coordinate. Hence, the whole covering is cylindrical and sampled.

Note that none of the coverings is cylindrical using the variable ordering $y \prec x$.



(a) Non-cylindrical covering **(b)** Cylindrical covering **(c)** Sampled covering

Figure 2.4.1: Three examples of coverings for $\mathbb{R}^2$. Each one consists of cells indicated by the coloured areas, black lines, and black points. Red points illustrate possible samples.

2.4.1 CAlC Method

With the CAlC method we check a finite set of polynomial constraints for consistency, i.e. we check whether their conjunction is satisfiable.

For the whole computations, the method maintains a sample point s, i.e. an assignment, at which satisfiability is checked. The sample, however, does not need to have full dimension n as in the CAD method. It might assign real values for the first $i < n$ variables with respect to the variable ordering but leaves out the other ones.

Starting with a zero-dimensional sample, i.e. an empty sample point $s = ()$, it is incrementally extended onto the next dimension. Each partial sample with assigned components $s_1, \dots, s_{i-1}$ is validated against the suitable constraints, namely those get evaluated, whose main variable is x_i. That way, we end up with univariate constraints. Two steps are necessary:

First, we exclude all the subsets of $\mathbb{R}$ that, when extending s with a value from them for s_i, lead to a violation of at least one constraint of level i. That way, the possible choice for s_i is restricted. Either there is no real value left for s_i, i.e. the whole set $\mathbb{R}$ is excluded, or a non-empty subset of $\mathbb{R}$ is remaining. In the latter case, s_i gets assigned and the next dimension is explored. However, if the excluded sets cover $\mathbb{R}$, a conflict appeared. Hence, the algorithm checks the assignments in inverse chronological order, i.e. it starts with s_{i-1} first, and calculates an interval around s_{i-1} that does not satisfy the constraints for the same reason. This region might be a point interval (called a *section*) $[s_{i-1}; s_{i-1}]$ or an open interval (called a *sector*) $(\ell; u) \subseteq \mathbb{R} \cup \{-\infty, \infty\}$ with $\ell < s_{i-1} < u$. We later discuss how these intervals are technically concluded. It might be that the resulting interval supplements previously found intervals on the level $i - 1$ towards a full covering of the real line again. If so, the backtracking step repeats recursively.

Termination of the algorithm takes place in two cases:

- If the sample point has full dimension and no constraint is unsatisfied, the algorithm terminates with SAT, indicating the satisfiability of the constraints.
- In case the algorithm backtracks to the empty sample with no uncovered region left for choosing s_1, the constraint set is proven to be unsatisfiable and, thus, the algorithm returns UNSAT.

Theoretically, this method is complete, however, the projection operator proj_{mc} is not. Therefore, the CAlC method might return UNKNOWN in some cases. We now discuss the four main algorithms of the CAlC method as presented by [ÁDEK21], slightly adapted to the previous notation. In Section 2.4.3 we illustrate the procedure by means of an example.

2.4.2 Algorithmic Procedure

The core of the procedure is `get_unsat_cover` (Algorithm 1). Given a partial sample $s = (s_1, \dots, s_{i-1})$ which initially is $s = ()$, `get_unsat_cover` first chooses an extension s_i of that sample point onto the following dimension. If afterwards the sample has full dimension, i.e. $i = n$, no conflict was found and SAT is returned. Otherwise, the recursive call to Algorithm 1 in Line 6 checks whether a satisfying extension of the current partial sample exists. If a sample cannot be extended, an UNSAT interval around the latest component s_i is characterised by Algorithm 3 and Algorithm 4. Sampling s_i within this interval would lead to the same conflict. The necessary information is encoded in an interval structure.

Definition 2.4.2 (Interval Structure)**.** Let $\ell \in \mathbb{R} \cup \{-\infty\}$, $u \in \mathbb{R} \cup \{\infty\}$ and $L, U, P_i \subseteq \mathbb{Q}[x_1, \dots, x_i] \setminus \mathbb{Q}[x_1, \dots, x_{i-1}]$ as well as $P_\perp \subseteq \mathbb{Q}[x_1, \dots, x_{i-1}]$. An *interval structure*

$$I = (\ell, u, L, U, P_i, P_\perp) \text{ or } I = (I_\ell, I_u, I_L, I_U, I_{P_i}, I_{P_\perp})$$

describes an UNSAT interval consisting of

- the lower bound ℓ of the interval,
- the upper bound u of the interval,
- polynomials L defining ℓ by one of their zeros (in case $\ell \neq -\infty$ otherwise $L = \emptyset$),
- polynomials U defining u by one of their zeros (in case $u \neq \infty$ otherwise $U = \emptyset$),
- a set P_i of polynomials with main variable x_i that appeared in a (previous) characterisation step, and
- a set $P_\perp$ of polynomials with main variables smaller than x_i.

The represented interval is open at both ends if $\ell \neq u$ and closed at both ends otherwise. Note that P_i and $P_\perp$ are trivially initialised by `get_unsat_intervals`, but they can be modified after calling `construct_characterisation`.

Algorithm 1: get_unsat_cover(s), as in [ÁDEK21]

Input: (Partial) sample point $s = (s_1, \dots, s_{i-1}) \in \mathbb{R}^{i-1}$ that does not evaluate any constraint to `False`.
Output: (SAT, O) for a satisfying sample $O \in \mathbb{R}^n$ or $(\text{UNSAT}, \mathbb{I})$ where $\mathbb{I}$ covers $\mathbb{R}$.

```
1  I := get_unsat_intervals(s)
2  while ⋃_{I∈I} I ≠ R do
3      s_i := sample_outside(I)
4      if i = n then
5          return (SAT, (s_1, ..., s_n))
6      (res, O) := get_unsat_cover((s_1, ..., s_i))
7      if res = SAT then
8          return (SAT, O)
9      else
10         R := construct_characterisation((s_1, ..., s_i), O)
11         I := interval_from_characterisation((s_1, ..., s_{i-1}), s_i, R)
12         I := I ∪ {I}
13 return (UNSAT, I)
```

Algorithm 2: get_unsat_intervals(s), as in [ÁDEK21]

Data: Global set of constraints C.
Input: (Partial) sample point $s = (s_1, \ldots, s_{i-1}) \in \mathbb{R}^{i-1}$.
Output: A set $\mathbb{I}$ of intervals such that $\{s\} \times I$ unsatisfies at least one constraint for all $I \in \mathbb{I}$.

```
1  𝕀 := ∅
2  C_i := the subset of C with main variable x_i
3  foreach c = p ~ 0 ∈ C_i do
4      c' := [[c]]^s                          // Yields truth value or univariate constraint
5      if c' = False then
6          I := (I_ℓ = −∞, I_u = ∞, I_L = ∅, I_U = ∅, I_{P_i} = {p}, I_{P_⊥} = ∅)
7          return 𝕀 := {I}
8      if c' = True then
9          continue
10     Z := real_roots(p, s)                  // Ordered list z_1 < ··· < z_k
11     Regions := {(−∞; z_1), [z_1; z_1], ..., (z_k; ∞)}   // (−∞; ∞) if Z was empty
12     foreach J = (ℓ; u) ∈ Regions do
13         Pick r ∈ J
14         if [[c']]^r = False then
15             L := ∅   U := ∅
16             if ℓ ≠ −∞ then
17                 L := {p}
18             if u ≠ ∞ then
19                 U := {p}
20             I := (I_ℓ = ℓ, I_u = u, I_L = L, I_U = U, I_{P_i} = {p}, I_{P_⊥} = ∅)
21             𝕀 := 𝕀 ∪ {I}                   // After simplifications
22 return 𝕀
```

Initially, `get_unsat_intervals` evaluates those input constraints $c : p \sim 0$ that become univariate after substitution of the current partial sample. It distinguishes three types of results:

- If the constraint is equivalent to `False` as in Line 5, e.g. after evaluating $x_{i-1}x_i > 0$ at $s = (x_{i-1} \mapsto 0)$, it can be concluded that no assignment for x_i will be satisfying. This is indicated by the resulting interval $(-\infty; \infty)$. Hence, the bounds are $\ell = -\infty$ and $u = \infty$ together with $L = U = \emptyset$. The responsible polynomial with main variable x_i is p meaning $P_i = \{p\}$, and no other polynomial, neither with main variable x_i, nor some x_j with $j < i$ is necessary, i.e. $P_\perp = \emptyset$. As the whole real line is already covered by a single interval, the procedure terminates.

- If the substituted constraint evaluates to `True` as in Line 8, no infeasible interval can be derived and the next constraint is tested.

- The most elaborate case takes place in Line 10, if the partial evaluation of p becomes univariate but is not unsatisfiable. Now the real zeros of the resulting univariate polynomial $\hat{p}$ are calculated, and every zero, every interval between two zeros, the open left, and the open right interval before and after the first respectively last zero of $\hat{p}$ are used to evaluate c and to check for satisfiable regions. For example, we would obtain the intervals $(-\infty; 0), [0; 0], (0; \infty)$ if $\hat{p} = x^2$.

  For each interval, a value is chosen at which $\hat{p}$ is evaluated and the truth value of the constraint c is determined. Note that the choice within the interval is

arbitrary as the satisfaction of c cannot change in between two zeros of $\hat{p}$. If c is violated, the interval is stored together with the responsible polynomial p as mentioned in the first case. Otherwise, the next interval is examined.

Thus, we obtain a set of intervals covering some part of the real line. Back in Algorithm 1, an assignment s_i outside of these intervals is chosen, unless the real line is covered in total. However, if the set $\mathbb{I}$ from Algorithm 1 covers the real line for x_i, UNSAT is propagated to the parent call, which will use `construct_characterisation` to generalise the conflict and to update the latest assignment s_{i-1}. In particular, there might be infinitely many values for s_{i-1} which run into the same conflict. Therefore, `interval_from_characterisation` deduces a hopefully large interval around s_{i-1} which can be neglected when reassigning x_{i-1}.

To understand the characterisation, it is helpful to consider the cell induced by an interval structure. To meet readability, from now on we assume that the $(i+1)$th dimension is fully covered and an interval around s_i, the highest assigned component of s, should be deduced. To that end, let I be an interval structure on level $i+1$ deduced in the CAlC method bounded by L and U. The variable x_{i+1} is the highest appearing in $L \cup U$ and $\hat{L} \cup \hat{U}$ is the set of all polynomials from $L \cup U$ partially evaluated at $(s_1, \dots, s_{i-1})$. The smallest set around s bounded by the zeros of $\hat{L} \cup \hat{U}$ and the ones of I's characterisation is a (two-dimensional view on a) cell. In Figure 2.4.2a two polynomial zeros are illustrated. The zero of $p \in U$ is bounding the corresponding cell induced by its interval structure I. The cell is highlighted as a crosshatched area of the same colour as the zero of p. The polynomial q is contained in $P_{i+1} \setminus (L \cup U)$, i.e. it is not defining for I's bounds. As an abuse of notation, we name the zeros as their responsible polynomials, i.e. we write p instead of $p = 0$. Note that the expansion of the crosshatched cell in Figure 2.4.2a is limited by the common zero of p and q. To identify such points, the other components of I have to be taken into account.

It remains to be clarified how an interval around s_i (a local view on the above cell) is inferred. To that end, a (possibly smaller) non-redundant subset of $\mathbb{I}$ is discovered by `compute_cover`, whereby the notion of non-redundancy is introduced below.

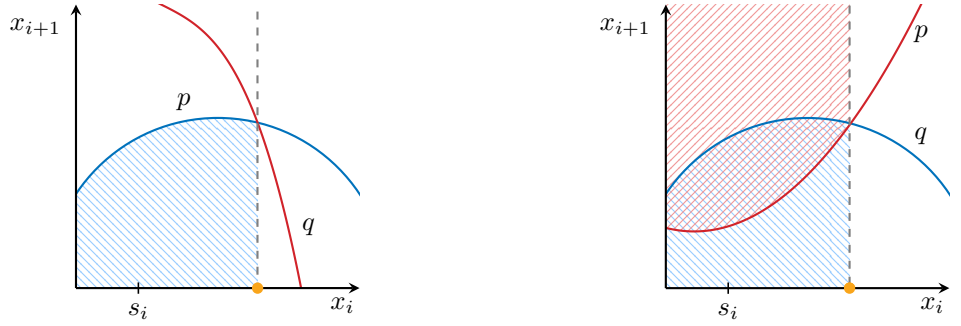
Definition 2.4.3 (Interval Redundancy and Ordering)**.** Let $\mathbb{I} = \{I_1, \dots, I_k\}$ be a set of interval structures covering the real line. The set $\mathbb{I}$ is *non-redundant* if for all I_j with $1 \leq j \leq k$ the subset $\mathbb{I} \setminus \{I_j\}$ does not cover the real line anymore. On the set $\mathbb{I}$ we make use of the total order $\preceq_{\mathbb{I}}$ which for $I, J \in \mathbb{I}$ is defined by

$$I \preceq_{\mathbb{I}} J \Leftrightarrow I_\ell \leq J_\ell \wedge (I_\ell < J_\ell \vee I_u \leq J_u).$$

The intervals I and J are called *consecutive* if $I \preceq_{\mathbb{I}} J$ or $J \preceq_{\mathbb{I}} I$ and there exists no $K \in \mathbb{I} \setminus \{I, J\}$ with $I \preceq_{\mathbb{I}} K \preceq_{\mathbb{I}} J$ respectively $J \preceq_{\mathbb{I}} K \preceq_{\mathbb{I}} I$.

To derive an interval on the level i over which the cells induced by the covering form a cylinder, a set of polynomials R is computed based on the non-redundant covering. It is defined the following way:

- Polynomials p appearing in the intervals $I \in \mathbb{I}$ with $\text{level}(p) < i+1$ are inherited from the intervals in $\mathbb{I}$ without modifications.
- The remaining polynomials need to be projected before they are stored in R to determine the lower and upper bounds in Algorithm 4 later on. The calculated projection is computed as explained in Definition 2.3.4. We now focus on the appearing resultants:



(a) Resultant characterising a cell

(b) Resultant for a covering

Figure 2.4.2: Both types of resultants appearing in the computation of a CAlC. Depicted are the zeros of two polynomials p and q and in orange the zero of their resultant. Crosshatched areas illustrate the induced (truncated) cells.

- Firstly, the *resultants characterising a cell* are calculated in Lines 8 and 9. One has to consider common zeros of the polynomials $p \in L$ and $q \in P_{i+1}$, respectively $p \in U$ with $q \in P_{i+1}$. However, these are only necessary in case $q(s \times m) = 0$ for some $m \leq \ell$ or $m \geq u$, respectively, where $s \times m$ extends s by $(s_{i+1} : x_{i+1} \mapsto m)$. They guarantee that the cell remains sign-invariant over the projection interval. The situation where $p \in U$ is depicted in Figure 2.4.2a.
- Secondly, *resultants characterising a covering* are computed which are built across the UNSAT intervals: To obtain a connected covering, we have to ensure that the deduced interval does not exceed the points where the covering cells overlap. Thus, in a first step, we need to identify the common zeros of neighboured interval bounds, i.e. we calculate resultants to check if the lower bound q of an interval reaches beyond the upper bound p of its predecessor. An illustration is given in Figure 2.4.2b.

Afterwards, simplification steps are performed, e.g. removing squares or coefficients as discussed in Examples 2.3.1 and 2.3.2.

Having calculated the polynomials of interest, the CAlC method determines an interval I around s_i such that for every choice $(x_i \mapsto s_i')$ for some $s_i' \in I$ not only a satisfying extension is impossible, but also this happens for the same reason as for s_i.

To that end, based on the current sample, the real roots Z of polynomials with main variable x_i resulting from the (previous) characterisations are identified. Moreover, fallback values $\pm\infty$ are stored in case no polynomial restricts an extension in any direction. The deduced interval is based on the zeros in Z closest to s_i.

The additional information is again stored in an interval structure, but this time, the sets P_i (polynomials with main variable x_i) and $P_\perp$ (polynomials with main variable smaller than x_i) are composed of polynomials that may not be defining for any input constraints.

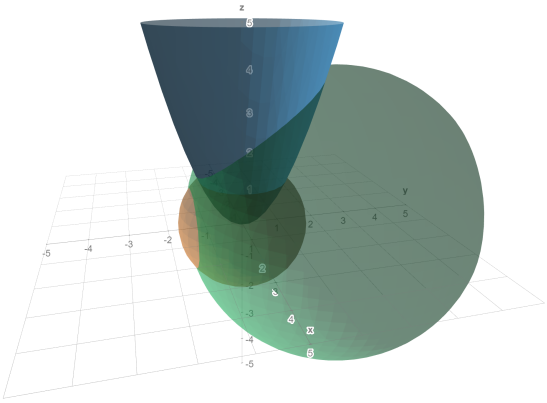


Figure 2.4.3: Illustration of the polynomial zeros belonging to the defining polynomials of the constraint set $\mathcal{C}$.

Algorithm 3: construct_characterisation$(s, \mathbb{I})$, as in [ÁDEK21]

Input: (Partial) sample point $s = (s_1, \ldots, s_i) \in \mathbb{R}^i$ and an UNSAT covering $\mathbb{I}$.
Output: A set $R \subseteq \mathbb{Q}[x_1, \ldots, x_i]$ of polynomials characterising an unsatisfiable region around s for the same reason.

$\mathbb{I} :=$ compute_cover($\mathbb{I}$)
$R := \emptyset$
foreach $I \in \mathbb{I}$ **do**
  Extract $\ell := I_\ell, u := I_u, L := I_L, U := I_U, P_{i+1} := I_{P_{i+1}}, P_\perp := I_{P_\perp}$
  $R := R \cup P_\perp$
  $R := R \cup \{\text{disc}(p) \mid p \in P_{i+1}\}$
  $R := R \cup \{\text{required\_coefficients}(p) \mid p \in P_{i+1}\}$
  $R := R \cup \{\text{res}(p, q) \mid p \in L,\ q \in P_{i+1},\ q(s \times m) = 0 \text{ for some } m \leq \ell\}$
  $R := R \cup \{\text{res}(p, q) \mid p \in U,\ q \in P_{i+1},\ q(s \times m) = 0 \text{ for some } m \geq u\}$
foreach $j \in \{1, \ldots, |\mathbb{I}| - 1\}$ **do**
  $R := R \cup \{\text{res}(p, q) \mid p \in U_j,\ q \in L_{j+1}\})$
return R `// After CAD simplifications`

Algorithm 4: interval_from_characterisation(s, s_i, P), as in [ÁDEK21]

Input: (Partial) sample point $s = (s_1, \ldots, s_{i-1}) \in \mathbb{R}^{i-1}$, an extension s_i, and a polynomial UNSAT characterisation P.
Output: Interval I around s_i so that the constraints are unsatisfiable on $\{s\} \times I$ for the same reason.

$P_\perp := \{p \in P \mid p \in \mathbb{Q}[x_1, \ldots, x_{i-1}]\}$
$P_i := P \setminus P_\perp$
$Z := \{-\infty\} \cup \text{real\_roots\_with\_check}(P_i, s) \cup \{\infty\}$
$\ell := \max\{z \in Z \mid z \leq s_i\}$
$u := \min\{z \in Z \mid z \geq s_i\}$
$L := \{p \in P_i \mid p(s \times \ell) = 0\}$
$U := \{p \in P_i \mid p(s \times u) = 0\}$
$I := (I_\ell = \ell, I_u = u, I_L = L, I_U = U, I_{P_i} = P_i, I_{P_\perp} = P_\perp)$
return I

2.4.3 Running Example

We conclude this chapter by introducing the running example of this thesis. It will be reused after each approach to illustrate the differences from the original method. Let $\mathcal{C}$ be the set of constraints

$$\mathcal{C} := \{ \quad \overbrace{\underbrace{z - y^2 - x^2}_{p_1} > 0}^{c_1}, \quad \overbrace{\underbrace{z^2 + y^2 + x^2 - 3}_{p_2} < 0}^{c_2}, \quad \overbrace{\underbrace{z^2 + (y - 2.5)^2 + x^2 - 16}_{p_3} > 0}^{c_3} \quad \}$$

Animation

whose polynomial zeros are depicted in Figure 2.4.3. The zero $p_1 = 0$ is illustrated in blue, $p_2 = 0$ in orange, and $p_3 = 0$ in green. The constraint c_1 is satisfied inside the paraboloid, c_2 inside the smaller sphere, and c_3 outside of the larger one. The zeros, i.e. the surfaces, are never satisfying. That way, every pair of constraints is satisfiable, but all together are not. A 3D-illustration is given in [Bär22].

$s = ()$
The procedure is started by calling `get_unsat_cover` with an empty sample $s = ()$. The set $\mathbb{I}$ of UNSAT intervals for the current dimension is set to $\mathbb{I} := \emptyset$ and by a call to `get_unsat_intervals`, every constraint in $\mathcal{C}$ is substituted by the current partial sample s and the univariate ones are analysed. However, without assigning any variable, no constraint becomes univariate and the set $\mathbb{I}$ remains empty, in particular, $\bigcup_{I \in \mathbb{I}} I \neq \mathbb{R}$. Therefore, any real number is suitable for being assigned to x as s_1. For simplicity we choose $(s_1 : x \mapsto 0)$. The updated sample $s = (s_1 : x \mapsto 0)$ has not yet full dimension, so a recursive call to `get_unsat_cover` is executed.

$s = (s_1 : x \mapsto 0)$
The situation is similar to the one before, so we choose $(s_2 : y \mapsto 0)$.

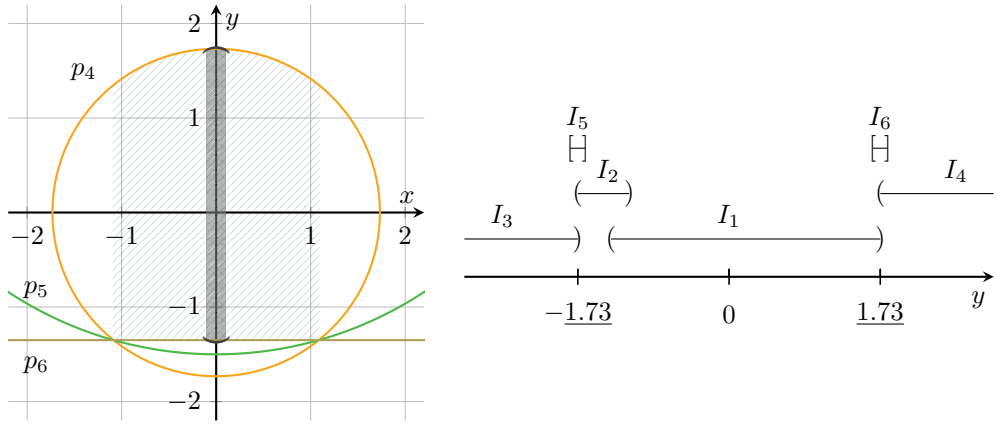
$s = (s_1 : x \mapsto 0, s_2 : y \mapsto 0)$
This time, `get_unsat_intervals` examines univariate constraints after substitution of s and all three constraints yield UNSAT intervals.

Constraint	Evaluated	Unsatisfied Intervals
c_1	$z > 0$	$(-\infty; 0), [0; 0]$
c_2	$z^2 < 3$	$(-\infty; \underline{-1.73}), [\underline{-1.73}; \underline{-1.73}], [\underline{1.73}; \underline{1.73}], (\underline{1.73}; \infty)$
c_3	$z^2 > 9.75$	$[\underline{-3.12}; \underline{-3.12}], (\underline{-3.12}; \underline{3.12}), [\underline{3.12}; \underline{3.12}]$.

The corresponding interval structures contain more information on the conflicts, e.g. instead of storing $(-\infty; 0)$ from c_1, we store $(\ell = -\infty, u = 0, L = \emptyset, U = \{p_1\}, P_3 = \{p_1\}, P_\perp = \emptyset)$. As these structures are simple to infer, we omit them on the z-level.

The UNSAT intervals obtained from c_2 and c_3, here denoted by $\mathbb{I}$, cover the real line and, thus, `get_unsat_cover` returns $(\text{UNSAT}, \mathbb{I})$.

$s = (s_1 : x \mapsto 0)$
Back in the previous routine `get_unsat_cover`, the conflict is handed over to



(a) The interval structure I_1

(b) Full covering of the y-axis

Figure 2.4.4: (a) Two-dimensional visualisation of the zeros of the projection polynomials obtained by the first characterisation. The obtained interval above $(s_1 : x \mapsto 0)$ is illustrated over $x = 0$. The crosshatched region belongs to the cell induced by I_1. (b) A full covering of the y-axis which is achieved after six sampling steps.

`construct_characterisation` yielding three non-trivial polynomials

$$\begin{aligned} p_4 &:= \operatorname{disc}(p_2) = y^2 + x^2 - 3, \\ p_5 &:= \operatorname{disc}(p_3) = y^2 - 5y + x^2 - 9.75, \\ p_6 &:= \operatorname{res}(p_2, p_3) = y + 1.35, \end{aligned}$$

simplified according to Section 2.3. Finally, `interval_from_characterisation` generates an interval structure. At first, the zeros of p_4, p_5, and p_6 are calculated being

$$Z = \{-\infty, \underbrace{-\underline{1.73}}_{p_4}, \underbrace{-1.5}_{p_5}, \underbrace{-1.35}_{p_6}, \underbrace{\underline{1.73}}_{p_4}, \underbrace{6.5}_{p_5}, \infty\}.$$

The closest zeros around $(s_2 : y \mapsto 0)$ are -1.35 and $\underline{1.73}$, resulting in $I_1 := (-1.35; \underline{1.73})$ together with the sets $L = \{p_6\}$, $U = \{p_4\}$, $P_2 = \{p_4, p_5, p_6\}$, and $P_\perp = \emptyset$. A visualisation of the current situation is given in Figure 2.4.4a. The crosshatched area belongs to the cell induced by the interval structure I_1. It is bounded by the zeros of p_4 and p_6. The expansion of the cell in x-dimension is limited by the common zeros of p_4 and p_6. If we focus on the space $I_1 \times \mathbb{R}$, the zeros of p_2 and p_3 in Figure 2.4.3 do not change in their number and order. This invariant remains unchanged at any s_1 and s_2 chosen inside the coloured cell.

Unfortunately, I_1 does not cover the whole y-axis yet, so we have to continue with a sample $s = (s_1 : x \mapsto 0, s_2 : y \mapsto a)$ for some $a \in \mathbb{R} \setminus I_1$. When using the sequence $a_1 := 0, a_2 := -1.5, a_3 := -2, a_4 := 2, a_5 := -\underline{1.73}, a_6 := \underline{1.73}$ to choose y based on $(s_1 : x \mapsto 0)$, one obtains the intervals depicted in Figure 2.4.4b. Now the y-axis is covered and characterisation steps take place again.

Based on $I_1, \ldots, I_6$ some polynomials are calculated, i.e. the characterisation is performed based on intervals that do not stem from constraints directly. We do not discuss all of them but the subset of resultants characterising a cell. When deducing

I_1, we used resultants characterising a covering. Now we also compute resultants characterising a cell, i.e. polynomials from a single interval structure are used. Doing so, we obtain

$$p_7 := \text{res}(p_6, p_5) = p_8 := \text{res}(p_6, p_4) = p_9 := \text{res}(p_4, p_5) := x^2 - 1.1775.$$

As these are equal, they would be stored once, but we keep them all separately for reference purposes.

Other characterisation polynomials as discriminants and resultants characterising a covering are computed as well, but $x = -\underline{1.09}$ and $x = \underline{1.09}$ from $p_7 = p_8 = p_9$ are closest to s_1. Hence, we deduce the interval $(-\underline{1.09}; \underline{1.09})$.

We do not continue concluding other intervals. However, if one would do so, the whole x-axis can be covered with UNSAT intervals. The initial call to `get_unsat_cover` is not able to select a sample anymore and overall UNSAT is constituted.

Chapter 3

Closed Intervals in the CAlC Method

Although the CAlC method as presented in Section 2.4 outperforms the naive CAD method in many cases [ÁDEK21], the algorithm does not consider the strictness of its input constraints. In `get_unsat_intervals`, strict constraints allow for additional sections, i.e. the zeros of the defining polynomial, but that information is used only on the constraint level. In any subsequent deduction step, the strictness of the ancestor constraints is not taken into account.

In Section 3.2, we shortly introduce an existing approach of how to exploit strict constraints in the CAD. Afterwards, Sections 3.4 and 3.5 propose two different ways of how the strictness of input constraints might be used within the CAlC method.

3.1 Motivation

Using Cylindrical Algebraic Coverings in a conflict-driven framework as the CAlC method, is an efficient advancement of CAD techniques compared to the standard Cylindrical Algebraic Decomposition.

In theory, assigning a value to the current variable, i.e. choosing a sample from a set $R \subseteq \mathbb{R}$, is an arbitrary choice. Despite the incompleteness of the McCallum projection operator, every $r \in R$ is sufficient to end up with the same satisfiability result, but in practise, this choice is limited to the proper subset $\mathbb{R}_{alg} \subsetneq \mathbb{R}$ of real numbers having an algebraic representation. It holds $r \in \mathbb{R}_{alg}$ if there exists a univariate polynomial p with real roots $\xi_1, \ldots, \xi_n$ and $\xi_i = r$ for some $1 \leq i \leq n$. This subset of real numbers is sufficient to determine the satisfiability of polynomial constraints.

In practise, sampling a rational number $r \in \mathbb{Q} \subsetneq \mathbb{R}_{alg}$ can usually be done much more efficient than sampling a non-rational $r \in \mathbb{R}_{alg} \setminus \mathbb{Q}$. Also, the isolation of real roots is time-consuming if the current partial sample is not rational. That is because the latter one can only be represented as the root of some polynomial, e.g. r might be the ith real root of a univariate polynomial p.

Therefore, one aims at reducing the number of samples based on a non-rational number as far as possible. For example, a heuristic choice could be to sample only rationals until $\mathbb{R}$ is covered up to finitely many non-rational numbers. These are the endpoints of open intervals which do not yield a full covering yet. However, note

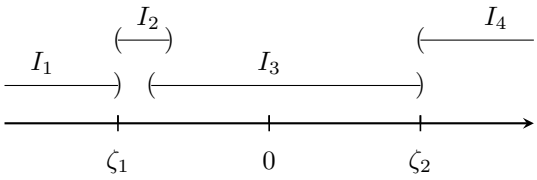


Figure 3.1.1: The intervals I_1 to I_4 nearly cover the entire real line. The only uncovered numbers are ζ_1 and ζ_2. These could but need not to be non-rational numbers. One would like I_1 to have a closed upper bound or I_2 to have a closed lower bound. Respectively, this holds for I_3 and I_4. Note that the upper bound of I_2 and the lower one of I_3 are open, but no sampling is needed as the endpoints are covered vice versa.

that not all of the open bounds have to be non-rationals. Nevertheless, one has to sample these points although algebraic numbers involve higher computational effort. A visualisation of a partial covering with two real numbers left is given in Figure 3.1.1.

As open interval bounds, which are not covered by some other interval, are the only situation in which non-rational samples are indispensable, it would be helpful to check these for satisfiability, more precisely for their extensibility towards a satisfying sample, without needing to perform all computations. When proving SAT by sampling an interval bound, this is not that costly because the computationally expensive characterisation steps are not performed anyway. However, for a partial UNSAT result, there are situations in which one can conclude an interval with closed bounds instead of open ones. In these cases, the non-rational endpoints do not need to be taken into account at all.

Intuitively, this is possible if an interval stems from strict constraints. Due to the strictness of the constraints' comparison operators, a zero of such a defining polynomial is not a satisfying assignment. While this intuition works for top-level intervals, a more elaborated view is necessary for lower levels.

In the following, we investigate two approaches, one mathematically proven way to infer closed interval bounds and one conjecture that makes use of a different strategy. The first one concludes closed bounds based on a property of those intervals that build up the current covering and the second one is based on some property of polynomials that are associated with the intervals of a covering.

3.2 Related Work

The idea of exploiting strict input constraints is not limited to the CAlC method. For example, in the CAD method one can preselect the generated cells and distinguish between those that need to be sampled and those that can be left out [Bro15].

As we did not introduce the complete CAD method, we give a brief intuition of how to make use of strict constraints: Normally, the CAD method generates sign-invariant full-dimensional cells which are sampled afterwards. All input constraints are evaluated at these samples and SAT is returned, if one is satisfying all constraints. However, all sections can be left out if the input constraints are all built up on strict comparison operators. A sample on a zero of a defining polynomial unsatisfies the corresponding strict constraint and, thus, the conjunction of input constraints.

A two-dimensional example is given in Figure 3.2.1. The coloured regions illustrate cells concluded by the CAD method. All cells are sampled, but not all are needed.

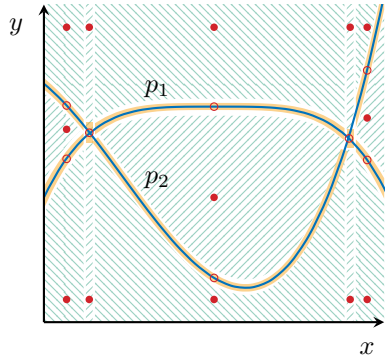


Figure 3.2.1: Illustration of the polynomial zeros of two constraints $c_1 : p_1 \sim_1 0$ and $c_2 : p_2 \sim_2 0$. The comparison operators $\sim_1$ and $\sim_2$ are strict. Hence, no satisfying solution can be found on the zeros of p_1 or p_2. The crosshatched regions and orange areas indicate the cells computed by the CAD method, sampled by the red points. There holds $p_1 = 0$ or $p_2 = 0$ within cells coloured in orange, hence no evaluation of the constraints at the corresponding (circled) samples is needed.

3.3 Deducing Closed Interval Bounds

When exploiting the strictness of input constraints, there are at least two fundamentally different approaches how to deducing and pushing down closed interval bounds.

While calculating a covering, two different data types occur, namely polynomials and intervals. Polynomials are used to maintain the properties of a covering when projecting down onto the next inspected level. Intervals are (local) results of the CAlC method regarding a specific dimension. These are the germs of cells, i.e. sign-invariant regions.

We present ideas on how to infer closed bounds both by means of intervals and polynomials. However, before diving into the approaches, we extend the definition of interval structures and cells used in the CAlC method. This facilitates the readability in both approaches.

So far, we represented intervals $(\ell; u)$ or $[b; b]$ by a structure I consisting of bounds $\ell < u$ or $b = \ell = u$, defining sets of polynomials L and U, a set P_i containing the projection polynomials with main variable x_i, and $P_\perp$ storing the remaining polynomials that appeared within the projection, i.e. polynomials with main variables smaller than x_i. There was no need to maintain whether an endpoint is closed or open as it could be reasoned from the context. To allow for other types of intervals, however, we have to extend the interval representation used in the CAlC method. To distinguish between these structures, we name the extension of interval structures differently.

Definition 3.3.1 (General Interval Structure)**.** Let $\ell \in \mathbb{R} \cup \{-\infty\}$, $u \in \mathbb{R} \cup \{\infty\}$ be endpoints, $b_\ell, b_u \in \mathbb{B}$ Boolean flags, and $L, U, P_i \subseteq \mathbb{Q}[x_1, \ldots, x_i] \setminus \mathbb{Q}[x_1, \ldots, x_{i-1}]$ as well as $P_\perp \subseteq \mathbb{Q}[x_1, \ldots, x_{i-1}]$ sets of polynomials.
The *general interval structure* I with

$$I = (\ell, u, b_\ell, b_u, L, U, P_i, P_\perp) \text{ or } I = (I_\ell, I_u, I_{b_\ell}, I_{b_u}, I_L, I_U, I_{P_i}, I_{P_\perp})$$

defines an interval based on the endpoints ℓ, u whereby

- the endpoints ℓ and u together with the sets L, U, P_i, and $P_\perp$ are defined analogously as for the interval structure $(\ell, u, L, U, P_i, P_\perp)$, and

- the Boolean flags b_ℓ and b_u indicate whether the bounds of the represented interval are closed. The endpoint $e \in \{\ell, u\}$ is closed if and only if b_e is `True`.

Note that every interval structure I can be extended to a general interval structure by assigning the bound flags b_ℓ and b_u to `True` if and only if $\ell = u$.

Based on a general interval structure, a cell is induced similar as for interval structures. This time, however, statements about the satisfiability of the cell's bounds can be made. To do so, we first extend the notion of coverings onto sets.

Definition 3.3.2 (Specialisation of an Interval)**.** Let $s = (s_1, \ldots, s_i)$ be a partial sample, $s' = (s_1, \ldots, s_{i-1}, s_i')$ similar to s apart from the last component, and $I = (\ell, u, b_\ell, b_u, L, U, P_i, P_\perp)$ a general interval structure based on s. The *specialisation* I' of I over s' results from I by evaluating the zeros of L and U at s' instead of s, i.e. all components of I and I' are the same except ℓ and u.

Note that the existence of the zeros belonging to L and U has to be taken into account before calculating I', which we have to ensure in the later adaptions.

An illustration of an interval specialisation is given in Figure 3.4.1a. The same interval structure is evaluated once over s_i and once over s_i'.

Definition 3.3.3 (Covering over a Set)**.** Let $n \in \mathbb{N}$, $S \subseteq \mathbb{R}^n$, $s \in S$ be a partial sample, and $\mathbb{I} := \{I_1, \ldots, I_k\}$ a set of general interval structures, such that $I_1, \ldots, I_k$ form a covering over s. If for all $s' \in S$ the specialisations of $I_1, \ldots, I_k$ provide a covering over s', then $\mathbb{I}$ forms a *covering over* S.

Finally, we introduce two functions used to precisely state when a constraint is strict and when an interval bound is finite.

Definition 3.3.4. Let $\mathbb{R}^\infty$ denote $\mathbb{R} \cup \{-\infty, \infty\}$ and let $a \in \mathbb{R}^\infty$. The function $\text{finite} : \mathbb{R}^\infty \to \mathbb{B}$ determines whether a is a real number, meaning

$$\text{finite}(a) := \texttt{True} \Leftrightarrow a \in \mathbb{R}.$$

Definition 3.3.5. Let $c : p \sim 0$ be a constraint based on a polynomial p compared to zero by an operator $\sim \in \{<, \leq, =, \neq, \geq, >\}$. The strictness of $\sim$ is determined by the function $\text{strict} : \{<, \leq, =, \neq, \geq, >\} \to \mathbb{B}$ with

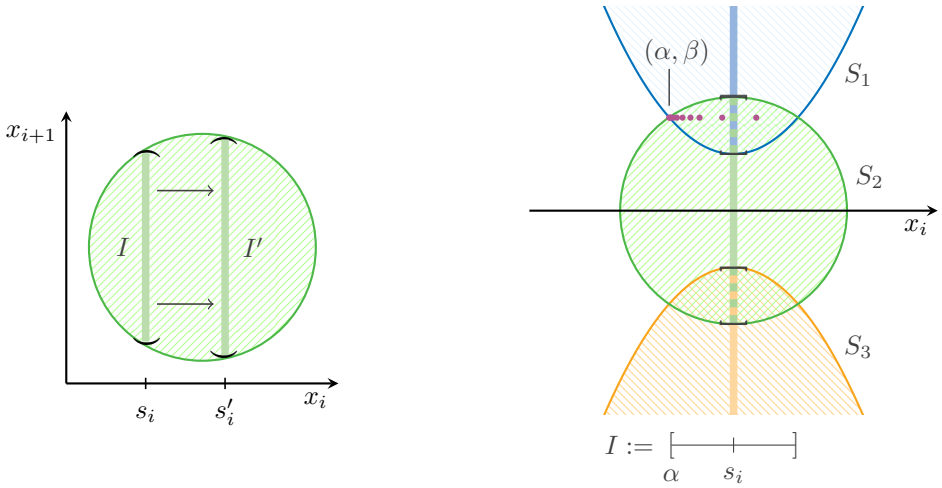
$$\text{strict}(\sim) = \texttt{True} \Leftrightarrow \sim \in \{<, \neq, >\}.$$

3.4 Closed Bounds Based on Intervals

In this section, we deal with the first approach of how to deduce closed interval bounds. It makes use of the intervals forming a covering over some partial sample $s = (s_1, \ldots, s_i)$. If all of these intervals rely on strict ancestor constraints only, the inferred interval around s_i will be closed.

In the following, we start by giving an intuitive argument on how the approach works, then dive into formalism and the theoretical background before updating the algorithmic procedures from Section 2.4.

The most comprehensible way is, to inspect a covering obtained directly from constraints, i.e. input constraints whose defining polynomials become univariate after



(a) Specialisation of an interval

(b) Proof illustration

Figure 3.4.1: (a) A cell bounded by the green zero. The interval I based on s_i is specialised to I' over some s_i'. (b) Three polynomials defining the interval bounds of a covering over s_i. The generalisation around s_i yields the interval I, the view on S based on the current partial sample. The filled regions belong to the cells $S_1, \ldots, S_k$ corresponding to the intervals $I_1, \ldots, I_k$ forming a covering over s_i. The intervals are highlighted above s_i. The sequence (ν, β) is illustrated by the purple points converging to (α, β). The convergence point is laying in closure(S_1) and closure(S_2). In this example it suffices to choose $q = 1$, i.e. the whole sequence is laying within S_1 as well as S_2.

substitution of the current partial sample $s = (s_1, \ldots, s_i)$. When generalising around s_i, an UNSAT interval I for dimension i is inferred. For all $s_i' \in I$ there holds that the specialisations of some intervals $I_1, \ldots, I_k$ which form a covering over s_i, still cover the real line over s_i'. Thus, the degree of extension around s_i is limited by the polynomial zeros defining the intervals $I_1, \ldots, I_k$: If they disperse too far away from each other, the covering of the real line breaks up. That way, the polynomial zeros defining the interval bounds of $I_1, \ldots, I_k$ are responsible for how far an extension is possible. Note that the zeros obtained from P_i and $P_\perp$ of the covering intervals have a co-responsibility when defining I, but we omit these for now as they are trivial when inferring a covering only based on constraints.

In the previous section we noted that a (general) interval structure induces a cell. Moreover, given an interval I_j, the cells of all specialisations of I_j are equal because the defining polynomials do not change.

In the original CAlC method, it holds that for a sector I the corresponding cell is UNSAT for the conjunction of its ancestor constraints. Remember that such a cell is open-bounded, i.e. it is an open set. In case I is a section, the related cell consists only of the zeros of its defining polynomials. Again, the cell is UNSAT for the conjunction of parent constraints.

Now to the case where all intervals $I_1, \ldots, I_k$ stem from strict constraints. In that particular case, the cells induced by $I_1, \ldots, I_k$ can be extended to contain their bounds, i.e. the interval perspectives on these cells have (at least) one closed endpoint. If I is a sector and in the next iteration one chooses s_i to be a finite endpoint α of I, then UNSAT over α can be constituted without needing to infer a new covering first. That is because $I_1, \ldots, I_k$ already form a valid covering over α: Since I has been

extend up to α, the cells over I mainly cover the real line over α. The only problem is that the cell bounds intersect or vanish over α. But as the bounds are part of the corresponding cells, the intersections of polynomial zeros above α are still covered and the properties of an UNSAT covering are maintained. That way, the related endpoint of I can be closed.

The above intuition relies on the fact that all intervals of the covering stem directly from input constraints. However, this does not apply to the majority of computed coverings. Having deduced UNSAT intervals from constraints, we deduce some interval I for the next lower level. After repetition of that procedure, a covering of the second-highest dimension might be concluded and an interval for the third-highest level is inferred. This time, its relationship to the strictness of input constraints is not obvious anymore. Thus, we formalise the approach to make inductive statements.

Similar to the previous intuition for a covering on the constraint level, the inference of closed interval bounds can be reduced to statements about the underlying cells. If we extend an open cell onto its bounds, we obtain the closure of the original cell. Note that this one is a cell as well.

Definition 3.4.1 (Closure of a Cell)**.** Let $m \in \mathbb{N}$ and S be a cell in $\mathbb{R}^m$. The set closure(S) is the smallest closed set that contains S, i.e.

$$\text{closure}(S) := \{s \mid s \in \mathbb{R}^m \text{ and } B_\varepsilon(s) \cap S \neq \emptyset \text{ for all } \varepsilon > 0\},$$

whereby $B_\varepsilon(s)$ is an open ball around s of radius ε. S is called *closed* if $S = \text{closure}(S)$.

Instead of arguing based on intervals, we use cells in the following theorem.

Theorem 3.4.1 (As in [Bro22])**.** Let $m > 0$, $k \geq 0$, $S \subseteq R^{m-1}$ the cell to be constructed, and let $\mathcal{F}$ be a formula. Furthermore, let $I_1, ..., I_k$ be general interval structures with the corresponding cells $S_1, \ldots, S_k$ forming a covering over S such that for each $1 \leq j \leq k$, closure(S_j) is UNSAT for $\mathcal{F}$. Then closure(S) is UNSAT for $\mathcal{F}$, too.

Proof (As in [Bro22]). Let $S \subseteq \mathbb{R}^{m-1}$ be a set over which $I_1, \ldots, I_k$ form a covering and $\alpha \in \text{closure}(S) \setminus S$ be a sample on the bounds of S. Next, let n be the index of the highest variable according to the variable ordering and $\beta, \gamma_1, \ldots, \gamma_{n-m} \in \mathbb{R}$ be arbitrary real numbers for extending α towards a full-dimensional sample. We have to show that $\mathcal{F}$ is unsatisfied at the point $P := (\alpha_1, \ldots, \alpha_{m-1}, \beta, \gamma_1, \ldots, \gamma_{n-m})$.

The unsatisfaction of $\mathcal{F}$ at P can be transferred to the cells induced by the general intervals $I_1, \ldots, I_k$. Therefore, let $(\nu_j)_{j \in \mathbb{N}}$ be a sequence of partial samples in S converging towards α. Then the extensions $(\nu_{j_1}, \ldots, \nu_{j_{m-1}}, \beta, \gamma_1, \ldots, \gamma_{n-m})$ for $j \in \mathbb{N}$ converge to P.

If we extend ν_j by β only, we obtain a point $P'_j \in \mathbb{R}^m$ laying inside one of the cells $S_1, \ldots, S_k$. Due to the convergence of $(\nu_j)_{j \in \mathbb{N}}$, there exists a $q \in \mathbb{N}$ and a $1 \leq r \leq k$, such that for the subsequence $(\nu_j)_{j \geq q}$ the points P'_j lie in the cell S_r of *one* interval I_r. In particular, the convergence point (α, β) is in closure(S_r) and by assumption closure(S_r) is UNSAT for $\mathcal{F}$. Hence, every $P \in \text{closure}(S)$ is violating $\mathcal{F}$. □

A visualisation of the proof idea is given in Figure 3.4.1b. Based on a partial sample s_i, a covering of three intervals has been obtained. The cells bounded by some polynomials are coloured corresponding to their defining zeros. A base interval I is deduced after characterisation steps. It is a local representation of the cell S, i.e. fixed to specific values for $x_1, \ldots, x_{i-1}$, namely the ones belonging to the sample's

components $s_1, \ldots, s_{i-1}$. Therefore, the chosen sample point α is equal to one of I's bounds. The convergence point (α, β) of the purple sequence $(\nu_j, \beta)_{j \geq q}$ is laying within the closure of the green cell. That way, reasoning about the assignment $s_i := \alpha$ can be postponed to the cells constructed before.

The depicted situation aims at the advantage of the closure property: Since the defining zeros of the green and the blue cell belong to their closures and these are unsatisfying some formula $\mathcal{F}$, a statement about the unsatisfiability based on (α, β) can be made.

In general, it holds that if a cell can be closed, then it is unsatisfying for the conjunction of all input constraints, i.e. $\mathcal{F}$ could be chosen trivially. However, the converse does not hold: It might be mandatory that a cell, which unsatisfies the conjunction of input constraints, is not closed.

Example 3.4.1. We consider the satisfiable constraint set $\{x + y = 0, -x + y = 0\}$. When choosing $s = (s_1 : x \mapsto 1)$, from the two constraints we obtain an UNSAT covering for the y-dimension that projects down to the interval $(0; \infty)$. Its corresponding cell is bounded by the zero of $\text{res}_y(y + x, y - x) = x$. If the cell, respectively the interval, is not closed, the sample $s = (s_1 : x \mapsto 0)$ on its left bound can be extended towards $s' = (s_1 : x \mapsto 0, s_2 : y \mapsto 0)$. The latter one witnesses the satisfiability of the constraint set. However, if the previous cell would be closed, the result of the CAlC method would be UNSAT. That is because the *only* satisfying assignment s' would not be taken into account.

We take this for a reason to choose $\mathcal{F}$ such that it supports reasoning about closed cell bounds. To that end, we make use of a set associated with every appearing interval in the CAlC method. It is used to build up $\mathcal{F}$ inductively.

Definition 3.4.2 (Ancestor Constraint Set)**.** Let I be an interval deduced by the CAlC method. As every interval is deduced from a covering which itself consists of intervals or from a constraint, a tree with I as the root is obtained.

The *ancestor constraint set* $\mathcal{A}_I$ contains the input constraints used to deduce I, i.e. those constraints responsible for the leaf nodes in the mentioned tree.

When deducing an interval I in the CAlC method, we can use $\mathcal{A}_I$ to define $\mathcal{F}$.

Corollary 3.4.1. Let $k \geq 0$ and I be an interval deduced from a covering $I_1, \ldots, I_k$ $(k > 0)$ or from a constraint $(k = 0)$ within the CAlC method. If all constraints in the ancestor constraint sets $\mathcal{A}_{I_i}$ are strict for all $1 \leq i \leq k$, then the expansion of I towards its bounds is a valid UNSAT interval.

Proof by Structural Induction. We prove that there exists a formula $\mathcal{F}$ for I based on $\mathcal{A}_I$ such that the endpoints of I unsatisfy $\mathcal{F}$.

IB In that case, I is inferred from a strict constraint c in `get_unsat_intervals`, i.e. $\mathcal{A}_I = \{c\}$. Hence, no previous intervals need to be considered and so we choose $\mathcal{F} := c$. By the strictness of c, the closure of the corresponding cell S of I is UNSAT for $\mathcal{F}$. Thus, I can be expanded towards its endpoints as it is a local view on S.

IS Let $I_1, \ldots, I_k$ be the intervals used to non-redundantly cover the current dimension and $\mathcal{A}_{I_1}, \ldots, \mathcal{A}_{I_k}$ their corresponding ancestor constraint sets. Furthermore, let $S_1, \ldots, S_k$ be the induced cells of $I_1, \ldots, I_k$.

By the induction hypothesis, there exist formulae $\mathcal{F}_1, \ldots, \mathcal{F}_k$ based on input constraints $\mathcal{A}_{I_1}, \ldots, \mathcal{A}_{I_k}$ such that closure(S_j) is UNSAT for $\mathcal{F}_j$ and all $1 \leq j \leq k$. The conjunction

$$\mathcal{F} := \mathcal{F}_1 \wedge \cdots \wedge \mathcal{F}_k$$

is unsatisfied by the closures as well, as each one unsatisfies a subformula of $\mathcal{F}$. By use of Theorem 3.4.1, the closure of the deduced cell S is UNSAT for $\mathcal{F}$, too. Thus, the endpoints of the interval I, which is a local view on S, are allowed to be closed. □

3.4.1 Algorithmic Adaptions

The above ideas are used to update the CAlC method. Regarding the proof of Corollary 3.4.1, the *existence* of formulae $\mathcal{F}_1, \ldots, \mathcal{F}_k$ is sufficient to argue that the deduced interval can be closed. That is why the algorithmic procedures do not have to maintain a formula for each interval but a flag assuring that all constraints in the ancestor sets are strict.

To that end, we once again modify the definition of general interval structures.

Definition 3.4.3 (General Interval Structure (Updated))**.** A *general interval structure* I with $I = (\ell, u, b_\ell, b_u, L, U, P_i, P_\perp)$ is supplemented by a Boolean flag $\Delta \in \mathbb{B}$ stating whether all ancestor constraints of I are strict, i.e.

$$\Delta = \texttt{True} \Leftrightarrow \text{all constraints in } \mathcal{A}_I \text{ are strict.}$$

Note that the flags b_ℓ and b_u are no longer mandatory to determine closed bounds: Considering Δ, whether the endpoints are finite, and whether they are equal would suffice as well.

As the proof of Corollary 3.4.1 suggests, the inheritance of the flag Δ happens inductively. Thus, we first adapt `get_unsat_intervals` (Algorithm 2), afterwards `interval_from_characterisation` (Algorithm 4) is adapted.

Currently, `get_unsat_intervals` yields sections and sectors, general interval structures, however, allow for fewer but more extended intervals. That is because sections can be merged with consecutive sectors stemming from the same constraint, if both characterise unsatisfiable intervals based on the current partial sample.

The only sections not being able to be merged with neighbour intervals are those resulting from strict constraints which have no unsatisfying neighbours. Typically, these are constraints c of the form $c : p \neq 0$. As the zero of p will not lead towards a satisfying sample, but any value in an ε-region around the zero of p might do so, no consecutive sectors exist. Other comparison operators can cause a similar behaviour, e.g. the inequation $-x^2 < 0$ is satisfied for all choices of x except $x = 0$.

Regarding the preliminary considerations, we obtain Algorithm 5. Differences to Algorithm 2 are coloured in blue.

The method `compute_regions` creates general interval structures with $L = U = P_i = P_\perp = \emptyset$ for intervals induced by a set of zeros. It closes up finite endpoints of sectors if the corresponding constraint is strict. That way, sections are merged with

Algorithm 5: get_unsat_intervals(s) (Updated)

Data: Global set of constraints C.
Input: (Partial) sample point $s = (s_1, \ldots, s_{i-1}) \in \mathbb{R}^{i-1}$.
Output: A set $\mathbb{I}$ of general interval structures such that $\{s\} \times I$ unsatisfies at least one constraint for all $I \in \mathbb{I}$.

$\mathbb{I} := \emptyset$
$C_i :=$ the subset of C with main variable x_i
foreach $c = p \sim 0 \in C_i$ **do**
  $c' := [\![c]\!]^s$ `// Yields truth value or univariate constraint`
  if $c' =$ `False` **then**
    $I := (I_\ell = -\infty, I_u = \infty, I_{b_\ell} =$ `False`$, I_{b_u} =$ `False`$, I_\Delta = \text{strict}(\sim),$
      $I_L = \emptyset, I_U = \emptyset, I_{P_i} = \{p\}, I_{P_\perp} = \emptyset)$
    return $\mathbb{I} := \{I\}$
  if $c' =$ `True` **then**
    continue
  $Z :=$ real_roots(p, s) `// Ordered list` $z_1 < \cdots < z_k$
  Regions := compute_regions$(Z, \sim)$ `//` $(-\infty; \infty)$ `if` $Z = \emptyset$
  foreach $J \in$ Regions `//` J `has the form` $(\ell; u), [\ell; u), (\ell; u]$ `or` $[\ell; u]$
  do
    Pick $r \in J$
    if $[\![c']\!]^r =$ `False` **then**
      $L := \emptyset \quad U := \emptyset$
      if $\ell \neq -\infty$ **then**
        $L := \{p\}$
      if $u \neq \infty$ **then**
        $U := \{p\}$
      $I := (I_\ell = J_\ell, I_u = J_u, I_{b_\ell} = J_{b_\ell}, I_{b_u} = J_{b_u}, I_\Delta = \text{strict}(\sim),$
        $I_L = L, I_U = U, I_{P_i} = \{p\}, I_{P_\perp} = \emptyset)$
      $\mathbb{I} := \mathbb{I} \cup \{I\}$ `// After simplifications`
return $\mathbb{I}$

Algorithm 6: interval_from_characterisation$(s, s_i, P, \mathbb{I})$ (Updated)

Input: (Partial) sample point $s = (s_1, \ldots, s_{i-1}) \in \mathbb{R}^{i-1}$, an extension s_i, a polynomial UNSAT characterisation P, and the set of general interval structures $\mathbb{I}$ forming the covering over s_i.
Output: A general interval structure I around s_i so that the constraints are unsatisfiable on $\{s\} \times I$ for the same reason.

$P_\perp := \{p \in P \mid p \in \mathbb{R}[x_1, \ldots, x_{i-1}]\}$
$P_i := P \setminus P_\perp$
$Z := \{-\infty\} \cup$ real_roots_with_check$(P_i, s) \cup \{\infty\}$
$\ell := \max\{z \in Z \mid z \leq s_i\}$
$u := \min\{z \in Z \mid z \geq s_i\}$
$L := \{p \in P_i \mid p(s \times \ell) = 0\}$
$U := \{p \in P_i \mid p(s \times u) = 0\}$
$b_\ell := \left(\text{finite}(\ell) \wedge \bigwedge_{I' \in \mathbb{I}} I'_\Delta\right) \vee (\ell == u)$
$b_u := \left(\text{finite}(u) \wedge \bigwedge_{I' \in \mathbb{I}} I'_\Delta\right) \vee (\ell == u)$
$\Delta := \bigwedge_{I' \in \mathbb{I}} I'_\Delta$
$I := (I_\ell = \ell, I_u = u, I_{b_\ell} = b_\ell, I_{b_u} = b_u, I_\Delta = \Delta, I_L = L, I_U = U, I_{P_i} = P_i, I_{P_\perp} = P_\perp)$
return I

neighboured sectors whenever possible. We do not give an implementation of this method as it is straightforward.

Calculating projection polynomials in `construct_characterisation` is independent of the interval bounds and, thus, no modification is necessary.

Next, the inductive step is performed by `interval_from_characterisation`. Based on a covering $\mathbb{I}$ of the real line, the characterisation polynomials are calculated and an interval I is deduced. The covering $\mathbb{I}$ over s_i is inspected and the conjunction of flags I'_Δ of the intervals $I' \in \mathbb{I}$ determines whether all constraints in the ancestor set of I are strict.

Finally, `get_unsat_cover` is adjusted to fit the parameter list of Algorithm 6, i.e. the set of intervals $\mathbb{I}$ is handed over to `interval_from_characterisation` in Line 11. As the rest remains unchanged, we do not present the updated procedure.

3.4.2 Example

We head back to the running example introduced in Section 2.4.3. The constraint set

$$\mathcal{C} := \{ \quad \overbrace{\underbrace{z - y^2 - x^2}_{p_1} > 0}^{c_1}, \quad \overbrace{\underbrace{z^2 + y^2 + x^2 - 3}_{p_2} < 0}^{c_2}, \quad \overbrace{\underbrace{z^2 + (y - 2.5)^2 + x^2 - 16}_{p_3} > 0}^{c_3} \quad \}$$

remains unchanged.

As before we discuss the first deduced interval for the y-dimension in detail. When choosing the sample $s = (s_1 : x \mapsto 0, s_2 : y \mapsto 0)$, all three constraints can be partially evaluated. From that, we infer UNSAT intervals for the z-dimension.

Constraint	Evaluated	Unsatisfied Intervals
c_1	$z > 0$	$(-\infty; 0]$
c_2	$z^2 < 3$	$(-\infty; -\underline{1.73}], [\underline{1.73}; \infty)$
c_3	$z^2 > 9.75$	$[-\underline{3.12}; \underline{3.12}]$.

Note that sections and sectors are merged by `compute_regions`. This time, we exemplary give the general interval structure obtained from c_3 explicitly which is $(\ell = -\underline{3.12}, u = \underline{3.12}, b_\ell = \texttt{True}, b_u = \texttt{True}, \Delta = \texttt{True}, L = \{p_3\}, U = \{p_3\}, P_3 = \{p_3\}, P_\perp = \emptyset)$. All four interval structures have the flag $\Delta = \texttt{True}$ as the input constraints are all strict.

A covering is provided by c_2 and c_3 and a characterisation of the conflict is computed. The calculated polynomials p_4, p_5, and p_6 are as before and yield the zeros

$$Z = \{-\infty, \underbrace{-\underline{1.73}}_{p_4}, \underbrace{-1.5}_{p_5}, \underbrace{-1.35}_{p_6}, \underbrace{\underline{1.73}}_{p_4}, \underbrace{6.5}_{p_5}, \infty\}.$$

The closest ones still are -1.35 and $\underline{1.73}$. We deduce the interval $I_1 := [-1.35; \underline{1.73}]$ because the tree of coverings above I_1 consists of three leaf intervals (two from c_2 and one from c_3) all with $\Delta = \texttt{True}$. The ancestor set of I_1 is $\mathcal{A}_{I_1} = \{c_2, c_3\}$. The conjunction of the three interval flags Δ is `True` and we obtain $I_1 = (\ell = -1.35, u = \underline{1.73}, b_\ell = \texttt{True}, b_u = \texttt{True}, \Delta = \texttt{True}, L = \{p_6\}, U = \{p_4\}, P_2 = \{p_4, p_5, p_6\}, P_\perp = \emptyset)$. Compared to Figure 2.4.4a, the bounds of the induced cell defined by the zeros of p_4

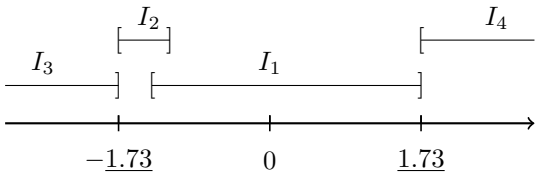


Figure 3.4.2: The intervals I_1 to I_4 cover the real line for the y-dimension. All finite interval bounds are closed as they stem from `True` flagged intervals only.

and p_6 are now part of that UNSAT cell. The possibility of reassigning s_2 is reduced by the values -1.35 and $\underline{1.73}$.

If one continues covering the y-axis, a set consisting of four instead of six general interval structures is deduced, all with $\Delta = \texttt{True}$. Since all bounds are closed, no point intervals occur as depicted in Figure 3.4.2.

Based on these four interval structures, characterisation steps are performed yielding the same polynomials as the original CAlC method does. In particular, the two intervals I_5 and I_6 from Figure 2.4.4b do not provide any missing insides.

That way, the first interval covering a part of the x-axis is $[-\underline{1.09}; \underline{1.09}]$. The conjunction of the corresponding flags $\Delta_1 \wedge \Delta_2 \wedge \Delta_3 \wedge \Delta_4$ is `True` and both endpoints are finite. Therefore, $b_\ell = \texttt{True}$ and $b_u = \texttt{True}$.

Note that all three constraints have to use strict comparison operators as otherwise one of the intervals $I_1, \ldots, I_4$ would have a `False` flag. Thus, the conjunction of $\Delta_1, \ldots, \Delta_4$ would be `False` as well.

3.5 Closed Bounds Based on Polynomials

Unlike the approach presented in Section 3.4, we introduce another possibility of deducing interval bounds, this time based on polynomials. The method above concludes the unsatisfiability of a closed interval bound by arguing on the unsatisfiability of ancestor constraints. That way, the argumentation is based on a property of the intervals forming the current covering.

The alternative approach presented in this section is based on a conjecture instead of a sophisticated mathematical argumentation. A significant difference to the previous approach is that the closure is based on polynomials rather than intervals when it comes to the inference of closed bounds. To that end, the following section focuses on the intuition level instead of a seamless proof.

Definition 3.5.1 (Flagged Polynomial Pair)**.** Let $p \in \mathbb{Q}[x_1, \ldots, x_n]$ be a polynomial and $\omega \in \mathbb{B}$ a Boolean truth value. The pair (p, ω) is called a *flagged polynomial pair*. Sometimes, we call these pairs polynomials themselves, in particular when speaking about projection steps. In that case, we refer to ω as the flag of p.

As in the method based on intervals, the strictness of input constraints is the basis for inferring closed bounds. This time, the definition of a general interval structure is reused for clarity only. By using flagged polynomial pairs, one can deduce the type of bounds based on the polynomial flags as well. Note that we use Definition 3.3.1 for general interval structures, not the updated version.

The setup, however, is not only adapted to deal with general interval structures but also to deal with flagged polynomial pairs. Thus, every polynomial p appearing in the computation of a CAlC, either from input constraints or from a characterisation, is replaced by a flagged polynomial pair (p, ω). The flag indicates whether an interval bound based on this polynomial can be closed ($\omega = \texttt{True}$) or has to remain open ($\omega = \texttt{False}$). If all polynomials defining the same interval bound are flagged `True`, the endpoint is inclusive, i.e. the bound is closed.

If p results from an input constraint $p \sim 0$, it is flagged based on the comparison operator $\sim \in \{<, \leq, =, \neq, \geq, >\}$, so $\omega := \text{strict}(\sim)$. However, if p is generated in the original algorithm, it is replaced by the pair (p, ω), whereby the flag ω is defined as a Boolean combination of the flags from p's polynomial parents.

As introduced in Section 2.3, four types of polynomials are calculated in the characterisation steps:

- Discriminants,
- Coefficients,
- Resultants for a covering, and
- Resultants for a cell.

We give a rule-based deduction system for how the flag of newly generated polynomials is set. As mentioned before, we do not propose that this system is sound. Moreover, we mainly focus on the intuition for resultants but give rules for discriminants and coefficients as well.

If (p, ω) is the discriminant of some polynomial (q, ϑ), we conjecture that the flag can be inherited directly.

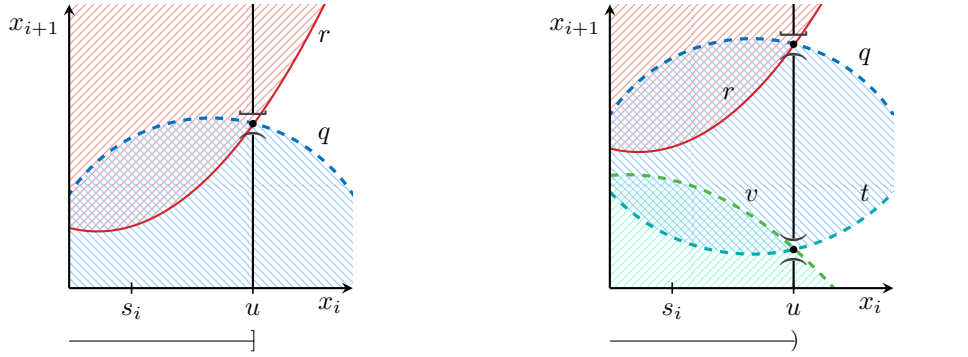
Rule 3.5.1 (Discriminant)**.** Let (q, ϑ) and (p, ω) be flagged polynomial pairs and $p = \text{disc}(q)$. The flag ω of p is straightly inherited from the parent polynomial q, i.e. there holds $\omega := \vartheta$.

Determining the flag of required coefficients of a polynomial q is more involved. Heading back to Section 2.4, we need coefficients to detect asymptotic behaviour which can restrict the size of the generalisation around a partial sample point. For now, we stick to `False` flags of these coefficients regardless of the parent polynomial's flag.

Rule 3.5.2 (Coefficient)**.** Let (q, ϑ) and (p, ω) be flagged polynomial pairs such that $p \in \texttt{required\_coefficients}(q)$ is a coefficient that evaluates to zero after substituting the current sample, i.e. it needs to be considered in the CAlC projections. The flag ω of p is set to $\omega := \texttt{False}$.

Finally, the mentioned two kinds of resultants of pairs (q, ϑ) and (r, λ) are computed. For resultants within a cell, two different variants are presented below. Independent from that, the flag of resultants characterising a covering is assigned using the same rule for both variants. Thus, we explain the latter one first.

Rule 3.5.3 (Resultant for a Covering)**.** Let (q, ϑ), (r, λ), and (p, ω) be flagged polynomial pairs and let I_j and I_{j+1} be consecutive intervals of a covering whereby (q, ϑ) is defining for the upper bound of I_j and (r, λ) for the lower bound of I_{j+1}. If $p := \text{res}(q, r)$, the flag ω is disjunctively inherited, i.e. $\omega := \vartheta \vee \lambda$.



(a) Interval bound u defined by a common zero.

(b) Interval bound u defined by multiple common zeros.

Figure 3.5.1: Visualisations of resultants characterising a covering. A solid line indicates the zero of a `True` flagged polynomial, a dashed line belongs to a `False` one. (a) A covering showing that it suffices for a common zero if it is contained in one of the intervals, i.e. one bound is inclusive. (b) Illustration for why one has to consider all common zeros over the deduced interval bound.

The intuition behind Rule 3.5.3 is, that it suffices for the common zero of q and r if it is contained in one of the cells bounded by either q or r. As it cannot be an inner point of the cells, we require at least one parent to have a closed endpoint. That way, the covering at the common zero still holds. In terms of flagged polynomial pairs that is equivalent to a `True` flag of q or r. The situation where $\vartheta = \texttt{False}$ and $\lambda = \texttt{True}$ is depicted in Figure 3.5.1a. We assume that x_i describes the dimension we want to project onto and for x_{i+1} a covering has been calculated before. Hence, all variables x_k with $k < i$ are substituted according to the current partial sample. Let s_i denote the current assignment for the x_i-dimension. If the zero of the resultant $p = \operatorname{res}(q, r)$ is defining for the upper bound of the interval in the next lower dimension, it is the closest zero to s_i. Therefore, the cells above (red area) and below (blue area) the common zero allow for a covering above and below it. The crucial zero itself is contained within the cell bounded by r, thus a full covering is achieved.

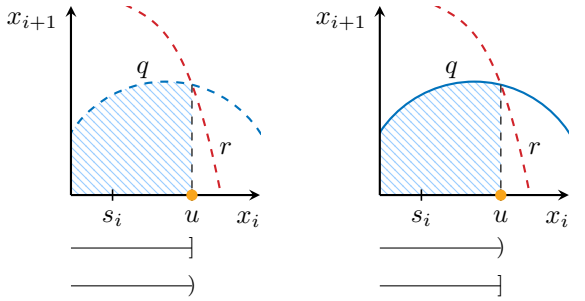
If multiple polynomials are defining for the bound u, the weakest flag is dominating, i.e. the bound of the deduced interval is closed if and only if all responsible polynomials are flagged `True`. Speaking locally, the common zero of q and r in Figure 3.5.1a *does not prevent* the interval below from being closed but does not guarantee it either. If another two polynomials t and v, both with `False` flags, have a common zero below the one of q and r, the conjunctive value of the resultants' flags is `False` as well. The bound at u remains open, because the covering above u is no longer seamless. An illustration is given in Figure 3.5.1b.

For resultants within a single cell, we present two different approaches.

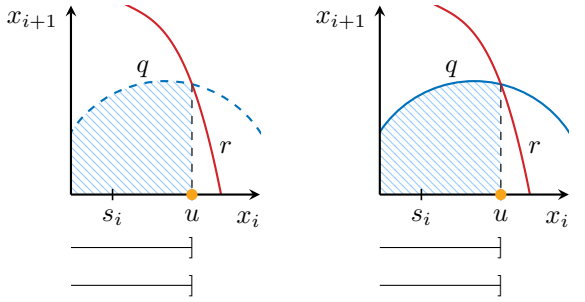
First Variant

This variant uses two different rules for the two types of resultants.

Rule 3.5.4 (Resultant for a Cell)**.** Let $I = (\ell, u, b_\ell, b_u, L, U, P_{i+1}, P_\perp)$ denote a general interval structure of a covering, let (q, ϑ), (r, λ), and (p, ω) be flagged polynomial pairs whereby $(q, \vartheta) \in U$ $[(q, \vartheta) \in L]$ and $(r, \lambda) \in P_{i+1}$ with $r(s \times m) = 0$ for some $m \geq u$ $[m \leq \ell]$. If $p = \operatorname{res}(q, r)$, then $\omega := \neg\vartheta \vee \lambda$.



(a) $\vartheta = \texttt{False}$, $\lambda = \texttt{False}$ **(b)** $\vartheta = \texttt{True}$, $\lambda = \texttt{False}$



(c) $\vartheta = \texttt{False}$, $\lambda = \texttt{True}$ **(d)** $\vartheta = \texttt{True}$, $\lambda = \texttt{True}$

Figure 3.5.2: Types of bounds appearing in the computation of resultants within a cell. The zero of q is defining the upper bound of a cell in dimension $i + 1$. The polynomial r appeared in earlier characterisation steps. Four combinations of ϑ and λ are possible. A solid zero refers to a `True` flag. The upper intervals are concluded using the first variant, the lower ones by applying the second variant.

W.l.o.g., we discuss a cell with $(q, \vartheta) \in U$ and $(r, \lambda) \in P_{i+1}$. Let $s = (s_1, \ldots, s_i)$ denote the current partial sample. The resultant (p, ω) of q and r is flagged according to Rule 3.5.4. The four combinations of ϑ and λ are illustrated in Figure 3.5.2.

The idea is, to ensure sign-invariance of a cell as mentioned in the original CAlC method: Thus, deducing a closed interval bound in Figure 3.5.2a and Figure 3.5.2c is no problem as the intersection of the zero of r and the cell bound defined by the zero of q is not part of the cell as the bound itself is not ($\vartheta = \texttt{False}$).

Other than that, in Figure 3.5.2d, the zero of q is part of the cell as $\vartheta = \texttt{True}$. Although this time the common zero occurs inside the cell, it is not disturbing as the flag of r is `True` as well. Thus, the sign change of r to zero meets weak sign-invariance ($\sigma \geq 0$ or $\sigma \leq 0$ instead of strict comparison) which is sufficient as $\lambda = \texttt{True}$.

Finally, Figure 3.5.2b constitutes the only problem: The cell bound, namely the zero of q, has a `True` flag, but this time weak sign-invariance is not enough because $\lambda = \texttt{False}$. Hence, a real sign change appears inside the cell and the deduced interval has to remain open such that it is not projected downwards.

Second Variant

This variant does not make a difference when flagging both resultants.

Rule 3.5.5 (Resultant for a Cell)**.** Let $I = (\ell, u, b_\ell, b_u, L, U, P_{i+1}, P_\perp)$ denote a general interval structure of a covering, let (q, ϑ), (r, λ), and (p, ω) be flagged polynomial pairs whereby $(q, \vartheta) \in U$ $[(q, \vartheta) \in L]$ and $(r, \lambda) \in P_{i+1}$ with $r(s \times m) = 0$ for some $m \geq u$ $[m \leq \ell]$. If $p = \text{res}(q, r)$, then $\omega := \vartheta \vee \lambda$.

W.l.o.g., we again discuss a cell with $(q, \vartheta) \in U$ and $(r, \lambda) \in P_{i+1}$. Let $s = (s_1, \ldots, s_i)$ be the partial sample and $p := \text{res}(q, r)$ the desired resultant with its flag ω according to Rule 3.5.5. Again, there are four possible combinations of the parent flags ϑ and λ illustrated in Figure 3.5.2. If the common zero of q and r is closest to s_i, the upper bound is based on it. Hence, it is defining for whether the endpoint of the deduced interval might be closed.

In the original approach, an open bound is inferred when extending the interval around s_i onto some interval $(\ell; u)$. Thus, the common zero defining u is not part of the covering above the interval. However, if the endpoint should be closed, the common zero must not compromise the cells above it. Although the sign of r within the cell bounded by q changes at the common zero, this does not need to have an impact on the unsatisfiability.

If q has a `True` flag as in Figure 3.5.2b and Figure 3.5.2d, the sign change of r at the common zero does not affect the truth of the UNSAT cell as its bound is already part of it. Thus, the intersection takes place inside the cell and the flag of p can be specified as $\omega :=$ `True`.

Now to the situation where q has the flag $\vartheta =$ `False` and r is flagged $\lambda =$ `True`, depicted in Figure 3.5.2c. As r is not defining for a cell, its semantics allow for fewer conclusions compared to the situation in Figure 3.5.1a. However, from the `True` flag one can infer that the common zero itself can be excluded from the search space. That is because the `True` flag of r ensures UNSAT for some input constraint on the zero of r. The reason is that at least one ancestor constraint causing the resultant r makes use of a strict comparison operator. Otherwise, r would be flagged $\lambda =$ `False`. That way, r is able to cause $\omega :=$ `True`.

In case both q and r have a `False` flag as in Figure 3.5.2a, meaning $\vartheta =$ `False` and $\lambda =$ `False`, none of the zeros guarantees the truth-invariance at the intersection: Neither the zero is part of the UNSAT cell, nor ensures r the unsatisfiability by its flag. Therefore, the flag ω of p is set to $\omega :=$ `False` and the derived interval must remain open.

Contrary to the first variant, the second one relays on the unsatisfiability at the common zero, i.e. truth-invariance of the conjunction of input constraints, and cuts down the interval accordingly. The first variant, however, deals with the original idea of sign-invariance. This affects that the interval bounds differ. Regardless of which variant might be used, both are unproven conjectures.

3.5.1 Algorithmic Adaptions

We reuse the definition of general interval structures as of Definition 3.3.1. Note that this choice is for convenience only. Considering the conjunction of flags belonging to polynomials in L and U that correspond to finite bounds, one can infer the same information.

As before, the inductive base of all computation steps is given by UNSAT intervals stemming from constraints. Thus, we update `get_unsat_intervals` first. Contrary to the first approach, no flag Δ is needed to indicate whether all ancestor intervals are based on strict constraints, but instead, flagged polynomial pairs are used. Modifications of the algorithm are highlighted in blue.

Algorithm 7: get_unsat_intervals(s) (Updated)

Data: Global set of constraints C.
Input: (Partial) sample point $s = (s_1, \dots, s_{i-1}) \in \mathbb{R}^{i-1}$.
Output: A set $\mathbb{I}$ of general interval structures such that $\{s\} \times I$ unsatisfies at least one constraint for all $I \in \mathbb{I}$.

$\mathbb{I} := \emptyset$
$C_i :=$ the subset of C with main variable x_i
foreach $c = p \sim 0 \in C_i$ **do**
  $c' := [\![c]\!]^s$ `// Yields truth value or univariate constraint`
  if $c' =$ `False` **then**
    $I := (I_\ell = -\infty, I_u = \infty, I_{b_\ell} =$ `False`$, I_{b_u} =$ `False`$, I_L = \emptyset, I_U = \emptyset,$
      $I_{P_i} = \{(p, \text{strict}(\sim))\}, I_{P_\perp} = \emptyset)$
    return $\mathbb{I} := \{I\}$
  if $c' =$ `True` **then**
    continue
  $Z :=$ real_roots(p, s) `// Ordered list` $z_1 < \dots < z_k$
  Regions := compute_regions$(Z, \sim)$ `//` $(-\infty; \infty)$ `if` $Z = \emptyset$
  foreach $J \in$ Regions `//` J `has the form` $(\ell; u), [\ell; u), (\ell; u]$ `or` $[\ell; u]$
  do
    Pick $r \in J$
    if $[\![c']\!]^r =$ `False` **then**
      $L := \emptyset \quad U := \emptyset$
      if $\ell \neq -\infty$ **then**
        $L := \{(p, \text{strict}(\sim))\}$
      if $u \neq \infty$ **then**
        $U := \{(p, \text{strict}(\sim))\}$
      $I := (I_\ell = J_\ell, I_u = J_u, I_{b_\ell} = J_{b_\ell}, I_{b_u} = J_{b_u}, I_L = L, I_U = U,$
        $I_{P_i} = \{(p, \text{strict}(\sim))\}, I_{P_\perp} = \emptyset)$
      $\mathbb{I} := \mathbb{I} \cup \{I\}$ `// After simplifications`
return $\mathbb{I}$

This time, we need to modify `construct_characterisation` too, because it is responsible for correctly inheriting the flags of projection polynomials. As we presented two different variants, we use an orange ♦ to highlight differences between the two variants. For the first variant, the orange marking is replaced by the formula $\neg\vartheta \vee \lambda$, while for the second one it would be $\vartheta \vee \lambda$.

As mentioned in Section 3.5, we stick to `False` flags for all coefficients and, therefore, assume that `required_coefficients` works accordingly.

The interval deduction takes place in `interval_from_characterisation`, where the bound flags b_ℓ and b_u are set with respect to the polynomials stored in L and U. The deduction is conjunctive as pointed out in Figure 3.5.1b. This time, however, there is no need for passing over the covering $\mathbb{I}$ since all necessary information is self-contained within the polynomial flags.

Algorithm 8: construct_characterisation$(s, \mathbb{I})$ (Updated)

Input: (Partial) sample point $s = (s_1, \ldots, s_i) \in \mathbb{R}^i$ and an UNSAT covering $\mathbb{I}$.
Output: A set $R \subseteq \mathbb{Q}[x_1, \ldots, x_i] \times \mathbb{B}$ of flagged polynomial pairs characterising an unsatisfiable region around s for the same reason.

$\mathbb{I} :=$ compute_cover($\mathbb{I}$)
$R := \emptyset$
foreach $I \in \mathbb{I}$ **do**
  Extract $\ell := I_\ell, u := I_u, L := I_L, U := I_U, P_{i+1} := I_{P_{i+1}}, P_\perp := I_{P_\perp}$
  $R := R \cup P_\perp$
  $R := R \cup \{(\text{disc}(p), \vartheta) \mid (p, \vartheta) \in P_{i+1}\}$
  $R := R \cup \{\text{required\_coefficients}(p) \mid (p, \vartheta) \in P_{i+1}\}$
  $R := R \cup \{(\text{res}(p, q), \blacklozenge) \mid (p, \vartheta) \in L, (q, \lambda) \in P_{i+1}, q(s \times m) = 0 \text{ for some } m \leq \ell\}$
  $R := R \cup \{(\text{res}(p, q), \blacklozenge) \mid (p, \vartheta) \in U, (q, \lambda) \in P_{i+1}, q(s \times m) = 0 \text{ for some } m \geq u\}$
foreach $j \in \{1, \ldots, |\mathbb{I}| - 1\}$ **do**
  $R := R \cup \{(\text{res}(p, q), \vartheta \vee \lambda) \mid (p, \vartheta) \in U_j, (q, \lambda) \in L_{j+1}\})$
return R `// After CAD simplifications`

Algorithm 9: interval_from_characterisation(s, s_i, P) (Updated)

Input: (Partial) sample point $s = (s_1, \ldots, s_{i-1}) \in \mathbb{R}^{i-1}$, an extension s_i, and a flagged polynomial UNSAT characterisation P.
Output: A general interval structure I around s_i so that the constraints are unsatisfiable on $\{s\} \times I$ for the same reason.

$P_\perp := \{(p, \omega) \in P \mid p \in \mathbb{Q}[x_1, \ldots, x_{i-1}]\}$
$P_i := P \setminus P_\perp$
$Z := \{-\infty\} \cup \text{real\_roots\_with\_check}(P_i, s) \cup \{\infty\}$
$\ell := \max\{z \in Z \mid z \leq s_i\}$
$u := \min\{z \in Z \mid z \geq s_i\}$
$L := \{(p, \omega) \in P_i \mid p(s \times \ell) = 0\}$
$U := \{(p, \omega) \in P_i \mid p(s \times u) = 0\}$
$b_\ell := \left(\text{finite}(\ell) \wedge \bigwedge_{(p,\omega) \in L} \omega\right) \vee (\ell \texttt{ == } u)$
$b_u := \left(\text{finite}(u) \wedge \bigwedge_{(p,\omega) \in U} \omega\right) \vee (\ell \texttt{ == } u)$
$I := (I_\ell = \ell, I_u = u, I_{b_\ell} = b_\ell, I_{b_u} = b_u, I_L = L, I_U = U, I_{P_i} = P_i, I_{P_\perp} = P_\perp)$
return I

3.5.2 Example

To illustrate both variants and their differences, we head back to the running example but modify it slightly: The constraints c_1 and c_3 are updated to compare via their corresponding weak operator '$\geq$' instead of '$>$', i.e.

$$\mathcal{C} := \{ \quad \overbrace{\underbrace{z - y^2 - x^2}_{p_1} \geq 0}^{c_1}, \quad \overbrace{\underbrace{z^2 + y^2 + x^2 - 3}_{p_2} < 0}^{c_2}, \quad \overbrace{\underbrace{z^2 + (y - 2.5)^2 + x^2 - 16}_{p_3} \geq 0}^{c_3} \quad \},$$

which still is unsatisfiable as the original constraint set.

Instead of dealing with polynomials, the corresponding flagged polynomial pairs are used, which can be deduced from constraints by inspecting the comparison operators: $(p_1, \omega_1 := \texttt{False}), (p_2, \omega_2 := \texttt{True})$, and $(p_3, \omega_3 := \texttt{False})$. To meet readability, we identify each flag ω_i by the index of the polynomial p_i it belongs to. In fact, these have to be interpreted as the pair (p_i, ω_i) because the same polynomial might appear several times with different flags. To distinguish pairs with the same polynomial, each time a non-trivial polynomial is deduced, the index is incremented causing some polynomials to appear multiple times with different names.

We recompute the first interval of a y-cover in detail and skip the others as done before. The first valuable call to `get_unsat_intervals` takes place after choosing $s = (s_1 : x \mapsto 0, s_2 : y \mapsto 0)$. All three constraints become univariate and yield UNSAT intervals.

Constraint	Evaluated	Unsatisfied Intervals
c_1	$z \geq 0$	$(-\infty; 0)$
c_2	$z^2 < 3$	$(-\infty; -\underline{1.73}], [\underline{1.73}; \infty)$
c_3	$z^2 \geq 9.75$	$(-\underline{3.12}; \underline{3.12})$.

Note that the sections resulting from c_2 are merged with neighboured sectors. We once give the general interval structure for $(-\infty; -\underline{1.73}]$ which is $(\ell = -\infty, u = -\underline{1.73}, b_\ell = \texttt{False}, b_u = \texttt{True}, L = \emptyset, U = \{(p_2, \texttt{True})\}, P_3 = \{(p_2, \texttt{True})\}, P_\perp = \emptyset)$ and forego the other ones on level z.

The UNSAT intervals obtained by c_2 and c_3 denoted as $\mathbb{I}$ cover the real line and, thus, `get_unsat_cover` returns $(\text{UNSAT}, \mathbb{I})$.

The conflict is handed over to `construct_characterisation` yielding

$$\begin{aligned} p_4 &:= \text{disc}(p_2) = y^2 + x^2 - 3 & \omega_4 &:= \omega_2 = \texttt{True}, \\ p_5 &:= \text{disc}(p_3) = y^2 - 5y + x^2 - 9.75 & \omega_5 &:= \omega_3 = \texttt{False}, \\ p_6 &:= \text{res}(p_2, p_3) = y + 1.35 & \omega_6 &:= \omega_2 \vee \omega_3 \equiv \texttt{True}. \end{aligned}$$

Thus, one derives the pairs $(p_4, \texttt{True}), (p_5, \texttt{False})$, and $(p_6, \texttt{True})$ stored in the sets of polynomials instead of p_4, p_5, p_6. Finally, `interval_from_characterisation` constructs a general interval structure. At first, the zeros of p_4, p_5, and p_6 are calculated, which again are

$$Z = \{-\infty, \underbrace{-\underline{1.73}}_{p_4}, \underbrace{-1.5}_{p_5}, \underbrace{-1.35}_{p_6}, \underbrace{\underline{1.73}}_{p_4}, \underbrace{6.5}_{p_5}, \infty\}.$$

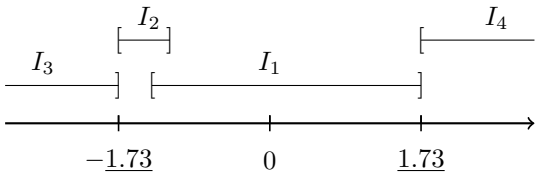


Figure 3.5.3: The intervals I_1 to I_4 cover the real line for the y-dimension. All finite interval bounds are closed as they stem from `True` flagged polynomials only.

Still, the closest zeros are -1.35 and $\underline{1.73}$. Every zero is obtained by a unique polynomial, so the conjunctions of the responsible flags are $b_\ell = \omega_6$ and $b_u = \omega_4$. Both endpoints are finite, so the first condition of Lines 8 and 9 are met.

The resulting interval $I_1 := [-1.35; \underline{1.73}]$ is stored together with $b_\ell = \texttt{True}$, $b_u = \texttt{True}, L = \{(p_6, \texttt{True})\}$, $U = \{(p_4, \texttt{True})\}$, $P_2 = \{(p_4, \texttt{True}), (p_5, \texttt{False}), (p_6, \texttt{True})\}$, and $P_\perp = \emptyset$. Compared to Figure 2.4.4a, the induced cell now contains its bounds. In the next step, one would reassign y, but due to the closed bounds of I_1 the possible assignments are reduced by -1.35 and $\underline{1.73}$.

If one continues covering the y-axis, a set consisting of four instead of six general interval structures is deduced since no point intervals occur. The bounds of all such intervals are closed, as depicted in Figure 3.5.3. Although not all constraints make use of strict comparison operators, the disjunctive connectives compensate that.

Based on these four intervals, characterisation polynomials are calculated. We do not discuss all of them but continue with the three resultants p_7, p_8, and p_9 obtained in the execution of the original CAlC method. Ignoring the flags, I_1 consists of the same polynomials as before. Hence, the characterisation polynomials remain unchanged, but although the polynomials are equal, their flags need not to be. That is why we keep all three resultants as individual polynomial pairs.

Moreover, the obtained polynomials p_7, p_8, and p_9 are resultants within a single cell, namely the one induced by I_1. That is why the flags depend on the chosen variant. We first discuss the rule with negation, i.e. Rule 3.5.4:

$$\begin{aligned} p_7 &:= \operatorname{res}(p_6, p_5) = x^2 - 1.1775 & \omega_7 &:= \neg\omega_6 \vee \omega_5 \equiv \texttt{False}, \\ p_8 &:= \operatorname{res}(p_6, p_4) = x^2 - 1.1775 & \omega_8 &:= \neg\omega_6 \vee \omega_4 \equiv \texttt{True}, \\ p_9 &:= \operatorname{res}(p_4, p_5) = x^2 - 1.1775 & \omega_9 &:= \neg\omega_4 \vee \omega_5 \equiv \texttt{False}. \end{aligned}$$

The zeros $x = -\underline{1.09}$ and $x = \underline{1.09}$ of these three resultants are closest to s_1, so for the deduced interval I, there holds $\ell = -\underline{1.09}, u = \underline{1.09}$ and $L = U = \{(p_7, \texttt{False}), (p_8, \texttt{True}), (p_9, \texttt{False})\}$. Because the conjunction $\omega_7 \wedge \omega_8 \wedge \omega_9 \equiv \texttt{False}$, the flags b_ℓ and b_u are `False` as well, so the interval endpoints remain open. Thus, the excluded interval is $I = (-\underline{1.09}; \underline{1.09})$ as in the original CAlC method's example.

If we repeat this characterisation step for the second variant using the purely disjunctive Rule 3.5.5, we obtain different flags:

$$\begin{aligned} p_7 &:= \operatorname{res}(p_6, p_5) = x^2 - 1.1775 & \omega_7 &:= \omega_6 \vee \omega_5 \equiv \texttt{True}, \\ p_8 &:= \operatorname{res}(p_6, p_4) = x^2 - 1.1775 & \omega_8 &:= \omega_6 \vee \omega_4 \equiv \texttt{True}, \\ p_9 &:= \operatorname{res}(p_4, p_5) = x^2 - 1.1775 & \omega_9 &:= \omega_4 \vee \omega_5 \equiv \texttt{True}. \end{aligned}$$

As the polynomial flags of the defining zeros are all `True`, this time the conjunction $\omega_7 \wedge \omega_8 \wedge \omega_9$ is `True`, i.e. $b_\ell \equiv$ `True` and $b_u \equiv$ `True`. Hence, we deduce the interval $I = [-\underline{1.09}; \underline{1.09}]$. Compared to the interval approach, the ratio of strict constraints in $\mathcal{C}$ is smaller but nevertheless sufficient to infer closed bounds.

Although the negation might seem obstructive in this example, it can be powerful: If c_1, c_2, and c_3 had been built up on weak inequations only, the first variant would flag p_7, p_8, and p_9 all `True` and a both sides closed interval I is inferred. The second variant would flag these polynomials `False` and, thus, concludes a both sides open interval.

Chapter 4

Benchmarks

In the previous chapter we introduced two different approaches for inferring closed bounds when deducing UNSAT intervals in the CAlC method. We revisit the approaches once again but this time from a practical point of view.

4.1 Dataset and Environment

This comparison aims to determine the impact of the two approaches on the results and on the strategy of the CAlC method. The basis for this comparison is the existing implementation of the CAlC method in the *Satisfiability Modulo Theories Real Algebra Toolbox* (`SMT-RAT`) [CKJ+15]. In the following, this original solver is called `CAlC`. We refer to the two adaptations of `SMT-RAT` as `CAlC-I` for the approach based on intervals and `CAlC-P` for the one based on the second variant of flagged polynomials.

We use the Quantifier-free Non-linear Real Arithmetic dataset `QF_NRA` from `SMT-LIB` as of April 2022, consisting of 11552 instances from twelve different families [BFT16]. The instances might not only be pure conjunctive constraint sets but Boolean combinations of them, i.e. the CAlC method is used as a theory solver in the context of SMT solving. For our experiments, we apply limits of 60 seconds and 4 GB of RAM per instance, each running on a CPU with 2.1 GHz frequency.

4.2 Experimental Results

4.2.1 Setup

The CAlC method in `SMT-RAT` is different from the pseudo algorithmic procedures presented in the algorithms of this thesis. In particular, the characterisation process differs. Instead of calculating characterisation polynomials independent from each other and in a linear manner, a rule system is used, which is introduced in [NÁS+22]. That is the reason why we implement the second variant of the polynomial approach: In that case, no difference between the flagging of resultants is made.

Moreover, transferring the above ideas to an implementation requires adjustments and decisions on heuristics. We do not deepen these aspects but exemplary focus on a heuristic for choosing the covering intervals.

Solver	SAT		UNSAT		UNKNOWN		TIMEOUT	Solved	
CAlC	4553	0.38 s	4625	1.29 s	171	4.78 s	1783	9178	79.5%
CAlC-I	4609	0.43 s	4648	1.29 s	160	5.34 s	1726	9257	80.1%
CAlC-P	4612	0.53 s	4654	1.48 s	171	5.11 s	1711	9266	80.2%

Figure 4.2.1: Overview of the satisfiability results of the three solvers. UNKNOWN indicates that the solver stopped due to the incompleteness of the McCallum projection operator proj_{mc}. The column of solved instances states the sum of SAT and UNSAT results.

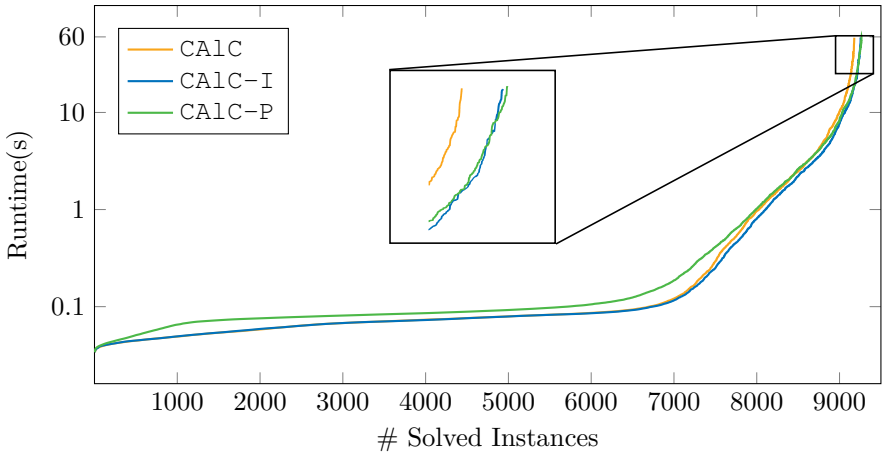


Figure 4.2.2: Accumulative illustration of solved instances and their mean runtimes.

After inferring a covering of the real line, characterisation steps are performed based on the polynomials stored in these intervals. However, not all intervals might be used as mentioned in [ÁDEK21]. The reason is that the covering should be non-redundant. The choice of an appropriate subset of intervals is likely to be not unique.

For the interval approach, a sophisticated choice could be to select many intervals stemming only from strict constraints. If the whole non-redundant covering consists of those intervals, for the deduced one the property holds as well.

Regarding the polynomial approach this choice is more involved. That is because one can only guarantee for closed endpoints if all polynomials appearing in some interval of the covering are flagged `True`. As this is unlikely, one has to go with less flagged polynomials. At least for us, no sophisticated preselection of intervals, which guarantees flagged polynomials for the closest zeros in the deduction, was obvious.

4.2.2 Overview

The overall performance of the three CAlC solvers is given in Figure 4.2.1. First of all, we want to remark that none of the solvers returns a wrong satisfiability result. The measures, however, differ between the three CAlC methods.

Both adaptations can increase the number of solved instances, i.e. SAT or UNSAT results. `CAlC-I` solves 79 instances and `CAlC-P` 88 instances more than `CAlC`. Inde-

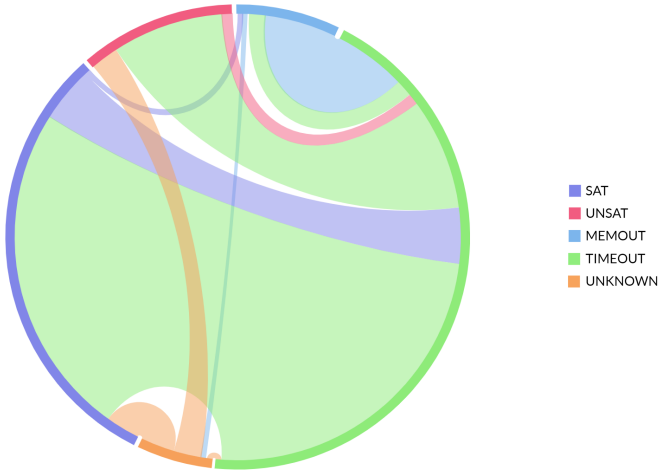


Figure 4.2.3: Transition chart showing movements in the results of `CAlC` and `CAlC-I` on instances with different results. The circle boundaries symbolise the different result types. Chords between the boundaries indicate how the result has changed. A chord of colour x between x and y indicates a transition from result x (`CAlC`) to result y (`CAlC-I`).

pendent of the approach, the share of SAT instances has increased more than the one of UNSAT instances. On the other hand, the mean runtime is raising simultaneously. Only for UNSAT instances, `CAlC-I` is slightly faster by 0.0034 s. We assume that the higher runtimes are related to the workload caused by maintaining the flags for interval bounds and the strictness of parent constraints. However, for the polynomial approach the hardness of additionally solved instances is relevant, too: Based on the instances that can be solved by both `CAlC` and `CAlC-P`, the runtime of `CAlC-P` is 0.909 s, i.e. 0.087 s higher than of `CAlC`, but 0.098 s lower than the average runtime of `CAlC-P` on all solved instances which is 1.007 s. For `CAlC-I`, no such gap is noticeable. Moreover, the new heuristics first check for the existence of intervals which are more promising for deducing closed bounds before actually computing a non-redundant covering. An accumulation of instances sorted by runtime is given in Figure 4.2.2.

4.2.3 Detailed Analysis of the Interval Approach

After giving an overview and a comparison of the approaches, in this section we focus on the deviations of the interval approach from the original CAlC method.

It is worthwhile to inspect where the surplus of SAT and UNSAT results comes from, depicted in Figure 4.2.3. To this end, we compare the instances with different results of `CAlC` and `CAlC-I`. As one might expect, a majority of the chords flowing to SAT and UNSAT results from instances that do not finish within 60 seconds. Note that the extra work that is performed to preselect promising intervals causes some former SAT or UNSAT instances to run into timeout.

Eleven instances turn from UNKNOWN to SAT or UNSAT and none the other way around. The reason for this behaviour is that by inferring an interval with closed bounds, sampling its endpoints (which have been nullifying in `CAlC`) becomes obsolete. However, it cannot be assumed that this is a general behaviour of `CAlC-I`, because all those newly solved instances come from the same dataset `zankl`.

Solver	# Overall Sample Components		# Sample Components on Bounds	
CAlC	165.281		109.745	
CAlC-I	155.751	−5.77%	102.478	−6.62%
CAlC-P	147.964	−10.48%	96.267	−8.77%

Figure 4.2.4: Number of executions of `sample_outside` to (re)assign a sample component. The number of such components laying on open interval bounds is a subset of the overall number of sample components. The percentages refer to the original number of samples needed by CAlC.

4.2.4 Detailed Analysis of the Polynomial Approach

The polynomial approach behaves rather similar to the interval one regarding the overall statistics measured beforehand. However, the number of UNKNOWN instances does not alter between CAlC and CAlC-P. Instead, the proportion of former timed out instances turning to SAT or UNSAT is larger with 62 and 39 instances. On the downside, the proportion of former SAT or UNSAT instances that time out is higher as well.

Intuitively, it might seem as if the polynomial approach can solve more instances than the interval one due to its purely disjunctive behaviour, but the difference is marginal. We have no clear explanation for why the polynomial approach can (only) solve nine more instances but suspect that the following three factors play a role:

- As mentioned in Section 4.2.1, the heuristic for choosing the non-redundant intervals is more involved and the optimality is not predictable: If one could predict which polynomials will induce the closest zeros next to the current sample before calculating the characterisation, one could save large parts of the characterisation step itself. Therefore, in the current implementation, intervals are preferred in which the proportion of polynomials flagged as `True` is high.
- Coefficients are always flagged `False`. It could be that this choice is too rigorous, but we did not discuss another flagging rule. Therefore, some bounds are not closed, although it might be possible in theory.
- Finally, the runtime of instances solved by CAlC-P is on average higher than the one of CAlC-I and CAlC. In particular, instances that can be solved in less than ten seconds are by far slowest on CAlC-P. The number of closed intervals per instance that can be deduced by CAlC-P but not CAlC, however, is on average 454.9 and therefore larger than 357.2 of CAlC-I. Thus, we assume that the polynomial based adaption lacks implementational efficiency rather than applicability.

Neglecting the drawbacks, the use of disjunctive connectives seems to be the reason why the polynomial approach can solve more instances than CAlC-I, namely 11%. This correlates with the number of partial samples needed to evaluate an average instance illustrated in Figure 4.2.4: The solver CAlC-P needs the fewest sample components to infer a satisfiability result, measured by the number of times a sample component is (re)assigned. That refers to both the total number of samples and those on open interval bounds.

Chapter 5

Conclusion

In this thesis we have recalled the concept of the Cylindrical Algebraic Coverings (CAlC) method and adapted it in two different ways in Chapter 3. We summarise the main properties of both approaches as well as the experimental results of the corresponding implementations. Afterwards, we discuss some ideas and open questions that could not be addressed in this thesis but are nevertheless of strong interest.

5.1 Summary

The idea of the CAlC method is, to check the satisfiability of a polynomial constraint set over the reals. To do so, (partial) sample points are chosen and for these it is checked whether a satisfying extension exists. This thesis mainly focuses on the situation where non-extensibility of the current sample has been shown. Regarding some variable ordering, the CAlC method deduces an interval around the value of the highest assigned variable in the current partial sample. This interval can safely be left out when searching for further extensions of the current lower-level assignments. The intervals are open ended in case they do not only contain a single point. However, different enhancements are conceivable.

The interval approach derives closed interval bounds from strict input constraints in the literal sense. If all constraints that led to the currently deduced interval make use of strict comparison operators, the zeros of the involved polynomials are surely not suitable to be chosen for a SAT sample. Hence, the endpoints of the interval do not need to be considered for further checks, i.e. the interval's bounds are allowed to be closed.

The polynomial approach abstracts from this straight inheritance relationship. It is based on strict constraints as well but does not encode the information in an interval but instead annotates polynomials with flags. This allows for a more refined procedure since the number of polynomials per covering is usually greater than the number of intervals, i.e. there might be more sources for closed interval bounds. In particular, fewer requirements have to be met when deducing closed bounds as the rules used to inherit polynomial flags are disjunctive.

In practise, both adaptions are able to increase the number of solved instances because fewer samples need to be checked. The difference between the practical impact of the two approaches is small: The polynomial approach does not outperform the one based on intervals.

5.2 Future Work

In this thesis, we introduced two approaches to infer closed interval bounds based on the strictness of input constraints. However, even though we presented intuitive arguments why we believe that the polynomial approach is correct, we could not yet formally prove its correctness. Hence, the next step is to substantiate and formalise this approach.

The two variants proposed for flagging the resultants characterising a cell in the polynomial approach have different semantics, thus we need to prove their correctness separately. A further interesting question is whether there are other rules for flagging that can be used.

Moreover, coefficients in the polynomial-based approach are currently flagged independent of their parent polynomial. It can be assumed that there is a less restrictive alternative that does not generally set the flags to `False`.

For the interval approach, we conjecture that it can be relaxed as well, i.e. the conjunctive behaviour might be reduced to weaker conditions. To us, it makes sense that it suffices if one of two adjacent cells is flagged `True`.

In addition, the implementation should be further improved. More fundamentally and independent from the specific approach, it is important to consider how to overcome the structural differences between the CAlC method introduced here and the rule-based system used by `SMT-RAT` [NÁS+22, CKJ+15].

Next, the heuristics for selecting the intervals that cover the real line should be optimised. Particularly concerning the polynomial approach, it can be assumed that this could unfold as yet unused potential. Currently, there is a low advantage of using disjunctive connectives. A new heuristic might also contribute to reducing the average runtime per instance, because there is a considerable difference in runtime between `CAlC-I` and `CAlC-P`, especially for easier instances.

Finally, the idea of how strict constraints are exploited in SMT solving might be transferable to other techniques. For example, first tests with the single cell construction introduced by Nalbach et al. in [NÁS+22] do seem promising.

Bibliography

[ÁDEK21] Erika Ábrahám, James H. Davenport, Matthew England, and Gereon Kremer. Deciding the Consistency of Non-Linear Real Arithmetic Constraints with a Conflict Driven Search Using Cylindrical Algebraic Coverings. *Journal of Logical and Algebraic Methods in Programming*, 119:100633, 2021.

[Bär22] Philipp Bär. Exploiting Strict Constraints in the Computation of Cylindrical Algebraic Coverings – Illustration. `https://doi.org/10.5281/zenodo.6738566`, 2022.

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

[Bro15] Christopher W. Brown. Open Non-Uniform Cylindrical Algebraic Decompositions. In *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '15, page 85–92. Association for Computing Machinery, 2015.

[Bro22] Christopher W. Brown. Personal Communication. May 12, 2022.

[CÁ11] Florian Corzilius and Erika Ábrahám. Virtual Substitution for SMT-Solving. In *Proceedings of the Fundamentals of Computation Theory*, pages 360–371. Springer Berlin Heidelberg, 2011.

[CKJ+15] Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp, and Erika Ábrahám. SMT-RAT: An Open Source C++ Toolbox for Strategic and Parallel SMT Solving. In *Proceedings of the Theory and Applications of Satisfiability Testing – SAT 2015*, pages 360–368. Springer International Publishing, 2015.

[Col75] George E. Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In *Proceedings of the Second GI Conference on Automata Theory and Formal Languages*, pages 134–183. Springer, 1975.

[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.

[FHT+06] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:209–236, 2006.

[Fit96] Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Texts and Monographs in Computer Science. Springer New York, 2nd edition, 1996.

[FOSV17] Pascal Fontaine, Mizuhito Ogawa, Thomas Sturm, and Xuan Tung Vu. Subtropical Satisfiability. In *Proceedings of the Frontiers of Combining Systems*, pages 189–206. Springer International Publishing, 2017.

[Kna08] Anthony W. Knapp. *Advanced Algebra*. Cornerstones. Birkhäuser Boston MA, 1st edition, 2008.

[Lev73] Leonid A. Levin. Universal Sequential Search Problems. *Problems of Information Transmission*, 9(3):115–116, 1973.

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

[NÁS+22] Jasper Nalbach, Erika Ábrahám, Philippe Specht, Christopher W. Brown, James H. Davenport, and Matthew England. Levelwise Construction of a Single Cylindrical Algebraic Cell. Under Review, 2022.

[Rot10] Joseph J. Rotman. *Advanced Modern Algebra*, volume 114 of *Graduate Studies in Mathematics*. American Mathematical Society, 2nd edition, 2010.

[Sne05] Gregor Snelting. Quantifier Elimination and Information Flow Control for Software Security. In *Proceedings of the A3L 2005*, pages 237–242. BoD, 2005.

[Stu06] Thomas Sturm. New Domains for Applied Quantifier Elimination. In *Computer Algebra in Scientific Computing*, pages 295–301. Springer Berlin Heidelberg, 2006.

[Tar51] Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry*. RAND Corporation, 1951.