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

PIECEWISE LINEAR UNDER-APPROXIMATION OF CELL BOUNDARIES IN MCSAT

Paul Tristan Wagner

Communicated by
Prof. Dr. Erika Ábrahám

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

Additional Advisor:
Jasper Nalbach

Aachen, 28.07.2023

Abstract

Satisfiability modulo theories (SMT) solving is the process of determining the satisfiability of a logical formula in some first-order theory. A particularly interesting and general first-order theory is called quantifier free non-linear real arithmetic (QFNRA). SMT solving of formulae in this theory can be used during the process of formal verification. In this thesis, we build upon a recent result that showed how polynomial bounds can be under-approximated to improve the performance of SMT solvers for quantifier free non-linear real arithmetic. In particular, we will present how piecewise linear functions can be used to achieve better under-approximations. We implemented two general variants of this approach into the open-source C++ toolbox for strategic and parallel SMT solving (SMT-RAT) that offer many settings for fine-tuning. Furthermore, we evaluated the implementation on the basis of SMT-LIB's QF_NRA benchmark dataset. Our evaluation revealed that our methods outperform the default approach, which does not use approximations.

Acknowledgements

I would like to thank Professor Erika Ábrahám for providing me with the opportunity to write my bachelor thesis at the Theory of Hybrid Systems research group on this very interesting but also challenging topic and for teaching me the necessary foundations on satisfiability checking. Furthermore, I would like to thank Professor Jürgen Giesl for being the second examiner for this thesis and for sparking my interest in formal verification. I am thankful to Jasper Nalbach for being my advisor for this thesis. I am very grateful for his endless patience and his ability to answer my every question.

Special thanks are addressed to my family and to my girlfriend Zofia, who have always supported me throughout my studies.

Contents

Chapter 1

Introduction

Software plays an increasingly relevant role in our day-to-day lives. Even if it is sometimes not obvious, a large proportion of people depend on the correctness of safety-critical software systems every day.

Some examples of such systems include autonomous or semi-autonomous vehicles, railway signaling, medical devices, air traffic control, and industrial control systems.

Safety-critical software systems need to be designed, developed, and tested rigorously in order to prevent severe consequences that could include loss of life, injury, or damage to the environment.

Traditional testing of software systems works by examining the behavior of the software on a set of example inputs. The problem with this is that even if the system behaves correctly for the selected set of example inputs, this does not give any guarantees that it behaves as intended for any possible input.

Formal verification is a rigorous and systematic process in which mathematical methods are used to determine whether a software system adheres to its intended behavior.

Formally verifying a software system can be divided into multiple steps. First, a model for the software system is created that abstractly represents the properties of the system one is interested in. Secondly, a specification is created that defines the intended behavior of the software system at hand. The model and the specification are then passed to a verification tool, which can rigorously deduce whether the model adheres to the given specification or not. In the end, the verification tool has either produced a formal proof showing the model always adheres to the given specification, or it provides a counterexample in the form of an input for which the model can behave in a way violating the specification.

Hybrid software systems are a class of systems that can behave in discrete as well as continuous ways.

Satisfiability modulo theories (SMT) is an approach to formal verification that has increasingly been used successfully in the past. In the SMT approach, both the system model and the specification are encoded in the form of logical formulae in some first-order theory. This makes the SMT approach particularly suitable for formally verifying the correctness of hybrid systems. The logical operators in such a formula are used to describe the system's discrete behavior, while the literals in the considered theory describe the system's continuous behavior.

One especially interesting theory is the theory of *quantifier free non-linear real*

arithmetic (QFNRA). It can be used to model hybrid systems exhibiting both linear and non-linear behavior. While this makes QFNRA extremely general and attractive for modeling hybrid software systems, it is also known in theoretical computer science that QFNRA is decidable, meaning there are algorithms that can always compute whether a given QFNRA formula is satisfiable or not.

In practice, however, it is the case that determining the satisfiability of QFNRA formulae presents an incredibly challenging task. One of the most prominently used algorithms for deciding the satisfiability of QFNRA in practice is the *cylindrical algebraic decomposition (CAD)* algorithm. Although it is known that the CAD algorithm has a runtime that grows doubly exponential in the worst-case, there are ongoing efforts to improve its runtime in practice using different heuristics.

1.1 Research Question

One particularly time-consuming task for most algorithms computing the satisfiability of QFNRA formulae is the computation of so-called *polynomial resultants*, which will be properly introduced in Chapter 2. A heuristic approach that aims to reduce the total amount of time spent computing these polynomial resultants has been developed by Promies [Pro22]. The presented approach is based on the idea of introducing new artificial polynomials which approximate the behavior of the original polynomials, aiming to compute simpler resultants instead of fewer. Promies' experimental results showed that this approach can significantly improve the time needed to determine the satisfiability of QFNRA formulae in many cases. And yet, it proved to be challenging to find polynomials that, on the one hand, match the behavior of the original polynomial more closely while also being of a form much simpler compared to the original polynomial in order to reduce the computational effort. For this reason, Promies proposed another idea that was deemed promising to investigate: In this thesis, we therefore want to build upon the ideas Promies developed and investigate the possibility of using piecewise defined linear functions to approximate the original polynomials.

1.2 Thesis Outline

In Chapter 2 we will present the necessary foundations as well as the work that has been done by Promies we are going to build upon. Chapter 3 will then cover the use of piecewise linear functions, propose two specific approaches on how they can be used for our purposes, and also show the ways in which we have implemented them in the *SMT-RAT* solver. In Chapter 4 we are going to evaluate our ideas and implementation using the SMT-LIB QF_NRA benchmarks and discuss our results. Lastly, we will conclude our thesis with an outlook on further improvements to our work and a summary in Chapter 5.

Chapter 2

Preliminaries

2.1 SMT Solving

Satisfiability modulo theories (SMT) is a variation of the well-known Boolean satisfiability (SAT) problem and generalizes it by allowing to form more complicated formulae containing literals from a broad selection of theories like real arithmetic, integers, bit-vectors, etc.

The process of determining the satisfiability of such a formula regarding some background theory is called *SMT solving*. The classical approach for SMT solving, also called *DPLL(T)*, consists of two phases. In the first phase, a SAT solver analyzes the structure of the Boolean skeleton of the formula. If it determines that the Boolean skeleton of the formula is satisfiable, it proposes a possible set of theory literals that need to be satisfied. In the second phase, this set of theory literals is then checked for consistency by a so-called theory solver, which either concludes that the set of theory literals is indeed consistent, which means that the original formula is satisfiable, or returns an explanation for the inconsistency in the form of an unsatisfiable subset of theory literals. These two phases alternate with each other until either the satisfiability or the unsatisfiability is concluded.

The classical approach has the advantage that advances and breakthroughs in SAT solving easily translate to improving SMT solvers.

2.2 Quantifier Free Non-Linear Real Arithmetic

In this thesis, we will deal with the theory of quantifier free non-linear real arithmetic (QFNRA) and the process of deciding the satisfiability of a given QFNRA formula. Quantifier free non-linear arithmetic encompasses real valued variables combined with multiplication and addition, as well as the comparison predicates $>$, $\geq$, $=$, $\leq$, $<$, $\neq$ and has been shown by Tarski in 1948 to be decidable even for quantified formulae. Although the proof for this is constructive, the presented algorithm has a non-elementary runtime complexity, meaning that it cannot be used in practice for deciding the satisfiability of QFNRA formulae.

We will now continue by formally defining a couple of important concepts so that it will be clear what exactly we are referring to later on.

Definition 2.2.1. *(Multivariate Polynomials) as presented in [Pro22]. Let R be a*

ring $(R, +, \cdot)$, $x_1, \ldots, x_n$ *variables and* $m \in \mathbb{N}$. *For all* $i \in \{0, ..., m\}$ *and* $j \in \{0, ..., n\}$ *let* $a_i \in R$ *and* $e_{i,j} \in \mathbb{N}$. *Then the term*

$$\sum_{i=1}^{n} a_i \prod_{j=1}^{n} x_i^{e_{i,j}}$$

is called a polynomial *over* R *in the variables* $x_1, \ldots, x_n$. *The set of all such polynomials is again a ring and denoted by* $R[x_1, \ldots, x_n]$. *We always assume that the given variables are ordered according to some total variable ordering* $\prec$ *with* $x_1 \prec ... \prec x_n$.

Definition 2.2.2. *(Polynomial Constraint). Let* $p \in \mathbb{Q}[x_1, \ldots, x_n]$ *and* $\sim \in \{<, \leq, =, \geq, >, \neq\}$. *We call* $p \sim 0$ *a* polynomial constraint.

Definition 2.2.3. *(Formula). A* quantifier free formula of non-linear real arithmetic (QFNRA formula) *is a Boolean combination of polynomial constraints using the operators* $\wedge, \vee, \neg$. *Every QFNRA formula* φ *has a logically equivalent formula* φ_{cnf} *in conjunctive normal form (CNF), meaning that there are* $c, k_1, \ldots, k_c \in \mathbb{N}$ *such that*

$$\varphi_{cnf} = \bigwedge_{i \in \{0,\ldots,c\}} \left(\bigvee_{j \in \{0,\ldots,k_i\}} l_{i,j} \right)$$

Where the $l_{i,j}$ *are polynomial constraints.*

The disjunctions in the CNF are called clauses, and the $l_{i,j}$ *are also referred to as literals.*

Definition 2.2.4. *(Semantics of QFNRA) as presented in [Pro22]. Let* $r \in \mathbb{R}^n, p \in \mathbb{Q}[x_1, \ldots, x_n]$ *be a polynomial and* $\sim \in \{<, \leq, =, \geq, >, \neq\}$.

- *We say* r *satisfies the constraint* $p \sim 0$, *if* $p(r) \sim 0$ *evaluates to true regarding the standard semantics of* $\mathbb{R}$. *This is also denoted as* $r \models (p \sim 0)$.
- *This notion of satisfaction is extended in the usual way. For formulae* φ, ψ *with free variables* $x_1, \ldots, x_n$ *we have* $r \models \varphi \wedge \psi$ *if* $r \models \varphi$ *and* $r \models \psi$, $r \models \varphi \vee \psi$ *if* $r \models \varphi$ *or* $r \models \psi$ *and finally* $r \models \neg\varphi$ *if* $r \not\models \varphi$.
- *A QFNRA formula* $\varphi(x_1, \ldots, x_n)$ *is satisfiable, if there exists a satisfying assignment* $r \in \mathbb{R}^n$ *with* $r \models \varphi$, *in this case we call* r *a model for* φ. *If there exists no such model, we call* φ *unsatisfiable.*

Example 2.2.1. *The formula*

$$\varphi := (x_2^2 - x_1^3 + x_1^2 + 3x_1 - 6 > 0) \wedge (x_1 x_2 - 4 < 0)$$

is a QFNRA formula in conjunctive normal form. The assignment $s = (-3, 1) \in \mathbb{R}^2$ *which assigns -3 to* x_1 *and 1 to* x_2 *satisfies* φ. *Therefore* φ *is satisfiable and we say that* s *is a* model *for* φ $(s \models \varphi)$.

Generally, algorithms are unable to deal with arbitrary real numbers. For reasoning about QFNRA it turns out that it suffices to search for solutions in a proper subset of the reals - the *real algebraic numbers.*

Definition 2.2.5. *(Univariate Polynomials). Let* R *be a ring and* $p \in R[x_1, \ldots, x_n]$. *We call* p *univariate in* $x \in \{x_1, ..., x_n\}$ *if* $p \in R[x]$, *that is,* p *contains no variables other than* x. *Otherwise,* p *is called multivariate.*

Definition 2.2.6. *(Real Algebraic Number). A* real algebraic number *is a real number that is the root of some univariate polynomial with rational coefficients.*

Let $r \in \mathbb{R}$. We say that r is algebraic if there exists $p \in \mathbb{Q}[x] \setminus \{0\}$ such that $p(r) = 0$. The set of all real algebraic numbers (RANs) is denoted by $\mathcal{R}$.

Real algebraic numbers are typically represented as tuples of a univariate polynomial $p \in \mathbb{Q}[x]$ and an interval $I \subset \mathbb{R}$ with rational endpoints such that exactly one root of p lies inside I. Note that there exist efficient methods that can isolate the real roots of any univariate polynomial.

Example 2.2.2. *The real number $\sqrt{3}$ is also an algebraic number, as it is the root of the univariate polynomial $x^2 - 3$. It can be represented as $\big(x^2 - 3,\ (1, 3)\big)$. Note, however, that many real numbers are not algebraic numbers, as they aren't the root of any polynomial. An example of this is π or Euler's number e.*

In 1975 further advancements in the field of computational real algebraic geometry have been made by Collins [Col75] who introduced the notion of a cylindrical algebraic decomposition (CAD) together with an algorithm for computing such a decomposition. The CAD algorithm can be used to decompose the multidimensional real space into a finite set of representative sample points with respect to a set of polynomial constraints. Given this set of sample points, it is then possible to evaluate the polynomial constraints for each sample point. The set of polynomial constraints turns out to be satisfiable if and only if at least one of the sample points satisfies it.

For these reasons, the CAD algorithm can be used as a theory solver in the classical DPLL(T) approach to SMT solving.

We will now explain the fundamental ideas that Collins' algorithm for computing a cylindrical algebraic decomposition relies on.

Definition 2.2.7. *(Cell) as presented in [NAS+ 22]. A* cell *is a non-empty connected subset of $\mathbb{R}^i$ for some $i \in \mathbb{N}$.*
We call a cell algebraic if it is a semi-algebraic set, meaning that it is the finite union of sets defined by polynomial equalities and inequalities.

Definition 2.2.8. *(Decomposition) as presented in [NAS+ 22]. A set $D = \{R_1, ..., R_k\}$ of cells of $\mathbb{R}^n$ is called a* decomposition *of $\mathbb{R}^n$ if*

- $\bigcup_{i=1}^{k} R_i = \mathbb{R}^n$ *and*
- $R_i \cap R_j = \emptyset$ *for $i \neq j$*

We call a decomposition algebraic if its cells are algebraic.

Definition 2.2.9. *(Cylindrical over a Decomposition) as presented in [NAS+ 22]. A decomposition D of $\mathbb{R}^n$ is called* cylindrical over a decomposition *D' of $\mathbb{R}^m$ with $m < n$ if all projections of cells $R \in D$ onto $\mathbb{R}^m$ are again cells in D'.*

Definition 2.2.10. *(Cylindrical Algebraic Decomposition) as presented in [NAS+ 22]. A* cylindrical algebraic decomposition (CAD) *of $\mathbb{R}^n$ is an algebraic decomposition D of $\mathbb{R}^n$ such that there exists a sequence $(D_1, ..., D_n)$ with $D_n = D$ such that each D_i is cylindrical over D_{i-1} for $i \in \{2, ..., n\}$.*

Definition 2.2.11. *(Sign of a polynomial) [Pro22]. The* sign of a polynomial *$p \in \mathbb{Q}[x_1, \ldots, x_n]$ at $r \in \mathbb{R}^n$ is defined as*

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

Definition 2.2.12. *(Sign Invariance) [NAS$^+$22]. A cell R is called* P-sign invariant *for a set $P \subset \mathbb{Q}[x_1, \ldots, x_n]$ if for all $p \in P$ and $r_1, r_2 \in R$ it holds that $sgn(p(r_1)) = sgn(p(r_2))$.*

Definition 2.2.13. *(Cylindrical Algebraic Decomposition for a Set of Polynomials) as presented in [Á14]. A* CAD of $\mathbb{R}^n$ for a finite set of polynomials P *is a CAD for which all cells are P-sign invariant.*

Definition 2.2.14. *(Delineability) [Col75] as presented in [NAS$^+$22].*
Let $i \in \mathbb{N}, R \subseteq \mathbb{R}^i$ be a cell, and $p \in \mathbb{Q}[x_1, \ldots, x_{i+1}] \setminus \{0\}$. The polynomial p is called delineable *on R if and only if there exist finitely many continuous functions $\theta_1, ..., \theta_k : R \to \mathbb{R}$ such that*

- *$\theta_1 < ... < \theta_k$,*
- *the set of real roots of $p(r, x_{i+1})$ is $\{\theta_1(r), ..., \theta_k(r)\}$ for all $r \in R$ and*
- *there exists constants $m_1, ..., m_k \in \mathbb{N}_{>0}$ such that for all $r \in R$ and all $j \in \{1, ..., k\}$, the multiplicity of the root $\theta_j(r)$ of $p(r, x_{i+1})$ is m_j.*

We say that a finite set of polynomials $P \subset \mathbb{Q}[x_1, \ldots, x_n]$ is delineable on a cell R of $\mathbb{R}^i$ if the product of the polynomials is delineable on R.

An intuition for the notion of delineability for a finite set of polynomials $P \subset \mathbb{Q}[x_1, \ldots, x_n]$ over a cell R is that the roots of P behave continuously over R, while also their number and multiplicity and the number of common roots stay constant.

Definition 2.2.15. *(CAD projection operator) [NAS$^+$22]. A* CAD projection operator *proj is a function $proj : 2^{\mathbb{Q}[x_1,\ldots,x_n]} \to 2^{\mathbb{Q}[x_1,\ldots,x_{n-1}]}$ such that for any $i \in \{1, ..., n\}$, any cell R of $\mathbb{R}^i$ and any set of irreducible polynomials $P \subset \mathbb{Q}[x_1, \ldots, x_n]$ each of level $i + 1$ it holds that*

- *proj(P) is a set of polynomials of level at most i and*
- *if proj(P) is sign-invariant on R then P is delineable on R.*

The CAD projection operator is the main component needed in the CAD algorithm. Although Collins originally presented a complete projection operator, meaning that it will always succeed in computing a CAD for any finite set of polynomials, it was relatively inefficient and has been subject to optimization and improvements since. In this thesis, we will mainly focus on the projection operator presented by McCallum [McC85]. It has the advantage of computing a much smaller set - even a strict subset - of polynomials compared to Collins' projection operator. This reduction in the number of polynomials vastly improves the efficiency of the CAD algorithm, but it comes at the cost of making the CAD algorithm incomplete, meaning that there are cases for which the algorithm fails to compute a decomposition. Albeit that these cases are statistically rare [BDE$^+$16].

McCallum's projection operator utilizes the concepts of polynomial *coefficients*, *discriminants* and *resultants*, for which we will now provide a formal definition as well as a geometric interpretation.

Definition 2.2.16. *(Coefficients and Degree) as presented in [Pro22]. Let $p \in \mathbb{Q}[x_1, \ldots, x_n]$. We are able to interpret p as a univariate polynomial in x_n with coefficients in $\mathbb{Q}[x_1, \ldots, x_{n-1}]$. Therefore let $m \in \mathbb{N}$, $c_0, \ldots, c_m \in \mathbb{Q}[x_1, \ldots, x_{n-1}]$ such that $p = \sum_{i=0}^{m} c_i x_i^m$*

- *We define the set* $coeff_{x_n}(p) := \{c_0, \ldots, c_m\} \setminus \{0\} \subset \mathbb{Q}[x_1, \ldots, x_{n-1}]$ *of polynomial* coefficients *of* p *with respect to* x_n*.*
- *We define the* degree *of* p *with respect to* x_n *as* $deg_{x_n}(p) = m$*.*

Definition 2.2.17. *(Resultants and Discriminants) as presented in [Pro22]. Let* p *and* q *be two polynomials in* $\mathbb{Q}[x_1, \ldots, x_n]$*.*

- *The* resultant *of* p *and* q *with respect to the variable* x_n *is a polynomial denoted as* $res_{x_n}(p, q) \in \mathbb{Q}[x_1, \ldots, x_{n-1}]$ *such that for all* $s \in \mathbb{R}^{n-1}$ *it holds that*

$$res_{x_n}(p, q)(s) = 0 \Leftrightarrow p(s, x_n) \text{ and } q(s, x_n) \text{ have a common complex root}$$

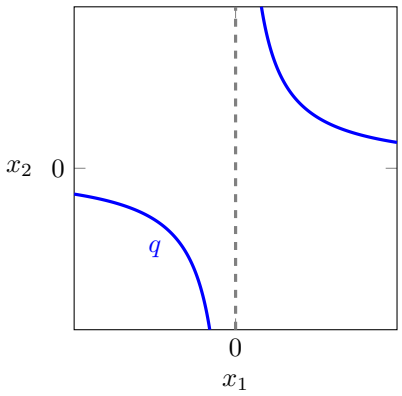
- *The* discriminant *of* p *with respect to the variable* x_n *is a polynomial denoted as* $disc_{x_n}(p) \in \mathbb{Q}[x_1, \ldots, x_{n-1}]$ *such that for all* $s \in \mathbb{R}^{n-1}$ *it holds that*

$$disc_{x_n}(p) = 0 \Leftrightarrow p(s, x_n) \text{ has a multiple complex root}$$

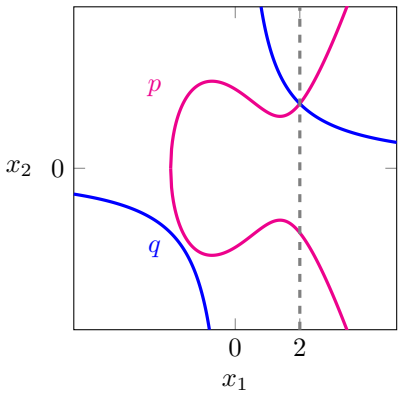
Polynomial resultants are also commonly defined as the determinant of the so-called *Sylvester matrix*. If the two polynomials are of degree m and degree n respectively, the Sylvester matrix is a matrix in $\mathbb{Q}[x_1, \ldots, x_{n-1}]^{(m+n)\times(m+n)}$ containing the coefficients of the two polynomials. In practice, there are more advanced methods for computing the resultant of two polynomials other than computing the determinant of the Sylvester matrix directly. One example of this is the *subresultant algorithm*, which is able to avoid much of the complexity that is caused by more straightforward implementations. Anyhow, it is still known that in the worst-case, the subresultant algorithm must perform a number of multiplications and additions that grows quadratically in the degree of the polynomials [Duc00]. It is important to understand that the resultant is typically of much higher degree compared to the original polynomials and that its computation and the subsequent isolation of its real roots, which often follows for our purposes, are the main things responsible for the complexity of the commonly used algorithms for deciding the satisfiability of QFNRA and something we would like to avoid whenever possible. Additionally, it is important to note that the methods for computing polynomial resultants and discriminants that are used in practice can only effectively work on *square-free* and pairwise *co-prime* polynomials. That is, a polynomial $p \in \mathbb{Q}[x_1, \ldots, x_n] \setminus \{0\}$ is square-free if it does not decompose into $p = g^2 * h$ with $g \in \mathbb{Q}[x_1, \ldots, x_n] \setminus \mathbb{Q}$ and $h \in \mathbb{Q}[x_1, \ldots, x_n]$. This means that a square-free polynomial does not contain the square of another non-constant polynomial as a factor. Two polynomials $p, q \in \mathbb{Q}[x_1, \ldots, x_n]$ are co-prime, if they do not compose into $p = g * h$, $q = g * f$ for $g \in \mathbb{Q}[x_1, \ldots, x_n] \setminus \mathbb{Q}$ and $h, f \in \mathbb{Q}[x_1, \ldots, x_n]$. Therefore, two polynomials are co-prime if they do not share any non-constant polynomial as a factor. For this reason, the presented algorithms expect sets of *irreducible* polynomials as inputs and regularly perform factoring into irreducible polynomials to ensure that the working set of polynomials stays irreducible. A polynomial $p \in \mathbb{Q}[x_1, \ldots, x_n]$ is called irreducible if it does not decompose into $p = g * f$ with $g, f \in \mathbb{Q}[x_1, \ldots, x_n] \setminus \mathbb{Q}$. An irreducible polynomial cannot be represented as the product of two non-constant polynomials. Note that an irreducible polynomial is always square-free and that two irreducible polynomials must either be equal or co-prime.

We will now briefly extend the definitions for coefficients, resultants, and discriminants with a useful geometric interpretation for the two-dimensional case using the two polynomials $p = x_2^2 - x_1^3 + x_1^2 + 3x_1 - 6$, $q = x_1 x_2 - 4 \in \mathbb{Q}[x_1, x_2]$. Also note that

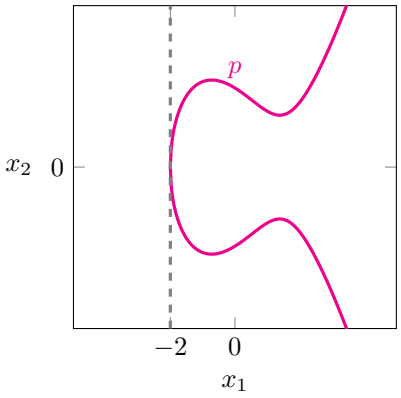
in the following graphs, we plot the set of real roots for the given polynomials. These sets are typically also called *polynomial varieties*.



(a) The set $\text{coeff}_{x_2}(q) = \{x_1, -4\}$ whose root in x_1 is exactly at the points at which q has a discontinuity.



(b) The resultant $\text{res}_{x_2}(p, q) = -x_1^5 + x_1^4 + 3x_1^3 - 6x_1^2 + 16$. Its root in x_1 is exactly at the intersection of the polynomial varieties of p and q



(c) The discriminant $\text{disc}_{x_2}(p) = 4(x_1^3 - x_1^2 - 3x_1 + 6)$ has its roots exactly at the turning point of the polynomial variety of p.

Figure 2.1: Geometric interpretation of the coefficients, resultants, and discriminants

Definition 2.2.18. *(McCallum's Projection Operator) [McC85] as presented in [NAS+22].* McCallum's projection operator *is defined for each* $i \in \{2, ..., n\}$ *and each set of irreducible polynomials* $P \subset \mathbb{Q}[x_1, \ldots, x_i] \setminus \{0\}$ *as*

$$proj_{mc}(P) = \bigcup_{\substack{p \in P \\ level(p)=i}} coeff_{x_i}(p) \cup \bigcup_{\substack{p \in P \\ level(p)=i}} \{disc_{x_i}(p)\} \cup \bigcup_{\substack{p,q \in P,\ p \neq q \\ level(p)=level(q)=i}} \{res_{x_i}(p, q)\} \cup \bigcup_{\substack{p \in P \\ level(p)<i}} \{p\}$$

Given both the intuition for delineability and the components in McCallum's projection operator, it is straightforward to see how it provides delineability: While the coefficients of the polynomial are used to separate discontinuities, the discriminants ensure that the roots of the polynomial have constant multiplicities, and the pairwise resultants of the polynomials guarantee that the number of common roots between the polynomials stays constant.

Definition 2.2.19. *(Level) [Pro22]. Let F be a polynomial, constraint or clause.*

$$\mathit{level}(F) := \mathit{max}(\{i \in \mathbb{N} \mid x_i \text{ appears in } F\} \cup \{0\})$$

Definition 2.2.20. *(Real Roots) [Pro22]. Let $p \in \mathbb{Q}[x_1, \ldots, x_n]$ and $r \in \mathbb{R}^n$. We call r a* real root *of p if $p(r) = 0$. The set of all real roots of p is denoted by realRoots(p).*

Definition 2.2.21. *(Indexed Root Expression) [Pro22]. Let $i, j \in \mathbb{N}$, $p \in \mathbb{Q}[x_1, \ldots, x_i]$ with level(p) $= i > 0$. Then $root_{x_i}[p, j] : \mathbb{R}^{i-1} \longrightarrow \mathbb{R} \cup \{undef\}$ is defined by*

$$root_{x_i}[p, j](s) = \begin{cases} undef, & \text{if } j > |realRoots(p(s, x_i))| \text{ or } p(s, x_i) = 0, \text{and otherwise} \\ \xi_j, & \text{where } realRoots(p(s, x_i)) = \{\xi_1, ..., \xi_k\} \text{ with } \xi_1 < ... < \xi_k. \end{cases}$$

We also call indexed root expressions of this form indexed root expressions of level i. Algorithmically, we address the index j of an indexed root expression `ir` with `ir.index`.

Definition 2.2.22. *(Symbolic Interval) [NAS$^+$22]. A* symbolic interval *I of level i can have two forms*

- *$I = (sector, l, u)$ where l is $-\infty$ or an indexed root expression of level i and u is ∞ or an indexed root expression of level i.*
- *$I = (section, b)$ where b is an indexed root expression of level i.*

Definition 2.2.23. *[NAS$^+$22] (Single Cell Data Structure). A* single cell data structure *is a sequence $R = (I_1, ..., I_n)$ where each I_i is a symbolic interval of level i.*

Now that we have introduced all the necessary prerequisites, we will present the algorithm for computing a cylindrical algebraic decomposition.

Algorithm 1: cad(P) [NAS+22]

Input: Finite set of polynomials $P \subset \mathbb{Q}[x_1, \ldots, x_n] \setminus \{0\}$
Output: Cylindrical algebraic decomposition for P
/* Step 1: Projection phase */
$P_n := \text{factors}(P)$
for $i = n-1, ..., 1$ **do**
  $P_i := \text{factors}(\text{proj}(P_i))$
  $P_{i+1} := \{p \in P_{i+1} \mid \text{level}(p) = i+1\}$
/* Step 2: Lifting phase */
$D_0 := \{()\}$
for $i = 1, ..., n$ **do**
  $D_i := \emptyset$
  foreach *cell* $R = (I_1, ..., I_{i-1}) \in D_{i-1}$ **do**
    $\Xi := \bigcup_{p \in P_i} \text{realRoots}(p(s, x_i))$
    sort elements of $\Xi = \{\xi_1, ..., \xi_k\}$ such that $\xi_1 < ... < \xi_k$
    foreach *interval* $I \in \{(-\infty, \xi_1), [\xi_1, \xi_1], (\xi_1, \xi_2), ..., (\xi_k, \infty)\}$ *for* $k > 0$ *and* $I \in \{(-\infty, \infty)\}$ *for* $k = 0$ **do**
      Define I_i as the symbolic interval that represents I
      $D_i := D_i \cup \{(I_1, ..., I_{i-1}, I_i)\}$
return D_n

Despite the algorithm for computing a CAD for a set of polynomials being a big improvement in solving non-linear real arithmetic, it still presents challenges.

The CAD algorithm exhibits a doubly exponential runtime complexity because, in the worst-case the number of CAD cells is doubly exponential, which also manifests in practical applications.

Therefore, there have been ongoing efforts to enhance the CAD algorithm by building upon its underlying ideas and aiming to achieve a more efficient approach.

For the purpose of deciding the satisfiability of QFNRA, an entire CAD provides more information than would be necessary to study the specific QFNRA formula at hand. This is because having the entire CAD can be used to reason about the satisfiability of any Boolean combination of the polynomial constraints present in the formula.

2.3 Model-Constructing Satisfiability Calculus

The *Model-Constructing Satisfiability Calculus (MCSAT)* was originally introduced as the prototype *nlsat* by Jovanović and de Moura in [JdM12] which is a decision procedure specifically for solving quantifier free non-linear real arithmetic. Nlsat was later generalized as the abstract decision procedure MCSAT in [JBdM13] for solving first-order logic.

It provides a more general alternative to classical DPLL(T) approach by combining the idea of incrementally constructing a model for the given QFNRA formula at hand with conflict resolution.

For simplicity, we will refer to the MCSAT instantiation for nonlinear real arithmetic simply as MCSAT in the remainder of this thesis.

In MCSAT, Boolean and theory searches are executed in parallel, and conflicts are distinguished between Boolean and theory conflicts. While Boolean conflicts are

addressed via Boolean resolution, theory conflicts are addressed using ideas from CAD.

The MCSAT algorithm incrementally tries to construct a sample point that satisfies all of the necessary constraints, which should hold according to the Boolean search. If, at some point, it finds itself unable to extend the sample point further, this is what we call a "theory conflict". Then the algorithm will compute an explanation in the form of a region containing the sample point on which the same constraints are violated for the same reasons. These regions are constructed as cylindrical algebraic cells and used to discard them from the search space in which MCSAT tries to construct sample points.

In this way, the MCSAT algorithm is also able to address some of the shortcomings of the classical DPPL(T) approach for SMT solving, as it is no longer necessary to compute an entire CAD.

Because the part of MCSAT where we generate explanations for theory conflicts has been designed in a relatively flexible way, it is also possible to apply many attempts at optimizing this process in various ways.

For a more in-depth overview of the MCSAT algorithm, we will refer to the explanation in [Pro22] as well as the original papers introducing the idea [JdM12, dMJ13, JBdM13].

In the next section, we will focus on computing explanations for the MCSAT algorithm.

2.4 Levelwise Construction of a Single Cylindrical Algebraic Cell

The task of *single cell construction* is: Given a finite set of polynomials $P \subset \mathbb{Q}[x_1, \ldots, x_n]$ and a sample point $s \in \mathcal{R}^n$, construct a cell $R \subseteq \mathbb{R}^n$ such that $s \in R$ and R is P-sign invariant. We want the resulting cell to be algorithmically computable and also to leverage the results of CAD theory. For these reasons, we want to focus on the construction of cylindrical algebraic cells. Our goal is to make the constructed cells as big as possible so that the MCSAT algorithm can exclude larger regions of its search space.

We will now present an approach to the construction of a single sign-invariant cylindrical algebraic cell that was recently proposed by Nalbach et al. [NAS+22]. It was named the "levelwise" single cell construction because, in contrast to its predecessors, it does not work by refining the cell polynomial by polynomial but rather by working down in the variable ordering, constructing the intervals of higher levels first.

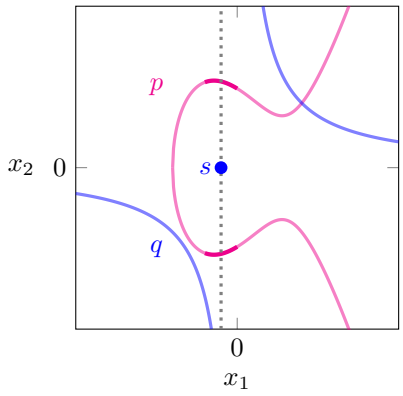
Algorithm 2: levelwise_single_cell(P, s) [NAS+22]

Input: Finite set of polynomials $P \subset \mathbb{Q}[x_1, \ldots, x_n] \setminus \{0\}$, sample $s \in \mathbb{R}^n$
Output: Single cell data structure $R = (I_1, ...I_i)$ of symbolic intervals such that $s \in R$ and R is P-sign invariant; or FAIL

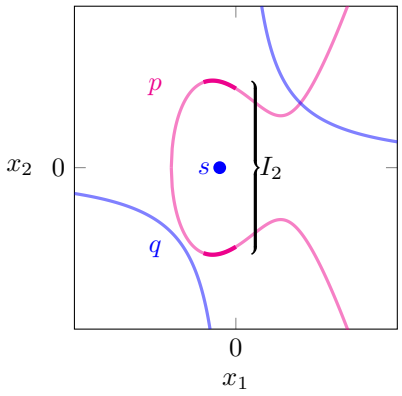
$P_n := \{p \in \text{factors}(P) \mid \text{level}(p) = n\}$
$P_\perp := \{p \in \text{factors}(P) \mid \text{level}(p) < n\}$
for $i = n, ..., 1$ **do**
  `// For these cases, McCallum's projection operator fails.`
  if $\exists p \in P_i : p(s_{[i-1]}, x_i) = 0$ **then**
    return FAIL
  $\Xi := \bigcup_{p \in P_i} \text{realRoots}(p(s_{[i-1]}, x_i))$ sort elements of $\Xi = \{\xi_1, ..., \xi_k\}$ such that $\xi_1 < ... < \xi_k$
  find $I \in \{(-\infty, \xi_1), [\xi_1, \xi_1], (\xi_1, \xi_2), ..., (\xi_k, \infty)\}$ for k > 0 and $I \in \{(-\infty, \infty)\}$ for $k = 0$ such that $s_i \in I$
  Define I_j as the symbolic interval that represents I
  if $i > 1$ **then**
    $P_{i-1} := \{p \in \text{factors}(\text{proj}_{\text{mc}}(P_i)) \cup P_\perp \mid \text{level}(p) = i - 1\}$
    $P_\perp := \{p \in \text{factors}(\text{proj}_{\text{mc}}(P_i)) \cup P_\perp \mid \text{level}(p) < i - 1\}$
return $(I_1, ..., I_n)$

The algorithm for the levelwise single cell construction is a slight modification to the CAD algorithm, where we only apply the so-called lifting phase over cells which correspond to the sample s [NAS+22].

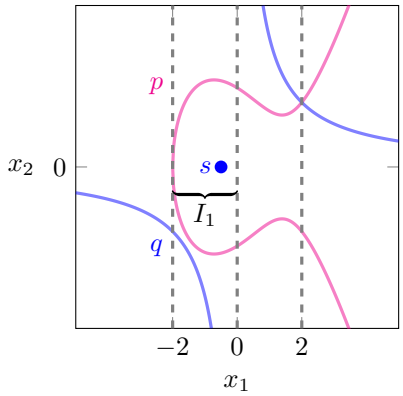
Example 2.4.1. *The following example depicts the single cell construction for our running example.*



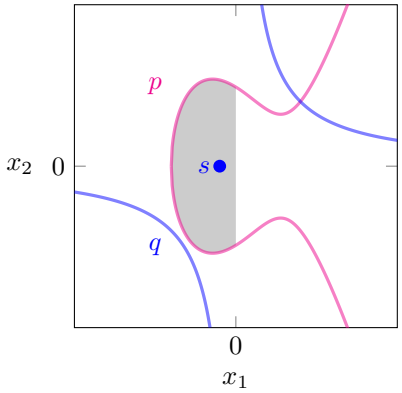
(a) We start out by substituting the sample point into all of the polynomials up to the highest level. This yields univariate polynomials in x_2 whose roots we can isolate and order.



(b) The two roots, which are directly above and directly below our sample point, define the symbolic interval I_2.



(c) After executing one McCallum projection, we are on the lowest level and can now isolate and order the roots of the projected univariate polynomials. The two roots, which are directly left and directly right of our sample point, define the symbolic interval I_1.

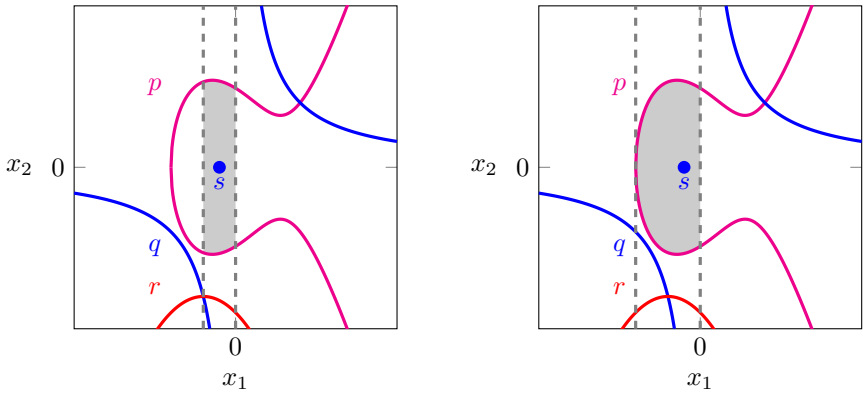


(d) The computed cell $R = (I_1, I_2)$. Note how the correctness of the projection ensures that I_2 is defined for all of the points in I_1.

Figure 2.2: Levelwise single cell construction

For the task of single cell construction, the pairwise resultant computation in McCallum's projection provides more resultants than are necessary, as we only need to ensure that no other root crosses into the computed cell. This observation enables us to omit the computation of many of the resultants, which saves time and generally results in bigger cells.

Example 2.4.2. *We added an additional polynomial r to show how some resultants in the projection can be ignored for single cell construction and how omitting them can result in bigger cells.*



(a) The full McCallum's projection also includes the resultant of q and r, even though their intersection is irrelevant for the sign-invariance of the resulting cell.

(b) We can omit computing the resultant of q and r which results in a bigger cell.

Figure 2.3: Cell size comparison between a full McCallum's projection and when we omit resultants

Of course, it is not the case that we can arbitrarily omit any resultants. We can only do so in a way that still guarantees the sign-invariance of the resulting cell. The idea of omitting resultants compared to the full McCallum's projection has been developed and formalized in [NAS+22] as an *indexed root ordering*. However, for our purposes of under-approximating the cell boundaries in the single cell construction, we are only going to use the so-called *"biggest cell heuristic"*. This heuristic only includes the resultants between the polynomial that defines the upper bound and all the polynomials that are above it at the sample point, as well as the resultants between the polynomial that defines the lower bound and all the polynomials that are below it at the sample point.

2.5 Under-Approximating Cell Boundaries

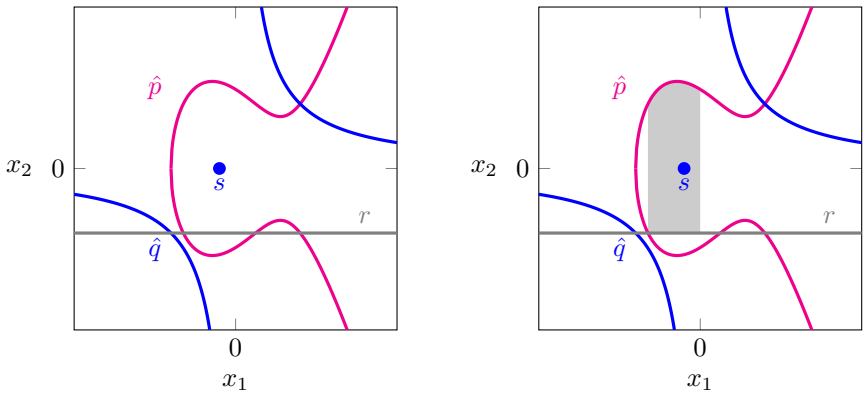
As we have already discussed, the computation of polynomial resultants is costly, because, most of the time, it yields polynomials of high degree. This especially gets worse if this process is iterated, as in the repeated projections in the CAD algorithm or the single cell construction. It is known that during the projection phase, the degrees of the polynomials grow doubly exponentially. This is the reason why Promies [Pro22] developed an approach for introducing new artificial polynomials of low degree during the single cell construction to simplify the computed resultants. This, however, comes at the cost of under-approximating the cell, at least on the level at which the artificial polynomial is introduced.

We will now present the basic ideas that Promies developed.

Example 2.5.1. *For demonstrative reasons, we replace the polynomials p and q from our running example with slightly perturbed versions of them. We define $\hat{p} := 2^6(x_1^6x_2^6 + 1)p + 1$ and $\hat{q} := 2^6(x_1^6x_2^6 + 1)q + 1$. While these are completely different polynomials, visually the difference between p and $\hat{p}$ as well as q and $\hat{q}$ is minuscule. Furthermore, this perturbation did not affect the root structure around the sample point s, meaning that the single cell construction would do exactly the same steps as in the original example. Note that $\hat{p}$ and $\hat{q}$ are still irreducible as well as co-prime, but that now $deg_{x_2}(\hat{p}) = 8$ and $deg_{x_2}(\hat{q}) = 7$.*

In the levelwise single cell construction using the biggest cell heuristic, we would now compute $res_{x_2}(p, q)$. But since $deg_{x_1}(res_{x_2}(p, q)) = 77$ it could potentially be beneficial if it is possible to avoid computing it in the first place. The idea that Promies experimented with was to introduce new artificial polynomials of low degree, therefore under-approximating the cell on the current level but also avoiding computing complicated resultants.

Assume, for instance, that we artificially introduce the polynomial $r := x_2 + 2$ to our example. Now, according to the biggest cell heuristic, we need to compute the two resultants $res_{x_2}(r, \hat{p})$ and $res_{x_2}(r, \hat{q})$ instead of one, but they are of much lower degree $deg_{x_1}(res_{x_2}(r, \hat{p})) = 9$ and $deg_{x_1}(res_{x_2}(r, \hat{q})) = 7$.



(a) We introduced the artificial polynomial $r := x_2 + 2$ to the perturbed version of our running example.

(b) In this case, the resulting cell from single cell construction is a strict subset of the cell that would be computed without the introduction of artificial polynomials. On the other hand, it is much easier to compute this new approximated cell.

Figure 2.4: Introduction of artificial polynomials of low degree to avoid the computation of complicated resultants

More specifically, Promies concentrated on the introduction of artificial polynomials whose roots at the sample point lie at a rational point with simple representation. This is the case because it was observed that simply choosing a rational very close to the bound of the polynomial we are trying to approximate can lead to increasingly complicated coefficients in our set of polynomials. We must therefore acknowledge that there exists a trade-off between how well we can mirror the behavior of the polynomial using our approximations and the complexity of the rational coefficients used.

Promies coined the term "naïve" approximation for describing the introduction of the presented type of polynomials, where we have $h := x_i - r$ with $r \in \mathbb{Q}$. It was noted further that if we continue to use this kind of approximation on each level for approximating both the upper and lower bounds, the resulting cell is a box [Pro22].

Promies also addressed termination, stating that without further restrictions, it is possible that MCSAT can do an arbitrary number of these approximations, therefore leading to non-termination.

Example 2.5.2. *Assume, for instance, that we start out with our usual running example with the initial sample point* $s_0 = (-\frac{1}{2}, 1)$. *Now we want to approximate the lower bound of the polynomial* p *and introduce the artificial polynomial* $h_0 := x_2$ *whose root is always at* $x_2 = 0$, *following the "naïve" approximation approach. This excludes a large region containing* s_0 *but does not prevent the MCSAT algorithm to choose the point* $s_1 = (-\frac{1}{2}, -\frac{1}{2})$ *as the next sample. We could now approximate the lower bound of the polynomial* p *again by introducing the artificial polynomial* $h_1 := x_2 + 1$. *If we do not restrict the process of approximating the cell boundaries in some way, there is nothing that prevents MCSAT from continuing this process with ever more artificial polynomials whose roots converge against the root of the approximated polynomial at*

the sample point but never reach it, thus leading to non-termination. It is apparent that the original levelwise single cell construction prevents this kind of problematic behavior, as it excludes the entire available interval on that level from the get-go.

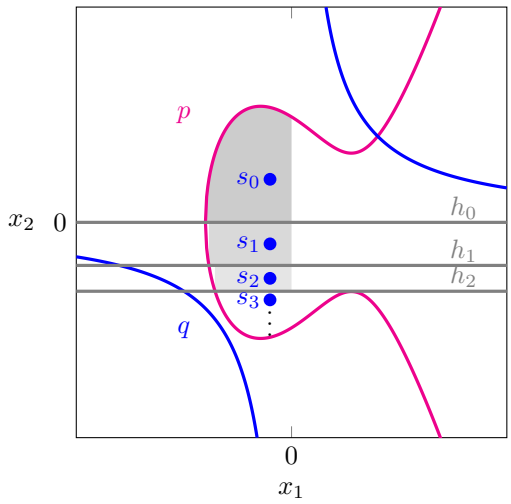


Figure 2.5: Example of non-termination caused by repeated, unrestricted approximation of cell boundaries

For this reason, a number of criteria were introduced to ensure termination. The most basic of these termination criteria simply limits the number of approximated cells.

An apparent downside of approximating cells in the levelwise cell construction is also depicted in the example for non-termination 2.5: If we regularly apply a cell approximation, potentially we have to compute a significantly higher number of total cells compared to the single cell construction without approximation. However, these cells are potentially easier to construct.

Generally, we have to note that the approach of under-approximating cell boundaries in the presented fashion is heuristic in nature, regardless of the kind of artificial polynomials that we use to approximate. There are no guarantees of any kind that the introduced artificial polynomials will actually have a positive effect on the MCSAT algorithm or if they will ultimately lead to complicated resultants being omitted.

It was hypothesized that further approximation approaches that embody the behavior of the approximated polynomial around the sample point more closely could be beneficial to improve the amount of search space that the MCSAT algorithm can consequently discard. Therefore, Promies further investigated other ways to introduce artificial polynomials that take into account additional information about the approximated polynomials.

An attempt was made to use the concept of the *Taylor expansion* to further incorporate the behavior of the polynomial at the sample point into the approximation. Roughly speaking, the nth-order Taylor expansion of a function at a specific point is another function whose value and first n derivatives match the original function at that point. Promies concentrated on the use of the first- and second-order Taylor expansions.

In the executed experimental results of these three variants, the "naïve" box-like approach as well as the first-order Taylor expansion performed significantly better than the levelwise single cell construction without approximation, with the "naïve" approach still beating the first-order Taylor expansion. Meanwhile, the second-order Taylor expansion even performed slightly worse than the single cell construction without approximation.

Promies concluded that the increased effort for calculating more accurate cell approximations in the form of Taylor expansions does not pay off because, on the one hand, additional derivatives need to be computed and, on the other hand, the following resultants are still more complicated than for the "naïve" approach.

Another issue that can potentially affect the quality of the generated cells when using the Taylor expansion is the limited amount of control that we have over the resulting polynomial. This is because it is entirely determined by the original polynomial and its partial derivatives at a single point. Therefore, it is pretty open whether the resulting approximation is actually a good under-approximation and does not immediately cross over the approximated polynomial, leading to small cells.

Promies suggested that one interesting approach that might be worth investigating is to develop a technique for computing piecewise defined linear descriptions of cell boundaries.

Chapter 3

Piecewise Linear Under-Approximation

3.1 Motivation

Promies has shown that the approach of under-approximating the cell boundaries of high-degree polynomials by low-degree ones can increase the efficiency of the levelwise single cell construction, but has also presented the limitations and challenges that come with this approach.

In this thesis, we want to investigate the possibility of using piecewise linear approximations of the cell boundaries instead of low-degree polynomials. While they are more flexible in approximating the cell boundaries and provide more control, their polynomial resultants are also easier to compute compared to the Taylor expansion ones.

3.2 Piecewise Linear Functions

We will first define *continuous piecewise linear functions* in a rigorous way.

Definition 3.2.1. *(Continuous Piecewise Linear Function). A continuous function* $f : \mathbb{R} \longrightarrow \mathbb{R}$ *is called a* (continuous) piecewise (affine) linear function *if there exists* $n \in \mathbb{N}$, *intervals* $\Omega_1, ..., \Omega_n \subseteq \mathbb{R}$ *and* $l_i = m_i x + b_i \in \mathbb{Q}[x]$ *for* $i \in \{1, ..., n\}$ *with* $m_i, b_i \in \mathbb{Q}$ *such that*

- $\bigcup_{i \in \{1,...,n\}} \Omega_i = \mathbb{R}$,
- *for all* $i \in \{1, ..., n-1\}$, $x \in \Omega_i$, $x' \in \Omega_{i+1}$ *it holds that* $x \leq x'$,
- *for all* $i,j \in \{1, ..., n\}$ *with* $i < j$ *it holds that* $|\Omega_i \cap \Omega_j| = 1$ *if and only if* $j = i+1$,
- *by* r_i *we denote the right endpoint of* Ω_i *for* $i \in \{1, ..., n-1\}$ *and*
- *for all* $x \in \mathbb{R}$ *it holds that* $f(x) = l_i(x)$ *where* $i = min\{j \in \mathbb{N} \mid x \in \Omega_j\}$.

In this definition, we basically expect n ordered intervals Ω_i that cover the real numbers such that neighboring intervals always have exactly one common boundary. Furthermore, we expect n linear terms l_i such that they exactly define the given function on their corresponding interval Ω_i. We call a continuous piecewise linear function consisting of n linear segments, n-segmented. Note that for $i \in \{1, ..., n-1\}$, the right endpoint r_i of Ω_i is also the left endpoint of Ω_{i+1}. Technically, we are not really dealing with linear functions here in the sense of linear algebra, as linear functions must fix the origin, but rather with affine linear functions. For the remainder of this thesis, we will simply ignore this difference for ease of language. We will also usually drop the specifier "continuous" because, for reasons of delineability but also for reasons of representability we will only deal with continuous piecewise linear functions in this thesis.

Example 3.2.1.

$$f : \mathbb{R} \longrightarrow \mathbb{R}, x \longrightarrow \begin{cases} 1 & \text{if } x < -1 \\ -x & \text{if } -1 \leq x < 1 \\ -1 & \text{if } x \geq 1 \end{cases}$$

is a continuous piecewise linear function, as it is continuous and the following exists: $\Omega_1 = (-\infty, -1]$, $\Omega_2 = [-1, 1]$, $\Omega_3 = [1, \infty)$ *and* $l_1 = 1$, $l_2 = -x$, $l_3 = -1$, *which fulfill the definition. Here we have* $r_1 = -1$ *and* $r_2 = 1$.

The graph of this piecewise linear function looks as follows:

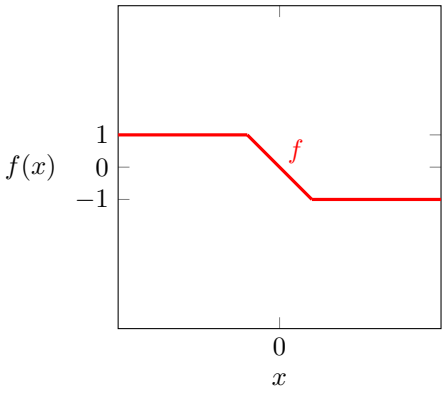


Figure 3.1: Graph of the piecewise linear function f

3.3 Computing Piecewise Linear Under-Approximations

The setup at this point is the following: During the levelwise single cell construction, we want to introduce artificial piecewise linear functions that approximate the upper or lower bound of the cell we are trying to construct. If we are currently constructing the interval for level i in the single cell construction, the piecewise linear function will be computed in terms of the variables x_i and x_{i-1}. Therefore, we must be at least on level 2.

This means that geometrically, our piecewise linear approximations are always only trying to capture the behavior of the approximated polynomial bound according to a two-dimensional cut that lies parallel to the coordinate axes of x_i and x_{i-1}.

In this context, we will call the associated variable x_{i-1} the *primary variable* and x_i the *secondary variable*. This is because the piecewise linear function will be defined in terms of x_{i-1} and is going to be used to bound the values in the variable x_i.

We will now present the process of piecewise linearly approximating a polynomial $p \in \mathbb{Q}[x_1, \ldots, x_n]$ around a sample point $s \in \mathcal{R}^n$.

The main idea is to generate a number of support points that resemble the behavior of the root of the polynomial we are trying to approximate. We will do this by first choosing a number of rational coordinates in the primary variable at which we want to probe the root of the polynomial. To make sure that at the coordinates at which we want to probe the polynomial, the root of the polynomial that we are interested in is available, we initially compute the so-called *delineable interval* given the polynomial around our sample point.

For a single polynomial, we obtain the delineable interval by computing, isolating, and ordering the roots of its discriminant and its coefficients. No resultant computation is necessary at this point because we don't take any other polynomials into account for probing p.

Additionally, it is worth mentioning that the calculation of the delineable interval generally does not incur any additional computational cost at this point, as its components are cached and will be used during McCallum's projection anyway.

Now that this delineable interval is computed, we can choose a number of rational coordinates inside the delineable interval, near the primary variable of the sample point s.

The exact way in which we choose these coordinates is not fixed at this point, and can also be subject to various different heuristics.

Then we isolate the roots of the polynomial at the chosen coordinate and select the specific root we are interested in. For every of the chosen coordinates, this results in a real algebraic number representing the root of the polynomial at that coordinate.

As we want to compute a piecewise linear function based on rational coefficients, we need to rationally approximate the obtained real algebraic number from the side from which we intend to approximate. This whole process finally yields a number of rational support points from which we can span the piecewise linear function.

The pseudocode for this algorithm is given below. For simplicity, we focus on the case where we approximate the upper bound of the cell.

Algorithm 3: approximate_upper_bound_piecewise_linearly

Input: Indexed root expression ir of a polynomial $p \in \mathbb{Q}[x_1, \ldots, x_i]$, Number of piecewise linear segments $n \in \mathbb{N}$, Sample $s = (s_1, ..., s_i) \in \mathcal{R}^i$
Output: Piecewise linear under-approximation for the upper bound or FAIL

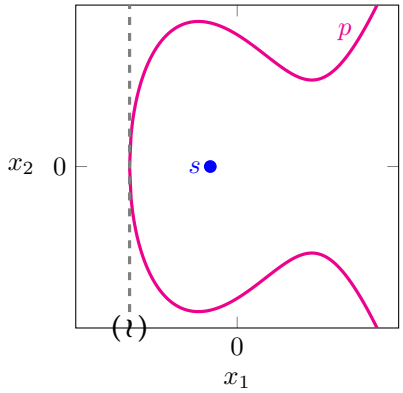
```
1  if level(p) < 2 then
2  |  return FAIL
3  if s_{i-1} ∉ ℚ then
4  |  return FAIL
5  Let delineable_interval be the delineable interval of p with respect to s.
6  if s_{i-1} is one of the bounds of delineable_interval then
7  |  return FAIL
8  Choose n rational coordinates {q_1, ..., q_n} in delineable_interval such that
     q_i ≠ s_{i-1} for all i ∈ {1, ..., n}
9  support_points := ∅
10 for q ∈ {q_1, ..., q_n} do
11 |  modified_sample := (s_1, ..., s_{i-2}, q)
12 |  roots := realRoots(p(modified_sample, x_i))
13 |  root := roots[ir.index]
14 |  Choose v ∈ ℚ such that v is close to root, but v < root
15 |  support_points := {(q, v)} ∪ support_points
16 modified_sample := (s_1, ..., s_{i-2}, s_{i-1})
17 roots := realRoots(p(modified_sample, x_i))
18 root := roots[ir.index]
19 Choose v ∈ ℚ such that v is close to root, but v < root and v > s_i
20 support_points := {(s_{i-1}, v)} ∪ support_points
21 return the piecewise linear function that is induced by support_points
```

Note that here we restrict ourselves by only computing piecewise linear functions that have a support point at the primary coordinate of the sample. We do this because it is a straightforward way to ensure that the piecewise linear function can be used for explanations in MCSAT, since we need to make sure that the explanation excludes the current sample point.

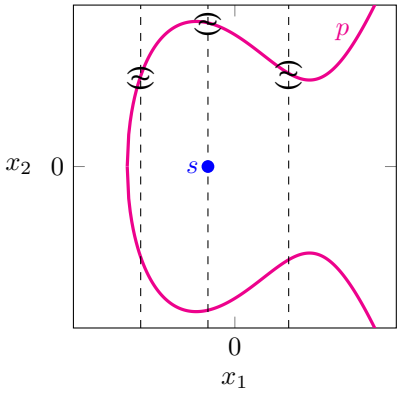
In the context of the algorithm and the piecewise linear function f that we compute, this means that it must be the case that $s_i < f(s_{i-1})$.

It is important to realize that not all piecewise linear functions that correctly exclude the sample point must also have a support point at the primary coordinate of the sample point. Therefore, this restriction is a choice we made that is rather pragmatic in nature.

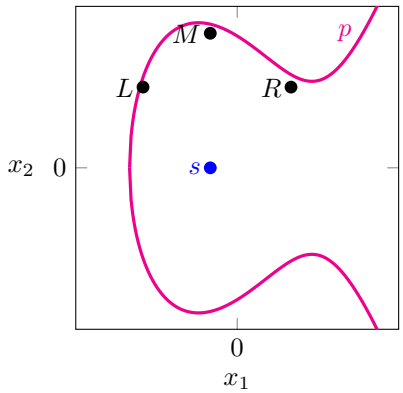
Example 3.3.1. *In this example, we show how the construction of a piecewise linear approximation for the upper bound would go along with the polynomial p in our running example.*



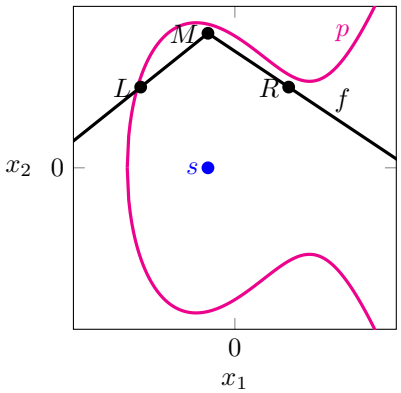
(a) Initially, we compute the delineable interval in x_1 with respect to the sample point s and the given polynomial we want to approximate. Here the delineable interval is $(-2; \infty)$.



(b) Next, we choose rational coordinates for x_1 in the delineable interval at which we isolate the roots of the polynomial, which yields real algebraic numbers that represent these roots in the x_2 coordinate.



(c) We choose rational approximations for these isolated roots from below. Therefore, we created a number of support points that roughly match the shape of the root of the polynomial.



(d) Lastly, we compute the piecewise linear function that is spanned by the given support points.

Figure 3.2: A visual representation of the process of piecewise linearly approximating the root of a polynomial

Fallbacks

As the algorithm reveals, there are multiple possible occasions at which either this restriction causes this approach to fail or where it does not make any sense to execute a piecewise linear approximation.

The first case in which a piecewise linear approximation does not make sense is when we are in the one-dimensional case during the single cell construction.

The second case is when the primary coordinate of the sample point is not rational, therefore causing our restriction to fail to compute a support point directly at the sample.

A third possible case is when there is no space around the sample point in the delineable interval. We will illustrate this possibility using an example.

Example 3.3.2. *Consider the example of the polynomial p from our running example, together with the sample point $s = (-2, 2)$. In this case, the delineable interval of p with respect to the sample point s is $[-2; -2]$ because the sample point lies exactly on the root of the discriminant $disc_{x_2}(p)$.*

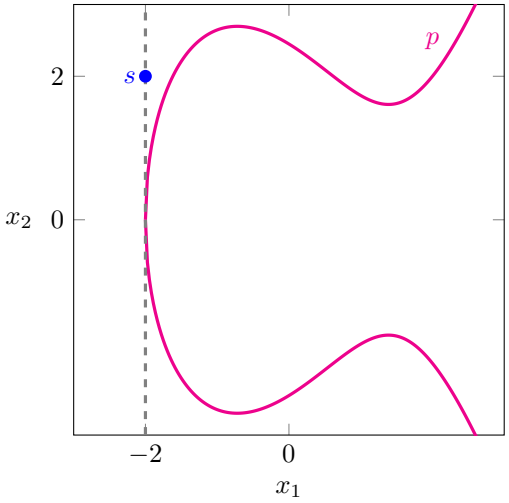


Figure 3.3: The delineable interval of p with respect to s does not have any space around s. Therefore, a piecewise linear approximation does not make sense here.

However, in all of these cases, not all hope is lost for executing an approximation. One option we still have is to simply execute one of the "naïve" approximations by Promies, thereby combining the two approaches.

3.4 Representation of Piecewise Linear Functions for MCSAT

In order to use a piecewise linear approximation as a cell boundary and feed it back into the MCSAT algorithm as part of an explanation, we need to be able to convert it into a logical formula.

It turns out that there is no unique way in which this task can be accomplished. For this thesis, we decided on using a form of representation that is also called a *conjunctive normal expression* for continuous piecewise linear functions. This form of representation has been extensively studied in [XVDBDS14, XvdBDSL15]. Later on, we will give more elaborate reasons for why this representation was deemed attractive for our purposes.

We will now introduce this form of representation formally.

Theorem 3.4.1. *Let f be an n-segment continuous piecewise linear function consisting of linear terms $l_1, ..., l_n$. There exists $k \in \{1, ..., n\}$ and sets $I_1, ..., I_k \subseteq \{l_1, ..., l_n\}$ such that for all $x \in \mathbb{R}$ it holds that*

$$f(x) = \min_{k}(\max_{l \in I_k}(l(x)))$$

For the remainder of this thesis, we will also refer to this as the *min-max representation* of a continuous piecewise linear function.

It is important to note that we can also represent continuous piecewise linear functions in terms of maxima over minima (*max-min representation*). Their construction is completely analogous, which is why we will mostly focus on talking about min-max representations.

In order to prove Theorem 3.4.1 and to show how we can compute such a representation, we will now introduce an important type of set.

Definition 3.4.1. *Given an n-segment piecewise linear function f and its corresponding intervals $\Omega_1, ..., \Omega_n$ and linear segments $l_1, ..., l_n$, we define the sets $I_{\leq,i}$ and $I_{\geq,i}$ as follows:*

$$I_{\leq,i} := \{l \in \{l_1, ..., l_n\} \mid \forall x \in \Omega_i : l(x) \leq l_i(x)\}$$

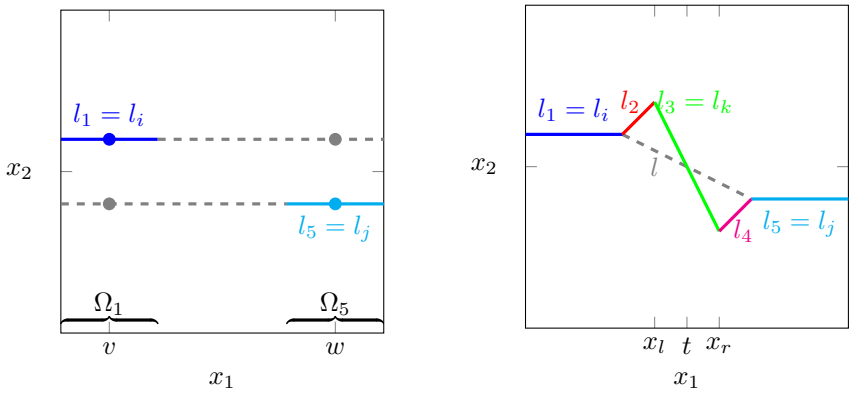
$$I_{\geq,i} := \{l \in \{l_1, ..., l_n\} \mid \forall x \in \Omega_i : l(x) \geq l_i(x)\}$$

Before we can prove Theorem 3.4.1, we will first introduce and prove a lemma that guarantees the existence of certain delimiting linear segments.

Lemma 3.4.2. *Given an n-segment continuous piecewsie linear function f with linear segments $l_1, ..., l_n$ and intervals $\Omega_1, ..., \Omega_n$. For every pair $l_i = m_i x + b_i, l_j = m_j x + b_j \in \{l_1, ..., l_n\}$ there exists some $l_k \in \{l_1, ..., l_n\}$ such that $l_k \in I_{\geq,i}$ and $l_k \in I_{\leq,j}$.*

Proof. If we have $l_i \in I_{\leq,j}$ we are immediately done, as $l_i \in I_{\geq,i}$. Symmetrically if we have $l_j \in I_{\geq,i}$ we are also immediately done, as $l_j \in I_{\leq,j}$. So we will consider the case where $l_i \notin I_{\leq,j}$ and $l_j \notin I_{\geq,i}$.
If i and j were direct neighbors, one of the first two conditions must directly hold for continuous piecewise linear functions, as l_i and l_j intersect at their common boundary point of Ω_i and Ω_j.
Furthermore we can assume without loss of generality that $j > i + 1$ as the proof for $i > j + 1$ is analogous.
Given the case we consider where $l_i \notin I_{\leq,j}$ and $l_j \notin I_{\geq,i}$ there must exist witnesses $v \in \Omega_i$ and $w \in \Omega_j$ such that $l_i(v) > l_j(v)$ and $l_i(w) > l_j(w)$.
It follows that if l_i and l_j have a point of intersection, its x-coordinate cannot lay in Ω_k for any $k \in \{i+1, .., j-1\}$, because their order does not change between v and w.
We now define the linear function $l = mx + b$ connecting the right endpoint r_i of Ω_i at $l_i(r_i)$ with the left endpoint r_{j-1} of Ω_j at $l_j(r_{j-1})$.
This means that $l(r_i) = l_i(r_i)$ and $l(r_{j-1}) = l_j(r_{j-1})$.
It must hold now that $l(r_i) = l_i(r_i) > l_j(r_i)$ and $l(r_{j-1}) = l(r_{j-1}) < l_i(r_{j-1})$ because l_i lies above l_j between v and w.
As we now know, where the intersections between l_i, l_j and l lay, we can deduce

that $m < m_i$ and $m < m_j$. Since the algebraic manipulations for showing this are straightforward we will omit them at this point.
Because the original piecewise linear function f is continuous and its linear segments connect $l(r_i)$ and $l(r_{j-1})$, there must now exist $l_k = m_k x + b_k \in \{l_{i+1}, ..., l_{j-1}\}$ such that $l_k(t) = l(t)$ at some $t \in \Omega_k$ and $x_l, x_r \in \Omega_k$ such that $x_l < x_r$, $x_l \leq t \leq x_r$, $l_k(x_l) \geq l(x_l)$ and $l_k(x_r) \leq l(x_r)$.
It follows that $m_k \leq m$ and that $l_k \in I_{\geq,i}$ and $l_k \in I_{\leq,j}$. □



(a) Example showing the ordering of the linear segments that is induced by $l_i \notin I_{\leq,j}$ and $l_j \notin I_{\geq,i}$

(b) Extension of the previous example by linear segments between l_1 and l_5 as well as the artificial connecting line l

Figure 3.4: Visualized proof idea for Lemma 3.4.2 using an example

Now we can use Lemma 3.4.2 to prove Theorem 3.4.1.

Proof. We claim that given an n-segment piecewise linear function the sets $I_{\leq,1}, ..., I_{\leq,n}$ suffice to fulfill the min-max representation.
First of all we prove that for all $i \in \{1, ..., n\}$ and $t \in \Omega_i$ it holds that $l_i(t) = \max_{l \in I_{\leq,i}}(l(t))$. It must be true that for all $t \in \mathbb{R}$ it holds that $l_i(r) \leq \max_{l_j \in I_{\leq,i}}(l_j(r))$ because $l_i \in I_{\leq,i}$. Let us now assume that for some $t \in \mathbb{R}$ it is the case that $l_i(t) < \max_{l_j \in I_{\leq,i}}(l_j(t))$. Then there must exist an $l \in I_{\leq,i}$ such that $l_i(t) < l(t)$. But this contradicts the definition of $I_{\leq,i}$.

Now we will assume that there exists some $t \in \mathbb{R}$ such that
$f(t) \neq \min_k(\max_{l \in I_{\leq,i}}(l(t)))$.
It holds that $t \in \Omega_i$ for some Ω_i. Therefore, it follows that $f(t) = l_i(t) = \max_{l \in I_{\leq,i}}(l(t))$ as we have just shown. It must be true that $f(t) = l_i(t) > \min_k(\max_{l \in I_{\leq,k}}(l(t)))$ because the previous maximum is one of the terms that the minimum includes.
Now there exist some j such that $l_i(t) > \max_{l \in I_{\leq,j}}(l(t))$
But from Lemma 3.4.2 we know that there exists some l_k such that $l_k \in I_{\geq,i}$ and $l_k \in I_{\leq,j}$
For that reason, $\max_{l \in I_{\leq,j}}(l(t)) \geq l_k(t)$ and $l_k(t) \geq l_i(t)$
We now obtain $l_i(t) > \max_{l \in I_{\leq,j}}(l(t)) \geq l_k(t) \geq l_i(t)$. ↯
Thus it must hold that for all $t \in \mathbb{R}$, $f(t) = \min_k(\max_{l_j \in I_{\leq,i}}(l_j(t)))$ □

Example 3.4.1. *The sets $I_{\leq,1}$, $I_{\leq,2}$ and $I_{\leq,3}$ look as follows for the piecewise linear function from Example 3.2.1:*

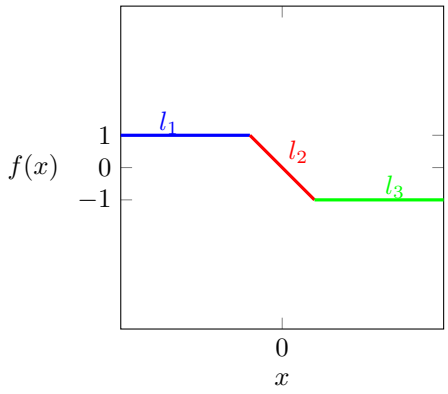


Figure 3.5: The piecewise linear function from Example 3.2.1 decomposed into its linear segments.

$$I_{\leq,1} = \{l_1, l_3\}, \quad I_{\leq,2} = \{l_2, l_3\}, \quad I_{\leq,3} = \{l_2, l_3\}$$

Which results in this representation:

$$\min\big(\max(1,-1), \max(-x,-1), \max(-x,-1)\big)$$

This can be simplified to

$$\min\big(1, \max(-x,-1)\big)$$

As we can observe, there can be a lot of redundancy in the sets $I_{\leq,i}$.

In [XVDBDS14, XvdBDSL15] the possibility of finding irredundant or even minimal conjunctive normal expressions has been investigated. For our purposes, however, we decided that we are okay with having redundancy in our min-max representation because computing an irredundant or even minimal representation of this form is much more involved. In the case of the minimal representation, the method presented in [XvdBDSL15] even requires an amount of time that grows exponentially in the number of segments of the piecewise linear function.

From the proof for Theorem 3.4.1 we can also derive another possibility to construct the sets $I_1, ..., I_n$:

$$I_j := \{l_j\} \cup \{l_i \in \{l_1, ..., l_n\} \setminus \{l_j\} \mid l_j \notin I_{\geq,i},\ l_i \in I_{\leq,j}\}$$
$$\cup\, \{l_k \mid l_i \in \{l_1, ..., l_n\} \setminus \{l_j\},\ l_j \notin I_{\geq,i},\ l_i \in I_{\leq,j},\ l_k \text{ as in Lemma 3.4.2 for } l_i \text{ and } l_j\}$$

The rationale behind this construction is to have $l_j \in I_j$ and $I_j \subseteq I_{\leq,j}$. From this directly follows the correctness of the first part of the proof. Additionally, we only add those l_k to the set I_j that the second part of the proof depends on. As we have already indicated, it holds that $I_j \subseteq I_{\leq,j}$, where in practice this subset relation is usually strict. For this reason, we will refer to this representation as the "advanced" representation in contrast to the former "simple" one. Because of this subset relation, one might think that this representation will always be at least as small as the one

using the sets $I_{\leq,i}$. It turns out, though, that this is only true before factoring out identical sets. This means that these two representations are actually incomparable after factoring out identical sets.

Example 3.4.2. *The resulting sets from the second construction, for Example 3.1 look as follows:*

$$I_{\leq,1} = \{l_1\}, \quad I_{\leq,2} = \{l_2, l_3\}, \quad I_{\leq,3} = \{l_2, l_3\}$$

They result in this representation:

$$\min\left(\max(1), \max(-x, -1), \max(-x, -1)\right)$$

We will now shortly present why Theorem 3.4.1 only holds for continuous piecewise linear functions.

Example 3.4.3. *Whichever sets I_1 and I_2 we choose for the depicted example, the expression of the minimum over maxima will always deteriorate into either 1 or -1 which does not equal the desired function.*

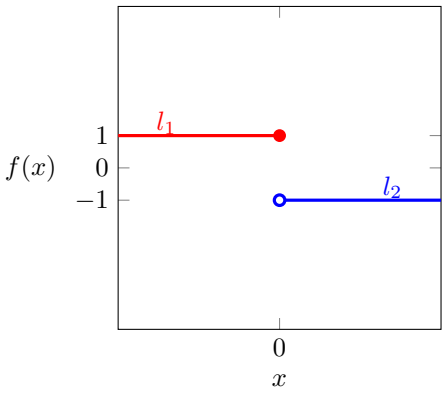


Figure 3.6: Non-continuous 2-segment piecewise linear function with $l_1 = 1$ and $l_2 = -1$ and a discontinuity at $x = 0$

At this point, we will shed some light on the reasons why we decided to use the presented form of representation for piecewise linear functions.

The first reason for this is that the min-max representation is advantageous for the conversion into a logical formula for the MCSAT algorithm, as it can be canonically converted. Consider the case where we want to use piecewise linear functions to approximate the upper bound of a cell. We can simply formulate a bound on the secondary variable from above by using the piecewise linear function at hand. This means that given a piecewise linear function f and its min-max representation $f = \min_k\left(\max_{l\in I_k}(l(x))\right)$ with sets $I_1, ..., I_n$, a primary variable x and a secondary variable y, we can formulate that

$$y < f = \min_k\left(\max_{l\in I_k}(l(x))\right)$$

From which we can derive

$$\bigwedge_k\left(y < \max_{l\in I_k}(l(x))\right) \equiv \bigwedge_k \bigvee_{l\in I_k}(y < l(x))$$

The second reason for using the min-max representation is that it behaves fairly nicely when the region we want to exclude is convex. This is because if the region is convex, all of the sets $I_{\leq,i}$ will only contain l_i. Therefore, the resulting min-max representation is simply the minimum over all linear segments, which is consistent with the region that we want to exclude. The resulting logical formula is then simply the conjunction over the constraints $y < l_i$. Because the MCSAT algorithm wants to exclude the described region, it actually applies a negation to the entire conjunction. Note that after applying the negation, the resulting formula is a clause, which can be easily recorded by MCSAT.

3.5 Implementation

SMT-RAT is an ongoing research project of the Theory of Hybrid Systems research group at RWTH Aachen University. The acronym SMT-RAT stands for satisfiability-modulo-theories real arithmetic toolbox. The project revolves around the implementation of the open source C++ toolbox for strategic and parallel SMT solving, which is called SMT-RAT as well [SMTb].

For this thesis, we implemented two concrete approaches for piecewise linear under-approximations into the SMT-RAT solver. They are called the static approach and the refinement-based approach. These two approaches can be fine-tuned and adapted using various settings.

For both of these approaches, we initially compute the delineable interval for the polynomial we intend to approximate with respect to our sample point to determine the interval in which it is possible to probe the polynomial.

In the static probing variant, we determine a fixed, predefined number of rational coordinates that are evenly distributed inside the computed delineable interval. Since it is possible that the delineable is unbounded in one or even both directions, we use a fallback distance, which is measured from the sample point, to artificially restrict these unbounded intervals. This fallback distance can be configured in the approximation settings beforehand.

The second implemented approach is called the refinement-based approach and builds upon the static approach. In the refinement-based approach, we first apply the static approach and then preemptively calculate the resultants between the given linear segments and the approximated polynomial and determine if these have any relevant roots. If this is the case, we choose a new rational coordinate in the middle of the two endpoints of the original linear segment, at which we again probe the polynomial. This new support point is then incorporated into the piecewise linear function, thereby refining the original approximation. This process is then recursively repeated up to a predefined depth, if necessary. We should note that the preemptive calculation of the resultants by itself does not incur an additional computational cost on its own at this point, as the resultant and its root are again cached. In a way, we only move up this process in order to gain an information advantage. Only if we decide to actually refine the piecewise linear function at this point by introducing another support point will this resultant calculation act as an additional cost.

We will now show how the construction of min-max representations can be implemented. Specifically, we will present, given an n-segment piecewise linear function f containing two linear segments l_i and l_j together with their respective intervals Ω_i and Ω_j, how to compute whether $l_i \in I_{\leq,j}$.

Algorithm 4: is_under

Input: Two linear segments $l_i = m_i x + b_i$, $l_j = m_j x + b_j \in \mathbb{Q}[x]$ of an n-segment piecewise linear function f together with their respective intervals Ω_i and Ω_j

Output: Whether $l_i \in I_{\leq,j}$

```
1 if i = j then
2 |  return true
  /* Here we denote by r_i the right endpoint of the
     interval Ω_i of f for i ∈ {1,...,n−1} as in Definition
     3.2.1                                                */
3 if j = 1 then
4 |  return l_i(r_1) ≤ l_j(r_1) and m_i ≥ m_j
5 else if j = n then
6 |  return l_i(r_{n−1}) ≤ l_j(r_{n−1}) and m_i ≤ m_j
7 else
8 |  return l_i(r_{j−1}) ≤ l_j(r_{j−1}) and l_i(r_j) ≤ l_j(r_j)
```

This algorithm is based on evaluating the linear segments at the two endpoints of Ω_j if possible and otherwise additionally using information about the slope of the segments. Its correctness is directly given by the fact that two non-identical affine linear functions have at most one point of intersection.

We recall at this point that we proposed two different possible ways of computing the index sets that satisfy the min-max representation from Theorem 3.4.1. Both of these were implemented into the SMT-RAT solver.

3.6 Adapted McCallum Projection Operator

One thing that should be addressed at this point is that if we want to introduce artificial piecewise linear functions, we cannot use the default variant of McCallum's projection operator anymore. This is because McCallum's projection operator is only defined for polynomials.

Furthermore, if we have a piecewise linear function that we want to use during the levelwise single cell construction, it obviously does not suffice to simply deconstruct it into its linear segments and treat them as individual polynomials, because this would not yield the cell that we intended when using the piecewise linear bound. What we need to do instead is to treat the piecewise linear function as its own object and compute the necessary resultants according to the biggest cell heuristic for each linear segment.

During this process, it is necessary to apply filters to the roots of the computed resultants to capture the intended meaning of the piecewise linear function. We will now depict the idea for this adapted projection operator.

Example 3.6.1. *Consider the piecewise linear function f that was used to describe how to probe a polynomial to compute a piecewise linear approximation. If we use the adapted version of McCallum's projection operator, the computed resultants $res_{x_2}(p, l_1)$ and $res_{x_2}(p, l_2)$ have roots that do not correspond to common roots of f and p because they are outside the interval for which the linear segment defines the piecewise linear function f. Therefore, we need to filter out these roots.*

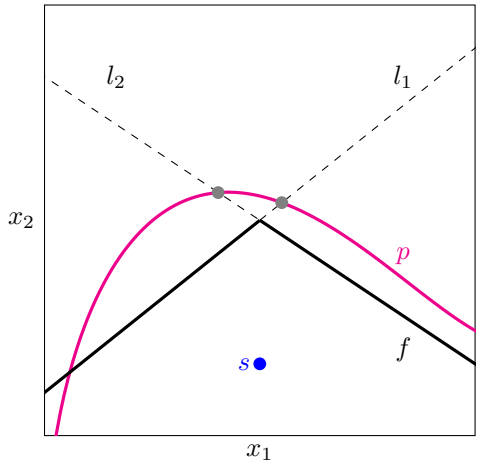


Figure 3.7: In the adapted version of McCallum's projection operator, $\text{res}_{x_2}(p, l_1)$ and $\text{res}_{x_2}(p, l_2)$ have unnecessary roots that need to be filtered out because they do not capture the intended meaning of the piecewise linear function f.

As the exact details of the implementation of the adapted McCallum's projection operator are out of scope for this thesis, we will not further elaborate on them at this point.

It suffices to mention that an efficient version of this adapted projection operator has been implemented by Nalbach in the SMT-RAT solver, which utilizes the knowledge of the intervals on which the individual linear segments are defined to efficiently filter out unnecessary roots of the resultants.

Chapter 4

Experimental Results

In this chapter, we will present the experimental results that were obtained while comparing different variants of our implemented approaches for piecewise linear cell approximations with the default single cell construction without approximation.

We used the SMT-LIB QF_NRA dataset [SMTa] to evaluate the performance of the different solver variants. At the time of writing, it consists of 12134 different QFNRA problems that are specified in the SMT-LIBv2 language.

The problems originate from a wide range of applications. Some of them are categorized as industrial problems, stemming from the qualitative analysis of systems of ordinary differential equations in the natural sciences; others are generated as proof obligations by frameworks for program analysis; again, others are families of crafted problems like the Kissing family, which aim to investigate a special case of sphere packing.

While there is not really an objective way to determine important or representative QFNRA problems, the SMT-LIB benchmark sets try to cover interesting, practical problems, some of which should be challenging to solve by modern SMT solvers.

Of the 12134 benchmark problems, 5248 are labeled as satisfiable, 5534 are labeled as unsatisfiable, and 1352 are labeled as unknown, meaning that it could not be determined yet whether they are satisfiable or unsatisfiable.

All the benchmarks were executed on the high-performance computing cluster of the RWTH Aachen University using the Benchmax benchmarking tool, which is bundled into the SMT-RAT solver.

The timeout was set to 60 seconds and the memory limit to four gigabytes. These cut-offs are somewhat arbitrarily chosen but fall into a favorable area in which it is still very feasible to run extensive benchmarks of the entire dataset as well as cover an overwhelming majority of the problems where we can actually decide the satisfiability of the given problem. The cluster nodes on which the benchmarks were executed are equipped with two Intel® Xeon® Platinum 8160 processors, each having 24 cores and a processor base frequency of 2.1 GHz.

Promies equipped the approximation framework inside the SMT-RAT solver with a number of different settings to configure when and how an approximation will be used during the single cell construction. We want to mention just a few of these settings here, on which we focused while evaluating the performance of our implementation. The first setting that we used was the maximum number of total approximations that were applied. Secondly, we also utilized the threshold on the degree

of polynomials, which suffices to apply an approximation. The idea is that for polynomials of high degree, applying an approximation is more useful because they also cause polynomial resultants of higher degree. However, one has to be careful at what value this threshold is set. If it is set too high, the number of problems to which any approximation is applied gets very small. On the other hand, if the threshold is set too low, potentially too many approximations will be applied to problems that could normally also be solved without applying an approximation, therefore leading to unnecessary overhead or even to the solver not being able to solve them in the set time limit.

The Baseline

We started by benchmarking the SMT-RAT solver with the levelwise single cell construction without approximations using the biggest cell heuristic. This solver will act as the baseline for the following comparisons, and we will abbreviate it as "LW".

The benchmark results showed that 10010 of the 12134 problems could be solved by the levelwise single cell construction without approximations. More interestingly, we observed that 8748 of the problems could be solved in less than one second. This reveals that a significant portion of the benchmark set consists of rather "easy" problems.

To get an impression of the reliability of the benchmarking results, we wanted to see how much the runtime for the instances differed between different runs. Therefore, we executed another benchmark of the levelwise single cell construction without approximations, using the exact same build as before.

The results of the two benchmark runs on the identical build showed that even though for about 70 of the problems the total runtime in the two different benchmarks differed by more than a second, overall, for the overwhelming majority of the problems, this difference was below 200 milliseconds. In Figure 4.1 the runtime distribution for the two benchmark runs on the identical build can be observed.

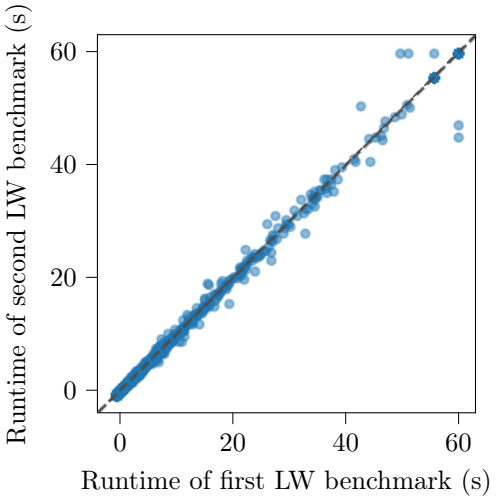


Figure 4.1: Scatter plot showing the runtime comparison between the first and second benchmark run of the identical LW build

More importantly, both benchmark runs are able to solve the same number of problems. Both benchmark runs differed in their results for four problems, with the first run solving two problems that the second couldn't, and vice versa. This effect can be explained by problems that the solver is able to solve in a time frame that is close to the time limit. In this case, small deviations in the solver's speed between different runs can cause different runs on the same build to yield slightly different results.

This observation lets us conclude that, overall, the benchmark results between different runs are relatively comparable. For this reason, we are going to execute every benchmark in the following comparisons only once.

We decided to use the following approximation settings for the upcoming benchmarks: Approximate at most 50 cells, approximate at most twice per polynomial, and approximate only polynomials that have at least a degree of five. Furthermore, we decided to use the advanced way of representing the piecewise linear functions as logical formulae for MCSAT and also defined the fallback distance for probing polynomials in the case of an unbounded delineable interval to be equal to one.

While these decisions for the approximation settings are relatively arbitrary, we hope they are in a suitable area to work out the differences between the different approaches.

Different Number of Linear Segments

We now executed three benchmarks using piecewise linear approximations using the static approach and a different number of linear segments. It is important to note that in the case of fallbacks, we do not apply any of Promies' "naïve" approximations here, as we are purely interested in the effect that our piecewise linear approximations have on the benchmarks.

The first benchmark was executed using 2-segmented piecewise linear approximations, the second using 4-segmented piecewise linear approximations, and the third using 6-segmented piecewise linear approximations. While we could have also compared using a different number of segments, we want to explore whether a change in the number of segments has a large effect on the benchmarks and if a general trend is visible.

Solver	solved	SAT	UNSAT	timeout	memout	segfault	gained	lost
LW	10010	5049	4961	2083	40	(1)	-	-
2-segment	10055	5081	4974	2039	38	(2)	111	66
4-segment	10068	5095	4973	2027	37	(2)	123	65
6-segment	10075	5102	4973	2020	39	0	133	68

Figure 4.2: Comparison between variants of piecewise linearly approximating solvers using different numbers of linear segments and the baseline solver, which uses the levelwise single cell construction without approximation (**LW**). The columns "gained" and "lost" indicate the number of problems the respective solver solved that the baseline solver could not, and vice versa.

Given these results, we mainly observe three effects. Firstly, we see that using piecewise linear approximations does actually have a positive effect on the performance of the SMT-RAT solver. Secondly, we observe that if we use a higher number

of segments, the number of solved instances increases slightly. We presume that this effect can be attributed to the fact that by using more segments, we can better approximate the bound of the polynomial, therefore increasing the likelihood of producing larger cells. Lastly, we can see that the increase in solved problems when using piecewise linear approximations cannot simply be attributed to the solver solving purely more instances. While the piecewise linearly approximating solvers gain many problems that they were able to solve compared to the levelwise single cell construction without approximation, they also lose many problems that could have been solved before. Because this is an effect that we generally observe when making changes to the SMT-RAT solver, we assume one should not overinterpret this effect, as it is most likely caused by the nature of the algorithm at hand: Whenever the solver decides to take a different path during its search for a solution to the problem compared to before, there is always the potential that the decision is better or worse just by chance.

Another thing that is notable is the occurrence of segmentation faults. Usually, the presence of segmentation faults indicates bugs in the solver. In this case, we are certain that these segmentation faults do not correspond to bugs in the solver but are actually caused by the benchmarking tool for technical reasons in rare cases where the actual result should have been a memout instead. We conclude this by the fact that for none of the indicated segmentation faults we could reproduce the problem, even when using the exact build used for benchmarking, and by the fact that for all of these segmentation faults the recorded peak memory usage lies upwards of the specified four gigabytes. For these reasons, we will denote these segmentation faults in parentheses. The reader should keep in mind that these segmentation faults should most likely be attributed to the memout column.

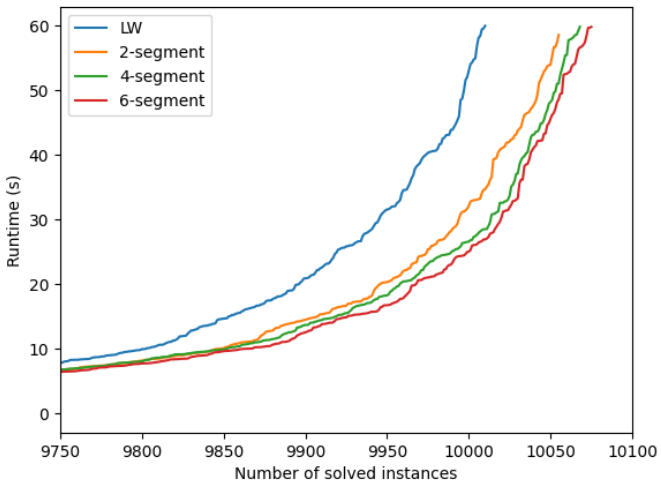


Figure 4.3: Performance profile showing the impact of using piecewise linearly approximating solvers with different numbers of segments compared to the levelwise single cell construction without approximation (**LW**).

Different Representations for Piecewise Linear Functions

Next, we want to investigate how much the two different ways of representing the piecewise linear functions as logical formulae affect the performance of the solver. For this, we compare two solvers using 6-segmented piecewise linear approximations using the same settings as before, with the only difference that one is using the simple and the other is using the advanced form of representing piecewise linear functions as logical formulae for MCSAT. For comparison, we will again provide the solver using the levelwise single cell construction without approximation as a baseline.

Solver	solved	SAT	UNSAT	timeout	memout	segfault	gained	lost
LW	10010	5049	4961	2083	40	(1)	-	-
6-segment simple representation	10069	5094	4975	2026	35	(4)	125	66
6-segment advanced representation	10075	5102	4973	2020	39	0	133	68

Figure 4.4: Comparison between two different 6-segmented piecewise linearly approximating solvers equipped with different ways of representing piecewise linear functions and the levelwise single cell construction without approximation (**LW**).

As we can see, the advanced representation shows a small edge over the simple representation. When discussing the two different representations for piecewise linear functions, we claimed that they are in fact incomparable after factoring out identical sets. Because this factoring out of identical sets is in fact not implemented yet in the SMT-RAT solver, the observation that the advanced representation has a small edge over the simple representation seems logical, given that its index sets are always subsets of the simple representation ones.

Static and Refinement-Based Approach

Now we are going to compare the two different implemented approaches to piecewise linear approximation. To do this, we will compare the benchmarking results of the solver using four linear segments with the static probing approach against the solver using four linear segments with the refinement-based approach. Therefore, the solver using the refinement-based approach starts out by doing exactly the same thing as the static probing variant, but then it preemptively computes the resultants of the linear segments and the polynomial it is trying to approximate, refining the piecewise linear function if necessary. Hence, the refinement-based solver may end up with a maximum number of eight segments if it refines every segment once.

Solver	solved	SAT	UNSAT	timeout	memout	segfault	gained	lost
LW	10010	5049	4961	2083	40	(1)	-	-
4-segment static	10068	5095	4973	2027	37	(2)	123	65
4-segment refinement	10071	5098	4973	2024	39	0	124	63

Figure 4.5: Comparison between two different 4-segmented piecewise linearly approximating solvers of the two different approaches and the levelwise single cell construction without approximation (**LW**).

In this comparison, the refinement-based approach has only a very slight edge over the static approach. This indicates that precomputing and refining the segments is only rarely advantageous in the way that we implemented them.

Addressing Fallbacks via "Naïve" Approximations

Another aspect we addressed was the presence of fallbacks during the process of piecewise linearly approximating the polynomials. We mentioned that in these cases, we still have the option to execute one of Promies' "naïve" approximations. Therefore, we want to investigate whether additionally using Promies' "naïve" approximations in cases of fallbacks can further improve the benchmark performance of our approach. For this we benchmarked the performance of a solver primarily using the 6-segmented piecewise linear approximation applying a "naïve" approximation whenever piecewise linear approximations do not make sense. We also compared these results with the performance of a solver that only used "naïve" approximations.

Solver	solved	SAT	UNSAT	timeout	memout	segfault	gained	lost
LW	10010	5049	4961	2083	40	(1)	-	-
6-segment	10075	5102	4973	2020	39	0	133	68
6-segment + "naïve"	10081	5099	4982	2014	36	(3)	147	76
"naïve"	10108	5113	4995	1986	40	0	153	55

Figure 4.6: Comparison between the 6-segmented piecewise linearly approximating solver, a combination of the 6-segmented piecewise linearly approximating solver using Promies' "naïve" approximation during fallbacks, a solver only using Promies' "naïve" approximations, and the levelwise single cell construction without approximation (**LW**).

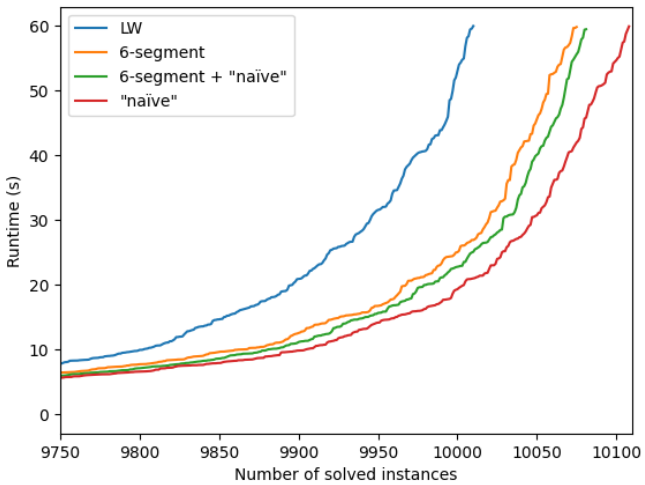


Figure 4.7: Performance profile showing the difference between the solvers from Figure 4.6.

We observe that combining piecewise linear approximations with "naïve" approximations during fallbacks further improves the performance. However, we have to acknowledge the fact that the solver only using "naïve" approximations not only outperforms the regular 6-segmented piecewise linear approximations but also the

combination of them with the "naïve" approximations during fallbacks.

We suspect that in general, the "naïve" approximation approach is much better suited for the QF_NRA benchmark set because, for many of the problems, it suffices to compute a simple approximation. Piecewise linear approximations, on the other hand, are more costly to compute and potentially only pay off for more complicated problems.

Chapter 5

Conclusion

5.1 Future Work

Further Approaches for Piecewise Linear Approximations

The first aspect we consider worthy of further research is the development of different approaches for computing piecewise linear approximations. For example, one could develop a method that works differently than the two presented approaches and does not start out by evenly distributing support points, but rather works from the sample point outward. Potentially, such an approach can also implement a heuristic capable of removing support points that it decides are unnecessary later on. As we have discussed while presenting the refinement-based approach, the precomputation of the resultants between the linear segments of the piecewise linear function and the polynomial that is approximated does not incur an additional computational cost as long as we don't change the piecewise linear function anymore. Therefore, we presume that it is a promising aspect to investigate how to take further advantage of this fact.

Simple Rational Representations

A second aspect we think is important to investigate is the effect of the approximation methods for real algebraic numbers and the rational representations we use for the computed support points from which we span the piecewise linear functions. As our goal from the beginning was to find approximations that better resemble the behavior of the approximated polynomial, one has to investigate how to reconcile the two aspects of finding piecewise linear functions that represent a good approximation of the approximated polynomial but at the same time have simple representations so that they don't introduce complicated piecewise linear functions into the MCSAT algorithm.

Irredundant or Minimal Representation for Piecewise Linear Functions

We were surprised by the fact that using the advanced min-max representation had a considerable positive effect on the piecewise linear approximation. Thus, we believe that investigating the effects of irredundant or even minimal min-max representations,

as presented in [XVDBDS14, XvdBDSL15], could be worth researching. We would like to note at this point that even though the method presented in [XvdBDSL15] requires an amount of time that grows exponentially in the size of the piecewise linear function since it requires computing a covering, it is known that reasonable approximation algorithms exist for this problem [Vaz01].

Identifying Problems for Which More Costly Approximations Pay Off

As we have seen, the "naïve" approximation approach by Promies' is generally better suited for the QF_NRA benchmark dataset than the piecewise linear approximations. This is why we believe that future work should be done in trying to identify QFNRA problems for which more costly approximations like piecewise linear ones pay off. A staggered approach could then be developed, applying piecewise linear approximations when it is indicated that they are better suited than "naïve" approximations.

5.2 Summary

We began this thesis by motivating formal verification and introduced SMT solving for quantifier free non-linear arithmetic. In the preliminaries, we formally defined the problem at hand, presented the necessary groundwork, and recapped Promies' ideas and findings on under-approximating polynomial bounds using polynomials of low degree. We continued by giving a definition for piecewise linear functions, and we showed the general idea of how one can use probing to compute a piecewise linear approximation for a polynomial bound. Then we discussed how it is possible to feed back representations of piecewise linear functions into the MCSAT algorithm and proved the existence of such representations for continuous piecewise linear functions. Furthermore, we proposed two different variants of the general idea that we implemented into the SMT-RAT solver. Lastly, we evaluated our implementation using SMT-LIB's QF_NRA benchmarking dataset, which revealed that it outperforms the levelwise single cell construction, which does not use approximations.

Bibliography

[Á14] Erika Ábrahám. Lecture on satisfiability checking, winter semester 2014/2015. https://ths.rwth-aachen.de/wp-content/uploads/sites/4/teaching/vorlesung_satchecking/ws14_15/09c_cad_handout.pdf, 2014.

[BDE+16] Russell Bradford, James H. Davenport, Matthew England, Scott McCallum, and David Wilson. Truth table invariant cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 76:1–35, 2016.

[Col75] George E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In H. Brakhage, editor, *Automata Theory and Formal Languages*, pages 134–183, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg.

[dMJ13] Leonardo de Moura and Dejan Jovanović. A model-constructing satisfiability calculus. In Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 1–12, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Duc00] Lionel Ducos. Optimizations of the subresultant algorithm. *Journal of Pure and Applied Algebra*, 145(2):149–163, 2000.

[JBdM13] Dejan Jovanović, Clark Barrett, and Leonardo de Moura. The design and implementation of the model constructing satisfiability calculus. In *2013 Formal Methods in Computer-Aided Design*, pages 173–180. IEEE, 2013.

[JdM12] Dejan Jovanović and Leonardo de Moura. Solving non-linear arithmetic. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning*, pages 339–354, Berlin, Heidelberg, 2012. Springer.

[McC85] Scott McCallum. An improved projection operation for cylindrical algebraic decomposition. In Bob F. Caviness, editor, *EUROCAL '85*, pages 277–278, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

[NAS+22] Jasper Nalbach, Erika Ábrahám, Philippe Specht, Christopher W. Brown, James H. Davenport, and Matthew England. Levelwise construction of a single cylindrical algebraic cell, 2022.

[Pro22] Valentin Promies. Underapproximating cell bounds in mcsat using low-degree polynomials. Master's thesis, RWTH Aachen University, 2022.

[SMTa] SMT-LIB benchmarks in QF_NRA logic. `https://clc-gitlab.cs.uiowa.edu:2443/SMT-LIB-benchmarks/QF_NRA`.

[SMTb] SMT-RAT, a toolbox for strategic and parallel satisfiability modulo theories solving. `https://github.com/ths-rwth/smtrat`.

[Vaz01] Vijay V Vazirani. *Approximation algorithms*, volume 1. Springer, 2001.

[XVDBDS14] Jun Xu, Ton JJ Van Den Boom, and Bart De Schutter. Irredundant lattice piecewise affine representations and their applications in explicit model predictive control. In *53rd IEEE Conference on Decision and Control*, pages 4416–4421. IEEE, 2014.

[XvdBDSL15] Jun Xu, Ton JJ van den Boom, Bart De Schutter, and Xiong-Lin Luo. Minimal conjunctive normal expression of continuous piecewise affine functions. *IEEE Transactions on Automatic Control*, 61(5):1340–1345, 2015.