hybr2d Theory of Hybrid Systems Informatik 2 | RWTH AACHEN UNIVERSITY

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

MASTER OF SCIENCE THESIS

UNDERAPPROXIMATING CELL BOUNDS IN MCSAT USING LOW-DEGREE POLYNOMIALS

Valentin Promies

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

Additional Advisor:
Jasper Nalbach

Aachen, 9th August 2022

Abstract

The *model constructing satisfiability calculus (MCSAT)* is an abstract decision procedure for determining the satisfiability of first-order logic formulas with respect to some theory. This process is known as *satisfiability modulo theories (SMT)* solving. In particular, MCSAT can be applied to the theory of *quantifier free non-linear real arithmetic (QFNRA)*, whose formulas are Boolean combinations of polynomial constraints. A crucial step of the algorithm is the construction of a single cell around a given point in the multidimensional real space so that a given set of polynomials is sign-invariant on that cell.

In this thesis, we modify a recently proposed method for single cell construction, showing how artificial low-degree polynomials can be introduced during the construction to speed up the computation of resultants, with the drawback of underapproximating the cell. Experimental results suggest that our approach can outperform the original method in the context of MCSAT.

Contents

Chapter 1

Introduction

Over the last decades, computer programs have progressively found their way into numerous aspects of our everyday lives and they continue to do so ever more rapidly. And there is good reason for that, as software can not only take over simple, repetitive tasks, but it can also efficiently solve problems that would be too complex or too laborious to solve manually. With significant advances in the field of artificial intelligence, even tasks previously deemed to be only manageable by humans now seem within reach of computers. But especially in critical applications, such as self-driving cars or medical equipment, it is crucial to confirm that the software, unlike humans, will never make any mistakes.

The process of formally proving the correctness of a computer program is called software verification. Considering the amount and complexity of program code produced every day, it is obvious that only automated approaches are suitable for this task. Usually, these approaches use some kind of formalism to encode software systems and their properties in order to make them suitable for formal reasoning. One example of this is satisfiability modulo theories (SMT), which expresses properties as logical formulas so that analysing their satisfiability allows to infer when a property is fulfilled. Depending on the context in which SMT is used, the formulas have different syntax and semantics, which are determined by a background theory. One particularly interesting theory is quantifier free non-linear real arithmetic (QFNRA), where formulas describe relations between real number variables by so-called polynomial constraints. It yields a very expressive and versatile logic for which the satisfiability problem is still decidable. Taking into account that the satisfiability problem for propositional logic is already NP-complete, tools for solving SMT problems are often confronted with generally very challenging tasks. However, there are many strategies that prove to be efficient and helpful in practice and thus, there has been vivid research in the field of SMT solving.

A noteworthy result is the *model constructing satisfiability calculus* (MCSAT) proposed by Jovanović and De Moura in 2013, along with its application to QFNRA [dMJ13, JdM12]. A crucial part of this method is the *single cell construction*. For a set of multivariate polynomials with n input variables and a sample point s in the multidimensional real space $\mathbb{R}^n$, a connected set $S \subseteq \mathbb{R}^n$ is constructed around s so that the polynomials have constant sign when evaluated at any point in S. The implementation by Jovanović and de Moura builds upon the theoretical results of Collins' CAD method [Col75] and was later improved by Brown and Košta [BK15]

whose ideas inspired a recent approach by Nalbach et al. [NSÁ+22].

The single cell construction subroutine makes up a significant portion of the runtime in MCSAT and one reason for this is the repeated computation of polynomial resultants, which can lead to a doubly exponential blow-up, depending on the degrees of the involved polynomials. Accordingly, there have been efforts to improve the efficiency of single cell construction methods and to reduce the number of computed resultants.

In this thesis, we will tackle this problem from a different angle, aiming to compute simpler resultants instead of fewer.

Contributions We modify the *levelwise* method for single cell construction recently proposed by Nalbach et al. ([NSÁ+22]) with the aim of reducing the complexity of resultant calculations, which make up a significant portion of the method's runtime. In particular, this thesis contains the following contributions:

- Firstly, we show how, during the levelwise single cell construction, new polynomials can be introduced to the working set of polynomials so that the computed resultants are far less complex. The resulting cell will always be an underapproximation of the maximal sign-invariant cell for the given input but not necessarily a subset of the cell computed by the unmodified procedure.
- We present two exemplary options for constructing the additional polynomials, one of which is a naive box-like approach, while the other is based on the multivariate Taylor expansion.
- Our main motivation is to improve the single cell construction in the context of the MCSAT method for SMT-solving. Therefore, we show that MCSAT is generally not complete when using our modified procedure, but we also present simple measures that can be employed to guarantee completeness, after all.
- There are many heuristic decisions within our approach, for which we give a number of possible choices and strategies. We also implemented our approach with the different heuristics within the SMT-RAT toolbox for SMT solving [SMTb, CKJ+15].
- Finally, we evaluate our approach in the context of MCSAT using the SMTLIB benchmark set [SMTa, BST10]. We compare it to the original levelwise method and investigate the performance of different heuristics and settings.

Structure In Chapter 2, we describe the foundations of our work and set up some basic definitions. First, the satisfiability problem for quantifier free non-linear real arithmetic (QFNRA) is introduced. We then show how the model constructing satisfiability calculus (MCSAT) can be used to solve that problem by utilizing sign-invariant cells. In Section 2.3, we delve further into the construction of single sign-invariant cells and present the levelwise approach by Nalbach et al.

Building upon this basis, we modify the levelwise method in Chapter 3. After giving an intuition for the idea of our approach and presenting a rather simple implementation, we provide a general formulation and show its correctness. This is followed by a more complex implementation using the multivariate Taylor expansion. In Chapter 4, we evaluate our approach based on experimental results.

Finally, Chapter 5 concludes this thesis, summarizing our work and providing some thoughts about future research based on our approach.

Chapter 2

Preliminaries

2.1 SMT Solving for Non-linear Real Arithmetic

Satisfiability modulo theories (SMT) is a variation of the satisfiability problem for first-order logic (FO), where the formulas are interpreted with respect to one or more background *theories*. In practice, the theory often predetermines the used variable domain, predicates and function symbols, as well as their semantics. This means that in SMT, we can view formulas as Boolean combinations of theory atoms (*constraints*), each of which is evaluated over its respective theory.

We will focus on one of the more well-known theories, named *real arithmetic*, which uses the real numbers $\mathbb{R}$ as underlying variable domain, along with the function symbols $+$ and $\cdot$ and comparison predicates $<$, $\leq$, $\geq$, $>$, $=$, $\neq$ with their standard semantics for $\mathbb{R}$. Before we define the formulas and satisfiability notion of quantifier free non-linear real arithmetic, we fix some basic notations and concepts used throughout the thesis.

We denote by $\mathbb{N}$ the set *natural numbers* including 0, by $\mathbb{Q}$ the *rational numbers* and by $\mathbb{R}$ the *real numbers*, respectively. Given $k \in \mathbb{N}$, we define the set of natural numbers from 1 to k as $[k] := \{1,\ldots,k\} \subset \mathbb{N}$. Moreover, for $k \in \mathbb{N}$ and $j \in [k]$, the first j components of a multidimensional point $r = (r_1,\ldots,r_k) \in \mathbb{R}^k$ are written as $r_{[j]} := (r_1,\ldots,r_j)$. In the remainder of this thesis, let $n \in \mathbb{N}$ be an arbitrary positive integer. We fix a set $X := \{x_1,x_2,\ldots,x_n\}$ of distinct (real valued) variables and an ordering $x_1 \prec x_2 \prec \ldots \prec x_n$ according to the index labels of the variables. Furthermore, we use lowercase letters like y to denote an arbitrary variable in X.

Definition 2.1.1 (Polynomial)**.** *Let R be a ring, e.g. $\mathbb{R}$ or $\mathbb{Q}$, and $m \in \mathbb{N}$. For all $i \in [m]$ and $j \in [n]$, let $a_i \in R$ and $e_{i,j} \in \mathbb{N}$. Then a term of the form*

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

is a polynomial *over R in variables $x_1,\ldots,x_n$. The set of all such polynomials is again a ring and denoted by $R[x_1,\ldots,x_n]$.*

Definition 2.1.2 (Univariate Polynomial)**.** *Let R be a ring and $p \in R[x_1,\ldots,x_n]$. We call p* univariate *in $y \in X$ if $p \in R[y]$, i.e. p mentions no variables other than y. Otherwise, p is called* multivariate.

We can always interpret $p \in R[x_1,\ldots,x_n]$ as a univariate polynomial in x_n with coefficients $R[x_1,\ldots,x_{n-1}]$, that is as $p \in R[x_1,\ldots,x_{n-1}][x_n]$. In particular, it can then be written as $p = \sum_{i=0}^{m} c_i x_n^i$ for a unique $m \in \mathbb{N}$ and unique polynomial coefficients $c_0,\ldots,c_m \in R[x_1,\ldots,x_{n-1}]$.

Definition 2.1.3 (Coefficients and Degree). *Let R be a ring, $p \in R[x_1,\ldots,x_n]$. Moreover, let $m \in \mathbb{N}$ and $c_0,\ldots,c_m \in R[x_1,\ldots,x_{n-1}]$ be so that $p = \sum_{i=0}^{m} c_i x_n^i$.*

- *We define the set $coeff_{x_n}[p] := \{c_0,\ldots,c_m\} \subset R[x_1,\ldots,x_{n-1}]$ of* polynomial coefficients of p w.r.t. x_n.
- *The* degree *of p with respect to x_n is $deg_{x_n}(p) = m$.*

The terms built from rational numbers, variables $x_1,\ldots,x_n$, addition and multiplication form the set $\mathbb{Q}[x_1,\ldots,x_n]$ of polynomials with rational coefficients. We can evaluate a polynomial $p \in \mathbb{Q}[x_1,\ldots,x_n]$ at a point $r \in \mathbb{R}^n$ by substituting r_1 for x_1, r_2 for x_2 and so on, up to x_n. The result is denoted by $p(r) \in \mathbb{R}$.

Definition 2.1.4 (Real Roots). *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 $realRoots(p)$.*

Note that, if a (non-zero) polynomial p with rational coefficients is univariate in some variable y, i.e. $p \in \mathbb{Q}[y] \setminus \{0\}$, then the set $realRoots(p)$ is finite. The roots of univariate polynomials let us define the real algebraic numbers, which are important for algorithmic reasons. In contrast to the real numbers, all algebraic numbers are finitely representable, which makes them suitable for computations.

Definition 2.1.5 (Real Algebraic Numbers). *A real number $r \in \mathbb{R}$ is* algebraic *if there is some $p \in \mathbb{Q}[y]$ (y being any variable) with $p(r) = 0$. The set of all* real algebraic numbers (RANs) *is denoted by $\mathcal{R}$. Every RAN r can be represented as a pair $r = (p,I)$, where $p \in \mathbb{Q}[y]$ with $p(r) = 0$ and I is an open interval with rational boundaries, so that $r \in I$ is the only root of p contained in I.*

Example 2.1.1 (RAN). *The irrational number $\sqrt{2}$ is algebraic with a possible representation $(y^2 - 2, (0,2))$. Note that there are real numbers which are not algebraic, for example π, and thus $\mathcal{R} \subsetneq \mathbb{R}$.*

There is an effective procedure for isolating the real roots of a univariate polynomial and representing them as RANs. Given algebraic numbers $r_1,\ldots,r_{n-1} \in \mathcal{R}$ and $p \in \mathbb{Q}[x_1,\ldots,x_n]$, we can isolate the real roots of $p(r_1,\ldots,r_{n-1},x_n)$. That is, we can compute the set $\{r' \in \mathbb{R} \mid p(r_1,\ldots,r_{n-1},r') = 0\} \subset \mathcal{R}$. For $r = (r_1,\ldots,r_{n-1})$, we also write $p(r,x_n)$ to represent the univariate polynomial resulting from substituting $r_1,\ldots,r_{n-1}$ for $x_1,\ldots,x_{n-1}$ in p. Similarly, given an additional value $r' \in \mathbb{R}$, we write $p(r,r')$ to denote $p(r_1,\ldots,r_{n-1},r')$.

With these definitions in place, we now define formulas of non-linear[1] real arithmetic. Polynomial constraints compare a polynomial to zero and they are the most basic kind of formula, also called theory atoms. They can be combined to create more complex formulas using the standard Boolean connectives.

[1] *Linear* real arithmetic is a fragment of the here presented logic, where only linear polynomials are used. It can be solved more efficiently with specialized algorithms, which is why we make the explicit distinction.

Definition 2.1.6 (Polynomial Constraint). *Let $p \in \mathbb{Q}[x_1,\ldots,x_n]$ and $\sim \in \{<, \leq, =, \neq, \geq, >\}$ be a comparison symbol. Then we call the term*

$$p \sim 0$$

a polynomial constraint.

Since we do not deal with other theories in the thesis, this is the only kind of considered constraints and therefore, we will sometimes omit the specifier "polynomial" and simply say *constraint*.

Definition 2.1.7 (Formula). *A quantifier free* formula of non-linear real arithmetic (QFNRA-formula) *is a Boolean combination of polynomial constraints using the operators $\wedge, \vee, \neg$.*

If a formula φ is built from constraints over variables $x_1,\ldots,x_n$, then we call these the *free variables* of φ and also write $\varphi(x_1,\ldots,x_n)$ to indicate this.

Every QFNRA-formula φ has an equivalent formula φ_{CNF} in *conjunctive normal form (CNF)*, that is, there are $c, k_1,\ldots,k_c \in \mathbb{N}$ so that $\varphi_{CNF} = \bigwedge_{i\in[c]}(\bigvee_{j\in[k_i]} l_{i,j})$, where each $l_{i,j}$ is a constraint or the negation of a constraint. The disjunctions in a CNF are called *clauses* and the disjuncts are also referred to as *literals*. Note that the negation of a constraint $p > 0$ is equivalent to the constraint $p \leq 0$ and similarly for the other comparison symbols.

The semantics of QFNRA are straightforward: we can assign real values to the free variables of a formula and evaluate it handling polynomials and comparisons as usual in $\mathbb{R}$ and using the standard semantics for the Boolean operators.

Definition 2.1.8 (Semantics of QFNRA). *Let $r \in \mathbb{R}^n$, $p \in \mathbb{Q}[x_1,\ldots,x_n]$ be a polynomial and $\sim \in \{<, \leq, =, \neq, \geq, >\}$.*

- *We say that r* satisfies *the constraint $p \sim 0$, if $p(r) \sim 0$ evaluates to true under the standard semantics of $\mathbb{R}$. In that case, we write $r \models (p \sim 0)$.*
- *This notion of satisfaction is extended in the usual way. For formulas 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 $r \models \varphi$ (also called* model*) and* unsatisfiable *otherwise.*

Example 2.1.2 (QFNRA-Formula). *The QFNRA-formula*

$$\varphi = \underbrace{(1 = 0 \vee x_1 > 0)}_{clause} \wedge (\quad \underbrace{x_1 \leq 0}_{constraint/literal} \quad \vee \underbrace{x_1^2 + x_2^2 - 1 < 0}_{polynomial})$$

is in conjunctive normal form. For the point $s = (\frac{1}{2},0) \in \mathbb{R}^2$, assigning $\frac{1}{2}$ to x_1 and 0 to x_2, we have $s \models \varphi$ and thus the formula is satisfiable.

Finding a model for QFNRA-formulas algorithmically seems tricky at first, considering that not all real numbers are finitely representable. Fortunately, it holds that for any QFNRA-formula $\varphi(x_1,\ldots,x_n)$ there exists $s \in \mathbb{R}^n$ with $s \models \varphi$ if and only if there exists $s \in \mathcal{R}^n$, $s \models \varphi$. Therefore, all algorithms presented in this thesis will in fact work on RANs and not on arbitrary real numbers.

2.2 MCSAT

Tarski showed in 1948 that the satisfiability problem for real arithmetic is decidable, even for quantified formulas [Tar98]. However, the algorithm he presented was not suitable for practical applications as it suffered from a non-elementary runtime complexity. The first somewhat applicable approach followed in 1975, when Collins introduced the idea of *cylindrical algebraic decomposition (CAD)* [Col75], showing that for a given finite set of polynomials in n variables, the infinite space $\mathbb{R}^n$ can be partitioned into a finite number of cells over which those polynomials have constant sign. The satisfiability of any formula built from constraints using those polynomials can then be determined by evaluating it on one sample point from each of the corresponding cells. Although the CAD method is a great achievement, it still leaves room for improvement, having a doubly exponential worst-case complexity which can also be experienced in practical applications. Since then, gradual improvements to the method were achieved by McCallum, Lazard and Brown [McC98, Laz94, MPP19, BM20] and various SMT-solving methods for QFNRA have been developed which use the theoretical insights related to CAD. Rather recently, in 2013, Jovanović and de Moura presented a *model constructing satisfiability calculus* (MCSAT) [dMJ13, JBd13] that can be used to solve the satisfiability problem for a row of theories. In particular, they provided an instantiation of the abstract MCSAT method for solving QFNRA and called the resulting procedure NLSAT [JdM12][2].

The general idea of MCSAT is to construct a model for the given input formula through a close interplay between theory and Boolean reasoning. In addition to deciding the truth value of a literal, it allows to temporarily decide the value of theory variables, which in turn impacts the possible Boolean decisions. When the search for a model encounters a conflict, MCSAT tries to generalize that conflict, gaining information on both the theory and Boolean level, and employs clause learning. We will now give a brief overview of an implementation of this procedure specifically for the theory of QFNRA, mainly based on the description in [Kre19]. A rough sketch of the algorithm flow is also illustrated in Figure 2.1. While MCSAT is the name of the general approach, which is applicable to many theories, we will use it to refer to the specific implementation for real arithmetic.

Algorithm Flow MCSAT takes as input a QFNRA-formula $\varphi(x_1, \ldots, x_n)$ in conjunctive normal form $\varphi = \bigwedge_{c \in C} c$ for a clause set C and it returns SAT along with a model $s \models \varphi$, if the input is satisfiable and $UNSAT$ otherwise. The procedure can be structured into levels corresponding to the theory variables $x_1, \ldots, x_n$.

Definition 2.2.1 (Level)**.** *Let F be a polynomial, constraint or clause. The* level *of F is the index label of the highest variable occurring in F, i.e.*

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

Moreover, if $lvl(F) = i$, then we call x_i the main variable *of F.*

According to their level, i.e. the highest variable they contain, the clauses are sorted into sets $C_0, \ldots, C_n \subseteq C$ with $C_i := \{c \in C \mid lvl(c) = i\}$ for $i \in \{0, \ldots, n\}$. Each level of the algorithm deals with the corresponding clauses, trying to find a partial model that is also consistent with the lower levels.

[2] In fact, they first published the NLSAT procedure and then generalized its ideas.

Construction of a Partial Model More precisely, the bottom level considers C_0, which are the clauses that do not contain any theory variables and thus immediately evaluate to the Boolean constants *true* or *false*. If level 0 is consistent, i.e. none of the clauses evaluate to *false*, the algorithm moves on to level 1, where it tries to satisfy C_1.

MCSAT employs Boolean reasoning to determine a set of literals which are contained in the clauses in C_1 and which are assumed to be true. The truth value of a literal is either determined by Boolean propagation or by making a decision, which can then lead to propagation for other literals. If the resulting Boolean assignment satisfies C_1, a value $s_1 \in \mathbb{R}$ for x_1 is chosen, which is consistent with the asserted literals and therefore $s_1 \models C_1$. This value is a partial model which is to be extended for the other variables in the subsequent levels.

For level 2, this means that MCSAT searches a value $s_2 \in \mathbb{R}$ for x_2, so that $(s_1,s_2) \models C_2$. Since s_1 is temporarily fixed, this is again a univariate problem and can be handled in the same way as for level 1. Note that the extended model should satisfy the same literals from C_1 as before and thus, the Boolean decisions of level 1 are still in effect, potentially propagating truth values for the literals at level 2.

In general, the partial model $(s_1,\ldots,s_i) \in \mathbb{R}^i$ is consistent with level i and is extended by an assignment s_{i+1} for x_{i+1} so that C_{i+1} is satisfied. This process is repeated level by level until either a model for C_n is found, meaning that the input is satisfiable, or until the procedure fails to find a satisfying assignment for the current level and thus encounters a conflict.

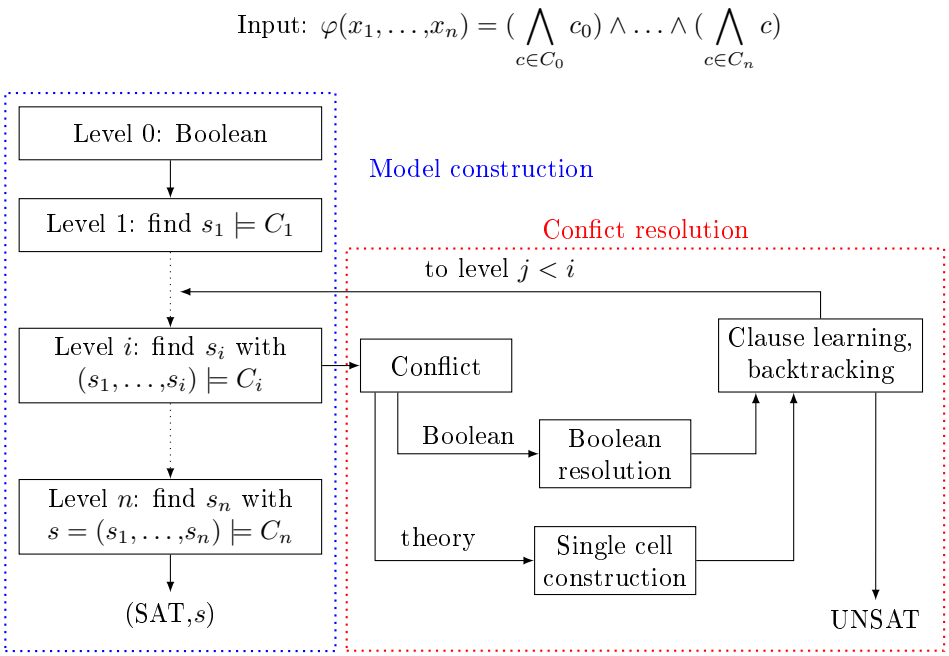


Figure 2.1: Schematic illustration of the NLSAT procedure

Conflicts We distinguish two kinds of conflicts. If all literals of a clause are forced to be false by the previous Boolean decisions and propagations, then that clause cannot be satisfied under the current assumptions and we call this situation a *Boolean conflict*. If the partial model cannot be extended to satisfy the currently asserted literals, i.e. there is no feasible value s_i for x_i, it is a *theory conflict*. In either case, MCSAT tries to resolve the issue by employing conflict driven clause learning (CDCL) and backtracking to one of the lower levels, undoing Boolean decisions and theory assignments in the process. Backtracking stops if a level is reached where the learned clause is satisfiable consistently with the decisions made up to that level, or if it reaches level 0, in which case it can be deduced that the input formula is unsatisfiable. While the conflict resolution and clause learning for Boolean conflicts can be done in the usual way like in standard SAT solvers, it is a bit more involved for the theory conflicts.

When the current partial model $(s_1, \dots, s_i)$ cannot be extended with an assignment for x_{i+1}, MCSAT generates an *explanation* for this conflict, with the purpose to guide the later search for a model. The more general this explanation is, the better it can guide the search. Simply learning that the partial model $(s_1, \dots, s_i)$ fails is barely helpful, considering that it only excludes one of infinitely many possible assignments. Instead, the conflict is generalized in two ways. First, a *conflicting core* is identified, which is a minimal subset K of the asserted literals on level $i+1$, so that already for K, no satisfying assignment $(s_1, \dots, s_i, s_{i+1}) \models K$ exists. Then, the partial model is generalized to a whole set $S \subseteq \mathbb{R}^i$ of assignments that would fail in the same way and cannot be extended to satisfy the conflicting core K. The description of such a set S can be computed using *single cell construction*, which we introduce in Section 2.3. Its origin lies in the CAD method and the related theoretical concepts, but instead of an entire decomposition, only a cell containing s is constructed.

The result of the conflict explanation process is a clause stating that, if the literals in K hold, the partial model (restricted to the variables $x_1, \dots, x_i$) must not be contained in S. This clause is added to the input formula, thereby excluding S from the later search. The part of the clause expressing membership of an assignment to the set S might introduce literals which were not present in the original input. It is worth pointing out that the termination and hence completeness of MCSAT is only guaranteed if for each input formula only finitely many new literals will be introduced by any number of generated explanations. This is important because, in the main part of this thesis, we will modify a procedure used for generalizing s to a set S, which will result in a different explanation. Consequently, we change what new literals are introduced and thus need to watch out for completeness.

Example 2.2.1. *We again consider the formula φ from Example 2.1.2:*

$$\varphi = \underbrace{(1 = 0 \vee x_1 > 0)}_{level\ 1} \wedge \underbrace{(x_1 \leq 0 \vee x_1^2 + x_2^2 - 1 < 0)}_{level\ 2}$$

At level 1, the first clause is considered and obviously, $x_1 > 0$ needs to hold to satisfy it. MCSAT now chooses a value for x_1, say $s_1 = 1$. At level 2, the second clause is considered. Under the current partial model, $x_1 \leq 0$ does not hold and therefore, the literal $l := x_1^2 + x_2^2 - 1 < 0$ needs to be satisfied. However, it simplifies to $x_2^2 < 0$ when substituting s_1 for x_1, which is unsatisfiable and therefore a theory conflict is encountered. For the conflicting core $\{l\}$, we could generalize the assignment $s_1 = 1$ to the set $S = \{s_1 \in \mathbb{R} \mid s_1 \geq 1\}$ and learn the clause $(\neg l \vee x_1 < 1)$. After backtracking

to level 1, Boolean propagation reveals that the new literal $x_1 < 1$ needs to hold. A different value, e.g. $s_1 = 1/2$ is chosen, which allows the choice of $s_2 = 0$ for x_2 and hence, the input is satisfiable.

2.3 Constructing a Single Sign-Invariant Cell

A crucial part of the MCSAT method is the generation of explanations for theory conflicts. Jovanović and De Moura formulated MCSAT as an abstract decision procedure, leaving a lot of variability for specific implementations. This includes the explanation function, for which they specified certain properties it must possess but did not fix it entirely. Nevertheless, they did also provide a concrete implementation in NLSAT, which at its core uses the concepts of cylindrical algebraic decomposition (CAD) to compute a single sign-invariant cell around the partial model that led to a conflict [JdM12]. In this section, we will describe the corresponding task of single cell construction and present an approach recently proposed by Nalbach et al. [NSÁ+22].

Definition 2.3.1 (Cell)**.** *We call a set $S \subseteq \mathbb{R}^i$ ($i \in \mathbb{N} \setminus \{0\}$) a* cell*, if it is non-empty and connected. A cell S is* algebraic*, if it is a semi-algebraic set, meaning that it is the solution set of a conjunction of polynomial constraints.*

Definition 2.3.2 (Sign-invariance)**.** *Let $i \in \mathbb{N}$ and $p \in \mathbb{Q}[x_1,\dots,x_i]$. The polynomial p is* sign-invariant *on a set $R \subseteq \mathbb{R}^i$ if*

$$\text{either } \forall r \in R.\ p(r) > 0 \quad \text{or} \quad \forall r \in R.\ p(r) < 0 \quad \text{or} \quad \forall r \in R.\ p(r) = 0 \text{ holds.}$$

*A set $P \subset \mathbb{Q}[x_1,\dots,x_i]$ of polynomials is sign-invariant on R if all $p \in P$ are sign-invariant on R. In that case we also say that R is P-*sign-invariant*.*

The task of *single cell construction* is the following: given a non-empty finite set $P \subset \mathbb{Q}[x_1,\dots,x_i]$ of polynomials and a point $s \in \mathbb{R}^i$, compute a description of a P-sign-invariant algebraic cell $S \subset \mathbb{R}^i$ with $s \in S$.

There are two additions to this formulation which are important in practice. Firstly, note that the cell description should be suitable for algorithmic purposes, meaning that it should be computable, have a coherent structure with other cell descriptions and be accessible for other algorithms, in our case for MCSAT. For this reason, we will compute algebraic cells with a cylindrical structure, so that they can be represented by a conjunction of so-called *extended polynomial constraints* (see Def. 2.3.13). Secondly, we would like the cell to be as large as possible, again for algorithmic reasons. In the context of MCSAT, a larger cell will exclude more from the search space and might thus lead to a quicker result. While there is no obvious way to quantify the size of a cell and we will not define the notion formally, we hope that the reader will get an intuition for it throughout this section and the next chapter.

2.3.1 Connection to MCSAT

Before we explain how single cell construction can be solved, we want to motivate its usage in the explanation process of MCSAT.

In the case of a theory conflict, the following situation occurs: there is a set K of constraints with level up to $i+1$ and a partial model $s = (s_1,\dots,s_i) \in \mathbb{R}^i$ so that for all $s_{i+1} \in \mathbb{R}$ holds $(s,s_{i+1}) \not\models K$. In order to use single cell construction, x_{i+1} is

eliminated from K, producing a set $P \subset \mathbb{Q}[x_1,\ldots,x_i]$ of polynomials of level at most i. This set has the property that, if $S \subseteq \mathbb{R}^i$ is a P-sign-invariant cell containing s, then for all $s' \in S$ and for all $s_{i+1} \in \mathbb{R}$ holds $(s',s_{i+1}) \not\models K$, i.e. all $s' \in S$ lead to the same conflict as s. Accordingly, computing a sign-invariant cell for the input (P,s) yields the set S generalizing the conflict.

The input polynomials P can be constructed using CAD technology, in particular the concept of delineability, which ensures the crucial properties of P. As the single cell construction itself also utilizes those concepts, we will not go into more detail here and instead explain them in the context that we are most interested in, i.e. single cell construction. For more information on the original CAD method and the theory surrounding it, see [Col75] and for details on how to integrate it into MCSAT, see [JdM12] or [NSÁ+22].

Finally, the output cell description will be easily transformable into a formula for MCSAT.

2.3.2 The Levelwise Method

In their NLSAT implementation, Jovanović and De Moura compute sign-invariant cells with a slightly adapted version of Collins' CAD method, which does not return a decomposition of the entire space, but only the desired cell. A more refined alternative was later published by Brown and Košta [BK15], who developed an approach which is better tailored to the specific task of computing a single cell around the given sample point. Their ideas inspired further research and lead to the very recent development of the so-called *levelwise* approach by Nalbach et al. [NSÁ+22], which provides more control over the generated cells and related optimizations. We now present this approach and it will be the basis of our considerations in Chapter 3.

We start with a motivating example, providing some intuition before we define the method formally.

Example 2.3.1 (Single Cell Construction)**.** *We want to construct a cell around the point* $s = (s_1,s_2) = (-\frac{1}{2},0) \in \mathbb{R}^2$ *on which the input polynomials* $P = \{p,q\} \subset \mathbb{Q}[x_1,x_2]$ *with*

$$p = x_1^2 + x_2^2 - 1 \text{ and } q = -2x_1x_2 + x_2 - 1$$

are sign-invariant. Figure 2.2 depicts the maximal P-sign-invariant cell around s, in the sense that no connected superset is also P-sign-invariant. The intuition for this is that every such superset will contain a point at which p or q has a different sign than at s. It is maximal in the sense that no connected superset is also P-sign-invariant, which is due to the fact that every such superset will contain a point on the boundary of the cell and thus a point at which p or q has a different sign than at s.

Unfortunately, it is often not possible to compute a maximal cell like this or to find a suitable description. Therefore, we can in general only construct the description for a sub-cell, which is included in, but not equal to the maximal cell.

As the name suggests, the levelwise method is structured into levels corresponding to the variables and their ordering. Given $P \subset \mathbb{Q}[x_1,\ldots,x_i]$ and $s \in \mathcal{R}^i$, it starts at the top level, which corresponds to x_i, and determines a symbolic interval I_i bounding x_i with respect to $x_1,\ldots,x_{i-1}$. Then, in the *projection step*, it derives from P a set $P_{i-1} \subset \mathbb{Q}[x_1,\ldots,x_{i-1}]$ of polynomials which are the input for the next level. This process is repeated for each level $j = (i-1),\ldots,1$, determining symbolic bounds I_j depending on $x_1,\ldots,x_{j-1}$ and projecting the polynomials. On the bottom level, the

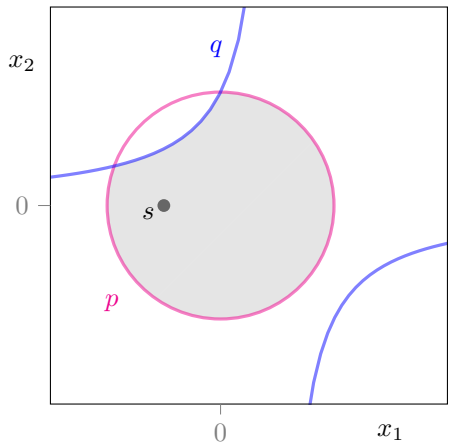


Figure 2.2: Graph of the implicit equations $p = 0$ (in red) and $q = 0$ (in blue), together with the maximal P-sign-invariant cell (grey area) around $s = (-\frac{1}{2},0)$. This cell does is bounded by, but does not include the contours defined by the roots of p and q. Notice that, throughout this thesis, we will label the graph of an implicit equation $p = 0$ (also called the *variety* of p) simply with the name of the polynomial, despite not plotting the non-zero function values of p.

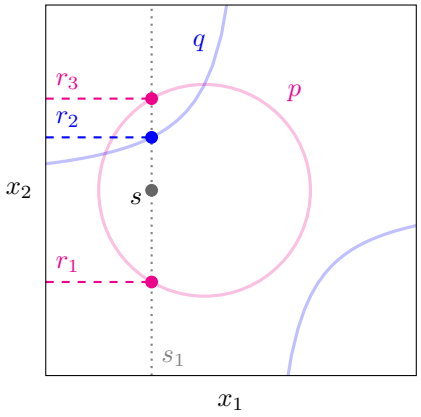
description I_1 does not depend on any variables and thus defines a concrete interval in $\mathbb{R}$. Crucially, the projections are chosen in a way ensuring that for all j, the description I_j is well-defined and correct when evaluated on any point $(r_1,\dots,r_{j-1})$ that lies in the underlying cell described by the lower levels $I_1,\dots,I_{j-1}$.

Example 2.3.2. *In our example, the top level corresponds to x_2. We begin by determining which polynomials define the cell boundary above and below the sample s, in the x_2-direction. Note that the algorithm is not aware of the plots which we use to illustrate the boundaries and it does not compute them. Instead, as is shown in Figure 2.3a, we fix the x_1-coordinate to s_1 and isolate and order the roots r_1,r_3 of $p(s_1,x_2)$ and r_2 of $q(s_1,x_2)$, which is a univariate problem. At the sample coordinate s_1, the desired cell is bounded in the x_2-direction by r_1 and r_2, belonging to p and q, respectively. We use this information to construct a symbolic interval I_2 depending on x_1, whose lower bound is defined by p and whose upper bound is defined by q, generalizing the root structure found at s_1.*

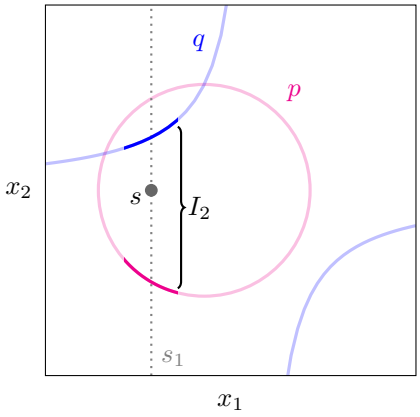
We then move to the next level, where we try to find an interval I_1 for x_1, on which the symbolic description I_2 for the upper level is still correct and well-defined. Graphically, I_1 is a neighbourhood of s_1 so that for each $s' \in I_1$, the root of $q(s',x_2)$ defines the upper bound and the smallest root of $p(s',x_2)$ defines the lower bound of the cell, just like at s_1. This is shown in Figure 2.3c.

We will now, step by step, formalize the intuition from the example and explain how the levelwise method works in the general case.

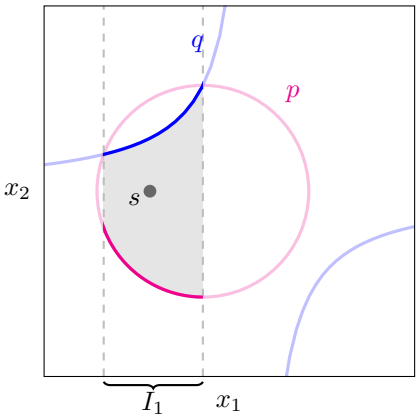
A primary observation is that the roots of the involved polynomials play a crucial role for sign-invariant cells as they define the boundaries of the maximal sign-invariant cells. In fact, the symbolic intervals we use for the cell description are defined via a symbolic representation of roots, called *indexed root expressions*.



(a) The roots of p and q at $x_1 = s_1$ are isolated and ordered. r_1, belonging to p, is the only root *below* s_2 and r_2, belonging to q is the closest root *above* s_2.



(b) At s_1, the lower and upper cell bound is defined by p and q, respectively. This is generalized to the symbolic interval I_2.



(c) The interval I_1 for x_1 is constructed around s_1 so that the root structure is preserved.

Figure 2.3: Intuition for the levelwise construction in the context of Example 2.3.1.

Definition 2.3.3 (Indexed Root Expression)**.** *Let $i, j \in \mathbb{N}$ and $p \in \mathbb{Q}[x_1,\dots,x_i]$ with $lvl(p) = i > 0$. An* indexed root expression *has the form $root_{x_i}[p,j]$ and we can evaluate it at any point $s \in \mathcal{R}^{i-1}$ as follows:*

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

Given $s \in \mathbb{R}^{i-1}$ and $P \subset \mathbb{Q}[x_1,\dots,x_i]$, we define the set of indexed root expressions for the real roots of P at s as

$$irExp(P,s) = \{root_{x_i}[p,j] \mid p \in P, j \in [|realRoots(p(s,x_i))|]\}.$$

Here, *undef* denotes that the value of the indexed root expression is *undefined* and in that sense, it induces a partial function.

The levelwise method uses indexed root expressions to construct a *locally cylindrical cell*. This means that the cell description can be structured into levels $I_1,\ldots,I_i$ for the variables $x_1,\ldots,x_i$ so that for $j \in [i]$, I_j restricts x_j with respect to $x_1,\ldots,x_{j-1}$, but independent of the variables $x_{j+1},\ldots,x_i$. More precisely, I_j either states that the value of x_j should be in between an upper and lower bound given by indexed root expressions derived from polynomials of level j, or it states that the value of x_j should be equal to one of those expressions. In the first case, we say that I_j defines a *sector* and in the second case a *section*. Note that in the case of a sector, the bounds can also be $\pm\infty$, indicating that there is no real bound.

To simplify the following definition, we define

Definition 2.3.4 (Single Cell Data Structure)**.** *A* single cell data structure *or* cell description *is a sequence $D = (I_1,\ldots,I_n)$, where for each $i \in [n]$ the element I_i is an object of one of the following forms:*

- *$I_i = (sector,l,u)$, where l is either an indexed root expression depending on the variables $x_1,\ldots,x_{i-1}$, or $-\infty$ and u is either an indexed root expression depending on the variables $x_1,\ldots,x_{i-1}$, or ∞.*
- *$I_i = (section,b)$, where b is an indexed root expression depending on the variables $x_1,\ldots,x_{i-1}$.*

We can evaluate I_j at any point $r \in \mathbb{R}^{j-1}$ for which the corresponding root expressions are defined and obtain a real interval, which in the case of a section will always contain only a single point. Now, graphically speaking, the lower levels $I_1,\ldots,I_{j-1}$ define an *underlying cell* $S(I_1,\ldots,I_{j-1}) \subseteq \mathbb{R}^{j-1}$ and adding I_j defines a slice of the cylinder $S(I_1,\ldots,I_{j-1}) \times \mathbb{R}$ which has the underlying cell as its base. With these concepts in mind, we define the set of points contained in the cell induced by a cell description. To simplify the definition, we set $-\infty(r) := -\infty$ for all $i \in \mathbb{N}$ and $r \in \mathbb{R}^i$ and do so symmetrically for ∞. This is only to avoid additional case distinctions for sectors without two real bounds and we will use $\pm\infty$ only as interval boundaries, which has a well-defined meaning.

Definition 2.3.5 (Set Represented by a Single Cell Data Structure)**.** *A description $D = (I_1)$ represents the cell*

$$S(D) = S(I_1) := \begin{cases} \{r \in \mathbb{R} \mid r \in (l,u)\} & \text{if } I_1 = (sector,l,u) \\ \{r \in \mathbb{R} \mid r = b\} & \text{if } I_1 = (section,b). \end{cases}$$

Note that in this case l, u and b are constant total functions which we identify with their constant value. A description $D = (I_1,\ldots,I_n)$ with $n > 1$ represents the cell

$$S(D) := \begin{cases} \{(r,r') \in \mathbb{R}^n \mid r \in S(I_1,\ldots,I_{n-1}), r' \in (l(r),u(r))\} & \text{if } I_n = (sector,l,u) \\ \{(r,r') \in \mathbb{R}^n \mid r \in S(I_1,\ldots,I_{n-1}), r' = b(r)\} & \text{if } I_n = (section,b). \end{cases}$$

In accordance to the prior comment, if $l = -\infty$, then $l(r) = -\infty$ as well and analogously for u.

Example 2.3.3. *In our leading example, the roots r_1,r_3 correspond to the indexed root expressions $\xi_1 := root_{x_2}[p,1]$ and $\xi_3 := root_{x_2}[p,2]$ associated with p, while r_2*

corresponds to $\xi_2 := root_{x_2}[q,1]$. *In particular, we have* $\xi_k(s_1) = r_k, (k \in \{1,2,3\})$. *Accordingly, the symbolic interval* I_2 *for the cell description is given by*

$$I_2 = (sector, root_{x_2}[p,1], root_{x_2}[q,1]) = (sector, \xi_1, \xi_2).$$

Note that the bounds ξ_1,ξ_2 *describe partial functions depending on* x_1. *Therefore, the underlying cell* $S(I_1)$ *for* x_1, *which is constructed on the next level, must only contain values for which* ξ_1 *and* ξ_2 *are defined. For* ξ_1, *this restricts the values to* $[-1,1]$ *and* ξ_2 *is undefined on* $x_1 = \frac{1}{2}$. *Furthermore, we need to identify the points where the graphs cross and* q *is no longer the correct upper bound of the cell. The* x_1*-coordinates of those intersection points pose additional restrictions on the extent of* I_1.

We can generalize the requirements for the underlying cell stated in the example. For every $j \in [i]$, we need that I_j is *well-defined* over $S(I_1,\dots,I_{j-1})$. This primarily means that for all $s' \in S(I_1,\dots,I_{j-1})$, the indexed root expressions in I_j have a defined value at s', but we will in fact ensure that each of these expressions induces a continuous function on the domain $S(I_1,\dots,I_{j-1})$. Furthermore, the described cell should be *correct* in the sense that, if $I_j = (sector, l, u)$, then for all $s' \in S(I_1,\dots,I_{j-1})$, the indexed root expressions l,u give the closest roots below and above s_j among all roots of polynomials involved in level j. On the other hand, if $I_j = (section, b)$, then no polynomial in P_j should change signs on the graph $\{(s', b(s')) \mid s' \in S(I_1,\dots,I_{j-1})\}$ of the root function induced by b.

For this purpose, we need to identify the points (or in general: regions) in the underlying space $\mathbb{R}^{j-1}$, where the root structure w.r.t x_j of the polynomials changes and where their varieties intersect. This can be done by examining the real roots of certain polynomials derived from P_j.

Definition 2.3.6 (Coefficients, Resultants and Discriminants)**.** *Let* $i \in \mathbb{N}$, $i > 0$, $p, q \in \mathbb{Q}[x_1,\dots,x_i]$ *and let* $k \in \mathbb{N}$, $c_0,\dots,c_k \in \mathbb{Q}[x_1,\dots,x_{i-1}]$ *be so that* $p = \sum_{j=0}^{m} c_j x_i^j$ *and* $c_k \neq 0$.

- *The set of* coefficients *of* p *w.r.t* x_i *is* $coeff_{x_i}[p] = \{c_0,\dots,c_k\} \subset \mathbb{Q}[x_1,\dots,x_{i-1}]$.
- *The* leading coefficient *of* p *w.r.t.* x_i *is* $ldcf_{x_i}[p] = c_k \in \mathbb{Q}[x_1,\dots,x_{i-1}]$.
- *The* resultant *of* p *and* q *w.r.t.* x_i *is a polynomial* $res_{x_i}[p,q] \in \mathbb{Q}[x_1,\dots,x_{i-1}]$ *so that for all* $s \in \mathbb{R}^{i-1}$*:*

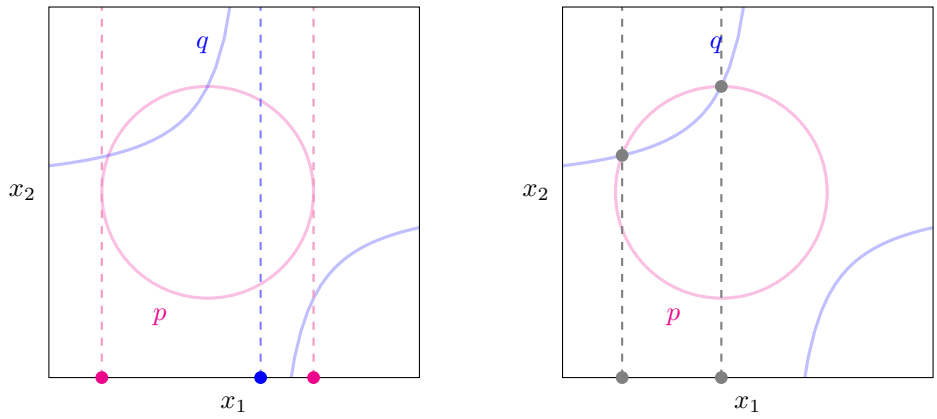
  $$res_{x_i}[p,q](s) = 0 \Leftrightarrow p(s,x_i) \text{ and } q(s,x_i) \text{ have a common complex root.}$$

- *The* discriminant *of* p *w.r.t.* x_i *is a polynomial* $disc_{x_i}[p] \in \mathbb{Q}[x_1,\dots,x_{i-1}]$ *so that for all* $s \in \mathbb{R}^{i-1}$*:*

  $$disc_{x_i}[p](s) = 0 \Leftrightarrow p(s) \in \mathbb{Q}[x_i] \text{ has a multiple complex root.}$$

The resultant can be computed as the determinant of the Sylvester matrix for the polynomials and the discriminant is equal to the resultant of p and its derivative with respect to x_i.

There is a helpful geometrical interpretation for those polynomial operations. For a polynomial $p \in \mathbb{Q}[x_1,\dots,x_i]$, a root $r \in \mathbb{R}^{i-1}$ of p's leading coefficient indicates a point where p's variety describes a singularity. This means that a part of the graph tends to infinity or negative infinity when approaching that root and thus, the root structure



(a) At the roots of $disc_{x_2}[p]$ (in red), the root structure of p changes due to turning points. The variety of q has a singularity at the root of $ldcf_{x_2}[q]$ (in blue).

(b) The resultant of p and q has its roots at $x_1 \approx -0.937$ and $x_1 = 0$ (marked grey). For these values, p and q have a common root.

Figure 2.4: Roots of the discriminant, leading coefficient (left) and resultant (right) of the polynomials p and q from Example 2.3.1.

of p changes at r as one of its roots "disappears". The roots of the discriminant of p indicate points where the root structure changes not due to singularities, but to self-intersections or turning points of the graph. Finally, the resultant of p and q describes the intersections of the respective graphs. This interpretation is illustrated in Figure 2.4.

It is important to note that we can only use these polynomial operations effectively if the respective input polynomials are square-free and pairwise coprime. That is, none of the polynomials has the square of a non-constant polynomial as a factor and no two polynomials share a common non-constant factor. Otherwise, the respective discriminant or resultant would be constantly zero and would not allow us to infer the desired information. Therefore, the *projection step*, in which resultants, discriminants and coefficients are gathered, is not applied directly to the polynomials of the current level, but to the set of their *irreducible* factors, which are square-free and coprime.

With these definitions in place, we can now describe the projection step of the levelwise method, which constitutes the transition from one level to the next. After the representation I_j of level j has been computed, the polynomials are *projected* to a set $P_{j-1} \subset \mathbb{Q}[x_1,\ldots,x_{j-1}]$ for which the underlying cell is then generated. If the polynomials in the projection fulfil certain properties of sign-invariance or order-invariance on the underlying cell S', then I_j is well-defined and correct over S'. Here, order-invariance is a stronger version of sign-invariance.

Definition 2.3.7 (Order-invariance)**.** *Let $i \in \mathbb{N}$, $p \in \mathbb{Q}[x_1,\ldots,x_i]$ and $r \in \mathbb{R}^i$. For $k \in \mathbb{N}$, let $D_k(p)$ denote the set of partial derivatives of p with total order k. The* order of p at r *is*

$$ord_r(p) := \min(\{k \in \mathbb{N} \mid \exists d \in D_k(p).\ d(r) \neq 0\} \cup \{\infty\})$$

Let $R \subseteq \mathbb{R}^n$, then p is order-invariant *on R if for all $r,r' \in R$ holds $ord_r(p) = ord_{r'}(p)$.*

Note that order-invariance implies sign-invariance.

To generate an underlying cell with the right properties, the algorithm proceeds with level $j-1$ as before on level j, though considering only the polynomials in P_{j-1}. After repeating this process (computing the level representation and projecting) for all levels below, the cell $S(I_1,\ldots,I_{j-1})$ provides the required properties of the polynomials in P_{j-1} and by design of the projection ensures the definedness and correctness of the description I_j.

To clarify the concept of projection, we first look at *McCallum's projection operator*, which was originally developed for the CAD method [McC98] and which forms the basis of the projection used in the levelwise approach.

Definition 2.3.8 (McCallum's Projection [McC98]). *Let $i \in \mathbb{N}$ with $i > 1$ and let $P \subset \mathbb{Q}[x_1,\ldots,x_i] \setminus \{0\}$ be a set of irreducible polynomials. Furthermore, we define $P_i = \{p \in P \mid lvl(p) = i\}$ and $P_{<i} = P \setminus P_i$. The* McCallum-projection *of P is*

$$proj_{mc}(P) := P_{<i} \cup \{disc_{x_i}[p] \mid p \in P_i\} \cup \{res_{x_i}[p,q] \mid p,q \in P_i\} \cup \bigcup_{p \in P_i} coeff_{x_i}[p].$$

If the discriminants, pairwise resultants and coefficients of all polynomials in P_i are order-invariant on a cell S, then the root structure of P_i does not change and as a result, the same indexed root expressions are defined over the entirety of S. Moreover, the graphs corresponding to the roots do not cross over S, thanks to the order-invariance of the resultants. This is the crucial property for the projection step, which we will now formalize using the concept of *analytic delineability*. The following definition uses two notions which we will not define formally, but only give a rough intuition, similar to what is described in [NSÁ+22]. An *analytic submanifold* of $\mathbb{R}^n$ is a non-empty subset $R \subseteq \mathbb{R}^n$ which for some $i \in [n]$ "looks locally like $\mathbb{R}^i$". A function $f : U \to \mathbb{R}$ with an open subset $U \subseteq \mathbb{R}^i$ as its domain is called *analytic* if it has a multiple power series representation about each point of U. For more detailed explanations, see [McC98]. For our purposes, it is enough to know that analytic submanifolds are cells, all the cells constructed by the levelwise method are analytic submanifolds and that analytic functions are continuous and have continuous partial derivatives of all orders.

Definition 2.3.9 (Analytic Delineability [McC98]). *Let $i \in \mathbb{N}$ with $i > 1$. A polynomial $p \in \mathbb{Q}[x_1,\ldots,x_i]$ of level i is called* analytically delineable *over an analytic submanifold $R \subseteq \mathbb{R}^{i-1}$ if there exist finitely many analytic functions $\theta_1,\ldots,\theta_k : R \to \mathbb{R}$ (for some $k \in \mathbb{N}$) so that*

- *$\theta_1(r) < \ldots < \theta_k(r)$ for all $r \in R$,*
- *for all $r \in R$ holds $realRoots(p(r,x_i)) = \{\theta_1(r),\ldots,\theta_k(r)\}$ and*
- *for each $j \in [k]$, the multiplicity of the root $\theta_j(r)$ in $p(r,x_i)$ is invariant with respect to $r \in R$.*

The intuition for this concept is in line with our prior considerations. If a polynomial $p(x_1,\ldots,x_i)$ is analytically delineable over an underlying cell $R \subseteq \mathbb{R}^{i-1}$, then we have a fixed number of different well-defined, continuous functions $(\theta_1,\ldots,\theta_k)$ with domain R. Each of these functions corresponds to an indexed root expression associated with p, which then is in turn also defined over R. McCallum's projection guarantees that if $proj_{mc}(P_j)$ is order-invariant on the underlying cell $R := S(I_1,\ldots,I_{j-1})$, then

the polynomials in P_j are analytically delineable on R and hence, I_j will be well-defined. In addition, the resultants in the projection ensure that no root functions of different polynomials cross at any value in R, which guarantees the correctness of the bounds.

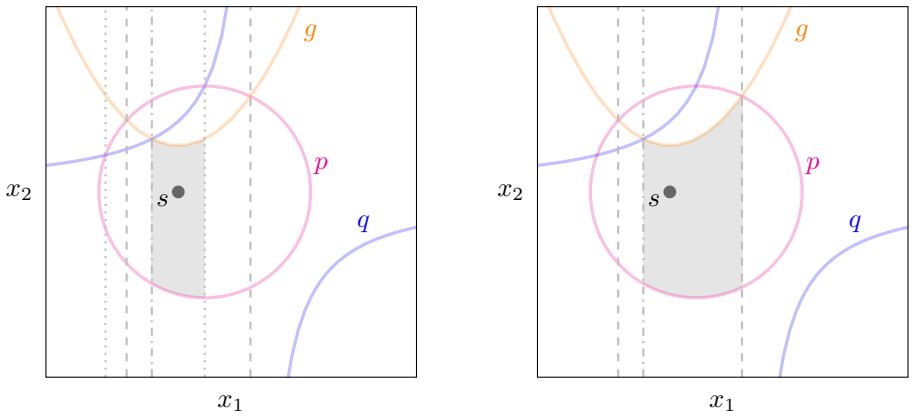
There is one case in which this statement fails, namely if any of the polynomials are *nullified* over a point within the underlying cell. Then, an order-invariant cell for the projection does not imply the wanted delineability properties.

Definition 2.3.10 (Nullification)**.** *Let $i \in \mathbb{N}$. A polynomial $p \in \mathbb{Q}[x_1,\dots,x_i]$ is* nullified *on a set $R \subseteq \mathbb{R}^{i-1}$, if $p(r,x_i) = 0$ for all $r \in R$.*

During the construction, the coefficients added to the projection can restrict the underlying cell in order to exclude regions where the polynomials are nullified. However, this does not work if nullification occurs at the underlying sample, that is, if there is $p \in P_j$ with $p(s_1,\dots,s_{j-1},x_j) = 0$. Since s must be included in the constructed cell, there is no chance to avoid the point of nullification and hence, McCallum's projection cannot be applied. This means that the method is not complete in general, but only for *well-oriented* sets of polynomials, where this kind of nullification does not occur (even after projection).

For our task, McCallum's projection actually computes more than is necessary. In particular, we only need that no root function crosses the bounds defined by I_j, but we may allow intersections of functions outside the cell. Accordingly, we can omit some of the resultants, which is very desirable as their computation is quite expensive in terms of runtime and memory. A second advantage of omitting resultants is that the resulting cell is potentially bigger, because fewer polynomials are restricting it. To illustrate how we can discard certain resultants, we revisit our running example.

Example 2.3.4. *We consider the known example, but with an additional polynomial $g = 2x_2 - 2x_1^2 - x_1 - 1$, whose variety is graphed in orange. McCallum's projection would compute the three pairwise resultants for p, q and g, resulting in the cell shown in Figure 2.5a. If we only compute the resultants of g with p and g with q and omit the resultant of p and q, we get a bigger, but still correct cell.*



(a) Using all pairwise resultants. (b) Using only resultants including g.

Figure 2.5: Sign-invariant cells computed for Example 2.3.4 with different sets of resultants. Dashed lines correspond to the roots of $res_{x_2}[g,p]$, dotted lines to the roots of $res_{x_2}[p,q]$ and the dash-dotted line indicates the root of $res_{x_2}[g,q]$.

The levelwise approach determines the set of necessary resultants using an *indexed root ordering*. Suppose that Ξ is the set of indexed root expressions defined at the underlying sample $s_{[j-1]} = (s_1, \ldots, s_{j-1})$ and each corresponding to a polynomial in P_j. Further, assume $I_j = (sector, l, u)$ for some $l, u \in \Xi$, meaning that the current level is a sector. Then we want to find a partial order on Ξ so that the ordering of any $\xi \in \Xi$ with l and u is consistent with their ordering at $s_{[j-1]}$. In particular, the partial order does not require comparability of such a ξ with any element different to the bounds. Moreover, we can use transitivity implicitly and thus only need a relation $\preceq$ whose transitive closure fulfils these requirements.

Definition 2.3.11 (Indexed Root Ordering [NSÁ+22]). *Let $i \in \mathbb{N}$, $P \subset \mathbb{Q}[x_1, \ldots, x_{i+1}]$ be a set of irreducible polynomials of level $i+1$ and $s \in \mathbb{R}^i$ be a sample such that no $p \in P$ is nullified over s. Moreover, let $\Xi \subseteq irExp(P,s)$ and $\preceq_s \subseteq \Xi \times \Xi$ be an ordering such that for all $\xi, \xi' \in \Xi$ holds $\xi \preceq_s \xi' \Leftrightarrow \xi(s) \leq \xi'(s)$. An* indexed root ordering (i.r.o.) *on Ξ for s is a relation $\preceq \subseteq \preceq_s$.*

Let additionally $s_{i+1} \in \mathcal{R}$ and either $I = (sector, l, u)$ for some $l, u \in \Xi \cup \{-\infty, \infty\}$ with $l(s) < s_{i+1} < u(s)$, or $I = (section, b)$ with $b \in \Xi$ and $b(s) = s_{i+1}$ respectively. An indexed root ordering $\preceq$ with transitive closure $\preceq^*$ matches *I if it fulfils the following properties:*

- *If $I = (section, b)$ then*

$$\begin{aligned} &\text{for all } \xi \in \Xi \text{ holds } (\xi(s) \leq b(s) \implies \xi \preceq^* b) \text{ and} \\ &\text{for all } \xi \in \Xi \text{ holds } (b(s) \leq \xi(s) \implies b \preceq^* \xi). \end{aligned}$$

- *If $I = (sector, l, u)$ then*

$$\begin{aligned} &\text{either } l = -\infty \text{ or for all } \xi \in \Xi \text{ holds } (\xi(s) \leq l(s) \implies \xi \preceq^* l) \text{ and} \\ &\text{either } u = \infty \text{ or for all } \xi \in \Xi \text{ holds } (u(s) \leq \xi(s) \implies u \preceq^* \xi). \end{aligned}$$

In this thesis, we only consider indexed root orderings that are constructed to match the description of the respective current level. Consequently, we assume this property implicitly whenever we use the term *indexed root ordering*. Note that there can be multiple possible indexed root orderings to choose from and in the presented approach, this choice is made by a heuristic. Nalbach et al. propose three heuristics for the sector case and one for the section case. We will focus on the sector case and one of the three heuristics in particular, which is called "biggest cell heuristic".

Definition 2.3.12 (Biggest Cell Heuristic). *Let P, s, I, Ξ and $\preceq_s$ be as in Definition 2.3.11. In the case of $I = (sector, l, u)$, we set $L := \{(\xi, l) \mid \xi \in \Xi \setminus \{l\}, \xi \preceq_s l\}$ if $l \neq -\infty$ and $L := \emptyset$ otherwise. Similarly, we set $U := \{(u, \xi) \mid \xi \in \Xi \setminus \{u\}, u \preceq_s \xi\}$ if $u \neq \infty$ and otherwise $U := \emptyset$. The* biggest cell heuristic *yields the indexed root ordering*

$$\preceq_{BC} = \begin{cases} L \cup U & \text{if } I = (sector, l, u), l, u \in \Xi \\ \{(\xi, b) \mid \xi \preceq_s b\} \cup \{(b, \xi) \mid b \preceq_s \xi\} & \text{if } I = (section, b). \end{cases}$$

Like in the example, the idea is that we only need to ensure the order compared to the boundaries of the cell. After an indexed root ordering $\preceq$ has been determined, the resultant between two polynomials p and q is only computed if there is a pair

$(\xi,\xi') \in \preceq$, where ξ belongs to p and ξ' belongs to q. One exception to this are the polynomials inducing the cell bounds. If both the upper and lower bound exist (i.e. they are not infinite) and they are induced by different polynomials, then the corresponding resultant is always computed regardless of the indexed root ordering. Moreover, note that several pairs in the i.r.o. can lead to the same resultant, as the polynomials may yield multiple root expressions. However, the resultant is only computed once because it only depends on the polynomials (not the root expression) and is symmetric (up to a constant factor).

There are many other small optimizations, which allow to discard most of the coefficients and some discriminants from the projection, especially in the section case. We do not describe these optimizations here, as they require complex case distinctions and also do not interfere with our work in Chapter 3. The resulting projection operator is denoted by $proj_{LW}$. Its utility is more nuanced and complex than McCallum's projection, as it does not simply guarantee delineability for all involved polynomials on the basis of order-invariance. Instead, it is defined via sets of properties for each of the involved polynomials and for the constructed cell, as well as rules for deriving these properties. This way, for example, only a weaker version of analytic delineability called *projective analytic delineability* is guaranteed for some of the polynomials and only sign-invariance instead of not order-invariance is required for some elements of the projection. The idea is to have more control over the required and guaranteed properties, hoping to reduce the size of the projection.

Finally, the levelwise method is defined in Algorithm 1. Note that in the original source, it is formulated much more abstractly and as a kind of proof system. However, for the purpose of this thesis, we disregard some of the subtleties and instead focus on the practical algorithm flow.

Algorithm 1: Levelwise single cell construction: `scc`(P,s)

Input : $P \subset \mathbb{Q}[x_1,\dots,x_i], s \in \mathcal{R}^i$
Output: $D = (I_1,\dots,I_i)$ or FAIL

$P_\perp := P$
for $j = i, i-1,\dots,1$ **do**
  $P_j := \{p \in P_\perp \mid lvl(p) = j\}$
  `/* check for nullification */`
  if $\exists p \in P_j : \ p(s_{[j-1]}) = 0$ **then**
    return $FAIL$
  $\Xi := irExp(P_j, s_{[j-1]})$
  Determine I_j from $s_{[j]}$ and Ξ
  if $i > 1$ **then**
    Determine an indexed root ordering $\preceq$ w.r.t. $s_{[j]}$ and Ξ
    Project to the next level w.r.t. $\preceq, I_j, s$: $P_\perp := proj_{LW}(P_\perp)$
return $(I_1,\dots,I_i)$

Theorem 2.3.1 (Correctness (Theorem 7.1 in [NSÁ+22])). *The levelwise single cell construction is correct. That is, if* `scc`$(P,s) \neq FAIL$*, then it returns a cell description* `scc`$(P,s) = D = (I_1,\dots,I_i)$ *with* $s \in S(D)$ *and* $S(D)$ *is* P*-sign-invariant.*

Going back to the context of MCSAT, we define the formula derived from a cell description.

Definition 2.3.13 (Extended Polynomial Constraint)**.** *Let $i, j \in \mathbb{N}$, $p \in \mathbb{Q}[x_1, \ldots, x_i]$ and $\xi = root_{x_i}[p,j]$ be an indexed root expression. Further, let x_k, $(k \in [n])$ be a variable and $\sim \in \{<, \leq, =, \neq, \geq, >\}$. Then*

$$x_k \sim \xi$$

is an extended polynomial constraint *with the following semantics:*

$$\forall s \in \mathbb{R}^n : (s \models x_k \sim \xi \Leftrightarrow \xi(s) \in \mathcal{R} \text{ and } s_k \sim \xi(s)).$$

Note that the free variables of ξ do not include x_i.

Definition 2.3.14 (Cell Description Formula)**.** *Let $i \in \mathbb{N}$. The formula derived from a cell description $D = (I_1, \ldots, I_i)$ is $\varphi_D := \varphi_1 \wedge \ldots \wedge \varphi_i$ so that for each $j \in [i]$ holds*

$$\varphi_j := \begin{cases} x_j = b & \text{if } I_j = (section, b) \\ true & \text{if } I_j = (sector, -\infty, \infty) \\ x_j < u & \text{if } I_j = (sector, -\infty, u) \\ x_j > l & \text{if } I_j = (sector, l, \infty) \\ x_j > l \wedge x_j < u & \text{if } I_j = (sector, l, u). \end{cases}$$

Chapter 3

Underapproximating Sign-Invariant Cells

One of the most time-consuming operations in the levelwise single cell construction is the computation of polynomial resultants. In the worst case, each resultant requires a quadratic number of polynomial multiplications with respect to the degrees of the involved polynomials [Duc00]. While this might be manageable for a single projection step, the situation becomes quite drastic when iterating projections like in the CAD method or in single cell construction. Then, the degree of the derived polynomials (w.r.t their main variable) grows doubly exponential with the number of levels and hence, the computation times for the respective resultants do so, too [BDE+16].

Consequently, reducing the number of resultants and discriminants in those projections has been one of the main research motivations in the field. In fact, McCallum's projection was already a significant improvement compared to the projection Collins originally used for CAD. The levelwise single cell construction now builds upon this and allows to cut down even further on the number of computed resultants, which is one of its main advantages over prior approaches. As we have seen in Section 2.3 of the previous chapter, this is achieved by using the concept of an *indexed root ordering*.

In fact, this optimization opens up yet another way of tackling the complexity of resultants. When using an indexed root ordering, resultants are not computed canonically for all pairs of involved polynomials, but only for a small subset of those pairs, depending on the ordering of their real roots at the underlying sample. As a result, the choice of the indexed root ordering influences not only the number of computed resultants, but also their complexity. Nalbach et al. addressed this and presented an alternative to the biggest cell heuristic, called *lowest degree barriers heuristic*, which tries to minimize the degrees of the polynomials in each induced resultant input pair. However, its advantages are limited by the degree distribution present in the current set of polynomials. If there is a low variance or if the root structure at the sample enforces certain expensive resultants, then the gain in efficiency is only marginal. Moreover, using the lowest degree barriers heuristic might lead to a smaller cell at the lower levels, when compared to the biggest cell heuristic.

Our goal now is to construct indexed root orderings for which the induced resultants always have at least one low-degree polynomial as input and can thus be computed more efficiently. As we have pointed out for the lowest-degree-barriers-heuristic, the polynomials of the current level set strong limitations for this kind of

optimization.

In this chapter, we show how to overcome those limitations by introducing artificial low-degree polynomials to the working set, which have a root at a favourable position. Then for the resulting indexed root ordering, each induced resultant is easier to compute and the degrees of the polynomials in the subsequent levels do not grow as quickly. However, the additional polynomials change the boundaries of the constructed cell, making it an underapproximation.

This idea is first illustrated by a naive approach and then generalized in Section 3.2, where we also prove its correctness and analyse its properties with regards to completeness. We then present which parts of our method allow variations and heuristics, before we provide a more complex variant using the multivariate Taylor expansion in Section 3.4.

3.1 A Naive Approach

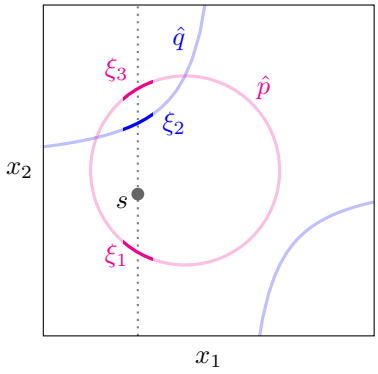
To illustrate our idea, we revisit the Example 2.3.1 from Chapter 2.

Example 3.1.1. *Instead of the usual input* $p = x_1^2 + x_2^2 - 1$ *and* $q = x_2 - 2x_1x_2 - 1$, *we consider the following polynomials:*

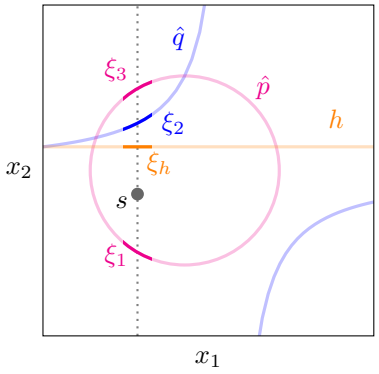
$$\hat{p} = 2^6(x_1^6x_2^6 + 1)p + 1 \text{ and } \hat{q} = 2^6(x_1^6x_2^6 + 1)q + 1.$$

Note that these polynomials have degrees $deg_{x_2}(\hat{p}) = 8$ *and* $deg_{x_2}(\hat{q}) = 7$ *in* x_2 *and that they are irreducible. They produce a graphically very slightly perturbed version of the problem in our running example, as can be seen in Figure 3.1a. In fact, the single cell construction would proceed in the same manner as shown before, only with* $\hat{p}$,$\hat{q}$ *instead of* p,q *and with slightly different real values for their roots. Importantly, the resultant of* $\hat{p}$ *and* $\hat{q}$ *will always be computed independently of the chosen indexed root ordering, because they give the lower and upper bound of the cell. However,* $res_{x_2}[\hat{p},\hat{q}]$ *is a very complex polynomial with degree* $deg_{x_1}(res_{x_2}[\hat{p},\hat{q}]) = 70$ *in* x_1 *and thus, its construction, determining its real roots and evaluating it on sample points will take a long time.*

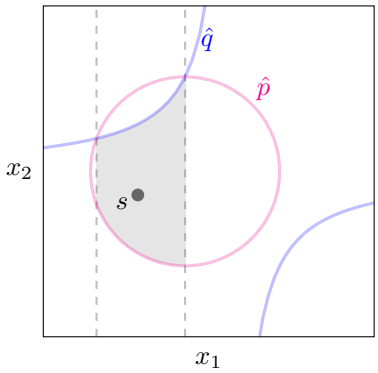
To avoid this expensive resultant, we can introduce the polynomial $h = x_2 - \frac{1}{4}$ *which has a root at* $(s_1, \frac{1}{4})$ *and thus becomes the new upper bound of the cell, as depicted in Figure 3.1b. Using the biggest cell heuristic now yields the indexed root ordering* $\preceq = \{(\xi_h,\xi_2),(\xi_h,\xi_3)\}$ *and accordingly, only the resultants* $res_{x_2}[h,\hat{p}]$ *and* $res_{x_2}[h,\hat{q}]$ *need to be computed, which is quite simple since* h *is linear. As a result, we get two resultants instead of one, but they have degree* 8 *and* 7 *in* x_1 *and thus are much easier to deal with in later computations. The drawback is that the constructed cell with the additional polynomial is much smaller in the* x_2*-direction. In fact, over the underlying cell for* x_1, *the originally constructed cell is equal to the maximal sign-invariant cell, which is not true for the new cell. In that sense, we only compute an underapproximation.*



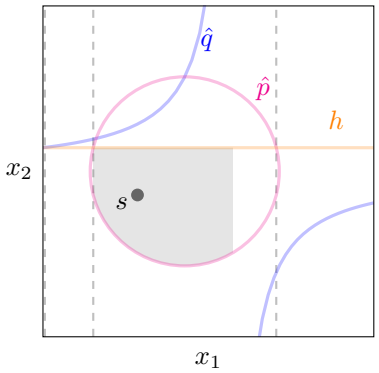
(a) The root structure is basically the same as in Example 2.3.1.



(b) The added polynomial h introduces a new root closer to the sample.



(c) The originally constructed cell without h. The bounds for x_1 are defined by the resultant of $\hat{p}$ and $\hat{q}$.



(d) The cell constructed with h. The dashed lines indicate the roots of the resultants related to h. Note that the upper bound for x_1 stems from the singularity of q.

Figure 3.1: Adding a new polynomial to the construction

In the leading example, we use a very simple construction for introducing artificial polynomials like h. The general idea is to use the biggest cell heuristic, but if one of the cell bounds has rather high degree, we create a new, approximate bound to avoid heavy resultant calculations. Assume we wanted to approximate the upper cell bound given by ξ_u at the underlying sample $s_{[i-1]}$ at level i. This means that we need to find a polynomial h which, at $s_{[i-1]}$, has a real root between the sample coordinate s_i and the bound $\xi_u(s_{[i-1]})$. For this purpose, we first choose a rational point $r \in (s_i, \xi_u(s_{[i-1]}))$ and then construct the artificial polynomial $h = x_i - r$. Now, the real roots of h are given exactly by $\{(s', r) \mid s' \in \mathbb{R}^{i-1}\}$ and in particular, we have $s_i < r < \xi_u(s_{[i-1]})$. As a result, the indexed root expression $root_{x_i}[h,1]$ now induces a constant upper bound on x_i because it is closer to s_i than ξ_u at the underlying sample $s_{[i-1]}$. The same idea can be applied for the lower bound.

After the artificial polynomials have been introduced to the working set, the indexed root ordering is constructed with the biggest cell heuristic, which now induces only resultants taking at least one of the simple bound polynomials as input. These

kinds of resultants are very easy to compute, as $res_{x_i}[p,x_i - r]$ essentially simplifies to $p(x_1,\ldots,x_{i-1},r)$, i.e. substituting r for x_i in p.

It is worth noting that the resulting approximated cell has a particular structure. If all bounds at each level were approximated, then the cell would be a box, i.e. the symbolic intervals $I_1,\ldots,I_i$ are in fact simple rational intervals, whose boundaries do not depend on the values of the variables at the lower levels. This is also why we call this approach *naive*. The construction of the artificial bounds does not take the original polynomials into account, only their roots at the sample.

The following algorithm is a modification of the levelwise single cell construction incorporating our naive approach.

Algorithm 2: Naive approximation approach: `naive-scc-apx(`P,s`)`

```
   Input  : P ⊂ ℚ[x_1,...,x_i], s ∈ ℛ^i
   Output: D = (I_1,...,I_i) or FAIL
1  P_⊥ := P
2  for j = i, i−1,...,1 do
3  |  P_j := {p ∈ P_⊥ | lvl(p) = j}
   |  /* check for nullification                                   */
4  |  if ∃p ∈ P_j : p(s_[j−1]) = 0 then
5  |  |  return FAIL
6  |  Ξ := irExp(P_j,s_[j−1])
7  |  Determine the lower bound ξ_l ∈ Ξ∪{−∞} and upper bound ξ_u ∈ Ξ∪{∞}
8  |  if ξ_l ≠ ξ_u then
   |  |  /* only approximate in the sector case                   */
9  |  |  if ξ_l = root_{x_i}[p,j] with deg_{x_i}(p) sufficiently large then
10 |  |  |  Let r_l = ξ_l(s_1,...,s_{j−1}) ∈ ℝ
11 |  |  |  Choose r ∈ (r_l, s_j)
12 |  |  |  P_j := P_j ∪ {x_j − r}
13 |  |  |  Ξ := Ξ ∪ {root_{x_j}[x_j − r,1]}
14 |  |  if ξ_u = root_{x_i}[p,j] with deg_{x_i}(p) sufficiently large then
15 |  |  |  Let r_u = ξ_l(s_1,...,s_{j−1}) ∈ ℝ
16 |  |  |  Choose r ∈ (s_j,r_u)
17 |  |  |  P_j := P_j ∪ {x_j − r}
18 |  |  |  Ξ := Ξ ∪ {root_{x_j}[x_j − r,1]}
19 |  Determine I_j from s_[j] and Ξ
20 |  if i > 1 then
21 |  |  Use the biggest cell heuristic to construct the i.r.o. ≺ w.r.t. s_[j] and Ξ
22 |  |  Project to the next level w.r.t. ≺, I_j, s: P_⊥ := proj_LW(P_⊥)
23 return (I_1,...,I_i)
```

The main difference to the original method is that if the current level is a sector and if its bounds would be defined by polynomials with high degree, we add a simple approximation inducing a new bound. This is done in lines 8 to 18 of the algorithm. Moreover, we fix the indexed root ordering to the biggest cell-heuristic. There are several aspects of this approach which we now want to consider more closely.

Sector vs. Section Case Notice that we only introduce artificial polynomials when the current level of the cell is a sector. If at level j there is $\xi_b \in \Xi$ with $\xi_b(s_{[j-1]}) = s_j$, then I_j will define a section and for every $s' \in S(I_1,\ldots,I_{j-1})$ in the underlying cell, there is a unique value $\xi_b(s')$ in I_j. Graphically speaking, there is no space to fit additional roots in between the cell boundary and the sample. Introducing a new polynomial with a root exactly at the sample would mean that the resultant with the section defining polynomial is also zero at the sample and thus, the next level would collapse to another section. To put this into more formal terms, we have

$$p \in \mathbb{Q}[x_1,\ldots,x_j],\, p(s_{[j]}) = 0 \Rightarrow res_{x_j}[p,x_j - s_j](s_{[j-1]}) = 0.$$

This behaviour is undesirable since we aim to produce cells that are as big as possible. Therefore, the section case is handled in the same way as in the original method.

Approximation Criteria So far, we described only vaguely in which situations this approximation approach is used by checking the condition whether the cell bounds are defined by a polynomial with "rather high degree". The reason is that there is no obvious way to characterize the instances in which simplifying the resultants is beneficial in the context of MCSAT. In fact, this may depend on other implementation details like the efficiency of resultant computations and real root isolation. Therefore, we only make heuristic choices, depending on the degrees of the involved polynomials. As we will see in Chapter 4, approximating polynomials with degree five or higher seems to be a good choice in practice, at least within the implementation of MCSAT that we used. While this threshold might not seem that high, recall that the degrees of the polynomials grow doubly exponential in the subsequent levels and hence, it can be worth to approximate early to mitigate that growth. There are many other criteria which take not only the degree of the respective bounding polynomial into account and we will explain some of them later in Section 3.3. For now, we use the simple threshold criterion.

Cell Size As can be seen in Example 3.1.1, the introduction of artificial polynomials leads to an underapproximation of the cell. We can formalize this observation as follows:

Theorem 3.1.1. *Let $i \in \mathbb{N}$, $P \subset \mathbb{Q}[x_1,\ldots,x_i]$ be a set of polynomials at level i and $s \in \mathcal{R}^i$. Moreover, let $D = (I_1,\ldots,I_i) = \texttt{scc}(P,s)$ be the cell description returned by the original levelwise method and $D' = (I'_1,\ldots,I'_i) = \texttt{naive-scc-apx}(P,s)$ be the cell description returned by the modified method, so that at level i, the bounds have been approximated. For the maximal P-sign-invariant cell $S_{max} \subseteq \mathbb{R}^i$ around s we have*

$$S(D) = S_{max} \cap (S(I_1,\ldots,I_{i-1}) \times \mathbb{R}) \text{ and } S(D') \subsetneq S_{max} \cap (S(I'_1,\ldots,I'_{i-1}) \times \mathbb{R})$$

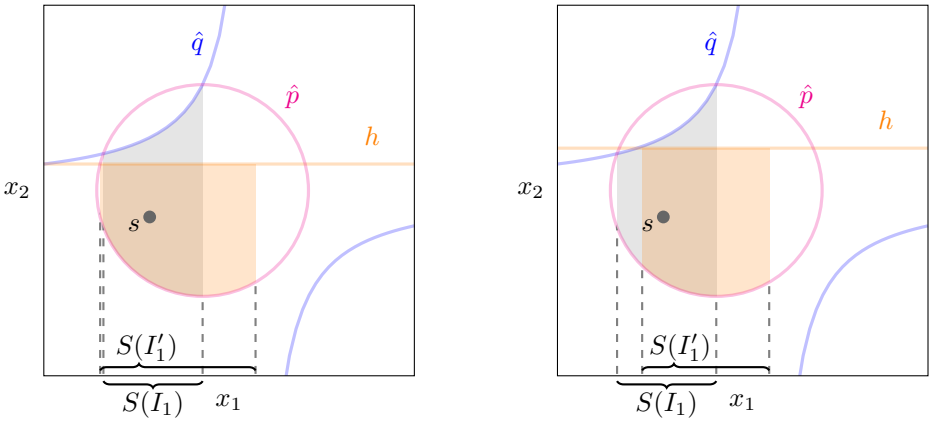
Proof idea. The first part of the statement follows obviously by construction of the cell. For the second part, observe that any point between an artificially created root and the original cell bound would be contained in S_{max}, but not in $S(D')$. □

This property can greatly harm the quality of the produced cells for their usage in MCSAT. Not only will a smaller cell exclude fewer possible assignments from the search, but the artificial polynomials introduce cell boundaries which are less closely tied to the actual problem structure, as they are only vaguely related to the original set of polynomials. This is the price we have to pay for a faster cell computation and,

depending on the given instance, that price might be too high and our approximation approach might not be worth it. This is why we do not always use the simplified cell bounds but only when certain criteria are met.

While our method always yields a smaller interval for the top level of the cell, compared to the original method, the lower levels are a bit more forgiving. In fact, it can happen that we produce a larger underlying cell because we avoid certain resultants. We illustrate this observation with another example.

Example 3.1.2 (Size of the Underlying Cell)**.** *Consider the leading example of this chapter again. Depending on where we choose the artificial root r, the underlying cell $S(I_1')$ of the approximation can be larger or smaller than the underlying cell $S(I_1)$ computed without approximation. Moreover, neither $S(I_1') \subseteq S(I_1)$ nor $S(I_1') \supseteq S(I_1)$ hold necessarily. This is illustrated in Figure 3.2.*



(a) For $r = \frac{1}{4}$, we have $S(I_1) \subsetneq S(I_1')$.

(b) For $r = \frac{2}{5}$, there is no subset relation between $S(I_1)$ and $S(I_1')$.

Figure 3.2: Comparison of the underlying cells $S(I_1)$ and $S(I_1')$ for different positions of the artificial root. $S(I_1)$ results from the original construction and $S(I_1')$ results from introducing the approximation polynomial $h = x_2 - r$ on the top level. Additionally, the original two-dimensional cell (grey) and approximated cell (orange) are indicated.

This concludes our presentation of the naive approximation approach. Our next step is to lift these ideas to a more general method with several possibilities for variations. Showing the correctness of the generalized approach will then also yield the correctness of the naive one.

3.2 General Formulation

In the previous section, we pointed out some parts of the naive approximation method that can be easily varied, but the constructed artificial polynomials and the resulting indexed root ordering were fixed. However, we will show that one can introduce any number of polynomials on each level of the single cell construction method and then choose an arbitrary indexed root ordering without damaging the correctness of the produced cell.

Definition 3.2.1 (Level Approximation)**.** *A* level approximation function *is a function*

$$\texttt{apx-level}(P,s) : \bigcup_{i\in\mathbb{N}} (\mathcal{P}(\mathbb{Q}[x_1,\ldots,x_i]) \times \mathcal{R}^i) \to \bigcup_{i\in\mathbb{N}} \mathcal{P}(\mathbb{Q}[x_1,\ldots,x_i])$$

so that for all $i \in \mathbb{N}$ *it holds*

$$P \subset \mathbb{Q}[x_1,\ldots,x_i] \text{ and } s \in \mathcal{R}^i \Rightarrow \texttt{apx-level}(P,s) \subset \mathbb{Q}[x_1,\ldots,x_i].$$

That is, for each set of polynomials at level i *with a corresponding sample, it returns a set of* approximation polynomials *of level at most* i*.*

Obviously, our naive approach can be formulated with a level approximation function which gives for each cell bound at level i a polynomial of the form $x_i - r \in \mathbb{Q}[x_1,\ldots,x_i]$. Note that this definition allows to construct any number of newly introduced polynomials and also does not restrict the position of their real roots. The important property is that the approximation polynomials do not have a higher level than the input and can thus be used in the corresponding level of the single cell construction. Now, we can formulate a more general algorithm with respect to some level approximation function `apx-level`(P,s).

Algorithm 3: General approximation approach: `scc-apx`(P,s)

Input : $P \subset \mathbb{Q}[x_1,\ldots,x_i], s \in \mathcal{R}^i$
Output: $D = (I_1,\ldots,I_i)$ or FAIL

$P_\perp := P$
for $j = i, i-1, \ldots, 1$ **do**
  $P_j := \{p \in P_\perp \mid lvl(p) = j\}$
  `/* check for nullification */`
  if $\exists p \in P_j : \; p(s_{[j-1]}) = 0$ **then**
    return $FAIL$
  $\Xi := irExp(P_j, s_{[j-1]})$
  Determine the lower bound $\xi_l \in \Xi \cup \{-\infty\}$ and upper bound $\xi_u \in \Xi \cup \{\infty\}$
  if $\xi_l \neq \xi_u$ **then**
    `/* only approximate in the sector case */`
    $H :=$ `apx-level`$(P_j, s_{[j]})$
    $P_j := P_j \cup H$
    $\Xi := \Xi \cup irExp(H, s_{[j-1]})$
  Determine I_j from $s_{[j]}$ and Ξ
  if $i > 1$ **then**
    Determine an indexed root ordering $\preceq$ w.r.t. $s_{[j]}$ and Ξ
    Project to the next level w.r.t. $\preceq, I_j, s$: $P_\perp := proj_{LW}(P_\perp)$
return $(I_1,\ldots,I_i)$

Correctness

The correctness of this algorithm follows from two important observations.

Lemma 3.2.1. *Let* $i \in \mathbb{N}$, $s \in \mathcal{R}^i$ *and* $P,H \subset \mathbb{Q}[x_1,\ldots,x_i]$*. If* $P \cup H$ *is sign-invariant on a cell* $S \subseteq \mathbb{R}^i$*, then so is* P*.*

Proof. This is an immediate result of the definition of sign-invariance. □

Lemma 3.2.2. *Let $i \in \mathbb{N}$, $s \in \mathcal{R}^i$ and $P \subset \mathbb{Q}[x_1,\dots,x_i]$. For all $j \in [i]$, let the set of approximation polynomials introduced by* `apx-level`$(P_j,s_{[j]})$ *at level j of the modified single cell construction be denoted by $H_j \subset \mathbb{Q}[x_1,\dots,x_j]$. For $H := \bigcup_{j\in[i]} H_j$, we have*

$$\texttt{scc-apx}(P,s) = \texttt{scc}(P \cup H,s).$$

Proof. To obtain this result, observe that at level j of the original single cell construction, only polynomials of level exactly j are considered. Therefore, any polynomial $p \in P$ of level j does not affect the construction of $I_{j+1},\dots,I_i$. As a result, both `scc-apx`(P,s) and `scc`$(P \cup H,s)$ compute the description I_j and the projection of level j based only on the polynomials $P \cup \bigcup_{k=j}^{i} H_k$. Since the modified algorithm does not differ from the original one after introducing the approximation polynomials, they do indeed yield the same description and projection. □

Theorem 3.2.3 (Correctness of the Modified Single Cell Construction)**.** *Let $i \in \mathbb{N}$, $s \in \mathcal{R}^i$ and $P \subset \mathbb{Q}[x_1,\dots,x_i]$. If* `scc-apx`$(P,s) = D \neq FAIL$, *then $S(D)$ is a P-sign-invariant cell with $s \in S(D)$.*

Proof. With the second lemma, we get that there is a set $H \subset \mathbb{Q}[x_1,\dots,x_i]$ with $S(D) =$ `scc`$(P \cup H,s)$. Since the original method is correct, it follows that $S(D)$ is a $(P \cup H)$-sign-invariant cell with $s \in S(D)$. Applying the first lemma then yields the wanted P-sign-invariance. □

There is actually an even more general version of the method in which the indexed root ordering and the computation of the symbolic interval do not need to consider all of the indexed root expressions belonging to the artificial polynomials. One could, for example, only use one particular artificial root expression to approximate the cell bound and disregard the other roots of the corresponding polynomial. Proving the correctness of this version is more involved and relies on techniques used in the proof for the original levelwise single cell construction. In fact, Nalbach et al. accounted for this idea in their definition of an indexed root ordering and their algorithm formulation based on properties and derivation rules. A closer look at the Rules 6.9 - 6.10 of [NSÁ+22] and the corresponding Lemmas reveals that they already implicitly allow this kind of modification, though it is not used in the original paper. As we focus on variations of the presented approximation method that use only linear polynomials, which only induce a single root expression anyway, we will not provide a detailed proof here.

Completeness

When investigating the completeness properties of our approach, we need to distinguish two different aspects. Firstly, the completeness of the modified single cell construction itself and secondly, its effect on the completeness of MCSAT. Concerning the first aspect, it is easy to see that the underapproximating single cell construction unfortunately inherits the incompleteness of the original method.

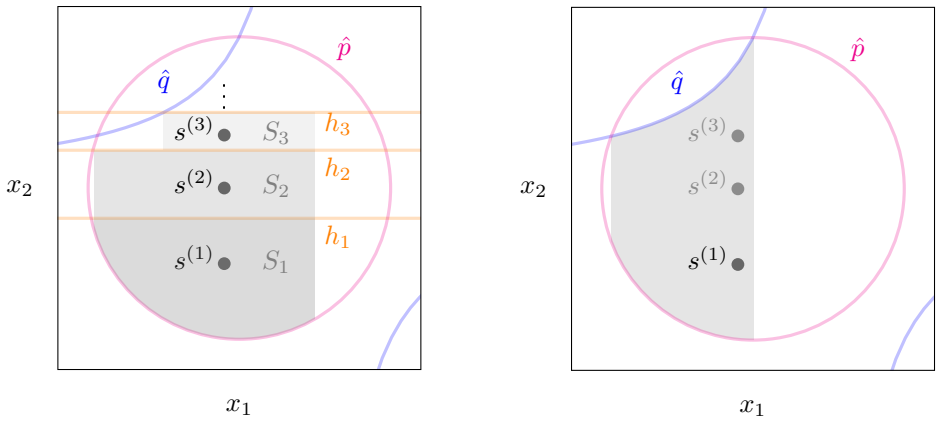
Theorem 3.2.4. *The modified single cell construction* `scc-apx` *is not complete. That is, there are $i \in \mathbb{N}$, $P \subset \mathbb{Q}[x_1,\dots,x_i]$ and $s \in \mathcal{R}^i$ with* `scc-apx`$(P,s) = FAIL$.

Proof. This follows from the incompleteness of the original levelwise single cell construction. If P is not well-oriented, then at some point in the algorithm, a polynomial in the projection is nullified and thus, the method fails. □

Interesting questions arising in this context are whether the artificial polynomials can lead to more cases of nullification or whether one can actively avoid nullification by constructing the approximations in a certain way. Obviously, no reasonable level approximation function should yield any nullified polynomials and this can, in fact, be easily avoided. Moreover, we can ensure that the resultants related to the approximation polynomials are not nullified either. However, it is not trivial to decide what impact an artificial polynomial will have on the lower levels after multiple projection operations. While our experimental results suggest that there is no significant increase in nullifications for the tested variants, we do not have a general, proved statement.

Application in MCSAT Employing the modified single cell construction in the explanation function of MCSAT at first yields another incompleteness result.

Example 3.2.1 (Completeness)**.** *The following is illustrated in Figure 3.3. Assume that $\hat{p}$ and $\hat{q}$ are derived from a set of literals which, together with the current sample $s^{(1)} = (\frac{1}{10}, -\frac{1}{2})$, form a theory conflict in MCSAT. We know that any other point s' in the maximal $\{\hat{p},\hat{q}\}$-sign-invariant cell S_{max} around $s^{(1)}$ will lead to the same conflict. In particular, MCSAT will derive the same polynomials $\hat{p},\hat{q}$ from the conflicting literals. Employing the naive underapproximating single cell construction yields a cell S_1, so that an artificial polynomial h_1 induces the upper bound for x_2. This cell is now excluded from the further search process of MCSAT and the algorithm backtracks, e.g. to the level for x_2. While the partial assignment for x_1 stays the same, a new value for x_2 could be chosen, which is outside of S_1 but still within S_{max}. Consequently, the same conflict occurs and a cell S_2 is constructed, again introducing an artificial polynomial h_2. This process can be repeated indefinitely as the cell bounds induced by the artificial polynomials are always strictly between the sample and the actual bound of S_{max}. Depending on what values MCSAT chooses for the partial assignments, it might not terminate when using the modified single cell construction. This behaviour does not occur with the original method, as it produces cells with a maximal top-level description, as shown in Theorem 3.1.1.*



(a) MCSAT chooses samples $s^{(1)}$,$s^{(2)}$ ... which converge towards the actual cell bound, but never reach it.

(b) The cell produced by the original level-wise method already includes all $s^{(k)}$, $k \in \mathbb{N}$ and hence avoids that problem.

Figure 3.3: Incompleteness of MCSAT when using underapproximation.

The example illustrates a great disadvantage of underapproximating the cells, namely that MCSAT might need to compute significantly more cells than normally needed, even leading to non-termination. It is important to note that such an infinite sequence of samples and cells can occur in general and not only for certain versions of the naive approximation. While the example gives a nice intuition for the incompleteness result, we can also describe it more formally, connecting it to the work of de Moura and Jovanović. They showed that MCSAT is complete if the used explanation function fulfils a *finite basis requirement*. That is, for every input formula φ, there exists a finite set $\mathbb{B}$ of theory literals so that in any run of MCSAT on φ, the clauses produced by the explanation function contain only literals from $\mathbb{B}$.

The original levelwise method yields an explanation function which indeed has this property, as long as we assume a common fixed variable ordering for all generated cells. The new literals it introduces are extended polynomial constraints, comparing a variable to a polynomial that has been constructed by repeatedly computing resultants, discriminants or coefficients of the polynomials present in the original literals. For a given starting formula, the set of all these polynomials is finite. When the new literals are involved in another conflict, the explanation function will again only yield literals built from the same set of polynomials. However, when we use the approximation method and introduce artificial polynomials, which also depend on the current sample, then we cannot assure the existence of a finite basis.

The solution to this problem is to restrict the level approximation function used in our modified method. If it yields only finitely many new approximations in any run of MCSAT, then we can guarantee completeness again (except in instances where nullification occurs). For this purpose, we let it take additional status information about the solving process in MCSAT into account. Then, the function only approximates a cell bound if this information meets certain criteria. With an appropriate choice of criteria and status information, we can ensure that only finitely many approximation polynomials will be generated.

Theorem 3.2.5. *Let $f(P,s;\mathit{info})$ be a level approximation function, which takes an additional parameter set info as input. For an arbitrary run of MCSAT, let $\mathit{Inputs} = \{(P,s,\mathit{info}) \mid f(P,s;\mathit{info})$ is called in the run$\}$. If $f(P,s;\mathit{info}) \neq \emptyset$ holds only for finitely many $(P,s;\mathit{info}) \in \mathit{Inputs}$, then the run terminates.*

Proof. If $f(P,s;\texttt{info}) \neq \emptyset$ for only finitely many inputs, then there is a point in the run after which the explanation function does not approximate any more. Therefore, all subsequent cells are computed essentially with the original levelwise construction. As only finitely many approximations have been added, the non-approximating single cell construction will still ensure a finite basis, though a larger one than for the initial input of MCSAT. Thus, the run terminates. □

We have implemented several realizations of this idea, which we will present in the next section, along with heuristics for other parts of the resulting algorithm.

3.3 Implemented Heuristics

The original levelwise single cell construction has been implemented in SMT-RAT, an open source C++ toolbox for SMT solving [SMTb, CKJ+15]. We built upon this basis and integrated our approach into that implementation. While we have shown more general results in theory, we have only put variants into practice that do not

differ too much from the naive approach. Still, there are several aspects which leave room for variability. The level approximation functions of our implemented methods all follow the same general structure:

- First, apply the criteria used to ensure termination. If they fail, do not generate any artificial polynomials.
- Then apply additional criteria to the polynomials of the current level to determine whether it is worth approximating the upper or lower bound of the cell. For example, check whether the degree of the polynomials inducing the cell bounds is greater than five.
- If one of the bounds meets the criteria, choose a point between the sample and the root defining that bound.
- For each of those points, generate an artificial polynomial that has a root at exactly that point.

That way, we only ever use artificial polynomials to simplify resultants with the cell bounds and accordingly, all our implemented methods use the biggest cell heuristic for computing the indexed root ordering. It would, of course, be possible to introduce polynomials with roots at other advantageous positions and then use other indexed root orderings, but this was not yet part of our considerations.

Termination Criteria

We start by providing some implementations of the strategy for ensuring termination given in Theorem 3.2.5. That is, we use criteria on additional status information to limit the number of approximations.

Number of Approximated Cells Probably the simplest idea is to count the number of (underapproximated) cells and to generate approximation polynomials only as long as that number is below a fixed threshold. In practice, one would not check this criterion in each call of the level approximation function, but rather once before the entire cell computation.

For more advanced criteria, we use additional properties of the level approximation function, which naturally arise from the desire to simplify the computations.

Definition 3.3.1 (Degree-bounded)**.** *A level approximation function* $f(P,s;\mathit{info})$ *is called* degree-bounded *if there is a fixed* $d \in \mathbb{N}$*, so that for all* $i \in \mathbb{N}$*,* $s \in \mathcal{R}^i$*,* $P \subset \mathbb{Q}[x_1,\ldots,x_i]$ *and additional status information* info*, we have*

- $deg_{x_j}(h) < d$ *for all* $h \in f(P,s;\mathit{info})$*,* $j \in [i]$ *and*
- $(\forall p \in P.\ deg_{x_i}(p) < d) \Rightarrow f(P,s;\mathit{info}) = \emptyset$.

This property expresses that the approximation polynomials do not have arbitrarily large degree, which is reasonable since we want to simplify resultants. Moreover, if all polynomials at the current level are already rather simple, then there is no approximation needed. When only approximating cell bounds induced by polynomials of degree five or higher, then the level approximation function corresponding to our

naive approach is degree bounded with $d \in \{2,3,4,5\}$. The benefit of this property is that the artificial polynomials alone do not cause any further approximations. That is, if MCSAT encounters a conflict consisting of literals derived only from artificial polynomials, then the computed cell will not be underapproximated and no new artificial polynomials are introduced.

Dynamic Threshold Degree-bounded level approximation functions allow for a more dynamic version of the criterion counting the approximated cells. Given a monotone function $g : \mathbb{N} \to \mathbb{N}$ with $n < m \Rightarrow g(n) < g(m)$ and the number n_{cells} of cells, we only introduce artificial polynomials if the current level contains a polynomial p with $deg(p) > g(n_{cells})$. Intuitively, the higher the degree of the polynomials, the more approximations we allow. This is motivated by the idea that for very high degree polynomials, the gain in efficiency might still outweigh the overhead of producing more cells. Since the level approximation function is degree-bounded, each run of MCSAT will reach a point at which all polynomials have a degree less than $g(n_{cells})$ and hence, no more artificial polynomials are generated.

Number of Approximations per Constraint Another option using degree-bounded functions aims to tackle the behaviour shown in the example, while being more adaptive to complex problem structures. If one only considers the overall number of approximated cells, it might happen that, after the threshold is reached, a conflict is encountered which consists of literals that were not involved in any of the prior conflicts. Using the underapproximation may be more efficient in this case, but is not even considered any more. To address this, one can limit the number of approximated cells for each constraint individually. That is, the level approximation function takes as status information the number of times each literal was involved in a conflict. If this number exceeds a certain threshold for any of the literals involved in the current conflict, then no artificial polynomials are generated. Degree-boundedness now ensures that each run of MCSAT will reach a point at which each conflict either contains a literal with exceeded threshold or the degrees of the polynomials in the corresponding cell construction are so small that no approximation is needed.

Approximation Criteria

The second variable aspect of the method is the approximation criterion, which is a heuristic to decide when an approximation will be useful. This is only a heuristic as there is no obvious or simple way to determine what resultants will be computed in the lower levels of the cell and how exactly the approximation will affect the solving process in MCSAT.

Degree of the Approximated Polynomial A simple choice, which we already presented, is to introduce an artificial polynomial only if the polynomial inducing the cell bound has a degree exceeding some threshold. Of course, the concrete value of that threshold impacts how well the method performs in the context of MCSAT. If it is too high, the approximation might never be used and there is no benefit to it. On the other hand, if it is too low, the method will approximate cells that could have been computed efficiently anyway and thereby just produce unnecessary overhead. In practice, a value of five performed quite well.

Estimating Overall Resultant Complexity An alternative idea is trying to estimate the overall complexity of the resultants that would be computed without approximation. When using the biggest cell heuristic for the indexed root ordering, it is fairly easy to determine for which pairs of polynomials a resultant would be added to the projection. A rough and easy to calculate guideline value for the complexity of a resultant of polynomials p and q is the product of their degrees, as this is the upper limit for the degree of the resultant. Consequently, one could use a criterion which allows approximation only if one of the potential resultants is deemed too expensive based on the guideline. However, it is worth noting that the reduction of resultant complexity is not the only benefit of the approximations. The simpler cell description can also speed up calculations and be helpful in cases where this criterion would not support an approximation.

Choice of the Artificial Root

A step of the algorithm that was kept vague in the description of the naive approach is the choice of the artificial root between the actual bound and the sample. The simplest method would be to use the midpoint, that is, if $(s_1,\ldots,s_j)$ is the sample and ξ is the indexed root expression corresponding to the bound:

$$r_{mid} = \frac{1}{2}(s_j + \xi(s_{[j-1]}))$$

However, this choice is not always valid because s_j or $\xi(s_{[j-1]})$ may be irrational and thus so can their midpoint. In that case, an artificial polynomial like $h = x_j - r_{mid}$ does not have rational coefficients and is hence unusable for our algorithms.

Rational Midpoint A simple workaround uses the representation of real algebraic numbers as roots of a polynomial within a certain interval with rational boundaries (see Def. 2.1.5). To find a rational point between two algebraic numbers, one can take the midpoint of those boundaries. More precisely, if $\alpha = (p,I)$ and $\beta = (q,J)$ are algebraic numbers with $\alpha < \beta$ and $I \cap J = \emptyset$, then we have

$$r'_{mid} = \frac{1}{2}(a_2 + b_1), \text{ for intervals } I = (a_1,a_2) \text{ and } J = (b_1,b_2).$$

Usually, the intervals in the representation are obtained by the procedure for real root isolation and they might overlap. In order to find a rational point between the algebraic numbers, we would then first need to refine the intervals until they are disjoint.

In practice, the rational midpoint is still not a good choice. SMT-RAT uses an implementation of rational numbers with arbitrary precision arithmetic, which entails that the representation of these numbers can become rather large. Consider two rationals $\frac{a}{b}$ and $\frac{c}{d}$ with coprime denominators b and d, which need $log(b)$ and $log(d)$ bits of storage. The denominator of the midpoint $\frac{ad+cb}{2bd}$ will need $log(b) + log(d) + 1$ bits and hence, the representation of numbers in our algorithm would grow and grow. This is even more severe when evaluating polynomials on the rational numbers, as $(\frac{a}{b})^k$ will need $k \cdot log(b)$ bits to store even the denominator. Accordingly, computations will also take a longer time. We will see later that this indeed has a significant impact on the performance of the algorithm.

Simple Representation Consequently, it is better not to use the midpoint, but instead choose a rational number with a representation that is as simple as possible. This can be achieved by utilizing the so called *Stern-Brocot tree* [Ste58, Bro61, GKPL94], which is a binary search tree containing all positive rational numbers. The path from the root ($\frac{1}{1}$) to any number q consists of fractions that have a smaller denominator than q and generally converge towards it. We can easily adapt its concepts to account for negative rational numbers, assuming that they are given in the form $\frac{z}{m}$ with $z \in \mathbb{Z}$ and $m \in \mathbb{N} \setminus \{0\}$. The following procedure then lets us find a rational number with rather simple representation within a given rational interval:

Algorithm 4: Simple representation point: `choose-point-sr(`q_1,q_2`)`

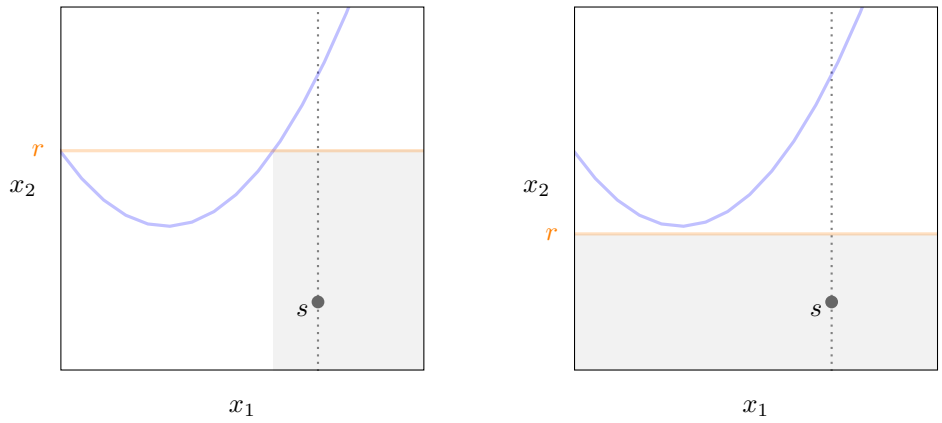
Input : Reduced fractions q_1, $q_2 \in \mathbb{Q}$ with $q_1 < q_2$
Output: $r \in (q_1,q_2)$ with a simple representation

```
if ∃z ∈ ℤ. z ∈ (q1,q2) then
    return some such z
Initialize n_l := ⌊q1⌋, d_l := 1 and n_r := 1, d_r := 0
Set n_c := n_l + n_r and d_c := d_l + d_r
c := n_c/d_c
while c ∉ (q1,q2) do
    if c < q1 then
        Set n_l := n_c and d_l := d_c
    if c > q2 then
        Set n_r := n_c and d_r := d_c
    n_c := n_l + n_r, d_c := d_l + d_r and c := n_c/d_c
return c
```

The procedure makes use of the *mediant* of two fractions, which is the sum of their numerators divided by the sum of their denominators and which always lies between the two fractions. In the Stern-Brocot tree, the children of each node are also obtained by taking the mediant of that node with some of its ancestors. The advantage of using the mediant instead of the midpoint is that it produces a smaller representation, where the denominator only needs $log(d_1 + d_2) < log(d_1) + log(d_2)$ bits of storage.

"Best" of Multiple Candidates A potential drawback of the simple representation method may be that the position of the artificial root can be rather arbitrary, while having more control over the position could allow for choices that are beneficial in other ways. Depending on the cell's boundary that is to be approximated, we might want to choose a root closer to or further away from it. Figure 3.4 illustrates this on an example, showing that the position of the artificial root will affect the size of the cell at the lower levels. In extreme cases, it can make the difference between a bounded and an unbounded level.

Therefore, it might be beneficial to look for an artificial root which will maximize the size of the cell. However, we cannot calculate the size of the cell without processing the lower levels and thus, we need to fall back to rough heuristics. One possibility for estimating the cell size is to only consider the current and next level. For a given artificial root r, we can construct the approximation polynomial and compute its resultant with the actual cell bound. Isolating the roots of the resultant will give an



(a) The root r is chosen rather close to the original bound defined by p. While this leads to a larger interval I_2 for x_2, the underlying cell for x_1 is bounded by the additional resultant.

(b) If r is chosen closer to s, then the cell is smaller in the x_2-direction, but unbounded on the lower level. This can be more desirable, as it simplifies the cell description and arguably produces a bigger cell.

Figure 3.4: Cells generated with the naive approach using different positions of the artificial root. The line in blue depicts a root function defining the boundary of a cell.

overestimation of the cell bounds at the next level. This yields a rough guideline for the quality of the artificial root. Since we cannot easily compute an exact value for r that would maximize the resulting estimated two-dimensional cell, we simply test several different values and take the best one. A possibility to define and compare the estimated cell sizes when using the naive approach is as follows:

Definition 3.3.2 (Root Position Heuristic)**.** *Let $i \in \mathbb{N}$ with $i > 1$, $s \in \mathcal{R}^i$ be a sample point and let $p \in \mathbb{Q}[x_1,\dots,x_i]$ define the upper or lower cell bound at level i by its root $r_p \in \mathbb{R}$ with $p(s_1,\dots,s_{i-1},r_p) = 0$. If $r_p > s_i$, let $J = (s_i,r_p)$ and otherwise $J = (r_p,s_i)$. For $r \in J$, we set $N_r := realRoots(res_{x_i}[p,x_i - r](s_1,\dots,s_{i-2}))$ and define the* heuristic size estimation *$hse(r,p,s)$ as follows:*

- *If there are $l,u \in N_r$ with $l < s_{i-1} < u$ and no element of N_r lies between l and u, then $hse(r,p,s) = |s_i - r| \cdot |u - l| \in \mathbb{R}$.*
- *If $N_r \neq \emptyset$ and either $\forall a \in N_r :\ s_{i-1} < a$ or $\forall a \in N_r :\ s_{i-1} > a$ holds, then $hse(r,p,s) =$ **`half-unbounded`**.*
- *If $N_r = \emptyset$, then $hse(r,p,s) =$ **`unbounded`**.*

*Let $r,r' \in J$ and $h := hse(r,p,s)$ and $h' := hse(r',p,s)$. We can compare h and h' by a partial ordering on the set $\mathbb{R} \cup \{$**`half-unbounded`**, **`unbounded`**$\}$. Elements of $\mathbb{R}$ are ordered as usual. For all $a \in \mathbb{R}$ we have **`unbounded`** $>$ **`half-unbounded`** $> a$. We say that r* provides a better heuristic value *than r' if $h > h'$. In the cases $a = a' =$ **`half-unbounded`** or $a = a' =$ **`unbounded`**, the values $|r - s_i|$ and $|r' - s_i|$ can be used as a tie break.*

To illustrate this idea, we revisit the example introduced in Figure 3.4.

Example 3.3.1. *The following process is depicted in Figure 3.5. We first choose a root r_1 rather close to the actual bound, construct the corresponding approximation polynomial h_1 and compute the resultant with the polynomial inducing the bound. The two-dimensional estimated cell E_1 is bounded in x_2-direction by r_1 and s_2 and in x_1-direction by the roots of that resultant. Notice that we bound it by s_2 with the reason that we do not take any potential lower bound into account. This makes our estimation easier to compute, but also less accurate. Next, we could try out two more roots r_2,r_3 at different positions and recognize that the estimation for r_3 is fully unbounded in x_1-direction and is hence more favourable.*

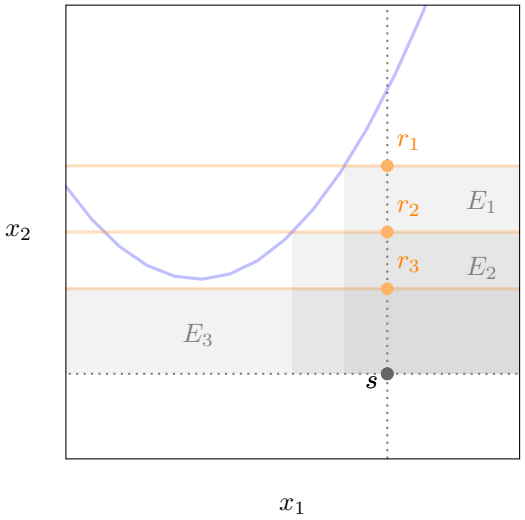


Figure 3.5: Illustration of the heuristic estimated cell size for different placements of the artificial root w.r.t. the upper cell bound. As lower bound in x_2-direction, s_2 is used. The estimations E_1 and E_2 for r_1 and r_2 are half-unbounded (in the x_1-direction), but r_1 is further away from s, so it yields the better heuristic value. The artificial bound built with r_3 would have no intersection with the actual bound and thus give the best value: unbounded.

Note that this kind of computation is only reasonable for very simple resultant calculations and root isolations. The overhead resulting from testing multiple candidates can be immense and thus this method should be handled with care.

Construction of the Polynomial

Finally, the last aspect of variation in our implementation is the construction of the approximation polynomial itself. In general, any polynomial that has a root at the chosen position would work, but in practice we want it to be a simple one. That is, we only consider polynomials with total degree one or two, so that the resultants and real roots can be computed efficiently. We already presented the simplest possible version in our naive approach, which only takes the root into account, but no properties of the approximated polynomial. Naturally, the question arises whether a better approximation leads to a more accurate cell and therefore to better results in MCSAT.

In the next section, we will present a more advanced approach based on the Taylor expansion of the approximated polynomial which will generate artificial cell bounds whose behaviour is closer to the original bounds.

3.4 Using the Taylor Expansion

For the purpose of generating a better approximation polynomial, we first need to clarify what it is that we want to approximate. Ultimately, our goal is an artificial cell bound which is represented by an indexed root expression. This expression induces a continuous function over the underlying cell which preferably behaves similar to the function corresponding to the original cell bound. However, it is not suitable to approximate root functions directly as their evaluation is quite time-consuming and requires real root isolation of a possibly very complex polynomial. Moreover, we do not know the extent of the underlying cell when constructing our approximations. Therefore, we resort to the less direct approach of approximating the polynomial which induces the cell bound.

More precisely, given the sample $s \in \mathcal{R}^i$, a polynomial $p \in \mathbb{Q}[x_1,\dots,x_i]$ and $r_p \in \mathcal{R}$ with $p(s_1,\dots,s_{i-1},r_p) = 0$, we find an artificial root $r \in \mathbb{Q}$ between s_i and r_p and construct another polynomial $h \in \mathbb{Q}[x_1,\dots,x_i]$ whose behaviour in the vicinity of $(s_1,\dots,s_{i-1},r)$ is similar to the behaviour of p around the point $(s_1,\dots,s_{i-1},r_p)$. To achieve this, we can simply compute an approximation $\hat{h}$ of p and then apply an appropriate shift it in the x_i-direction, i.e. $h(x_1,\dots,x_i) = \hat{h}(x_1,\dots,x_{i-1},x_i-(r-r_p))$.

There are several methods for polynomial approximation, many of which belong to the field of interpolation [GS12]. That is, they generate polynomials matching the interpolated function on a set of sample points. Hermite interpolation extends this by demanding that not only the function value, but also certain derivatives match on some of the samples. While interpolation has been studied thoroughly and has many applications, it does not seem suitable for our task. One of the difficulties of multivariate polynomial interpolation is to find an appropriate set of sample points. In our case, this would be particularly hard as we need a good approximation over the underlying cell, but do not yet know its extent. Therefore, it is unclear from which region the points should be chosen.

Instead, we use the *multivariate Taylor expansion* (see, e.g. [Edw73]), which is well-understood and one of the go-to methods for function approximation and which has already been applied in the context of algebraic curves [BHLH88]. Its advantage is that it only needs one sample point and yields a polynomial whose function value and derivatives match the approximated function at that point. Although the Taylor expansion can be defined for any order, we will only consider the first and second order cases, which correspond to linear and quadratic approximations.

Definition 3.4.1 (Taylor Expansion)**.** *Let $i \in \mathbb{N}$, $a \in \mathbb{R}^i$ and $F : \mathbb{R}^i \to \mathbb{R}$ be two times differentiable at a. The* first-order Taylor expansion *of F at a is*

$$T_1[F,a](x_1,\dots,x_i) := F(a) + \sum_{j\in[i]} \frac{\partial F}{\partial x_j}(a)(x_j - a_j).$$

The second-order Taylor expansion *of F at a is*

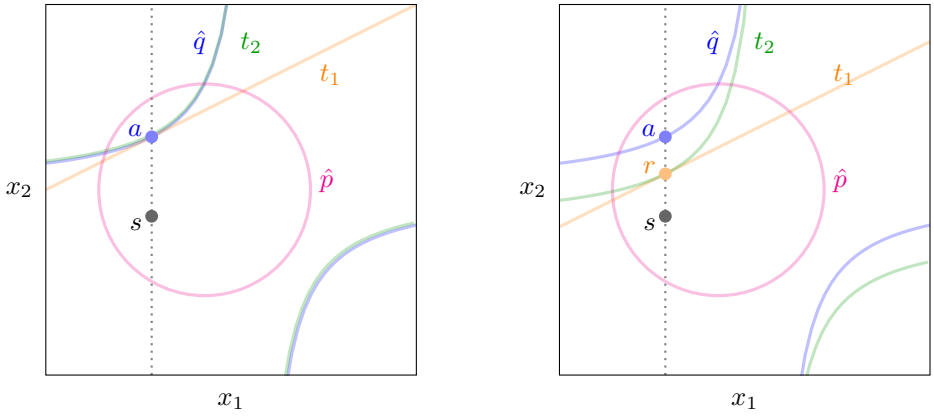
$$T_2[F,a](x_1,\dots,x_i) := T_1[F,a](x_1,\dots,x_i) + \frac{1}{2}\sum_{j,k\in[i]} \frac{\partial^2 F}{\partial x_j \partial x_k}(a)(x_j - a_j)(x_k - a_k).$$

The Taylor expansion has some practical properties, which we will not prove here as they can be easily verified.

Proposition 3.4.1. *Let $i \in \mathbb{N}$, $a \in \mathbb{R}^i$ and $F : \mathbb{R}^i \to \mathbb{R}$ be twice totally differentiable at a. It holds*

$$\begin{aligned} F(a) &= T_2[F,a](a) &&= T_1[F,a](a) \\ \frac{\partial F}{\partial x_j}(a) &= \frac{\partial T_2[F,a]}{\partial x_j}(a) &&= \frac{\partial T_1[F,a]}{\partial x_j}(a) \\ \frac{\partial^2 F}{\partial x_j \partial x_k}(a) &= \frac{\partial^2 T_2[F,a]}{\partial x_j \partial x_k}(a). \end{aligned}$$

Example 3.4.1 (Taylor Expansion)**.** *We go back to our running example and want to approximate the upper bound polynomial $\hat{q} = 2^6(x_1^6 x_2^6+1)(-2x_1x_2+x_2-1)+1$ at its root over the underlying sample $s_1 = -\frac{1}{2}$, that is at the point $a = (-\frac{1}{2}, root_{x_2}[\hat{q},1](-\frac{1}{2}))$ which is also shown in Figure 3.6. The first-order approximation is $t_1 = T_1[\hat{q},a]$ and similarly, $t_2 = T_2[\hat{q},a]$ for the second order. Note that in this example, the quadratic approximation is so close to the original polynomial that there is almost no visible difference. Before we can use the Taylor expansions as cell bounds, we need to introduce a "shift". Currently, their roots intersect the root of $\hat{q}$ at a and thus, the resultant would equal zero at the sample and force a section on the lower level. We choose a point r between s and a and modify the approximation so that their root is at r, which is shown in Figure 3.6b.*



(a) The usual expansions have a root at a, meaning that the resultant of $\hat{q}$ and $t_1(t_2)$ would have a root at s_1.

(b) This problem is solved by shifting the polynomials towards s.

Figure 3.6: Taylor-based approximation of cell bounds. The linear expansion t_1 is marked orange, the quadratic expansion t_2 is marked green.

It is important to emphasize that we approximated the polynomial, but are actually interested in its variety. As can be seen in the example, the varieties of the approximation also behave similarly to the original one. This follows from Proposition 3.4.1, as it implies that all first (respectively second) directional derivatives at a match the approximated polynomial.

Unfortunately, we need to make even more modifications to the construction to accommodate for some of the requirements and complications of our task. Similar to the choice of the root in the naive approach, we need to guarantee that the coefficients of the artificial polynomials are rational numbers. It can indeed happen that for some $j \in [i]$, the coordinate s_j is irrational, meaning that terms of the form $x_j - s_j$ are not suitable. In that case, we omit the summands containing x_j in the Taylor expansion and thereby lose the approximation properties related to the derivatives with respect to x_j. The reason why we cannot simply replace s_j by a rational approximation is that this could move the roots of the constructed polynomial to an unforeseen place, potentially rendering our method incorrect. Notice though that this problem will never occur for the top variable x_i since we replace $x_i - r_p$ by $x_i - r$ for the chosen rational point r.

Example 3.4.2. *Let $p \in \mathbb{Q}[x_1,x_2]$ be a polynomial of level 2 and $s \in \mathcal{R}^2$ so that s_1 is irrational. If we omit the summands containing x_1 in the first-order Taylor expansion of p at a point $a = (s_1,r_p)$, we get*

$$t_1 = p(a) + \frac{\partial p}{\partial x_2}(a)(x_2 - r_p).$$

If $p(a) = 0$ and we shift t_1 to have a root at (s_1,r) instead, we get

$$\tilde{t}_1 = \frac{\partial p}{\partial x_2}(a)(x_2 - r),$$

which has exactly the same roots as $x_2 - r$ and is thus equivalent to the polynomial produced by our naive approach.

The irrational samples also affect the derivatives which we compute for the Taylor expansion. If the approximation point a contains irrational coordinates, then $\frac{\partial p}{\partial x_j}(a)$ might be irrational, too. Now, however, we can use a rational approximation as the root structure is guaranteed by the terms $(x_j - a_j)$. While this does interfere with the gradient similarity, it does not fully destroy it, assuming a good enough rational approximation.

Another complication is the special case that the derivative of the original polynomial in direction of the highest variable x_i is zero. In that case, the Taylor expansion would have a level lower than i and would therefore not induce a valid cell bound. But we can deduce in this case that the discriminant of the original polynomial is zero at the underlying sample and therefore, the next level will collapse to a section anyway. Thus, we simply resort to the naive construction, which will not perform worse than the Taylor expansion in this situation.

With these modifications, we can define the new construction method.

Definition 3.4.2 (Taylor-based Bound Approximation)**.** *Let $i \in \mathbb{N}$, $s \in \mathcal{R}^i$ and let $p \in \mathbb{Q}[x_1,\dots,x_i]$. For a point $r_p \in \mathbb{R} \setminus \{s_i\}$ with $p(s_1,\dots,s_{i-1},r_p) = 0$, we set $a := (s_1,\dots,s_{i-1},r_p) \in \mathcal{R}^i$. If $r_p < s_i$, let $r \in (r_p,s_i)$ and otherwise $r \in (s_i,r_p)$. Furthermore, we set $a' := (s_1,\dots,s_{i-1},r)$ and $Rat(a') := \{j \in [i] \mid a'_j \in \mathbb{Q}\}$. The* linear Taylor-based bound approximation *of p at a is*

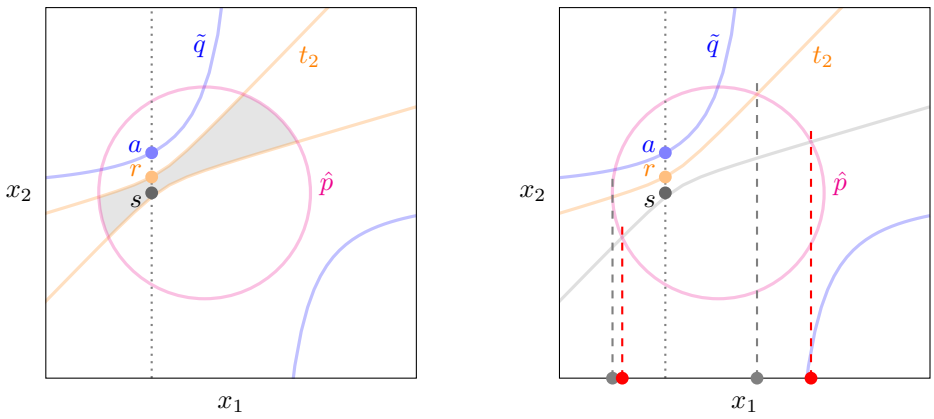
$$h_1[p,s,r](x_1,\dots,x_i) := \sum_{j \in Rat(a')} \frac{\partial p}{\partial x_j}(a)(x_j - a'_j).$$

The quadratic Taylor-based bound approximation *of* p *at* a *is*

$$h_2[p,s,r](x_1,\ldots,x_i) := h_1[p,s,r_p](x_1,\ldots,x_i)+\frac{1}{2}\sum_{j,k\in Rat(a')}\frac{\partial^2 p}{\partial x_j \partial x_k}(a)(x_j-a'_j)(x_k-a'_k).$$

The second-order Taylor expansion can have the unwanted side effect of introducing a second root function. Since the approximation is quadratic, it may have two roots over the underlying sample, only one of which is useful for us. While there could be applications of the approximation method which make use of both roots to, for example, underapproximate both the lower and upper cell bound at once, we will focus on the simple version of handling one root function at a time. And in this version, the second root is undesirable.

Example 3.4.3. *If we consider instead of* $\hat{q}$ *a different polynomial* $\tilde{q} = (1-2x_1)^7x_2^5-1$ *with a similar root structure, then its second-order Taylor-based approximation is significantly less close, as can be seen in Figure 3.7. In particular, it defines two root functions for all values of* x_1 *instead of one. As we only explicitly constructed the second root, adding the first one to the indexed root ordering would be disastrous. It can significantly shrink the cell and in extreme cases even lie exactly at the sample, meaning that the current level would suddenly be handled as a section instead of a sector. The solution to this is to only consider the explicitly constructed root in the indexed root ordering. As we mentioned earlier, this does not harm the correctness of our method, though the proof for this statement is more involved. Nevertheless, the second root will still interfere with the size of the underlying cell as the resultants indicate any common roots of the two input polynomials. Consequently, when we compute the resultant of our approximation with* $\hat{p}$ *and isolate its roots, we will get two roots belonging to intersections of the first, unwanted root function with the variety of* $\hat{p}$*. One of those additional roots actually shrinks the underlying cell.*



(a) When considering the additional root as lower bound of the cell, then this can shrink the cell immensely.

(b) Ignoring the additional root when computing the bounds provides a bigger cell, but it still causes intersections with other root functions which can shrink the next level (marked in red).

Figure 3.7: Additional root function introduced by $t_2 := h_2[\tilde{q},s,r]$. The second (upper) root of t_2 has been constructed explicitly, the first root has not.

As mentioned in the example, we can partially solve the problem of the second root by not including it in the indexed root ordering. That is, when we construct an artificial polynomial t_2 via the second-order Taylor expansion with an explicit root r over the underlying sample, we only add the indexed root expression corresponding to that root to the set Ξ from which the indexed root ordering will be derived. In particular, we can use the values of the derivatives computed in the Taylor expansion to determine whether we are interested in the first or second root, meaning that we can avoid real root isolation.

Lemma 3.4.2. *Consider the quadratic Taylor-based approximation $h := h_2[p,s,r]$ of a polynomial $p \in \mathbb{Q}[x_1,\ldots,x_i]$ $(i \in \mathbb{N} \setminus \{0\})$ with respect to $s \in \mathcal{R}^i$ and the chosen root $r \in \mathbb{Q}$. Let $r_p \in \mathcal{R}$ be a root of p at $s_{[i-1]}$ and $a := (s_1,\ldots,s_{i-1},r_p)$. Further, let $d_1 := \frac{\partial p}{\partial x_i}(a)$ and $d_2 := \frac{\partial^2 p}{\partial^2 x_i}(a)$. It holds*

$$root_{x_i}[h,1](s_{[i-1]}) = r \Leftrightarrow d_1 = 0 \text{ or } d_2/d_1 < 0.$$

Proof. To determine the roots of h at the underlying sample $s_{[i-1]}$, we substitute s_1 for $x_1,\ldots,s_{i-1}$ for x_{i-1} and obtain

$$h(s_1,\ldots,s_{i-1},x_i) = \frac{\partial p}{\partial x_i}(a)(x_i - r) + \frac{\partial^2 p}{\partial^2 x_i}(a)(x_i - r)^2 = d_1(x_i - r) + d_2(x_i - r)^2.$$

If $d_1 = 0$, then this polynomial has a double root at r and we are done. Otherwise, the roots are r and $r - \frac{d_2}{d_1}$. Accordingly, if $d_1 \neq 0$, we have

$$root_{x_i}[h,1](s_{[i-1]}) = r \Leftrightarrow r < r - \frac{d_2}{d_1} \Leftrightarrow 0 > \frac{d_2}{d_1}.$$

□

Example 3.4.4. *In the situation of Figure 3.7 we would proceed as follows. During the construction of t_2, we identify and store d_1 and d_2 as in the lemma. Then, we add t_2 to the working set P_2 which will be projected. However, depending on the values of d_1,d_2, we only add one of the root expressions $root_{x_2}[t_2,1]$ or $root_{x_2}[t_2,2]$ to the set Ξ from which the root ordering and level description will be derived. In the example, it is $root_{x_2}[t_2,2]$, and accordingly we avoid that the other root would be used as the lower cell bound.*

This concludes the theoretical presentation of our modified levelwise single cell construction and of the many possible variations. In the next step, we see how they perform when put into practice.

Chapter 4

Experimental Results

In this chapter, we evaluate the performance of our approach and compare it to the original levelwise method on the basis of experimental results. The generation of explanations for the SMT-solving process in MCSAT was our main motivation and naturally, we intend to evaluate the measures specifically implemented with that regard, for example the criteria for ensuring termination. Therefore, we mainly restrict our considerations to the usage in MCSAT. Nevertheless, we will also extract some statistics that give insights on the single cell construction as a stand-alone procedure.

Tested Implementations

The levelwise algorithm and our adaptations are implemented in the SMT-RAT solver, which can use these methods as explanation backends for its implementation of MCSAT. One of the main principles of SMT-RAT is modularity and accordingly, it allows to combine backends in a sequential manner. For example, if the levelwise method fails due to nullification, then a cell construction using the much more time-consuming Collins projection as in NLSAT is employed instead. Similarly, other incomplete, but more efficient backends can be called prior to the single cell construction so that, if a backend fails, it passes the problem on to the next method in the sequence. In addition to the single cell construction methods, SMT-RAT currently offers implementations of the following backends: The *Fourier-Motzkin variable elimination (FM)*[JBd13], which can be used to analyse linear constraints, the *Virtual Substitution (VS)* method [ÁNK17], which is particularly suitable for problems with linear and quadratic polynomials and the *Interval Constraint Propagation (ICP)* [Kre19]. If they work, these approaches are often much faster than single cell construction. We compared variants of our approach with the levelwise method, each with the fallback to original NLSAT explanations in the case of nullification. In addition, we also evaluated their performances when employing the other backends first. Our considerations will focus on the following tested variants:

- **LW**: The levelwise single cell construction using the biggest cell heuristic for generating indexed root orderings when the current level is a sector. In the fail case, it resorts to the complete construction method from NLSAT.
- **APX-N**: The naive approximation approach, introducing an artificial bound whenever the polynomial corresponding to the actual bound has degree 5 or

higher. The artificial root is chosen so that it has a simple rational representation and to guarantee termination of MCSAT, a fixed threshold of 50 approximated cells per instance is used.

- **APX-Ti**: For $i = 1$, this denotes the Taylor-based method of order one and, similarly, for $i = 2$, it refers to the second-order version. We use the same set of additional settings for approximation criteria, artificial root position and termination as for the naive approach.

- **FM-*/VS-*/ALL-*** For any of the above solvers, say *[solver]*, we denote by *FM-[solver]* and *VS-[solver]* the variants resulting from calling the FM- or VS-backend prior to *[solver]*. The sequential combination of the FM-, ICP- and VS-backends, as well as *[solver]*, in that order is denoted by *ALL-[solver]*. We test these combinations to determine whether our approach can be beneficial in cases where the already known methods struggle. Moreover, as the approximation reduces the degree of the polynomials in the cell descriptions, it could even enhance the incomplete FM and VS methods, which thrive on linear and quadratic polynomials.

Note that we also tested different criteria and values for the various heuristics in the approximation approach. In Section 4.4, we will give some details on their comparison. Before that, we will only consider the settings as described for APX-N, as they seemed to work best.

Benchmark Setting

All solvers were tested under the same conditions. For the test cases, we used the SMT-LIB benchmark set for quantifier free non-linear real arithmetic [BST10, SMTa], which is a collection of 11552 individual problem instances. Each instance provides a QFNRA-formula whose satisfiability is to be determined. We executed the solvers on a machine with four 2.1 GHz AMD Opteron CPUs, each consisting of 12 cores. Every individual problem instance was run with a time limit of one minute and with 4 GB of memory available. Consequently, if a solver could not solve the instance within the allotted time (or memory), its result was classified as a *timeout* (or *memout*).

4.1 Overview

We begin our analysis with a general overview of the solved instances for each solver, summarized in Table 4.1. In addition to the above mentioned versions, it also contains a row VB for the *virtual best* of the solvers LW, APX-N, APX-Ti. That is, VB is the hypothetical method which employs on each instance the fastest of the actually implemented variants.

An important observation is that only a fraction of the test instances allow for meaningful comparisons between the cell construction methods. The fifth column of the table indicates the number of solved instances that did not require any cell construction at all. In the case of the first four solvers, this means that only Boolean conflicts were encountered in the solving process, if any. As can be seen, this applies to roughly half of the solved instances. For the solvers using the other, incomplete backends first, the situation is even more extreme as they are able to also solve

Solver	solved	sat	unsat	without scc	with scc	using approx.
LW	9221	4512	4709	4724	4497	
APX-N	9265	4535	4730	4724	4541	583
APX-T1	9249	4528	4721	4724	4525	569
APX-T2	9210	4505	4705	4724	4486	530
VB	**9272**	**4537**	**4735**	**4724**	**4548**	**441**
FM-LW	9410	4662	4748	7223	2187	
VS-LW	9514	4716	4798	8303	1211	
ALL-LW	9584	4729	4855	8495	1089	
FM-APX-N	9442	4673	4769	7221	2221	596
ALL-APX-N	9610	4737	4873	8492	1118	544
VS-APX-T2	9499	4698	4801	8304	1195	519
Total	11552*	5069*	5379*			

** For 1104 instances, it is not known whether they are satisfiable or unsatisfiable.*

Table 4.1: Details on the solved instances for each solver. The columns show the total number of solved instances, the number of satisfiable and unsatisfiable instances among them, the number of instances where no single cell construction (scc) and where at least one scc was used and the number of solved instances where a cell was approximated (in that order). The solvers based on LW did not use any approximation.

instances which do require theory explanations without single cell construction. For example, in about 88% of the instances solved by ALL-LW, no call to the levelwise method was made, leaving only slightly more than a thousand relevant test cases.

When we examine the instances in which our new methods introduced at least one artificial polynomial, the data set is shrunk down even more. With the chosen approximation criteria, an approximation was performed in no more than 600 of the solved instances. When counting both the solved and unsolved cases, we get 808 cases where it was used. Notice that if no artificial polynomial is introduced, the approximation methods are almost identical to the levelwise approach, the only difference being some additional overhead for checking the criteria. Of course, changing the criteria could increase the number of relevant cases, for example when already approximating polynomials of degree three, but we will see later that this also leads to a worse performance overall. An interesting observation in this context is that the number of instances solved using approximation does only decrease very slightly (by at most 6.7%) when using the additional backends. In the case of FM-APX-N, it even increases. This contrasts the fact that the number of cases in which single cell construction is used plummets by 50 to 75 percent. A possible explanation is that the instances which the other backends cannot solve have a high overlap with the instances in which the approximation methods thrive. Taking into account that the former are specialized on problems with low-degree polynomials, while the latter are designed to deal with high degrees, this seems even more plausible.

Considering only the pure numbers, the naive approach stands out as it solves the most instances among the non-virtual-best methods. It solves 44 instances more than the original levelwise method when used as primary backend. The combination with other backends reduces the gap slightly, only being 32 instances for FM-APX-N and FM-LW and being 26 for the ALL-* variants. While the linear Taylor-based

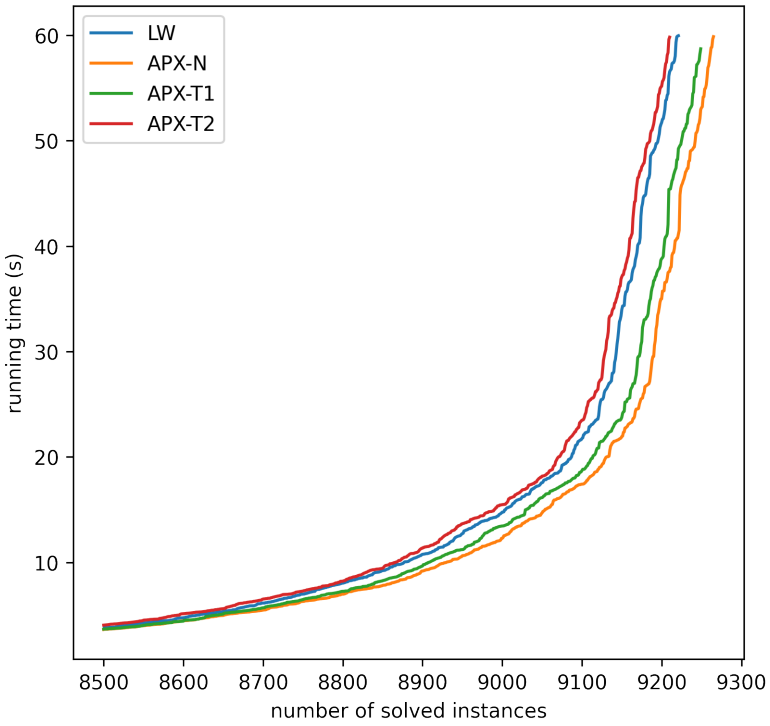


Figure 4.1: Performance profile with respect to the runtime of the solvers using only cell construction backends. The horizontal axis begins at 8500 since there is no significant difference between the solvers before that value.

method still performs better than the original solver, the quadratic version actually solves fewer instances than LW. This can also be observed in the performance profile in Figure 4.1.

Moving on from the pure numbers of solved instances, we take a closer look at specific statistics, with the aim of providing a better understanding of where and how much the methods differ. Table 4.2 considers those instances solved by LW and all APX-* methods for which the latter did indeed use approximation. The overlap is rather high, as this sums to 499 instances, while none of the variants individually solved more than 583.

We see that the linear approximation methods were on average faster than LW, despite computing many more cells. This means that not only do they solve more instances, they are also often more efficient in each individual cell construction and in total. One reason for the improved efficiency could be that our approach does indeed reduce the degrees of the involved polynomials, which can be observed by examining the mean maximum degree per instance. Moreover, we can see the effect of introducing artificial polynomials in the numbers of computed resultants, discriminants and leading coefficients. While APX-N produced about 51% more cells than LW, the mean number of resultants per instance increased by 124% and for discriminants and leading coefficients by 80% and 70%, respectively. The other approximation methods behave similarly, producing bigger projections in each cell. In the case of APX-N and

	LW	APX-N	APX-T1	APX-T2
mean runtime (s)	2.61	1.71	1.9	3.45
sum calls to scc	3462	5238	5599	5418
sum calls to NLSAT	9	9	9	9
mean max. degree	42.2	33.06	33.19	33.12
mean #resultants	12.42	27.83	30.92	28.29
mean #discriminants	23.37	42.14	45.61	43.16
mean #leading coefficients	30.99	52.68	57.41	53.92
mean runtime in scc (s)	0.51	0.28	0.29	0.54

Table 4.2: Statistics for the 499 instances on which the approximation methods did approximate at least one cell and which were solved by all listed variants.

APX-T1, this is outweighed by the simplification of the resultant computations, since they are still more efficient. This is also indicated by the fact that their mean time spent to compute a single cell is much lower than for LW. In that sense, they do work as intended. On the other hand, the quadratic Taylor-based method needs more time on average, both for individual cell constructions and in total. This suggests that the effort required to compute the Taylor expansion is too high compared to the payoff of simpler resultants. Interestingly, the reduction in degree is very similar to the linear methods.

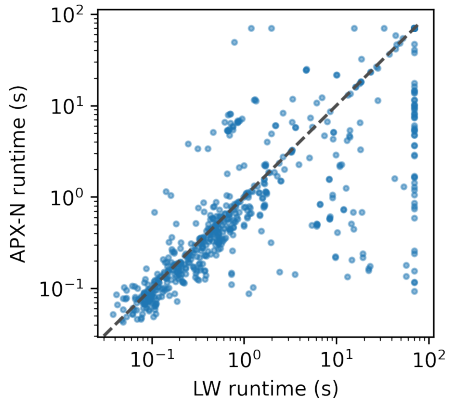
An open question was how the introduction of artificial polynomials affects the frequency of failure due to nullification. As we can see, such failures occurred exactly the same number of times for all variants, as in those cases the NLSAT backend was called. On all other instances where approximation was used, LW needed 315 and APX-N 212 such calls, even though it generated more cells. This suggests that our method does not have any significant positive or negative influence on nullification.

We will now take a closer look at the data for the naive and then the Taylor-based approaches, followed by some insights on the different heuristics that could be used.

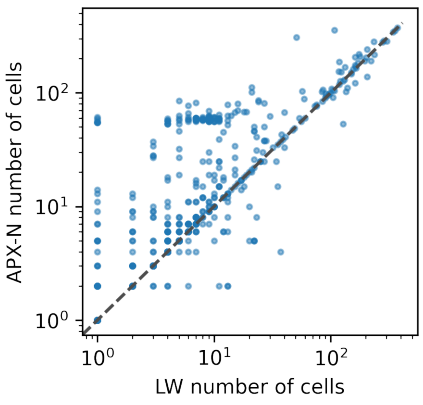
4.2 Naive Approach

First of all, the results of LW and APX-N only differ in a few instances. The number of test cases solved by both variants is 9215, meaning that there are only six cases in which LW found a solution but APX-N did not. It turns out that in two of those cases, LW found a solution just barely before reaching the time limit, while the approximation method failed simply due to the additional overhead of checking criteria without even approximating any cell. On the other hand, the approximation method was able to solve 50 new instances with regard to the original method. For the variants with other backends, the situation is similar. We have 9403, respectively 9577, commonly solved instances for the FM-* and ALL-* solvers, which translates to seven instances each "lost" by using approximation, but many more "gained". While these numbers do not show a great impact of our approach, they suggest that it almost never leads to a significantly worse performance.

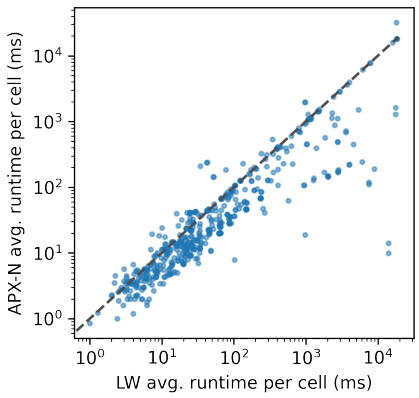
Statistics similar to those in Table 4.2 are visualized in Figure 4.2 with a data point for each instance on which approximation was applied, including those not solved by both solvers. They show that the differences in averages are not simply skewed by outliers, but do indeed follow from a general pattern.



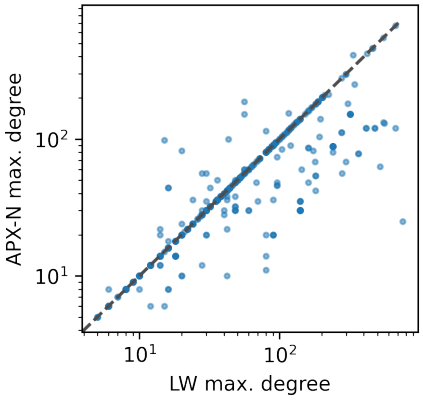
(a) Overall runtime in seconds. Timeouts are set to a value of 70 seconds and thus appear on the right and upper borders.



(b) The total number of constructed cells for each instance. That is, the number of calls to single cell construction.



(c) The average time needed to construct a cell for each instance. (time spend in single cell construction divided by number of cells)



(d) Maximum degree (in its main variable) of any polynomial in the input or constructed in the explanation process.

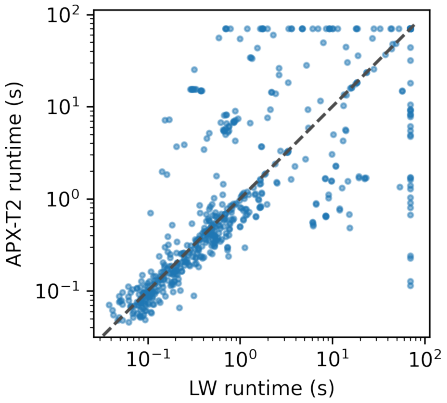
Figure 4.2: Scatter plots comparing LW and APX-N on the set of benchmarks where approximation is used.

4.3 Taylor-based Approach

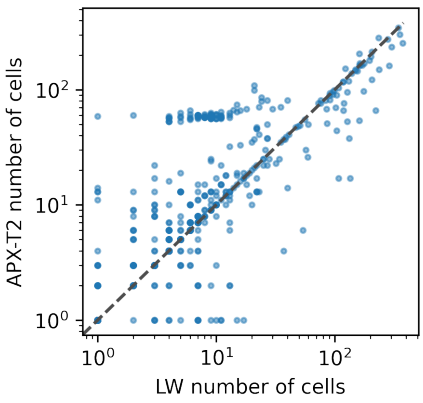
As can be seen in Tables 4.1 and 4.2, the Taylor-based methods performed worse than the naive approach in terms of solved cases and the quadratic version even solved fewer instances than the original levelwise method. Moreover, the more accurate approximation does not seem to give an advantage, even on subsets of the benchmarks. That is, almost no instance solved by APX-T1 or APX-T2 cannot be solved by the naive approach. This can also be seen in the numbers for the virtual best in Table 4.1, which only solves seven instances more than APX-N.

Furthermore, Table 4.2 shows that the Taylor-based methods actually required more cells than APX-N to solve the inputs, which contradicts the original motivation

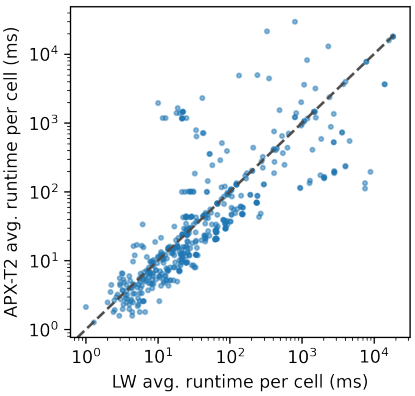
that a more accurate approximation would produce cells of higher quality for MCSAT. Accordingly, we suppose that the additional effort for computing the Taylor expansion and the related, more complex resultants does not pay off. While the linear version is not too far behind the naive approach, the additional overhead seems to be quite drastic in the second-order method APX-T2. Possible reasons for this behaviour could be a rather slow implementation of polynomial operations (e.g. taking the derivative and evaluating it) used by SMT-RAT or smaller cells due to the additional root of the quadratic polynomial. However, further investigations will be needed to see whether these assumptions are correct.



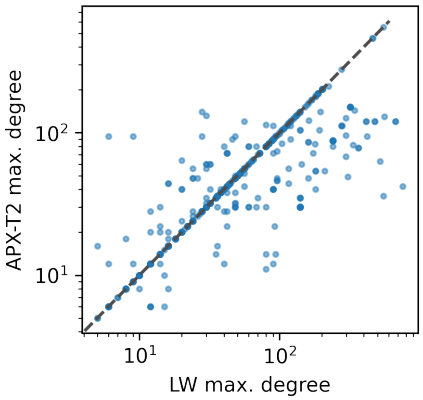
(a) Overall runtime in seconds. Timeouts are set to a value of 70 seconds and thus appear on the right and upper borders.



(b) The total number of constructed cells for each instance. That is, the number of calls to single cell construction.



(c) The average time needed to construct a cell for each instance. (time spend in single cell construction divided by number of cells)



(d) Maximum degree (in its main variable) of any polynomial in the input or constructed in the explanation process.

Figure 4.3: Scatter plots comparing LW and APX-T2 on the set of benchmarks where approximation is used.

The disadvantages of the quadratic Taylor approach also translate to the combination with other backends. We initially hoped that a combination with the vir-

tual substitution method (VS) could be synergistic, as VS thrives on problems with quadratic and linear polynomials. But again, VS-APX-T2 solved even fewer instances than VS-LW.

4.4 Heuristics

During the evaluation of our approach, we tried many different settings for the approximation criteria, termination criteria and placement of the artificial root. To be precise, we considered the following variations:

- Approximate all bounds induced by a polynomial of degree at least d for each $d \in \{3,4,5,6,7\}$.
- Approximate only bounds which would induce a resultant of degree 9 or higher. That is, check whether there would be a complicated resultant when approximation was not used.
- Only approximate a polynomial if its degree is higher than the value of a monotone function applied to the number of so far approximated cells.
- Stop using approximation after k cells for $k \in \{10,20,50,100\}$.
- Do not approximate a cell if it results from a conflict between constraints which already lead to k approximated cells, for $k \in \{5,10,20\}$.
- Place the artificial root at a point with a small rational representation.
- Place the artificial root at a point with a small rational representation within a certain interval. This way, one can force the root to be closer to or further away from the bound.
- Try several candidates for the root and use the one which gives the best estimated cell, as described in Section 3.3.

While we did not test all combinations of these parameters, we still acquired some insights on their performance. Of all tested versions, the one used for APX-N worked best and solved the most instances.

Approximating only polynomials with degree greater or equal to six made almost no difference, compared to a threshold of five. Setting the value to seven still changed the results only slightly, but there were some unsolved instances where approximating more eagerly would have helped. While a threshold lower than five led to fewer solved instances overall, the behaviour differed more from our default method. More precisely, there is a row of cases which only the earlier approximation allowed to solve, but there are even more cases which it did not solve despite them being efficiently solvable without approximation.

When approximating only a fixed number of cells, the experimental results showed that a threshold between 20 and 50 works well. Importantly, there are instances in which approximation does not help and it is useful to stop earlier. Of course, the threshold should depend on the concrete instance given and on other data sets, other strategies might work better. However, we found that setting a threshold for each constraint, allowing for more approximations in more complex problems, did not make a significant difference. A possible reason for this could be that the instances in which

approximation was used successfully have a rather simple Boolean structure in which only a few constraints cause conflicts. Indeed, the average number of disjunctions in the conjunctive normal form of the input formula is around 350 for the whole data set, around 150 in the cases where approximation was used and roughly 20 for the solved cases where approximation was used. A similar pattern emerges for the number of variables in the Boolean abstraction of the formula. Whether this is a general property of our approach or a lack in diversity in the benchmark set would need a more detailed investigation.

Regarding the root placement, the more advanced methods could not outperform the strategy of simply finding a rational with small representation. An unexpected reason for this lies in the MCSAT-implementation of SMT-RAT. When choosing a new sample, it prefers points with simpler representation, for example integers are preferred over fractions. If the artificial cell bound is very close to the actual bound, it is less likely that there is a rather simple point in between them and thus, a candidate which is unrelated to the approximated conflict might be chosen. This could lead to many different cells being approximated before one of those cells is actually covered by approximations, leading to longer runtime.

Although APX-N performed best individually, it is far from being optimal. As can be seen in Table 4.3, the number of instances solved by any variation is noticeably higher than the number for APX-N. This also holds true when only varying one parameter at a time, meaning that each of them has an impact on the number of solved instances and for each of them, APX-N is not yet optimal. Since the numbers for the virtual bests of each individual parameter variation are lower than for the overall virtual best, we assume that the parameters are to some extent orthogonal. That is, the solver performs better on different sets of instances when varying different parameters.

	APX-N	VB-deg	VB-apx-crit	VB-term-crit	VB-root	VB-all
solved	9265	9281	9307	9296	9286	9322

Table 4.3: Number of solved instances for APX-N and the virtual bests of all variations of the degree threshold (VB-deg), all variations of approximation criteria (VB-apx-crit), all variations of termination criteria (VB-term-crit), the different root placement strategies (VB-root) and of all tested variations (VB-all). Note that APX-N is taken into account for each of those virtual bests.

Chapter 5

Conclusion

5.1 Future Work

Our experimental results not only showed that the approximation approach can improve the SMT-solving process, but also that there is still unused potential, especially when varying the different parameters and heuristics. Therefore, we do think that pursuing the idea further is worthwhile and we already see several directions for future research.

Approximation Criteria

While our main implementation only used approximation in around 800 of the test instances, we have seen that changing the approximation criteria can increase this number. More importantly, it also showed that there are more instances for which our approach would be beneficial. On the flip side, the variation of approximation criteria also led to timeouts on problems that could previously be solved and which were due to unnecessary approximations. Accordingly, we assume that a clever way to identify the cases in which approximation is indeed helpful could have a big positive impact on our method. That is, one needs to find a good set of approximation criteria which are more dynamic than simply approximating all bounds of degree five or more. We already tried the idea of estimating the complexity of the resultants that would be computed without approximation and making the decision based on this estimation. Refining this idea could be promising, although we did not find an estimation and threshold that led to better results.

Termination Criteria

Similar to the question of whether or not a specific bound should be approximated, we see potential in the strategy for deciding whether a specific cell should be approximated. All our termination criteria rely on the fact that at some point, we simply stop using approximation and essentially resort to the normal levelwise method. Again, varying these criteria shows that more instances could be solved when applying the right strategy to each instance. Sometimes, many approximations are needed to come to a result, while at other times the approximation is not beneficial and stopping earlier would be the correct decision. Recognizing the cells for which an approxima-

tion is helpful is not an easy task, but finding an adequate heuristic potentially yields significant improvements.

Artificial Polynomial Construction

Even though the Taylor-based versions of our approach did not meet our expectations, we do not rule out the possibility that a better approximation of the cell bound could be advantageous. We suppose that non-linear approximations are only suitable if one finds a way to identify and ignore intersections of specific root functions. That is, if we could find out which roots of a resultant correspond to the intersections of the explicitly constructed root of our artificial polynomial, then its other roots would not harm the quality of the resulting cell. Alternatively, it might be interesting to approximate both bounds at once with a quadratic polynomial, so that it has two roots, each corresponding to one of the bounds.

To obtain a more accurate cell from linear bounds, one could develop a technique for using piecewise defined functions in the cell description. In particular, this would allow for linear descriptions of non-convex cells. This idea is illustrated in Figure 5.1 and would probably require some kind of interpolation.

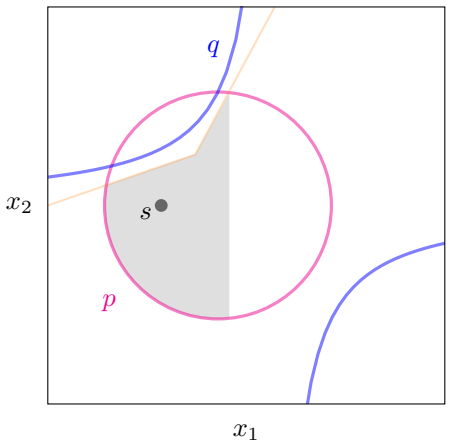


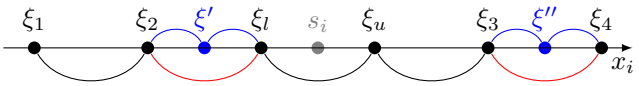
Figure 5.1: Approximation of cell bounds by piecewise linear functions.

However, all methods constructing an artificial bound close to the original one should be developed with regard to the strategies of the respective MCSAT implementation, because a rather small difference between the bounds can lead to numbers with bigger rational representation and thus more complex calculations.

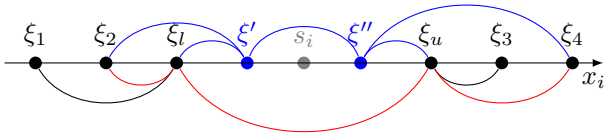
Other Indexed Root Orderings

We only considered the biggest cell heuristic for choosing an indexed root ordering. However, our general formulation of the approach works with arbitrary indexed root orderings and in fact, we can introduce artificial polynomials with roots at any point of the ordering without harming correctness. Therefore, the question arises of what strategies would be suitable for other orderings. For example, when using the "chain" heuristic ([NSÁ+22], Def.7.6), the indexed root expressions are ordered according to their value at the sample and then, only resultants of polynomials in neighbouring

root expressions are computed. Adding an artificial polynomial with a root in between two neighbours allows to replace their resultant with two simpler ones. Another idea again modifies the biggest cell heuristic, but it does not replace all resultants of the original bounds. Instead, only the more expensive ones are replaced by resultants with artificial bounds and the simpler ones still use the actual bounds. This way, the cell might be closer to the original one, but still easier to compute. Figure 5.2 illustrates the two ideas.



(a) Modification of the chain heuristic.



(b) Modification of the biggest cell heuristic, only replacing some of the resultants.

Figure 5.2: Ideas for introducing artificial roots in different indexed root orderings w.r.t. a sample s. Artificial roots and are marked blue. The arcs below indicate the pairs of root expressions which would originally induce a resultant. The resultants represented by red arcs are replaced by the ones corresponding to the blue arcs, when introducing artificial roots.

Usage in Incremental Linearisation Approaches

Finally, it would be interesting to see whether our approach can be used advantageously in incremental linearisation approaches, which construct a linear abstraction of formulas in non-linear real arithmetic, e.g. [CGI+18]. Solutions to the linear abstraction can be found more quickly, but not all of them solve the original problem. Therefore, the abstraction is incrementally refined to exclude these false solutions. Employing single cell construction and approximating all bounds by linear polynomials could yield a novel way to generate lemmas for refining the abstraction.

5.2 Summary

In this thesis, we presented a modification of the levelwise single cell construction method with the aim of reducing its complexity. We have shown how low-degree polynomials can be introduced at each level of the construction to define artificial cell bounds so that the resultants in the projection step are easier to compute and have lower degree. As these new bounds reduce the cell's size at the respective level, our approach can be understood as an underapproximation. However, it does yield correct results, meaning that the approximated cell still contains a specified point and leaves the sign of the input polynomials invariant. We proved this correctness result for a general formulation of our approach, but also gave concrete implementations and

pointed out what parts of the method leave room for variation. The main motivation for this thesis was to improve the single cell construction as part of the MCSAT algorithm for solving the satisfiability problem of non-linear real arithmetic. In this context, the underapproximation becomes relevant since smaller cells will exclude less of the search space for potential models and in the extreme case, underapproximating can lead to non-termination. We tackled this problem by providing measures that can be employed to guarantee termination. Finally, we implemented several versions of the modified single cell construction in the SMT-RAT toolbox and evaluated their performance in MCSAT by examining experimental results. They showed that our approach has the intended effect of speeding up cell construction and thereby allowing to solve more satisfiability problems quickly, though only on a fraction of the used test cases. Interestingly, the more complex implementations using the Taylor expansion were outperformed by the so-called naive version, which might deserve a less derogatory name, after all.

Bibliography

[ÁNK17] Erika Ábrahám, Jasper Nalbach, and Gereon Kremer. Embedding the virtual substitution method in the model constructing satisfiability calculus framework (work-in-progress paper). In M. England and V. Ganesh, editors, *Proceedings of the 2nd International Workshop on Satisfiability Checking and Symbolic Computation (SC^2 2017)*, 2017.

[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.

[BHLH88] C.L. Bajaj, C.M. Hoffmann, R.E. Lynch, and J.E.H. Hopcroft. Tracing surface intersections. *Computer Aided Geometric Design*, 5(4):285–307, 1988.

[BK15] Christopher W. Brown and Marek Košta. Constructing a single cell in cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 70:14–48, 2015.

[BM20] Christopher W. Brown and Scott McCallum. Enhancements to lazard's method for cylindrical algebraic decomposition. In François Boulier, Matthew England, Timur M. Sadykov, and Evgenii V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing*, pages 129–149. Springer International Publishing, 2020.

[Bro61] Achille Brocot. Calcul des rouages par approximation: nouvelle méthode. *Revue chronométrique. Journal des horlogers, scientifique et pratique*, (3):186–194, 1861.

[BST10] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB standard - version 2.0. In *Proceedings of the 8th international workshop on satisfiability modulo theories, (Edinburgh, Scotland)*, volume 13, page 14, 2010.

[CGI+18] Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, Marco Roveri, and Roberto Sebastiani. Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. 19(3), 2018.

[CKJ+15] Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp, and Erika Ábrahám. SMT-RAT: An open source C++ toolbox for strategic and parallel SMT solving. In Marijn Heule and Sean Weaver, editors,

Theory and Applications of Satisfiability Testing – SAT 2015, pages 360–368. Springer International Publishing, 2015.

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

[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.

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

[Edw73] C.H. Edwards. Multivariable differential calculus. In C.H. Edwards, editor, *Advanced Calculus of Several Variables*, pages 56–159. Academic Press, 1973.

[GKPL94] Ronald L. Graham, Donald E. Knuth, Oren Patashnik, and Stanley Liu. *Concrete Mathematics: A foundation for computer science*. Addison-Wesley, second edition, 1994.

[GS12] Mariano Gasca and Tomas Sauer. Polynomial interpolation in several variables. *Advances in Computational Mathematics*, 12:377–410, 2012.

[JBd13] Dejan Jovanovic, Clark Barrett, and Leonardo de Moura. The design and implementation of the model constructing satisfiability calculus. In *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.

[Kre19] Gereon Kremer. *Cylindrical algebraic decomposition for nonlinear arithmetic problems*. PhD thesis, RWTH Aachen University, 2019.

[Laz94] Daniel Lazard. *An Improved Projection for Cylindrical Algebraic Decomposition*, pages 467–476. Springer, New York, 1994.

[McC98] Scott McCallum. An improved projection operation for cylindrical algebraic decomposition. In Bob F. Caviness and Jeremy R. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 242–268, Vienna, 1998. Springer.

[MPP19] Scott McCallum, Adam Parusiński, and Laurentiu Paunescu. Validity proof of lazard's method for cad construction. *Journal of Symbolic Computation*, 92:52–69, 2019.

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

[Ste58] Moritz Stern. Über eine zahlentheoretische funktion. *Journal für die Reine und Angewandte Mathematik*, 1858(55):193–220, 1858.

[Tar98] Alfred Tarski. A decision method for elementary algebra and geometry. In Bob F. Caviness and Jeremy R. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 24–84, Vienna, 1998. Springer. Reprint of the original paper from 1948.