JAIST Repository
https://dspace.jaist.ac.jp/

Title	多項式制約解消のためのSMTソルバ
Author(s)	Vu, Tung Xuan
Citation	
Issue Date	2018-06
Type	Thesis or Dissertation
Text version	ETD
URL	http://hdl.handle.net/10119/15430
Rights	
Description	Supervisor:小川　瑞史，情報科学研究科，博士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

Doctoral Dissertation

SMT Solving for Polynomial Constraints

Vu Xuan Tung

Supervisor: Mizuhito Ogawa

School of Information Science
Japan Advanced Institute of Science and Technology

June, 2018

Abstract

The need for solving non-linear arithmetic arises from many applications in artificial intelligence and formal methods. Although the full first-order theory of real numbers is decidable, the best well-known decision procedure for it, namely quantifier elimination by cylindrical algebraic decomposition, has the complexity of double exponential with respect to the number of variables. This remains as an impediment for a solver supporting non-linear arithmetic. This thesis aims at an efficient complete framework for solving existential fragment of polynomial constraints by first developing (incomplete) efficient procedures as heuristics and then combine them with a complete procedure. Two efficient procedures proposed are

- an extension of the raSAT loop, which is, in turn, an extension of interval constraint propagation (ICP) with testing, with the application of the intermediate value theorem (IVT), and
- subtropical satisfiability.

Distinct procedures and their combinations are further integrated into a satisfiability modulo theories (SMT) framework by supporting features of SMT solving such as unsat core computation.

While raSAT loop (an extension of the ICP with testing) aims at quickly detecting satisfiability of inequalities, the application of the IVT aims at showing satisfiability of equations. We propose a scheme to combine interval arithmetic, testing, and the IVT to show satisfiability of combinations of inequalities and equations. SAT-directed heuristics are also proposed for the framework to quickly detect satisfiability while not affecting performances of detecting unsatisfiability. Experimental data shows that the proposed extensions increase the number of solved problems and the heuristics improve the running time on solved problems and also increase the number of solved benchmarks. Comparing with other SMT solvers, except for weaknesses in completeness, **raSAT** shows comparable running time on problems it solved.

The second procedure, i.e. subtropical satisfiability, aims at finding an assignment for variables which satisfies inequalities by examining the exponent vectors of polynomials appearing in the inequalities. From those exponent vectors, the method generates linear arithmetic constraints such that if they are satisfiable, then the original inequalities are also satisfiable. The solution of the generated linear constraints is further used to provide a witness for satisfiability of non-linear inequalities. Experimental results show that the procedure is quite fast at either detecting satisfiability or failing. In particular, it finds solutions for problems where other state-of-the-art non-linear arithmetic SMT solvers times out.

Both proposed procedures are incomplete and in order to produce a decision framework, we utilize quantifier elimination methods implemented in the computer algebra system Redlog/Reduce. We propose two kinds for combining the ICP-based methods

and the quantifier elimination, namely lazy and less lazy approaches. While the lazy approach uses ICP-based methods as pre-processing steps for the quantifier elimination, the less lazy one invokes the quantifier elimination on every box generated by the ICP framework. In both approaches, subtropical satisfiability is utilized first as an attempt to find a model for inequalities. Experimentally, the lazy method performs better than the less lazy one but we expect some future improvements for the less lazy approach so that several unsatisfiable boxes can be all discarded once the quantifier elimination method detects the unsatisfiability of one box. Experimental results also show that our framework is an efficient decision procedure to solve non-linear arithmetic SMT problems and complementary to implementations in other SMT solvers.
Keywords: *SMT solving, non-linear arithmetic, interval constraint propagation, subtropical satisfiability, complete efficient framework.*

Acknowledgments

This work was carried out during years 2015 – 2018 at Japan Advanced Institute of Science and Technology, Japan. It was also partially done at LORIA, Nancy, France from June 2016 to May 2017

I would like to express my sincere gratitude to my supervisor, Mizuhito Ogawa, for his continuous support of not only my thesis but also my life. His guidance into the world of formal methods and SMT solving have been essential during this work.

My sincere thanks also go to my second supervisor, Nao Hirokawa, for his detailed and constructive comments through this work. I also learned a lot from him about formalization and knowledge from other fields, especially term rewriting systems.

It is a pleasure to appreciate collaboration efforts from Haniel Barbosa, Pascal Fontaine, and Thomas Sturm during my off-campus research at LORIA, Nancy, France. Their great collaboration had helped me to fulfill the objective of my off-campus research.

This thesis would not have been completed and polished unless the referees of my dissertation committee, Pascal Fontaine, Nao Hirokawa, Mizuhito Ogawa, Kazuriho Ogata, Hiroyuki Seki, read and gave invaluable comments on the structure and contents of the thesis.

I also would like to acknowledge JAIST 5D scholarship for supporting expenses during my study at JAIST, and JAIST Off-Campus Research Grant for fully supporting the expenses during my stay at LORIA, Nancy, France.

Last but not least, I owe my loving thanks to my parents, my older brother, my older sister, and my wife for their constant encouragement and understanding. Thank you for always being beside me during happy moments and also difficult times.

Contents

Chapter 1

Introduction

Due to numerous applications (see [66] for more than 100 references), solving polynomial constraints has been an active area in research for a few decades. In 1948, Tarski proposed a decision procedure for the full first-order theory of real closed fields [80] with unfortunately non-elementary complexity. Since then, Tarski's procedure has been much improved. In 1975, Collins [12] gave a much more efficient method of quantifier elimination using cylindrical algebraic decomposition (CAD) which again has gone through many improvements [51, 39, 40, 13]. Despite those improvements, the worst-case complexity of doubly-exponential remains as an obstacle for their applications [18]. As a consequence, the attention turns to first developing (incomplete) efficient procedures for fragments of the full real closed fields theory and then combining them with complete methods to produce an efficient complete framework. Here we focus on the existential fragment of the theory and propose two efficient heuristic methods, namely extensions of the interval constraint propagation (ICP) and subtropical satisfiability. Then we combine them with a quantifier elimination procedure to produce an efficient complete framework for the existential fragment.

Starting from the work in [65], and then [28, 34, 32, 36], ICP showed practical efficiency although it is incomplete in solving polynomial constraints. In [46], testing was introduced into the ICP framework in order to make the framework quickly find satisfying assignments in the case that (strict) inequalities have solutions. The authors in [46] further suggested a combination of interval arithmetic (IA) and the intermediate value theorem (IVT) as a heuristic to show satisfiability of equations. This thesis further extends the idea to allow combining the result of testing with IA and IVT to show satisfiability of combinations between inequalities and equations. Several efficiency heuristics are also proposed in this thesis. The extensions were implemented inside the SMT solver **raSAT** which uses miniSAT as the SAT solver.

Observation on timed-out SMT-LIB benchmarks for the previous extension shows that on average one-third of the running time was spent on testing phase. As a result, a heuristic for the quick model finding is desirable for improvement of the framework. Recently, a method of subtropical real root finding is introduced in the context of symbolic computation [76] with the aim of computing real zeros of the single very large multivariate polynomial. The method takes an abstraction of polynomials as exponent vectors over the natural numbers tagged with the signs of the corresponding coefficients. It then uses linear programming to determine which monomial dominates others in a polynomial and computes two points at which the polynomial is positive and negative respectively. The

intermediate value theorem provides a guide for computing the root of the polynomial between such two points. Ignoring the task of finding roots, this thesis extends the idea to find points at which multiple polynomials are all positive. The task is reduced to solving linear constraints over reals. The existence of a solution for the generated linear constraints entails the existence of a point at which all the original polynomials are positive. The proposed procedure was implemented as the **STROPSAT** library.

Assuming the existence of the above two efficient methods, we combine them with a complete procedure to produce an efficient complete framework. For a complete procedure, we use quantifier elimination methods, namely CAD and virtual substitution, implemented in the Redlog/Reduce package that provides interfaces for interacting with SMT solvers. Two schemes of combinations are proposed and implemented inside the nonlinear theory solver of the SMT solver **veriT**. We experimentally evaluated the combination schemes on SMT-LIB benchmarks.

This thesis is organized as follows. Chapter 2 overviews general SMT solving and SMT solving for polynomial constraints. Various techniques for solving non-linear arithmetic are discussed in Chapter 3. An introduction for ICP is given in Chapter 4. Next two chapters propose heuristics for handling inequalities. While chapter 5 describes the extension of ICP with testing and efficiency heuristics for solving inequalities, an extension of the subtropical method for finding a model of inequalities is discussed in Chapter 6. Chapter 7 introduces the combination of testing, IA, and the IVT to solve combinations of inequalities and equations. Chapter 8 introduces two schemes of combining distinct components. Chapter 10 compares our ICP implementation with ones in other solvers. Before concluding in Chapter 12, we address some future work in Chapter 11.

Fig. 1.1 illustrates the contributions of the thesis. While raSAT loop and subtropical satisfiability aim at inequalities solving, the application of the IVT aims at equations solving. raSAT loop and the application of the IVT were implemented in the SMT solver **raSAT** which uses miniSAT as the SAT solver. Subtropical satisfiability was implemented as the **STROPSAT** library. When combining those procedures with complete methods, the theory reasoner of **raSAT** and the library **STROPSAT** are first integrated as modules of the SMT solver **veriT** by implementing required theory interfaces of **veriT**. Combining those modules with the complete framework is done inside the nonlinear theory solver of **veriT** by making calls to the implemented interfaces.

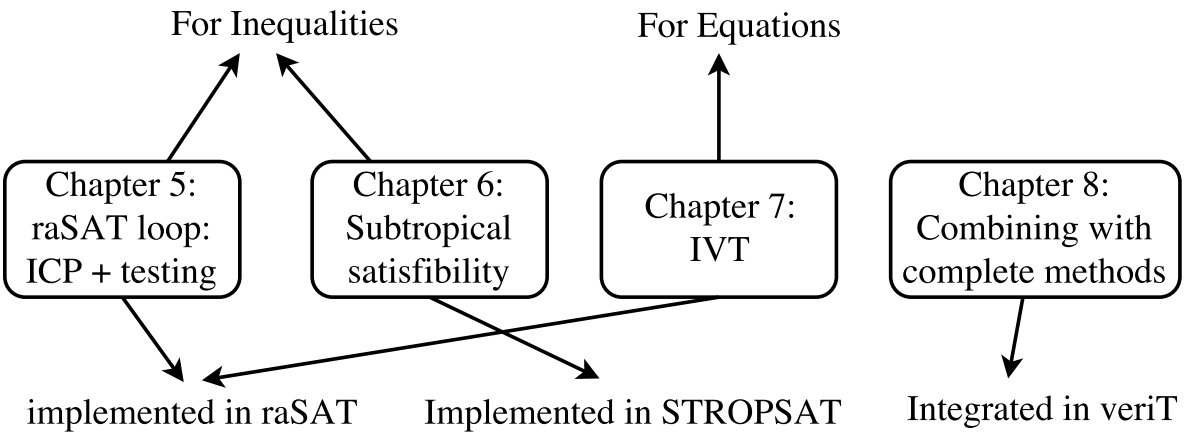


Figure 1.1: Contributions of the thesis

Chapter 2

SMT Solving for Polynomial Constraints

2.1 Satisfiability Modulo Theories

2.1.1 Syntax

The syntax of Satisfiability Modulo Theories (SMT) problems follows the classical first-order logic. To make the chapter self-contained, however, here we introduce all the relevant concepts and notations.

Definition 1 *A signature Σ is a 4-tuple (F, P, V, α) consisting of a set F of* function *symbols, a set P of* predicate *symbols, a set V of variables, and $\alpha : (F \cup P) \to \mathbb{N}$ is a function mapping each function or predicate symbol to one natural number indicating the arity of the symbol.*

We call the 0-arity symbols of F and P *constant* symbols and *propositional* symbols respectively.

A term is constructed by either a single variable or a function application with terms as arguments.

Definition 2 *Given a signature $\Sigma = (F, P, V, \alpha)$, a Σ-term t is defined as follows.*

$$\begin{array}{rcll} t & := & v & \text{where } v \in V \\ & | & f(t_1, \cdots, t_n) & \text{where } f \in F,\ \alpha(f) = n, \text{ and } t_1, \ldots, t_n \text{ are terms} \end{array}$$

A formula is either a relation between terms (applying some predicate symbol to terms), a propositional *bottom* $\bot$ symbol, a boolean combination between formulas, or a quantified formula.

Definition 3 *Given a signature $\Sigma = (F, P, V, \alpha)$, a Σ-formula φ is defined as follows.*

$$\begin{array}{rcll} \varphi & := & p(t_1, \cdots, t_n) & \text{where } p \in P,\ n = \alpha(p), \text{ and } t_1, \ldots, t_n \text{ are terms} \\ & | & \bot \mid \neg\varphi & \\ & | & \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 & \text{where } \varphi_1 \text{ and } \varphi_2 \text{ are formulas} \\ & | & \exists v\varphi \mid \forall v\varphi & \text{where } v \in V \end{array}$$

Atomic formulas *are of the form $p(t_1, \ldots, t_n)$ or $\bot$.*

Given a signature $\Sigma = (F, P, V, \alpha)$, we define the function $var(\_)$ as the set of (free) variables appearing in Σ-terms and Σ-formulas as following.

Definition 4 *The set of variables appearing in a term t, denoted by $var(t)$, is recursively defined as:*

1. $var(v) = \{v\}$ *where* $v \in V$,
2. $var(f(t_1, \ldots, t_n)) = \bigcup_{i=1}^{n} var(t_i)$ *where* $f \in F$ *and* $n = \alpha(f)$.

Definition 5 *The set of variables appearing in a formula φ, denoted by $var(\varphi)$, is recursively defined as:*

1. $var(p(t_1, \ldots, t_n)) = \bigcup_{i=1}^{n} var(t_i)$ *where* $p \in P$ *and* $n = \alpha(p)$;
2. $var(\bot) = \emptyset$, $var(\neg\varphi) = var(\varphi)$;
3. $var(\psi_1 \wedge \psi_2) = var(\psi_1) \cup var(\psi_2)$;
4. $var(\psi_1 \vee \psi_2) = var(\psi_1) \cup var(\psi_2)$;
5. $var(\exists v\varphi) = var(\varphi) \setminus \{v\}$, *and* $var(\forall v\varphi) = var(\varphi) \setminus \{v\}$.

2.1.2 Semantics

Formulas are given a truth value from the set $\{\text{TRUE}, \text{FALSE}\}$ by means of *interpretations* which are pairs of a *structure* and a *valuation*.

Definition 6 *Given a signature $\Sigma = (F, P, V, \alpha)$, a Σ-structure S is a tuple (U, P_u, F_u) where U is a* universe*, and*

- *F_u maps each predicate symbol $p \in P$ of arity $n = \alpha(p)$ to a function $I(p) : U^n \to \{\text{TRUE}, \text{FALSE}\}$,*
- *P_u maps each function symbol $f \in F$ of arity $n = \alpha(f)$ to a function $I(f) : U^n \to U$.*

Definition 7 *Given a signature $\Sigma = (F, P, V, \alpha)$, a Σ-interpretation I is a pair $(S, \mathbb{V})$ consisting of a structure $S = (U, P_u, F_u)$ and a valuation $\mathbb{V}$ which maps each variable $v \in V$ to an element $\mathbb{V}(v) \in U$.*

Given an interpretation, we can evaluate the values of terms.

Definition 8 *Given a signature $\Sigma = (F, P, V, \alpha)$ and a Σ-interpretation $I = (S, \mathbb{V})$ where $S = (U, P_u, V_u)$, the evaluation $t^I \in U$ of a term t is defined as followings.*

$$\begin{array}{lcll} v^I & := & \mathbb{V}(v) & \text{where } v \in V, \text{ and} \\ f^I(t_1, \cdots, t_n) & := & F_u(f)(t_1^I, \cdots, t_n^I) & \text{where } f \in F \text{ and } n = \alpha(f) \end{array}$$

Similarly, we define the evaluation of a formula with respect to a model.

Definition 9 *Given a signature $\Sigma = (F, P, V, \alpha)$ and a Σ-interpretation $I = (S, \mathbb{V})$ where $S = (U, P_u, V_u)$, the evaluation $\varphi^I \in \{\text{TRUE}, \text{FALSE}\}$ of a formula φ is defined as followings.*

$$
\begin{array}{lcl}
p^I(t_1, \cdots, t_n) & := & P_u(p)(t_1^I, \cdots, t_n^I) \text{ where } p \in P \text{ and } n = \alpha(p) \\
\bot^I & := & \text{FALSE} \\
(\neg\varphi)^I & := & \neg(\varphi^I) \\
(\varphi_1 \wedge \varphi_2)^I & := & \varphi_1^I \wedge \varphi_2^I \\
(\varphi_1 \vee \varphi_2)^I & := & \varphi_1^I \vee \varphi_2^I \\
(\exists v\varphi)^I & := & \begin{cases} \text{TRUE } \textit{if there exists } u \in U \textit{ such that } \varphi^{I[u/v]} = \text{TRUE}, \\ \text{FALSE } \textit{otherwise} \end{cases} \\
(\forall v\varphi)^I & := & \begin{cases} \text{TRUE } \textit{if for all } u \in U, \textit{ we have } \varphi^{I[u/v]} = \text{TRUE}, \\ \text{FALSE } \textit{otherwise} \end{cases}
\end{array}
$$

where $I[u/v]$ denotes the addition (replacing if already exists) of the mapping from variable v to value u into the evaluation $\mathbb{V}$ in I.

We say that a Σ-interpretation M *satisfies* (respectively *falsifies*) a Σ-formula φ if and only if φ^M is TRUE (respectively FALSE). In SMT solving, we are only interested in interpretations belonging to a given *theory* T which restricts the domain of variables, the interpretation of function symbols and predicate symbols. Traditionally one theory is defined by a set of axioms, and an interpretation is said to belong to such a theory if and only if it satisfies all the axioms of the theory. Following the more recent SMT literature, one Σ-theory is most generally defined as a set of (possibly infinitely many) Σ-interpretations. A Σ-formula φ is T-satisfiable for a Σ-theory T, if and only if there is an element of the set T that satisfies φ. Otherwise, φ is T-*unsatisfiable*.

Given a theory T, SMT solving is the problem of determining if a formula is T-satisfiable or not. When the theory is clear from the context, we often say that the formula is satisfiable, or unsatisfiable instead of T-satisfiable, or T-unsatisfiable respectively. Since the theory specifies the universe U and the interpretation of function and predicate symbols, SMT solving can be simply stated as the problem of finding an assignment from variables to values in the universe U such that the assignment makes the formula TRUE. Moreover, an assignment θ uniquely determines an interpretation when the theory is specified. Later on, we will use notations of assignments as interpretations. For example, φ^θ is the same as φ^I where I is the interpretation uniquely determined by θ.

2.1.3 Methods for SMT Solving

We can distinguish two main approaches for SMT solving, namely *eager* and *lazy* approaches.

Eager Approach

This method involves translating the original formula into an equisatisfiable Boolean formula in a single step. The translation by nature is theory-specific.

In [4], the authors show that if linear constraints over integers have a satisfying solution, then there exists a solution with size in bits is polynomially bounded in terms of the number of variables, the number of constraints, and the maximum constants appearing in the constraints. As a result, one can easily convert the original linear constraints into a propositional formula by representing each integer-valued variable as a sequence

of Boolean (bits) variables whose size is the above polynomial bound. Arithmetic and comparison operations are converted into propositional formulas (see [89]). In the end, we have an SAT formula which is equisatisfiable to the original linear constraint, and the generated SAT formula can be efficiently solved by state-of-the-art SAT solvers.

Another approach for solving linear constraints over rationals (or integers) was proposed in [74] which is based on the Fourier-Motzkin elimination method [16]. The Fourier-Motzkin elimination method works by eliminating variables one after the other until reaching comparisons between constants and thus the constraints can be decided. Suppose we are eliminating variable x_n. From each linear constraint containing x_n, we can compute either lower bound or upper bound of x_n as a linear expression of other variables. For example, the linear constraint $2x_1 + 4x_2 - 3x_3 \leq 0$ implies that $3x_3 \geq 2x_1 + 4x_2$, or x_3 has the lower bound as $\frac{2}{3}x_1 + \frac{4}{3}x_2$. For every pair (u, l) of lower bound and upper bound of x_n, i.e. $u \leq x_n$ and $x_n \leq l$, we can derive a new linear constraint $u \leq l$ which does not contain x_n. The method in [74] introduces new Boolean variables e_1, e_2, and e_3 for $u \leq x_n$, $x_n \leq l$, and $u \leq l$ respectively. After adding the new constraint e_3 (representing $u \leq l$) into the SAT formula, we need to further add $e_1 \wedge e_2 \rightarrow e_3$. It should be noted that the Fourier-Motzkin elimination method works on a conjunction of linear constraints, and the method in [74] apply the Fourier-Motzkin elimination method for the set of all linear constraints appearing in the original formulas which has an arbitrary Boolean structure. The conjunction of the generated SAT formula and the corresponding SAT formula of the original linear formula ensures the equisatisfiability.

Lazy Approach

The lazy approach integrates theory solvers into SAT solvers. In the simplest form, a theory solver for theory T is a procedure which takes as input a conjunction of atomic formulas with possibly negations and decides whether the conjunction is T-satisfiable or not. First, each atomic formula in the input is abstracted by a Boolean variable, resulting in an SAT formula. The SAT formula is passed to an SAT solver to enumerate the propositional models. Each of such models is a conjunction φ of atomic formulas with possibly negations and is checked by the theory solver.

If the solver detects unsatisfiability, it is desirable to find a (possibly minimal) subset of φ which is already unsatisfiable. Such a subset is called an *unsat core*. The negation of the unsat core is added into the SAT solver so that the propositional models containing the same cause of inconsistency will not be considered again by the SAT solver.

Otherwise, if the theory solver detects satisfiability, it is required to provide a witness, i.e. an assignment for variables which satisfies the conjunction. In some combinations of decision procedures, the theory solvers are also required to perform deductions of equalities between variables [20].

In a *less lazy* approach, the theory solver is integrated more tightly within the DPLL procedure of SAT solving where deciding literals and unit propagation are assisted by theory consistency ([28, 43]).

2.2 SMT Solving for Polynomial Constraints

2.2.1 Syntax

The signature $\Sigma = (F, P, V, \alpha)$ (for simplicity, we abuse the notations introduced in Sec. 2.1 for polynomial constraints) for polynomial constraints consists of

- a set $F = \{+, -, *, 1, 0\}$ of function symbols,
- a set $P = \{<, >, \geq, \leq, \neq\}$ of predicate symbols,
- a set V of variables, and
- the function $\alpha : (F \cup P) \to \mathbb{N}$, such that $\alpha(1) = 0$, and $\alpha(s) = 2$ for $s \in (F \cup P) \setminus \{1\}$.

A polynomial is a Σ-term (Definition 2), and a polynomial constraint is a Σ-formula with the exclusion of quantified formulas, i.e., we only interested in formulas without quantifiers $\exists$ and $\forall$. We call an atomic Σ-formula an *atomic polynomial constraint* (APC).

For technical convenience, any conjunction of APCs is represented as a set of APCs.

2.2.2 Semantics

In SMT solving for polynomial constraints, we are interested in interpretations belonging to the theory of real numbers $\mathbb{R}$ [55] equipped with the ordinary arithmetic operations $\{+, -, *\}$ and comparison operations $\{<, >, \geq, \leq, \neq\}$. Again, here we abuse the function and predicate symbols in Sec. 2.2.1, but it should be clear to distinguish when we are using them for syntax and when we are using them for semantics.

Definition 10 *A Σ-structure S for polynomial constraints is a tuple $(\mathbb{R}, P_u, F_u)$ where*

- *F_u maps each function symbol in $\{+, -, *\}$ to the corresponding arithmetic operation, e.g. $F_u(+) = +$, and maps $0, 1 \in F$ to the identity elements $0, 1$ respectively of $\mathbb{R}$,*
- *P_u maps each predicate symbol in $\{<, >, \geq, \leq, \neq\}$ to the corresponding comparison equipped with $\mathbb{R}$.*

Definition 11 *A Σ-interpretation I for polynomial constraints is a pair $(S, \mathbb{V})$ consisting of a structure $S = (\mathbb{R}, P_u, F_u)$ and a valuation $\mathbb{V}$ which maps each variable $v \in V$ to an element $\mathbb{V}(v) \in \mathbb{R}$.*

Given a polynomial constraint φ and an interpretation I, we can evaluate the truth value of φ with respect to I following the Definition 9.

To the best of our knowledge, there has not been any eager approach towards SMT solving for polynomial constraints. Although the method in [89] reduces polynomial constraints to SAT solving by only considering interpretation over rationals, the translation is an under-approximation which does not preserve equisatisfiability.

The state-of-the-art methodologies for polynomial SMT solving are mostly based on the lazy approach. We examine them in the next chapter and classify them by their properties.

Chapter 3

Existing Methodologies for Solving Polynomial Constraints

3.1 Algebraic Methods

Although methods described in this section can be used for general quantifier elimination of arbitrary first-order logic formula over real closed fields, we summarize them in the context of SMT solving, i.e., restricting the variables in the formulas to be existentially quantified.

3.1.1 Cylindrical Algebraic Decomposition

Cylindrical Algebraic Decomposition (CAD) is a complete decision procedure for deciding constraints over real closed fields. The basic idea of CAD is to split the space into a finite set of disjoint components (cells) such that in each cell, polynomials in the constraints have constant signs. The constraints are thus either TRUE or FALSE at all points in each cell.

Suppose we have a set $F_n = \{f_1, \ldots, f_m\}$ of m polynomials over n variables $\{x_1, \ldots, x_n\}$. CAD aims at partitioning $\mathbb{R}^n$ into connected cells such that in each of the cells, each polynomial in F_n has a constant sign. A set of polynomials is said to be *sign-invariant* over a set C if each polynomial in it has a constant sign at all points in C.

CAD consists of two steps, namely projection and lifting phases which are illustrated in Figure 3.1.

After each step of projection, a new set $F_{k-1} \in \mathbb{R}[x_1, \ldots, x_{k-1}]$ is constructed from $F_k \in \mathbb{R}[x_1, \ldots, x_k]$ such that if F_{k-1} is sign-invariant over $C \subset \mathbb{R}^{k-1}$, then for every $c_1, c_2 \in C$, and every $f \in F_k$, $f(c_1, x_k), f(c_2, x_k)$ both have the same number of different real and complex roots. In this case, F_k is said to be delineable on C. Technically, each polynomial in F_{k-1} is computed from coefficients of polynomials in F_k, certain heading coefficients of subresultants of each pair of polynomials in F_k and certain subresultants of each polynomial with their derivative. Details for those operations can be found at [12].

Example 1 *Consider* $F_2 = \{x_1^2 + x_2^2 - 1, -4x_1x_2 - 4x_1 + x_2 - 1\}$. *The projection phase creates the following new polynomials*

$$F_1 = \{1, x_1^2 - 1, -4x_1 + 1, -4x_1 + 1, -4x_1 - 1, -4(x_1^2 - 1), x(16x_1^3 - 8x_1^2 + x_1 + 16)\}$$

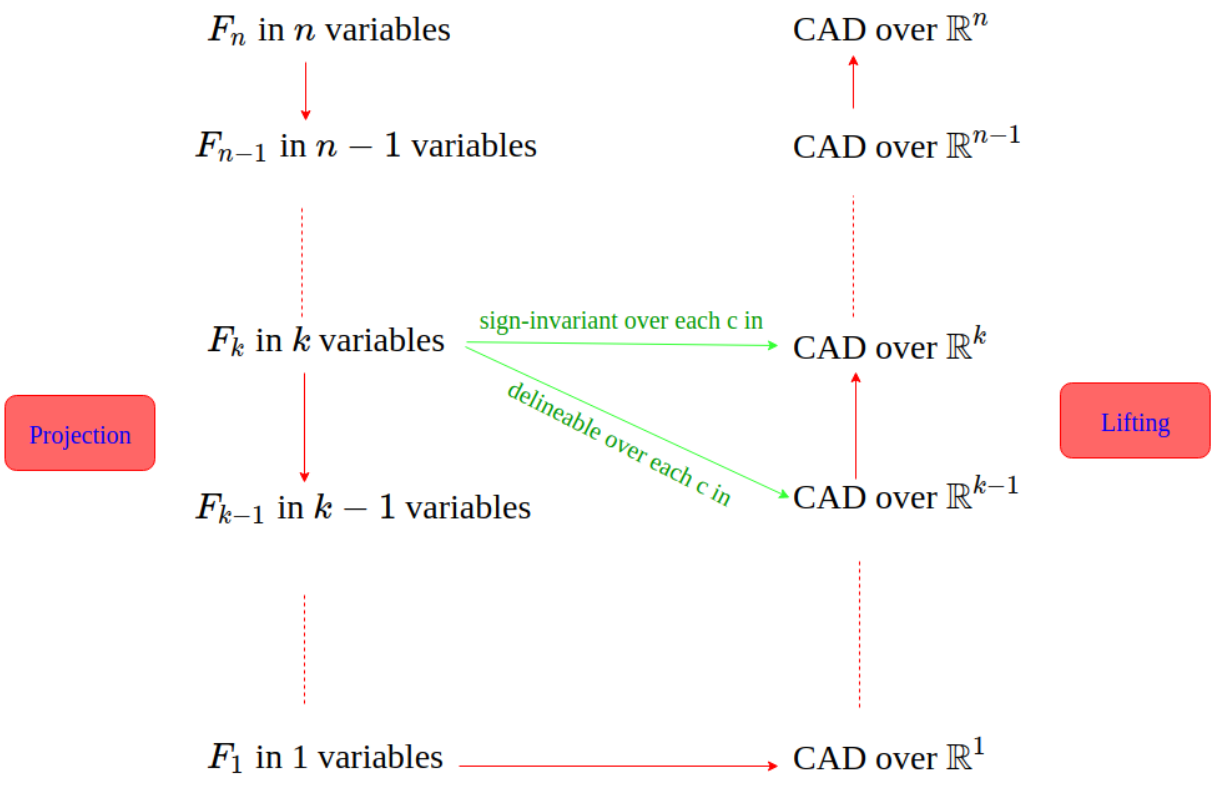


Figure 3.1: Three steps of CAD

At the end of the projection phase, we have a set F_1 of univariate polynomials. The lifting phase starts by constructing CAD of F_1 over $\mathbb{R}$ by isolating roots of polynomials in F_1 and partitioning $\mathbb{R}$ by those roots. Technically root isolation is done by iteratively splitting intervals and counting the number of roots (using Sturm sequence [55]) in each interval until we can isolate each root inside one interval [55]. Initially, bounds on roots are computed using Cauchy's inequality [55]. An algebraic number is usually represented as (l, r, f) where $l, r \in \mathbb{Q}$, f is the polynomial that has the considering algebraic number as one and only one root in $]l, r[$.

Example 2 *Example 1 cont. We need to compute roots of polynomials in* F_1. *Let us take* $16x_1^3 - 8x_1^2 + x_1 + 16$ *as an example. Cauchy's inequality [55] gives bounds for roots as* $]-2, 2[$. *Refinement of the interval [55] gives* $]-0.9, -0.8[$ *which also contains exactly one root of* $16x_1^3 - 8x_1^2 + x_1 + 16$. *Denote* $\alpha = (-0.9, -0.8, 16x_1^3 - 8x_1^2 + x_1 + 16)$. *The roots of polynomials in* F_1 *are*

$$\{-1, \alpha, -\frac{1}{4}, 0, \frac{1}{4}, 1\}$$

Then CAD of F_1 *over* $\mathbb{R}$ *is*

$$\{]-\infty, -1[, [-1, -1],]-1, \alpha[, [\alpha, \alpha],]\alpha, -\frac{1}{4}[, [-\frac{1}{4}, -\frac{1}{4}],]-\frac{1}{4}, 0[,$$

$$[0, 0],]0, \frac{1}{4}[, [\frac{1}{4}, \frac{1}{4}],]\frac{1}{4}, 1[, [1, 1],]1, \infty[\}$$

We can see that F_1 is sign-invariant over each cell in the computed CAD over $\mathbb{R}$. Sample points of the constructed CAD are computed by the following function

$$\text{SAMPLE}([l, r]) = l$$

$$\text{SAMPLE}(]l, r[) = \begin{cases} r - 1, & \textit{if } l = -\infty \\ l + 1, & \textit{if } r = \infty \\ \frac{l+r}{2}, & \textit{otherwise} \end{cases}$$

For general quantifier elimination, representation of algebraic cells is required to produce the final quantifier-free formula. Each cell in CAD over $\mathbb{R}^i$ can be uniquely defined by polynomial constraints which are computed from the polynomials in F_i [12]. However, in the context of SMT solving, such a requirement is unnecessary, and only the sample points of cells are enough to decide satisfiability of the existentially closed sentence. As a result, we here represent each cell only by its sample point.

Example 3 *Continue the Example 1, the CAD of F_1 over $\mathbb{R}$ can be uniquely defined by* $\{-2, -1, \frac{-1+\alpha}{2}, \alpha, \frac{\alpha-\frac{1}{4}}{2}, -\frac{1}{4}, -\frac{1}{8}, 0, \frac{1}{8}, \frac{1}{4}, \frac{5}{8}, 1, 2\}$.

For $1 < i \leq n$, CAD over $\mathbb{R}^i$ are constructed recursively. Because projection phase ensures the delineability of F_i on each cell C of CAD over $\mathbb{R}^{i-1}$, the roots of polynomials in F_i with respect to variable x_i cut the cylinder $C \times \mathbb{R}$ above C into finitely many connected cells. CAD over $\mathbb{R}^i$ is computed by taking all those cells arising from cells of CAD over $\mathbb{R}^{i-1}$. Sample points of new cells arising from the base cell C in $\mathbb{R}^{i-1}$ are computed by lifting the sample point of C into various cells of the cylinder $C \times \mathbb{R}$. First, the sample point of C is plugged into polynomials in F_i making them univariate ones in x_i with coefficients as algebraic numbers and root isolation is implemented to find the values of x_i in the new sample points for the corresponding new cells. It should be noticed that when plugging sample points into polynomials, we will have new polynomials with coefficients as algebraic numbers. The root isolation algorithm (Cauchy's inequality and Sturm sequence) works with any real closed field [55], and thus the real algebraic field. The readers are encouraged to read [55] for arithmetic operations over real algebraic numbers.

Example 4 *Continue the Example 1. Consider the cell C represented by* 0 *of CAD of F_1 over $\mathbb{R}$. Plugging $x_1 = 0$ into polynomials in F_2 gives $\{x_2^2 - 1, x_2 - 1\}$. Roots of the polynomials in this set are $\{-1, 1\}$. Thus the cylinder $C \times \mathbb{R}$ is decomposed into* 5 *cells $\{C\times]-\infty, -1[, C \times [-1, -1], C\times]-1, 1[, C \times [1, 1], C \times [1, \infty]\}$ which are represented by $\{(0, -2), (0, -1), (0, 0), (0, 1), (0, 2)\}$*

The complexity of CAD is doubly exponential in the number of variables [12]. It is implemented in RAHD [63, 62], Z3 4.3 [43], Yices [24], and SMT-RAT [15].

3.1.2 Gröbner bases

The fundamental idea of Gröbner bases is from a set of polynomials to construct a new set of polynomials (Gröbner basis) such that the ideal generated by such a new set is identical to the ideal generated by the original polynomials. As a consequence, the original

polynomials have some common zeros if and only if the newly constructed polynomials also do.

In the case of polynomials over complex numbers, the Hilbert's Nullstellensatz theorem [55] implies that the set of polynomials do not have any common zeros if and only if the Gröbner basis of the ideal generated by such polynomials contains 1. The procedure is then formulated by first computing the Gröbner basis of the polynomials in the equations, then check if 1 is in the Gröbner basis or not.

In the case of real numbers, we need to take more care since the Gröbner basis gives a sound but incomplete procedure for solving polynomial constraints since a set of polynomials having common zeros over complex numbers do not necessarily have common zeros over reals.

Example 5 *Consider the unsatisfiable equation over reals* $x^2 + 1 = 0$. *Gröbner basis of the ideal* $(\{x^2 + 1\})$ *is* $\{x^2 + 1\}$ *which does not contain* 1. *As a result,* $x^2 + 1$ *have some zeros over complex numbers, but in fact, those zeros are not reals.*

Let us give some definitions related to Gröbner basis first before explaining a method for deciding satisfiability of constraints over real numbers using Gröbner basis.

Let $\mathbb{Q}[x_1, \ldots, x_n]$ be the set of multivariate polynomials over the variables $x_1, \ldots, x_n$ with coefficients in $\mathbb{Q}$. A subset $I \subseteq \mathbb{Q}[x_1, \ldots, x_n]$ is an ideal iff I is a subgroup with respect to addition, and for all $f \in I$, $g \in \mathbb{Q}[x_1, \ldots, x_n]$ we have $fg \in I$. The ideal generated by a set $G \subseteq \mathbb{Q}[x_1, \ldots, x_n]$, denoted by (G), is the smallest ideal I containing G.

An admissible monomial order $\prec$ is a strict well-order on monomials such that if $u \prec v$ then $uw \prec vw$ for arbitrary monomials u, v, w. Admissible orders are extended to $\mathbb{Q}[x_1, \ldots, x_n]$ as a multiset order [7].

We say that a polynomial $f \in \mathbb{Q}[x_1, \ldots, x_n]$ is reduced to $g \in \mathbb{Q}[x_1, \ldots, x_n]$ with respect to a set $G \subset \mathbb{Q}[x_1, \ldots, x_n]$ iff there exist $f_0, \ldots, f_m \in \mathbb{Q}[x_1, \ldots, x_n]$ such that $f_0 = f, f_m = g$, for all $0 \leq i < m$, $f_{i+1} = f_i - h_i g_i$ for some $h_i \in \mathbb{Q}[x_1, \ldots, x_n]$, $g_i \in G$, and $f_{i+1} \prec f_i$. If g cannot be reduced any more with respect to G, then we write $g = red_G(f)$.

Definition 12 *Given an ideal* $I \subseteq \mathbb{Q}[x_1, \ldots, x_n]$, *a finite subset* $G \subseteq I$ *is a* Gröbner basis *if and only if* $I = (G)$ *and* $red_G(f)$ *is unique for any* $f \in \mathbb{Q}[x_1, \ldots, x_n]$.

Theorem 1 ([72]) *Let* R *be a real closed field, and* $G \in R[x_1, \ldots, x_n]$. *Then the set* $\{x \in R^n \mid g(x) = 0$ *for all* $g \in G\}$ *of common roots of polynomials in* G *is empty if and only if there exist polynomials* $s_1, \ldots, s_m \in R[x_1, \ldots, x_n]$ *such that* $1 + s_1^2 + \cdots + s_m^2 \in (G)$. *Moreover, if* $G \subseteq \mathbb{Q}[x_1, \ldots, x_n]$, *then* $s_1, \ldots, s_m$ *can be chosen among* $\mathbb{Q}[x_1, \ldots, x_n]$.

This theorem yields a complete decision procedure for solving polynomial constraints assuming a complete algorithm for finding witnesses $s_1, \ldots, s_m$. It should be noted that any inequality can be transformed into an equisatisfiable equation by introducing new variables. A method using semidefinite programming was proposed to find such a witness [64]. The idea is to parameterize the resulting polynomials $s_1, \ldots, s_m$ and encode the search in terms of semidefinite programs. Since degree bounds on witnesses exist [61], a decision procedure combining Gröbner basis and semidefinite programming for deciding polynomial constraints exists. However, these bounds are at least triply exponential [61], the search needs to incrementally increase the bound and search for polynomials with such a bounded degree [38, 64].

Example 6 *Consider the unsatisfiable constraint* $\varphi := x \geq y \wedge z \geq 0 \wedge xz < yz$. *First, it is transformed into a satisfiability-equivalent constraint* $\varphi' := x - y = a^2 \wedge z = b^2 \wedge (yz - xz) * c^2 = 1$ *consisting of only equations. The Gröbner basis of the ideal* $I = (x - y - a^2, z - b^2, (yz - xz) * c^2 - 1)$ *with respect to variables order* $a \prec b \prec c \prec x \prec y \prec z$ *is* $\{a^2 - x + y, b^2 - z, c^2x - c^2yz + 1\}$. *The method proposed in [64] finds a witness* $1 + a^2b^2c^2 \in I$. *By Theorem 1, polynomials in* $\{x - y - a^2, z - b^2, (yz - xz) * c^2 - 1\}$ *do not have any common zeros, thus* φ', *as well as* φ, *are unsatisfiable.*

3.2 Topological Methods

3.2.1 Virtual Substitution

The central idea of virtual substitution is to find a finite set of test points which are enough to decide the satisfiability of the constraints. Such points are computed from root formulas of polynomials appearing in the constraints. Since the method requires roots formulas, it theoretically limits the application to only polynomials whose degree is less than or equal to three.

Example 7 *Consider the constraint* $f_1 > 0 \wedge f_2 < 0$ *where* $f_1 := x^2 + xy + 1$ *and* $f_2 := xy$. *Consider* f_1 *as a polynomial over variable* x *leaving* y *as a parameter, and then consider the following possible cases and test points.*

- *If* $y^2 - 4 < 0$, *then* $x^2 + xy + 1$ *does not have any zeros, thus the test point* $-\infty$ *is enough to check the sign of* $x^2 + xy + 1$, *and thus decide the satisfiability of* $x^2 + xy + 1$ *over* $\mathbb{R}$.

- *If* $y^2 - 4 = 0$, $x^2 + xy + 1$ *has only one real root* $\frac{-y}{2}$. *Checking the two points* $\{-\infty, \frac{-y}{2}\}$ *is enough to conclude satisfiability of* $x^2 + xy + 1 > 0$. *In this case, the test point* $\frac{-y}{2}$ *can be safely discarded since it makes* $x^2 + xy + 1$ *zero.*

- *If* $y^2 - 4 > 0$, $x^2 + xy + 1$ *has two distinct real roots:* $e_1 = \frac{-y - \sqrt{y^2-4}}{2}$ *and* $e_2 = \frac{-y + \sqrt{y^2-4}}{2}$. *Checking* 5 *test points* $\{-\infty, e_1, e_1 + \varepsilon, e_2, \infty\}$ *is enough to conclude the satisfiability of* $x^2 + xy + 1 > 0$. *Further,* e_1 *and* e_2 *can be discarded because* $x^2 + xy + 1$ *becomes* 0 *at them.*

In this example, $0 < \varepsilon < \mathbb{R}^+$ is interpreted as a positive infinitesimal [86]. After testing points along with their existence conditions are computed, they are virtually substituted into the original constraints resulting in new constraints with one variable eliminated. We write $f[e/x]$ to denote the result of the substitution of variable x by expression e into polynomial f.

Example 8 *Continuing the previous example, the original constraint is equivalent to the*

following formula over reals.

$$\begin{aligned}
&(y^2-4<0 \wedge f_1[-\infty/x]>0 \wedge f_2[-\infty/x]<0)\\
\vee&(y^2-4=0 \wedge f_1[-\infty/x]>0 \wedge f_2[-\infty/x]<0)\\
\vee&(y^2-4=0 \wedge f_1[\frac{-y}{3}/x]>0 \wedge f_2[\frac{-y}{3}/x]<0)\\
\vee&(y^2-4>0 \wedge f_1[-\infty/x]>0 \wedge f_2[-\infty/x]<0)\\
\vee&(y^2-4>0 \wedge f_1[e_1/x]>0 \wedge f_2[e_1/x]<0)\\
\vee&(y^2-4>0 \wedge f_1[e_1+\varepsilon/x]>0 \wedge f_2[e_1+\varepsilon/x]<0)\\
\vee&(y^2-4>0 \wedge f_1[e_2/x]>0 \wedge f_2[e_2/x]<0)\\
\vee&(y^2-4>0 \wedge f_1[\infty/x]>0 \wedge f_2[\infty/x]<0)
\end{aligned}$$

In order to continue virtual substitution for remaining variables, the following three obligations need to be removed:

- the substitution of infinity,
- removing square-root expressions, and
- removing ε.

The detailed techniques for removing those expressions are presented in [86]. We are going to provide examples to illustrate the method.

Example 9 *The following example illustrates the removal of infinity.*

$$f_1[-\infty/x]>0 := x^2(1+\frac{y}{x}+\frac{1}{x^2})[-\infty/x]>0 := \infty>0 \equiv \text{TRUE}$$

The following example illustrates the removal of square-root expression.

$$\begin{aligned}
xy[\frac{-y-\sqrt{y^2-4}}{2}/x]<0 &:= \frac{-y^2-y\sqrt{y^2-4}}{2}<0\\
&\equiv y>0 \vee \big(y\le 0 \wedge y^4-y^2(y^2-4)>0\big)
\end{aligned}$$

Finally, the following example illustrates the removal of infinitesimal symbol for the constraint $f>0$ and the test point $e+\varepsilon$ for variable x. Define the function $v(f)$ as following with n is the degree of f with respect to variable x.

$$v(f)=\begin{cases} f>0 & \text{if } n=0\\ f>0 \vee (f=0 \wedge v(f')) & \text{otherwise}\end{cases}$$

Then, we have

$$f[e+\varepsilon/x]>0 := v(f)[e/x]$$

The intuition is that since f is continuous, and ε is arbitrarily small, f is greater than 0 at point $x=e+\varepsilon$ if and only if one of two following cases happens.

1. *Polynomial f is positive at the point $x=e$, then since f is continuous, it is still positive in a small neighborhood of e.*
2. *Polynomial f becomes 0 at $x=e$, and its derivative f' is positive at $x=e$. Since $f'[e/x]>0$, $f[e+\varepsilon/x]>f[e/x]=0$.*

The complexity of Virtual Substitution for SMT problem (existential quantified formulas) is exponential in the number of variables [86]. The method has been implemented in Redlog/Reduce [21], SMT-RAT [15].

3.2.2 Interval Constraint Propagation

This section briefly introduces Interval Constraint Propagation (ICP) [3], a numerical technique that provides an incomplete but practically very efficient method to check the satisfiability of a set of polynomial constraints [28, 65, 32]. The mechanism of the ICP is to iteratively evaluate polynomial constraints over boxes (Cartesian products of intervals), contract boxes using constraints, and decompose boxes until either SAT or UNSAT is detected. Evaluation of polynomial constraints is done by interval arithmetic which over-approximates the ranges of polynomials given intervals of variables.

Example 10 *Consider* $x^2 + xy - 4 < 0$ *and* $x \in [-3, 1], y \in [-4, -2]$*, then interval arithmetic [57] approximates the value of* $x^2 + xy - 4$ *as*

$$[-3, 1] * [-3, 1] + [-3, 1][-4, -2] - [1, 1] = [-11, 17]$$

Comparing $[-11, 17]$ *with* 0*, we cannot conclude if* $x^2 + xy - 4 < 0$ *is satisfiable.*

If evaluating constraints does not result in SAT/UNSAT answers, the constraints are in turn used to contract the boxes into possibly smaller ones.

Example 11 *Continue Example 10, since the constraint is* $x^2 + xy - 4 < 0$ *and the range of* $x^2 + xy - 4$ *is* $[-11, 17]$*, the value of* $x^2 + xy - 4$ *must be in* $[-11, 0]$ *at points in* $[-3, 1] \times [-4, -2]$ *that satisfy* $x^2 + xy - 4 < 0$*. Doing interval arithmetic in the backward manner gives the contracted box* $[-2, 1] \times [-4, -2]$ *(The detail of operations will be described in Chapter 4).*

If the box can be contracted successfully, the constraints will be evaluated over such a contracted box again and possibly the contraction will be done again until no changes of the box. If the contracted box is the same as the current box, then the box is decomposed into smaller ones and each of them will be processed in the same procedure as above.

ICP concludes unsatisfiability when no box satisfies the constraints and concludes satisfiability if some box makes the constraints valid (all points in the box satisfy the constraints).

Since the ICP is based on interval arithmetic which is an over-approximation, its applicability is limited to detection of satisfiability of inequalities and unsatisfiability of equations/inequalities. The ICP is almost incapable of proving satisfiable equations since it utilizes floating-point arithmetic which cannot represent real algebraic numbers. For some equations which have special structures, namely *strongly satisfiable equations*, the ICP can prove their satisfiability. More details are discussed in Chapter 10.

The ICP has been implemented in HySAT-II [28], dReal [34, 32], RSolver [65], and RealPaver [36].

3.3 Linearization

3.3.1 Linearization using CORDIC

A method of linearization was introduced in [30] using CORDIC algorithms [82] which are used for computing many elementary functions such as multiplication, division, SIN, COS,

EXP, LOG, etc [82]. The method is able to decide satisfiability of polynomial constraints given a precision requirement, which is a delta-complete problem [33]. The method is useful in verifying discrete-continuous and embedded systems with a tolerance of a given error bound. In the context of SMT solving, we can only obtain UNSAT answer from this method.

For nonlinear arithmetic, the method in [30] linearizes the multiplication $p = s \times t$ using a sequence of triples $(x_0, y_0, z_0), \ldots, (x_n, y_n, z_n)$ where $x_0 = s, y_0 = 0, z_0 = t$ and for $0 \leq k < n$, we have

$$y_{k+1} = y_k + \delta_k x_k 2^{-k}$$

$$z_{k+1} = z_k - \delta_k 2^{-k}$$

$$x_{k+1} = x_k$$

where

$$\delta_k = \begin{cases} 1 \text{ if } z_k \geq 0, \\ -1 \text{ otherwise} \end{cases}$$

Example 12 *Consider $s = 5, t = 1.125$. With $n = 8$, we have the following sequence of triples:* $(x_0 = 5, y_0 = 0, z_0 = 1.125), (x_1 = 5, y_1 = 5, z_1 = 0.125), \ldots, (x_8 = 5, y_8 = 5.6640625, z_8 = -0.0078125)$.

The resulting multiplication y_n has the error of $ex_0 2^{-(n-1)}$ [30] where e is in $[-1, 1]$, then we add the following *correction term* ec_{mult} to the result.

$$y'_n = y_n + ex_0 2^{-(n-1)} = y_n + ec_{mult}$$

We have

$$|ec_{mult}| \leq \left| x_0 2^{-(n-1)} \right|$$

Linearization is done by fixing the number n and transformation using the above sequence of triples and the resulting linear constraints can be decided efficiently [25, 31]. The number of variables, however, explodes with respect to the number of multiplication in the non-linear constraints.

The method has been implemented in CORD [30].

3.3.2 Linearization with Uninterpreted Function Symbols

Recently, a counterexample-guided abstraction refinement approach was introduced [8] for solving non-linear constraints using linearization with uninterpreted function symbols (EUF). The approach abstracts each multiplication xy as a term $fmul(x, y)$ where $fmul()$ is an uninterpreted function returning a real. For each $fmul(x, y)$ the following axioms are added.

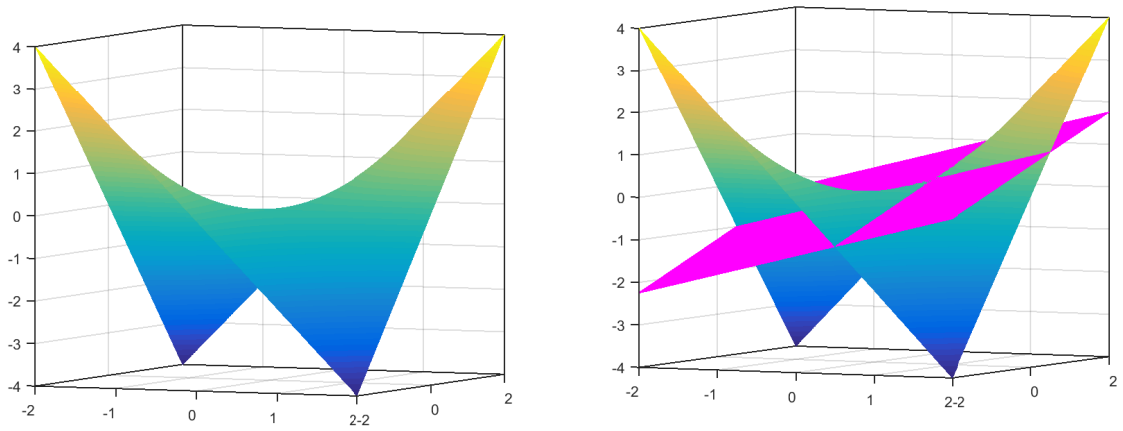
$$fmul(x, y) = fmul(y, x)$$

$$fmul(x, y) = fmul(-x, -y) \wedge fmul(x, y) = -fmul(-x, y) \wedge fmul(x, y) = -fmul(x, -y)$$

$$(x = 0 \vee y = 0 \leftrightarrow fmul(x, y) = 0)$$

$$((x > 0 \wedge y > 0) \vee (x < 0 \wedge y < 0)) \rightarrow fmul(x, y) > 0$$

$$((x > 0 \wedge y < 0) \vee (x < 0 \wedge y > 0)) \rightarrow fmul(x, y) < 0$$



(a) Graph of xy

(b) Graph of xy and the tangent plane at $(0.5, 0.5)$

Figure 3.2: Multiplication function and table plane

If SMT solving for LRA+EUF returns UNSAT, then we can conclude that the original formula is unsatisfiable. In the case of a SAT answer with a model π, we need to check if for each uninterpreted function $fmul(x, y)$, $\pi(fmul(x, y)) = \pi(x)\pi(y)$. If this is the case, then the original formula is satisfiable. Otherwise, lemmas are learned so that such a spurious model will not appear again. Consider the tangent plane $T(x, y, a, b)$ to xy at a point $(a, b) \in \mathbb{R}^2$ (the plane just touches the surface xy at (a, b)).

$$T(x, y, a, b) = bx + ay - ab$$

Figure 3.2 illustrates the surface xy and its tangent plane at the point $(0.5, 0.5)$. In the case of the previous spurious model, the following lemmas are added into the constraints to force the correct value of $fmul(x, y)$ at (a, b) and to provide bounds of $fmul(x, y)$ when x and y are not on the multiplication surface (which is useful to remove unsatisfiable regions).

$$fmul(a, y) = ay \wedge f(x, b) = xb$$

$$((x > a \wedge y < b) \vee (x < a \wedge y > b)) \rightarrow fmul(x, y) < T(x, y, a, b)$$

$$((x < a \wedge y < b) \vee (x > a \wedge y > b)) \rightarrow fmul(x, y) < T(x, y, a, b)$$

where a, b can be chosen as:

$$a := \pi(x) \text{ and } b := \pi(y)$$

$$a := \frac{1}{\pi(fmul(x, y))} \text{ and } b := \pi(y)$$

$$a := \pi(x) \text{ and } b := \frac{1}{\pi(fmul(x, y))}$$

The procedure is sound but incomplete. It has been implemented in CVC4 [9] and MathSAT [8].

3.4 Other Methodologies

3.4.1 Reduction to SAT

The method in [89] reduces the task of finding a satisfiable solution for non-linear constraints into an SAT problem by considering rational numbers $\mathbb{Q}$ as domains of variables. Each variable is represented as a pair (a, b) of an integer enumerator a and a positive integer denominator b. With a fixed number k of an initial number of bits, a and b can be represented as lists of k Boolean variables corresponding to their bits representations. Arithmetic over $\mathbb{Q}$ is transformed into arithmetic over integers $\mathbb{I}$ which is transformed into arithmetic over $\mathbb{N}$ which in turn is transformed into SAT formulas over those Boolean variables (see [89] for details). The resulting Boolean formula is solved by state-of-the-art SAT solvers. If the SAT solver returns SAT, then the original non-linear constraints are satisfiable and we can construct the model from the TRUE, FALSE values of Boolean variables. Encoding arithmetic also introduces new variables for the sums, carries, and shifting results [89].

Example 13 *Consider the bits representation* $(a_k, \ldots, a_1)$ *of* $a \in \mathbb{N}$*, and* $(b_k, \ldots, b_1)$ *of* $b \in \mathbb{N}$*. The sum of* a*, and* b *is defined as* $(c_k, s_k, \ldots, s_1)$ *where*

$$c_0 = \text{FALSE} \qquad s_i = a_i \otimes b_i \otimes c_{i-1} \qquad c_i = (a_i \wedge b_i) \vee (a_i \wedge c_{i-1}) \vee (b_i \wedge c_{i-1})$$

with $1 \geq i \geq k$ *and* $x \otimes y := \neg(x \leftrightarrow y)$*.*

Let us illustrate the addition by actual natural numbers.

Example 14 *Continue Example 13. Suppose* $k = 4$*,* $a = 3$ *and* $b = 14$ *which have representation respectively as* (FALSE, FALSE, TRUE, TRUE) *and* (TRUE, TRUE, TRUE, FALSE)*. Following the formulas in Example 13, we have the representation of* $a + b$ *as:*

$$(\text{TRUE}, \text{FALSE}, \text{FALSE}, \text{FALSE}, \text{TRUE})$$

The procedure is incomplete and unable to show unsatisfiability. It has been implemented in MiniSmt [89].

3.4.2 Subtropical Real Root Finding

Recently, *subtropical real root finding* was proposed in [76] which efficiently finds a root of equation $f = 0$ where f is an extremely large polynomial. The motivation is that some biology application results in solving $f = 0$ with f is extremely large, e.g. includes thousands of monomials in 10 variables, which is clearly beyond the classical methods in symbolic computation. The approach is finding two points at which f has different signs, then the intermediate value theorem asserts the existence of a solution for $f = 0$.

For the first task, without loss of generality, suppose we want to determine one point at which f is greater than 0. The method views the polynomial as the set of exponents vectors associated with the sign information on the coefficients. Consider the convex hull [69] of those exponent vectors (Newton polytope) of f and any exponent vector is a vertex of the convex hull, then the corresponding monomial will dominate others [76] with respect to some polynomial curve. When an exponent vector is a vertex of the Newton polytope, there exists a hyperplane that separates it and other exponent vectors. Such a hyperplane (specifically its normal vector) provides a witness of the polynomial curve.

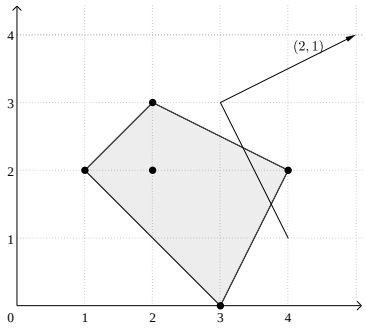


Figure 3.3: Newton polytope of $x^4y^2 - 3x^3 + x^2y^2 - 1000x^2y^3 - 9xy^2$

Example 15 *Consider* $f = 0$ *where* $f := x^4y^2 - 3x^3 + x^2y^2 - 1000x^2y^3 - 9xy^2$. *Figure 3.3 illustrates the Newton polytope of* f. *Since* $(4, 2)$ *is a vertex of the Newton polytope, the monomial* x^4y^2 *dominates* $-3x^3 + x^2y^2 - 1000x^2y^3 - 9xy^2$ *when* (x, y) *follows the curve* (a^2, a^1). *As a result, with a large enough* a, $f(a^2, a^1) > 0$.

The method employs linear programming to check if one exponent vector is the vertex of the Newton polytope. If it is, the model of the linear constraints provides the witness of the hyperplane that separates the vertex with other points.

After determining two points p, q at which f has different signs, the intermediate value theorem implies that there exists one point z on the line connecting p and q such that $f(z) = 0$. Such a z is the solution of the following system of equations

$$t = p + y(q - p) \wedge f(t) = 0$$

where y is a new variable such that $0 < y < 1$. Plugging $t = p + y(q - p)$ into $f(t)$ yields an equation over one variable y. Root isolation algorithms can be applied here to find a root of f.

The method has been implemented in Redlog/Reduce [76].

Chapter 4

Interval Constraint Propagation

4.1 Intervals and Interval Arithmetic

Let $\mathbb{R}$ be the set of real numbers and $\mathbb{R}^\infty = \mathbb{R} \cup \{-\infty, \infty\}$. The standard arithmetic operations on $\mathbb{R}$ are extended to those on $\mathbb{R}^\infty$ as presented in [57].

Definition 13 *The set of all* intervals $\mathbb{I}$ *is defined as* $\mathbb{I} = \{[l, h] \mid l \leq h \in \mathbb{R}^\infty\}$.

Definition 14 *A* box B *for a sequence of variables* $(x_1, \cdots, x_n)$ *is a Cartesian product of intervals, i.e.,* $B = I_1 \times \cdots \times I_n$, *where* $I_1, \cdots, I_n \in \mathbb{I}$ *are intervals of* $x_1, \cdots, x_n$ *respectively. We write* $B(x_j)$ *to denote* I_j *for* $j = 1, \cdots, n$.

Since most real numbers are not computable, representing them in a computer system is impossible. Numerical frameworks such as ICP generally approximate the set $\mathbb{R}$ by the set of floating-point numbers. In order to deal with round-off errors in floating-point arithmetic, ICP uses the notions of round-down and round-up for real numbers.

Definition 15 *Let* $r \in \mathbb{R}$. *Its* round-down *value, denoted as* $\underline{r}$, *is the largest floating-point number* f *such that* $f \leq r$. *Its* round-up *value, denoted as* $\overline{r}$, *is the smallest floating-point number* f *such that* $f \geq r$.

Definition 16 *An interval function* $G : \mathbb{I}^n \mapsto \mathbb{I}$ *is an* interval extension *of a function* $g : \mathbb{R}^n \mapsto \mathbb{R}$ *if and only if, for every box* $B \in \mathbb{I}^n$, *the inclusion* $\{g(x) \mid x \in B\} \subseteq G(B)$ *holds.*

An interval arithmetic *is a function mapping each polynomial function to one interval extension.*

There are many kinds of interval arithmetic based on symbolic transformations such as Affine Intervals (AIs) [14, 46], or numerical relaxations such as the Classical Interval (CI) [57].

Example 16 *CI maps* $g = x^2 + xy - 4$ *to the interval extension* $G(I_x, I_y) = I_x * I_x + I_x * I_y - [4, 4]$, *where* I_x *and* I_y *are intervals of* x *and* y *respectively, and interval operations are defined as:*

$$
\begin{aligned}
[l_1, h_1] + [l_2, h_2] &= [\underline{l_1 + l_2}, \overline{h_1 + h_2}] \\
[l_1, h_1] * [l_2, h_2] &= [\min(\underline{l_1 l_2}, \underline{l_1 h_2}, \underline{h_1 l_2}, \underline{h_1 h_2}), \max(\overline{l_1 l_2}, \overline{l_1 h_2}, \overline{h_1 l_2}, \overline{h_1 h_2})] \\
[l_1, h_1] - [l_2, h_2] &= [l_1, h_1] + [-h_2, -l_2].
\end{aligned}
$$

As a result, if $I_x = [-3, 1], I_y = [-4, -2]$*, we have*

$$\begin{aligned} G(I_x \times I_y) &= [-3, 1] * [-3, 1] + [-3, 1] * [-4, -2] - [4, 4] \\ &= [-11, 17]. \end{aligned}$$

The result $G(B)$ *can be sharpened with a consideration of power functions. More specifically,* $I_x * I_x$ *or* I_x^2 *can be better approximated as* $[0, 9]$*, which improves the value of* $G(B)$ *to* $[-8, 17]$*.*

For polynomial functions $f, g, \ldots$ we use $F_{\text{IA}}, G_{\text{IA}}, \ldots$ respectively to denote the interval extensions of these functions, which are mapped by a specific interval arithmetic "IA". When "IA" is clear from the context, we simply ignore the subscript and denote $F, G, \ldots$ respectively. We call $G(B)$ the *estimated range* of g over the box B by using "IA".

Definition 17 *Given an interval arithmetic and a box* B*, an APC* $g \diamond 0$ *is*

- IA-VALID *in* B *if and only if for all* $r \in G(B)$*,* $r \diamond 0$ *holds,*
- IA-UNSAT *in* B *if and only if for all* $r \in G(B)$*,* $r \diamond 0$ *does not hold, and*
- IA-SAT*, otherwise.*

Example 17 *(Fig. 4.1) Suppose an APC* $g > 0$ *and a box* B*.*

- *If* $G(B) = [-5, -2]$*, then* $g > 0$ *is* IA-UNSAT*.*
- *If* $G(B) = [-2, 5]$*, then* $g > 0$ *is* IA-SAT*.*
- *If* $G(B) = [2, 5]$*, then* $g > 0$ *is* IA-VALID*.*

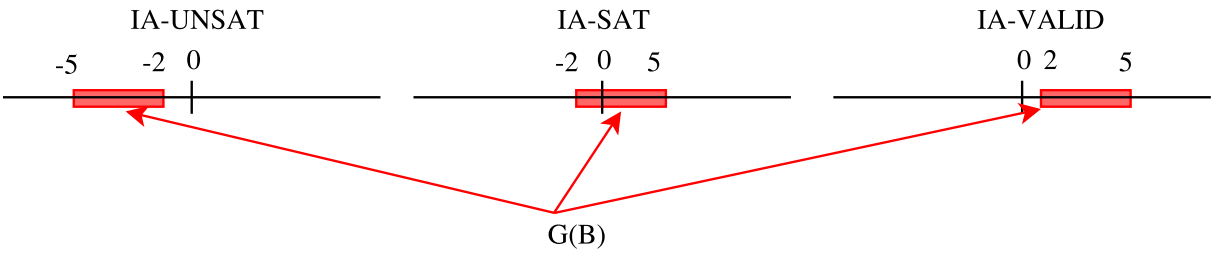


Figure 4.1: Example 17: IA-UNSAT, IA-SAT, and IA-VALID examples

Except for the IA-SAT case where we cannot conclude anything about satisfiability of the APC, we have the following theorem for relating IA-VALID and IA-UNSAT to satisfiability.

Theorem 2 *Suppose we have an interval arithmetic, a box* B*, and an APC* $g \diamond 0$*.*

- *If the APC is* IA-VALID*, then it is* SAT *at any point inside* B*.*
- *If the APC is* IA-UNSAT*, then it is* UNSAT *inside* B*.*

Proof 1

- *If the APC is* IA-VALID*, then for all $r \in G(B)$, we have $r \diamond 0$. In addition, by definition of interval arithmetic, we have for all $x \in B$, $g(x) \in G(B)$. As a result, for all $x \in B$, $g(x) \diamond 0$. In other words, the APC is satisfiable at any points inside B.*
- *If the APC is* IA-UNSAT*, then for all $r \in G(B)$, we have $\neg(r \diamond 0)$. In addition, by definition of interval arithmetic, we have for all $x \in B$, $g(x) \in G(B)$. As a result, for all $x \in B$, $\neg(g(x) \diamond 0)$. In other words, the APC is unsatisfiable inside B.*

Definition 18 *Given an interval arithmetic and a box B, an conjunction φ of APCs is*

- IA-VALID *in B, if and only if, each APC $g \diamond 0 \in \varphi$ is* IA-VALID*,*
- IA-UNSAT *in B, if and only if, some APC $g \diamond 0 \in \varphi$ is* IA-UNSAT*, and*
- IA-SAT*, otherwise.*

The following lemma follows directly from Definition 18 and Theorem 2.

Lemma 1 *Suppose we have an interval arithmetic, a box B, and a conjunction φ of APCs.*

- *If φ is* IA-VALID*, then it is* SAT *at any point inside B.*
- *If φ is* IA-UNSAT*, then it is* UNSAT *inside B*

4.2 Constraint Propagation

Constraint propagation consists of two steps: interval contraction and interval propagation.

Definition 19 *Given a relation $\diamond$ and an interval I, the* interval contraction *is the function* $\text{IC}(\diamond, I)$*, which returns the smallest interval I' such that*

$$\{r \in I \mid r \diamond 0\} \subseteq I'$$

Example 18 $\text{IC}(>, [-2, 5]) = [0, 5]$*,* $\text{IC}(>, [2, 5]) = [2, 5]$*, and* $\text{IC}(>, [-2, -5]) = \emptyset$*.*

Definition 20 *Given an APC $g \diamond 0$, a box B, and an interval $I \subseteq G(B)$,* interval propagation *is the function* $\text{IP}(g \diamond 0, I, B)$*, which returns a box $B' \subseteq B$ such that:*

- $G(B') \supseteq I$*, and*
- $B' \cap S = B \cap S$*,*

where $S = \{x \mid x \in B \text{ and } g(x) \diamond 0\}$ is the set of solutions of $g \diamond 0$ in B.

In practice, the operations of interval arithmetic, interval contraction, and interval propagation are executed along the syntax tree of a polynomial as stipulated in Example 19.

Example 19 *Consider the APC* $x^2 + xy - 4 < 0$ *and the box* $B = [-3, 1] \times [-4, -2]$ *for the variable sequence* (x, y) *in Example 16. Fig. 4.2 illustrates operations at lines* 18, 19, *and* 20 *in Algorithm 1. Constraint propagation promulgates interval contraction in a top-down manner. For example, consider three nodes* n_1, n_2, *and* n_3 *s.t.* $n_1 = n_2 + n_3$. *Interval arithmetic yields* $I_{n_1} = I_{n_2} + I_{n_3}$. *If constraint propagation reduces the interval of* n_1 *to* $I'_{n_1} \subset I_{n_1}$, *new intervals* I'_{n_2} *of* n_2 *and* I'_{n_3} *of* n_3 *can be inferred by the following operations.*

$$I'_{n_2} = I_{n_2} \cap (I'_{n_1} - I_{n_3}) \qquad I'_{n_3} = I_{n_3} \cap (I'_{n_1} - I_{n_2})$$

The right-hand-side tree in Fig. 4.2 indicates that the range of $n_1 = x^2 + xy$ *is contracted to* $[-4, 4]$. *Since the original ranges of* $n_2 = x^2$ *and* $n_3 = xy$ *are* $I_{n_2} = [0, 9]$ *and* $I_{n_3} = [-4, 12]$ *respectively, the range of* $n_3 = xy$ *is contracted to:*

$$I'_{n_3} = I_{n_3} \cap (I'_{n_1} - I_{n_2}) = [-4, 12] \cap ([-4, 4] - [0, 9]) = [-4, 12] \cap [-13, 4] = [-4, 4]$$

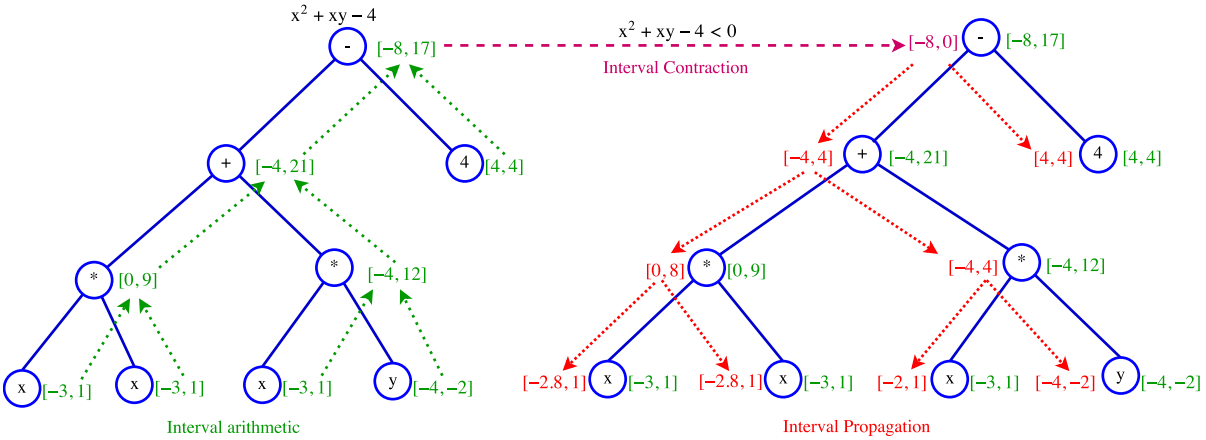


Figure 4.2: Example 19: interval arithmetic, interval contraction, and interval propagation.

4.3 ICP as a Theory Solver of an SMT Solver

Since ICP is based on interval arithmetic, which is an over-approximation, it can be applied to determine SAT/UNSAT of inequalities and UNSAT of equations. However, ICP generally does not produce interesting results on satisfiable equations. This section explains ICP as a theory solver (for inequalities) inside an SMT framework in the spirit of the very lazy approach [60]. An extension of ICP to handle constraints with the presence of equations is introduced in Chapter 7.

Fig. 4.3 outlines the SMT framework that utilizes ICP as a theory solver. Given a conjunctive normal form (CNF) formula, the SAT solver enumerates the propositional models of the CNF. Each such model is a conjunction of APCs and is delivered to the theory solver ICP. If ICP detects SAT, then the CNF is SAT. If ICP detects UNSAT, then conflict clause learning ("learn clause") is implemented in which the negation of the conjunction is passed to the SAT solver so that it will not be repeatedly selected in the future. In addition, theory solvers identify a subset of the conjunction that is

already UNSAT, whose negation will be added to the SAT solver as an optimized learned clause. If the SAT solver cannot find any further propositional model, it concludes the unsatisfiability of the CNF.

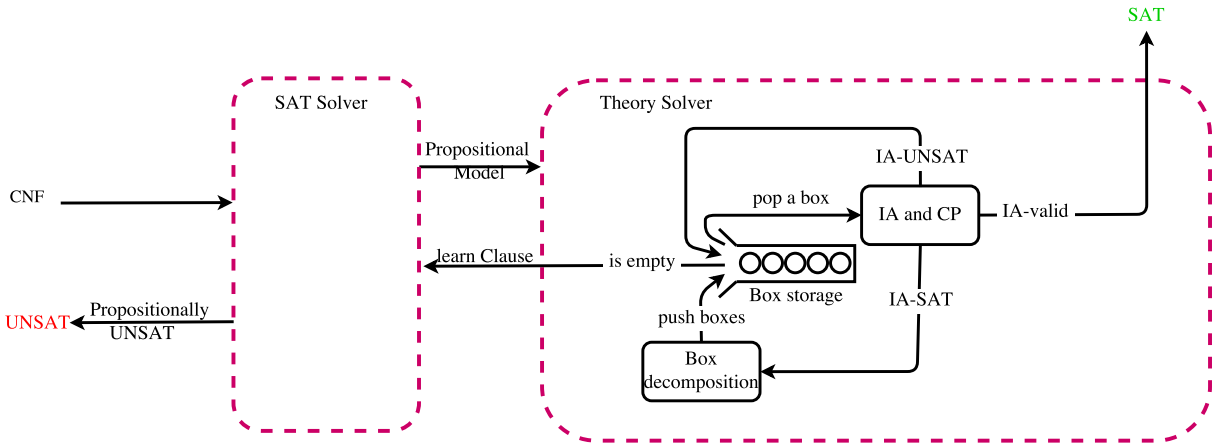


Figure 4.3: ICP-based SMT solver

The "Box storage" is a set which contains all the boxes generated by ICP. Starting with the "Box storage" that consists of a single box B ($[-\infty, \infty]^n$ by default), the ICP [3] theory solver tries to detect the satisfiability of the conjunction φ of APCs in B, by iteratively doing the following operations until successfully proving/disproving φ.

- Pop a box called the *current box* from the "Box storage".
- Evaluate φ and contract the current box with interval arithmetic and constraint propagation ("IA and CP"). The "IA and CP" module judges φ as either IA-VALID, IA-UNSAT, or IA-SAT.
  - If φ is IA-VALID, ICP outputs SAT.
  - If φ is IA-UNSAT, the control comes back to "Box storage" to either pop a new box if the storage is non-empty, or to implement clause learning otherwise.
  - If φ is IA-SAT, the control goes to "Box decomposition".
- Decompose the contracted current box into smaller ones ("Box decomposition"). New boxes created by box decomposition are pushed into "Box storage".

Algorithm 1 clarifies the module "IA and CP" in Fig. 4.3, which assumes a global variable *unsat_cores* to keep a set of sub-formulas of φ. The algorithm maintains an invariant that whenever φ is UNSAT, so is *unsat_cores*. In addition, since unsatisfiability of *unsat_cores* implies that of φ, it is advantageous to add $\neg$*unsat_cores* instead of $\neg\varphi$ to the SAT solver during conflict clause learning.

In addition to IA-VALID, IA-UNSAT, and IA-SAT, the algorithm also returns the box contracted from the original one by the constraints.

Since *interval arithmetic* (Definition 16) is an over-approximation and *constraint propagation* (Definition 20) preserves solutions, we obtain the soundness of IA_CP(_, _).

Theorem 3 *Suppose we have a conjunction of APCs φ and a box B.*

Algorithm 1 Interval Arithmetic and Constraint Propagation for proving φ in a box B

1: **function** IA_CP(φ, B)
2: $is\_valid \leftarrow$ TRUE
3: **for** $g \diamond 0$ in φ **do**
4: Compute $G(B)$ ▷ Compute range of g
5: **if** $g \diamond 0$ is IA-UNSAT inside B **then** ▷ Use Def. 17
6: $unsat\_cores \leftarrow unsat\_cores \cup g \diamond 0$
7: **return** IA-UNSAT, $\emptyset$
8: **end if**
9: **if** $g \diamond 0$ is IA-SAT inside B **then** ▷ Use Def. 17
10: $is\_valid \leftarrow False$
11: **end if**
12: **end for**
13: **if** $is\_valid =$ TRUE **then**
14: **return** IA-VALID, B
15: **end if**
16: $B_{init} \leftarrow B$ ▷ Store the initial value of B
17: **for** $g \diamond 0$ in φ **do**
18: $I \leftarrow G(B)$ ▷ Interval arithmetic
19: $I' \leftarrow \text{IC}(\diamond, I)$ ▷ Interval contraction
20: $B' \leftarrow \text{IP}(g \diamond 0, I', B)$ ▷ Interval propagation
21: **if** $B' \neq B$ **then** ▷ Contractions occurred
22: $unsat\_cores \leftarrow unsat\_cores \cup g \diamond 0$
23: **end if**
24: $B \leftarrow B'$
25: **end for**
26: **if** $B = B_{init}$ **then** ▷ No contractions occurred
27: **return** IA-SAT, B
28: **else**
29: IA_CP(φ, B) ▷ Continue the function until no contractions occurred
30: **end if**
31: **end function**

- *If* IA_CP(φ, B) *returns* IA-UNSAT, φ *is* UNSAT *inside* B.
- *If* IA_CP(φ, B) *returns* IA-VALID, φ *is* SAT *at any point inside* B.

Proof 2 *At line* 20 *of Algorithm 1, Definition 20 guarantees that* B' *contains all and only solutions of* $g \diamond 0$ *in* B. *As a result, after the loop at line* 17 *finishes, the set of solutions of* φ *in* B_{init} *is preserved in* B. *In other words, before each recursive call at line* 29, *the set of solutions is preserved, thus the soundness of the algorithm is guaranteed. Lemma 1 concludes the statements of the theorem.* *(Q.E.D.)*

In "Box decomposition", the contracted current box is decomposed into smaller ones which satisfy some properties described in Definition 21 to preserve the soundness and the progress of the ICP framework.

Definition 21 *Given a box B,* box decomposition *produces two boxes, B_1 and B_2, such that $B_1 \cup B_2 = B$, $B_1 \setminus B_2 \neq \emptyset$, and $B_2 \setminus B_1 \neq \emptyset$.*

We are ready to give the pseudo-code algorithm for ICP (Algorithm 2) for solving a conjunction φ.

Algorithm 2 ICP(f)or a set of polynomial constraints φ

```
1: function ICP(φ)
2:     S ← {]−∞, ∞[^n}
3:     while S ≠ ∅ do
4:         choose B ∈ S
5:         S ← S \ {B}
6:         IA-RESULT, B' ← IA_CP(φ, B)
7:         if IA-RESULT = IA-UNSAT then
8:             continue
9:         else if IA-RESULT = IA-VALID then
10:            return SAT
11:        end if
12:        B1, B2 ← decompose B'
13:        S ← S ∪ {B1, B2}
14:    end while
15:    return UNSAT
16: end function
```

Limitations of ICP in solving inequalities

Assume a conjunction of strict inequalities $\varphi = \bigwedge_i g_i > 0$ and a box B. The conjunction is either SAT or UNSAT in B.

In the first case where φ is SAT in B, there is a non-empty box $B' \subseteq B$ in which φ is IA-VALID (Fig. 4.4a). If intervals $I_1, \cdots, I_n$ in the box B are bounded and the ICP solver follows a breath-first-search manner (e.g. the "Box storage" is implemented as a queue), the ICP solver will eventually produce a box in which φ is IA-VALID. As a result, ICP can prove satisfiability of φ in B.

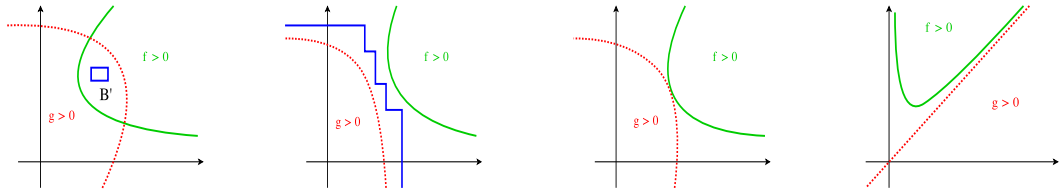
In the second case where φ is UNSAT in B, we consider three more possible cases: non-touching case, touching case, and convergence case. The ICP can prove the unsatisfiability of φ for the first case, but cannot do so for the last two cases.

If it is not a touching case (Fig. 4.4b) and $I_1, \cdots, I_n$ are bounded, there exists one finite set of small boxes whose union is B and inside each of them φ is IA-UNSAT. As a result, by box decomposition, the unsatisfiability of φ is consistently proved by ICP. In this case, Algorithm 1 might not terminate because of infinite contractions (the condition at line 26 might infinitely not be satisfied).

If it is a touching case (Fig. 4.4c), in order to prove unsatisfiability, a necessary condition is that ICP needs to compute the touching point to decompose boxes. This is generally not achievable because ICP uses floating-point numbers which are not enough to represent algebraic numbers.

If it is a convergence case (Fig. 4.4d), suppose that B is unbounded because otherwise it becomes a non-touching case and ICP will eventually prove the unsatisfiability of φ

in B. In this case, ICP will not terminate for the following two reasons. First, interval arithmetic and constraint propagation cannot conclude unsatisfiability of the φ within B because essentially there is not any box $B' \subseteq B$, which is contracted from B by "IA and CP", such that B' can separate satisfiable areas of APCs. Second, given an unbounded box B_1 in which the varieties of polynomials are convergent, the box decomposition will eventually create a new unbounded box $B_2 \subseteq B_1$ in which the varieties of polynomials are still convergent. As a result, there always exists an unbounded box in which the convergence occurs and thus ICP will not terminate.



(a) SAT detection (b) UNSAT detection (c) Touching case (d) Convergence case

Figure 4.4: Solvable and unsolvable cases of polynomial inequalities with ICP

Chapter 5

raSAT Loop for Inequalities

raSAT (refinement of approximations for satisfiability) loop [46, 85] was intended to accelerate ICP for SAT detection by *testing*. The framework is illustrated in Fig. 5.1. When interval arithmetic and constraint propagation cannot conclude satisfiability of the conjunction (i.e. IA-SAT), testing is implemented to find a satisfiable instance for the conjunction. While the first section of this chapter introduces testing in details, the second and third sections provide various kinds of interval arithmetic and heuristics in the framework, and the last section presents a modification of the framework for handling constraints over integers.

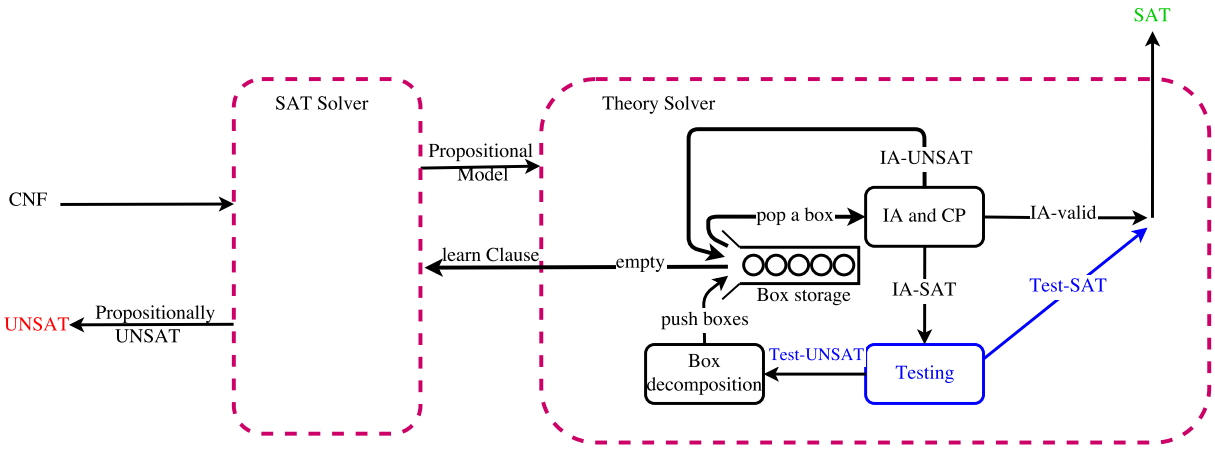


Figure 5.1: raSAT loop

5.1 Early Satisfiability Detection with Testing

For a conjunction φ of APCs and a box B, testing tries to find an assignment from points in B to variables, which satisfies φ. Let us start with definitions related to *test cases* and *test values*.

Definition 22 *A* test case *on a set V of variables is regarded as a function $\theta : V \mapsto \mathbb{R}$. We call $\theta(x)$ the* test value *of x. The domain of θ is denoted by* $\text{DOM}(\theta)$. *The test case is in a box B if $\theta(x) \in B(x)$ for all $x \in V$.*

Next, we give the definition for *(un)satisfying instances* of constraints.

Definition 23 *A test case θ on $var(\varphi)$ is a* satisfying (resp. an unsatisfying) instance *of φ if $\varphi^\theta = \text{TRUE}$ (resp. $\varphi^\theta = \text{FALSE}$).*

Example 20 *Let $\varphi = x_3x_4 - x_4 > 0$ and $B = [0, 1.5] \times [-5, -0.5]$ for (x_3, x_4). Testing may produce a test case θ in B with $\theta(x_3) = 1.2$ and $\theta(x_4) = -4.3$. Since $\varphi^\theta = \text{FALSE}$, θ is an unsatisfying instance of φ.*

Given a set of test cases, we define the notions of *Test*-SAT and *Test*-UNSAT for polynomial constraints.

Definition 24 *Let TC be a set of test cases on V such that $var(\varphi) \subseteq V$. TC concludes that φ is* Test-SAT *if there exists a* SAT *instance of φ in TC; otherwise,* Test-UNSAT.

We are ready to describe the box "Testing" in Fig. 5.1. It is formalized by the function TESTING(φ, B, θ) which is specified by Algorithm 3. Given a conjunction φ and a box B, the algorithm calls TESTING(φ, B, $\emptyset$) (i.e. θ is initialized as $\emptyset$) and incrementally constructs new test cases (when testing a new APC) in the box B from θ by extending the domain of θ with new variables, and checks each new test case to find a SAT instance of φ. The purpose of incremental construction is that if some test case do not satisfy some APCs, it is discarded and not extended further.

Algorithm 3 Testing φ in a box $B = [l_1, h_1] \times \cdots [l_n, h_n]$ for the variables $(x_1, \cdots, x_n)$

```
1: TESTING(φ, B, ∅)
2: function TESTING(φ, B, θ)
3:     if φ = ∅ then                          ▷ All inequalities are satisfied
4:         return Test-SAT
5:     end if
6:     (g_i > 0, φ') ← POP_APC(φ)             ▷ Pop an APC g_i > 0 from φ
7:     for θ' ∈ GENERATE_TEST_CASES(g_i > 0, B, θ) do
8:         if SATISFY(g_i > 0, θ') and TESTING(φ', B, θ')=Test-SAT then
9:             return Test-SAT
10:        end if
11:    end for
12:    return Test-UNSAT
13: end function
```

The function POP_APC(φ) returns an APC $g_i > 0$ in φ and update the current conjunction as $\varphi' = \varphi \setminus \{g_i > 0\}$. The function GENERATE_TEST_CASES($g_i > 0, B, \theta$) constructs new test cases by extending θ with newly generated test values of variables in $var(g_i) \setminus \text{DOM}(\theta)$.

Example 21 *Let $g_i = x_3x_4 - x_4$, $B = [-2, 3.5] \times [-5, 0] \times [0, 1.5] \times [-5, -0.5]$ for variables (x_1, x_2, x_3, x_4), and $\theta = \{(x_2, -3.9), (x_4, -2.5)\}$. The steps of* GENERATE_TEST_CASES*$(g_i > 0, B, \theta)$ are as follows.*

- *Generating test values (of the specified number) for variables in $var(g_i) \setminus \text{DOM}(\theta) = \{x_3\}$ in B. Assume such test values for x_3 are $\{1.1, 0.3\}$*

- *Extending θ with those new test values and returning the set of extended test cases. In this example, the returned set of test cases is $\{\theta_1, \theta_2\}$ where*

$$\theta_1 = \{(x_2, -3.9), (\mathbf{x_3}, \mathbf{1.1}), (x_4, -2.5)\}$$

$$\theta_2 = \{(x_2, -3.9), (\mathbf{x_3}, \mathbf{0.3}), (x_4, -2.5)\}$$

There are various choices for generating test values. Current **raSAT** applies random testing [46, 85], i.e., randomly selecting values from intervals that are assigned to variables. Section 5.3 proposes a heuristic based on affine intervals to guide the generation of test values.

The *subtropical method* was recently proposed [76], which heuristically constructs a satisfiable instance of an APC $g > 0$. The idea is to find in g a dominant monomial with a positive coefficient and construct an assignment for variables so that such a positive monomial dominates others and make g greater than 0. This might offer a promising alternative to random testing.

Example 22 *Continuing Example 21, θ_1 is an* UNSAT *instance of $g_i > 0$. As a consequence, θ_1 can be discarded to avoid useless extension on remaining variables, i.e., $\{x_1\}$.*

The number of generated test cases affects the performance of the framework in terms of the following sense. If the constraint is satisfiable, an insufficient number of test cases may cause raSAT miss to detect satisfiability. If the constraint is unsatisfiable, on the other hand, a large number of test cases will cause overhead because they are not able to satisfy the constraint.

raSAT restricts the number of test cases to at most 2^N (N is a constant) by selecting N most likely influential variables to have 2 test values and leaving a single test value for the rest. The heuristic to select such N variables is presented in Section 5.3. The value of N is chosen from the candidates $\{1, 5, 10, 20\}$ by experiments on SMT-LIB benchmarks. We found that 10 performed better than the others.

Box decomposition in **raSAT** is fundamentally the bisection at the half-way point on a selected dimension of the box B.

Example 23 *Given a box $B = [-2, 3.5] \times [-5, 0] \times [0, 1.5] \times [-5, -0.5]$ for variables (x_1, x_2, x_3, x_4). First, **raSAT** selects a variable, e.g., x_3. Then, its assigned interval, e.g., $[0, 1.5]$, is bisected at the half-way point. As a consequence, the resulting two boxes are $B_1 = [-2, 3.5] \times [-5, 0] \times [0, \mathbf{0.75}] \times [-5, -0.5]$ and $B_2 = [-2, 3.5] \times [-5, 0] \times [\mathbf{0.75}, 1.5] \times [-5, -0.5]$.*

An interval may be open-ended. The precise operation for the bisection in **raSAT** is as follows.

$$[-\infty, \infty] \xrightarrow{Decomposition} [-\infty, 0], [0, \infty]$$

$$[-\infty, h] \xrightarrow{Decomposition} [-\infty, h-8], [h-8, h]$$

$$[l, \infty] \xrightarrow{Decomposition} [l, l+8], [l+8, \infty]$$

$$[l, h] \xrightarrow{Decomposition} [l, \frac{l+h}{2}], [\frac{l+h}{2}, h]$$

where $l \neq -\infty$ and $h \neq \infty$. The number 8 here comes from our intention to create finite small boxes for the framework to quickly detect satisfiable instances. We made various experiments with different numbers such as 1, 8, 64, 128, and 1024; and we saw that 8 is the most likely effective value.

Section 5.3 introduces a heuristic to select the most likely influential variable whose interval is going to be decomposed.

Before presenting heuristics used in the raSAT framework, we show the pseudo-code of raSAT loop in Algorithm 4

Algorithm 4 raSAT loop for a set of polynomial constraints φ

```
1: function RASATLOOP(φ)
2:     S ← {]−∞, ∞[^n}
3:     while S ≠ ∅ do
4:         choose B ∈ S
5:         S ← S \ {B}
6:         IA-RESULT, B' ← IA_CP(φ, B)
7:         if IA-RESULT = IA-UNSAT then
8:             continue
9:         else if IA-RESULT = IA-VALID then
10:            return SAT
11:        end if
12:        φ_IA-SAT ← {φ_APC | φ_APC ∈ φ and φ_APC is IA-SAT in B'}      ▷ exclude IA-VALID APCs
13:        TEST-RESULT ← TESTING(φ_IA-SAT, B', ∅)
14:        if TEST-RESULT =Test-SAT then
15:            return SAT
16:        end if
17:        B_1, B_2 ← decompose B'
18:        S ← S ∪ {B_1, B_2}
19:    end while
20:    return UNSAT
21: end function
```

In addition to testing, **raSAT** further prepares various kinds of interval arithmetic which aims at a refined result of interval arithmetic.

5.2 Various Interval Arithmetic

A popular IA is CI [57] used in Example 16. The main weakness of CI is the loss of dependency among values. For instance, if $x \in [2, 4]$ then, $x - x$ is evaluated to $[-2, 2]$ instead of $[0, 0]$.

Affine intervals (AIs) [14, 54, 46] introduce *noise symbols* ϵ, which are interpreted as intervals $[-1, 1]$. Thus, an AI correctly cancels the subtraction among the same variable, e.g., $x \in [2, 4]$ above is represented as $3 + \epsilon$ and $x - x$ becomes 0. Consider an example with multiplications $x^2 - x \times y$ with $x \in [2, 4]$ and $y \in [0, 2]$. Then, an AI respectively represents the intervals of x and y as $[3, 3] + \epsilon_1$ and $1 + \epsilon_2$. The interpretation of $x^2 - x \times y$ is:

$$
\begin{aligned}
&([3,3]+\epsilon_1)^2 - ([3,3]+\epsilon_1) \times ([1,1]+\epsilon_2) \\
&= [9,9] + [6,6]\epsilon_1 + \epsilon_1^2 - ([3,3] + [3,3]\epsilon_2 + \epsilon_1 + \epsilon_1\epsilon_2) \\
&= [6,6] + [5,5]\epsilon_1 - [3,3]\epsilon_2 + \epsilon_1^2 - \epsilon_1\epsilon_2.
\end{aligned}
$$

In order to preserve the soundness of affine intervals w.r.t. Definition 16 (as an over-approximation), constants in Affine forms are represented by intervals, and CI is used during arithmetic of affine intervals.

Types of affine intervals vary due to choices of approximations on the multiplications ϵ_i^2 and $\epsilon_i\epsilon_j$.

1. AA [14, 73] replaces $\epsilon_i\epsilon_j$ with a fresh noise symbol.
2. AF1 and AF2 [54] prepare a fixed noise symbol for any $\epsilon_i\epsilon_j$.
3. EAI [59] replaces $\epsilon_i\epsilon_j$ with $[-1,1]\epsilon_i$ or $[-1,1]\epsilon_j$.
4. AF2 [54] replaces ϵ_i^2 with the fixed noise symbols ϵ_+ or ϵ_-.
5. The Chebyshev Approximation Interval (CAI) was proposed in [46], which uses linear Chebyshev approximations (Fig. 5.2) on ϵ_i^2 and $\epsilon_i|\epsilon_i|$.

The main drawback of AIs is that the estimates of the multiplications between different variables may be less precise than those of CI.

Example 24 *Consider xy for $x \in [2,4]$ and $y \in [0,2]$. Then, CI evaluates it as $[0,8]$, where AA computes*

$$([3,3]+\epsilon_1) \times ([1,1]+\epsilon_2) = [3,3] + [3,3]\epsilon_2 + \epsilon_1 + \epsilon_1\epsilon_2$$

which is evaluated to $[3,3] + [-3,3] + [-1,1] + [-1,1][-1,1] = [-2,8]$.

However, as mentioned previously, AIs partially preserve the dependency among values, which is lost in CI. The example below shows the value dependency.

Example 25 *Let $g(x_1,x_2) = x_1^3 - 2x_1x_2$ for $x_1 = [0,2] = 1 + \epsilon_1$ and $x_2 = [1,3] = 2 + \epsilon_2$.*

- *While CI [57] estimates $g(x_1,x_2)$ as $[-12,8]$,*
- *AF_2 does as $-[3,3] - \epsilon_1 - [2,2]\epsilon_2 + [3,3]\epsilon_+ + [3,3]\epsilon_\pm$ which is evaluated to $[-9,6]$.*

The Chebyshev approximations in CAI [46] on ϵ^2 and $\epsilon|\epsilon|$ (Fig. 5.3) are given by:

$$|x| - \frac{1}{4} \leq x^2 = |x|^2 \leq |x| \quad \text{and} \quad x - \frac{1}{4} \leq x|x| \leq x + \frac{1}{4}$$

which leads to:

$$\epsilon\epsilon = |\epsilon||\epsilon| = |\epsilon| + [-\frac{1}{4}, 0] \quad \text{and} \quad \epsilon|\epsilon| = \epsilon + [-\frac{1}{4}, \frac{1}{4}].$$

Detailed definitions are given in Appendix A.

Example 26 *Let $f = (x^2 - 2y^2 + 7)^2 + (3x + y - 5)^2$, $x \in [-1,1]$, and $y \in [-2,0]$. The bounds of f computed by AF1, AF2, and CAI are $[-98,220]$, $[-53,191]$, and $[-4.6875, 163.25]$ respectively.*

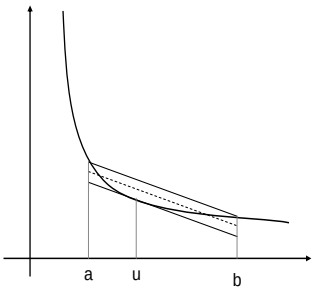


Figure 5.2: Chebyshev approximation

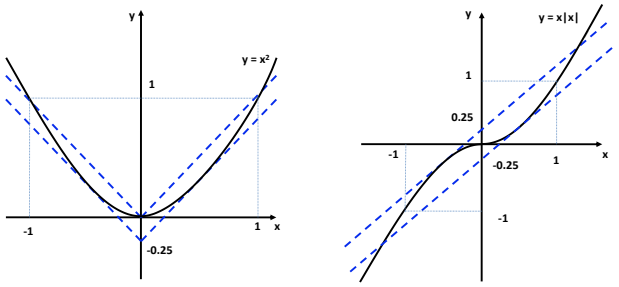


Figure 5.3: Chebyshev approximation of x^2 and $x\ |x|$

Example 27 *Let* $sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!}$ *(an initial segment of Taylor expansion) and* $x \in [0, 0.523598]$. *The bounds of* $sin(x)$ *computed by AF1, AF2, and CAI are* $10^{-6}[-6290.49099241,\ 523927.832027]$, $10^{-6}[-6188.00580507,\ 514955.797111]$, *and* $10^{-6}[-1591.61467700,\ 503782.471931]$ *respectively.*

The authors in [26] utilized a symbol δ, which is similar to ϵ in AIs, to handle strict linear inequalities over rationals. The domain of rationals $\mathbb{Q}$ is extended to $\mathbb{Q}_\delta = \{c + k\delta \mid c, k \in \mathbb{Q}\}$. In addition, the operations and comparison over $\mathbb{Q}_\delta$ are defined with the presence of the symbol δ [26]. While the symbolic transformations involving δ in [26] and ϵ in AIs share similarities, they are essentially different in their purposes. AIs introduce ϵ symbols to present bounded intervals and maintain the relationship between related intervals. However, δ in [26] is used to convert strict inequalities and strict bounds into corresponding non-strict ones with the following observation. A conjunction of strict inequalities $p_1 > 0 \wedge \cdots \wedge p_n > 0$ is satisfiable if and only if there exists a positive rational number δ such that $p_1 \geq \delta \wedge \cdots \wedge p_n \geq \delta$ [26, Lemma 1].

Although **raSAT** provides implementations of CI [57], AF1, AF2, EAI, and CAI [14, 54, 46], AF2 and CI are selected by using an empirical trade-off between efficiency and precision. Given an APC $g \diamond 0$, **raSAT** uses CI first. If it detects that the APC is either IA-VALID or IA-UNSAT, the interval arithmetic for the APC is complete. Otherwise, i.e., IA-SAT is returned by CI, AF2 is applied to prove or disprove the APC. The result of CI is further re-used to refine the range of g in the box B as $G(B) = G_{\text{CI}}(B) \cap G_{\text{AF2}}(B)$.

The computation of interval arithmetic is generally more expensive than that of floating-point arithmetic because the upper and lower bounds of the intervals must be rounded up and down respectively. In addition, AIs involve symbolic computations which are expensive. Since the procedure of decomposing a box B into B_1 and B_2 makes B different from new boxes in only one of its dimensions, most heavy computations in IA are repeated when reestimating the range of a polynomial g in one of the new boxes. **raSAT** avoids such repetitions by storing IA results at each node of the syntax tree of g and re-computing them only when the IA results at one of the direct child nodes was changed.

5.3 Various Heuristics to boost SAT Detection

This section proposes two heuristic measures to improve the performance of the raSAT loop framework, namely *SAT-likelihood* and *variables sensitivity*.

Suppose AI estimates the range of a polynomial g in a box B as $[c_1, d_1]\epsilon_1 + \cdots + [c_n, d_n]\epsilon_n$, which is evaluated by instantiating $[-1, 1]$ to ϵ_i for $i = 1, \ldots, n$.

Definition 25 SAT-likelihood *of an APC $g \diamond 0$ in a box B is:*

$$\frac{|\text{IC}(g \diamond 0, G(B))|}{|G(B)|}.$$

For a conjunction φ of APCs, the SAT-likelihood of the box B *is the least* SAT-likelihood *in B among APCs.*

Definition 26 *The* sensitivity *of a variable x_i in $g > 0$ is $max(|c_i|, |d_i|)$.*

Example 28 *Continue the Example 25, the SAT-likelihood of $g(x_1, x_2) > 0$ is $\frac{6}{9-(-6)} = 0.4$ by AF_2, and the sensitivities of x_1 and x_2 are 1 and 2 by AF_2, respectively.*

When popping a box from the box storage, **raSAT** pops the box with the largest value of *SAT-likelihood*. The box storage in Fig. 5.1 is consequently implemented as a priority queue.

When generating test values for the variable x_i, **raSAT** further observes the coefficient of the corresponding error symbol ϵ_i to guide the generation. The following example illustrates the details of the heuristic.

Example 29 *The AF2 transforms g as $[2, 2] - [6, 6]\epsilon_1 - [2, 2]\epsilon_2 + [4, 4]e_+ + [6, 6]\epsilon_\pm$ for the APC $g = x_1^2 + x_1x_2 - 4 < 0$ and the box $B = [-3, 1] \times [-4, 2]$ of (x_1, x_2). The coefficient corresponding to x_1 is $-[6, 6]$ which is negative. We consequently estimate that g is monotonically decreasing w.r.t. x_1, and similarly w.r.t. x_2. In addition, we need the test values such that the value of g become as small as possible because the APC is $g < 0$. We can select the upper bounds of x_1 and x_2 as the test values for these two reasons. The test case $\{(x_1, 1), (x_2, 2)\}$ in this example will satisfy the APC. If we rather take the lower bounds as test values, i.e., $\{(x_1, -3), (x_4, -4)\}$, the test case will not satisfy the APC.*

The concrete strategy for generating the test values of each variable in **raSAT** is summarized in Table 5.1. Note that when generating test values for some variables at line 6 of Algorithm 3, one APC $g_i \diamond 0$ is passed into the generation function.

If one test value is generated for a variable, the generation follows the strategy in Table 5.1. On the other hand, if two test values are generated, one follows the strategy in Table 5.1, and the other is randomly selected.

Sign of the error coefficient	$g_i > 0$ or $g_i \geq 0$	$g_i < 0$ or $g_i \leq 0$	$g_i = 0$
Positive	Upper bound	Lower bound	Random
Negative	Lower bound	Upper bound	Random
0	Random	Random	Random

Table 5.1: Strategy for generating one test value of a variable in **raSAT**

As mentioned, **raSAT** restricts the number of test cases to at most 2^{10} by selecting the 10 most influential variables for generating 2 test values. The selection proceeds as following two steps.

1. Select 10 APCs with the least SAT-likelihood.

2. For each selected APC, choose one variable with the highest sensitivity.

The premise behind this is that in incremental testing on APCs, test cases for an APC with the least SAT-likelihood will be more likely to be discarded, and test-UNSAT thus will be detected earlier. In addition, variables with smaller sensitivities will not make the polynomial change it values much when their values are changed.

These metrics are also used in "Box decomposition" where **raSAT** selects the variable with the largest sensitivity in the Test-UNSAT APC with the least SAT-likelihood to decompose its interval.

raSAT starts the search with a small initial box. The current choice is $[-8, 8]^n$, which is selected by experiments. If satisfiability is detected inside it, the search completes and returns SAT. If unsatisfiability is detected, **raSAT** restarts with the box $[-\infty, \infty]^n$. Because of our strategy in "Box decomposition" (Section 5.1), **raSAT** eventually produces the box $[-8, 8]^n$ after iterative decompositions, which is immediately discarded because it had been proven to not satisfy the constraint.

Since ICP may not terminate (viz., for the cases in Fig. 4.4c and 4.4d), in order to avoid local pitfalls selected by the heuristics, **raSAT** prepares a threshold on the size of boxes to avoid further decompositions and switches the search to other boxes. When all boxes are below the threshold, **raSAT** selects another propositional solution provided by the SAT solver. If **raSAT** fails to show either SAT or UNSAT on every propositional solution with the given threshold, it decreases the threshold and resets the search. The initial threshold is currently set to 0.125 and reset by dividing the previous threshold by 8.

The effect of heuristics was examined with 18 combinations of these metrics and random choices (details are in Appendix C). Of these, only the above combination demonstrated visible differences from the random choices, especially on SAT detection for quite large problems such as QF_NRA/zankl/matrix-[2-5]-all-*.smt2. The current heuristics choice detects 15 SAT (including nine problems marked "*unknown*" in the SMT-LIB benchmarks), whereas others detect at most 5 SATs (with at most 1 problem annotated as "*unknown*").

5.4 raSAT Loop for Constraints over Integers

We have mostly focused on polynomial constraints over reals (QF_NRA) in this paper. The required modifications to apply the techniques to those over integers (QF_NIA) are simple.

- In testing, test values are restricted to integer-valued ones.

- In box decomposition, the lower and upper bounds of intervals in each box are rounded up and down respectively, to the nearest integers. For instance, $[-9.6, 10.63]$ becomes $[-9, 10]$.

Since, we can directly compute exact equality on integers, the above modifications of **raSAT** loop are able to solve satisfiable equations.

Chapter 6

Subtropical Satisfiability

This chapter extends the idea in [76] to find a satisfiable instance for multiple inequalities. While Sect. 3.4.2 overviews about subtropical real root finding, we are going to illustrate our extension idea through an example.

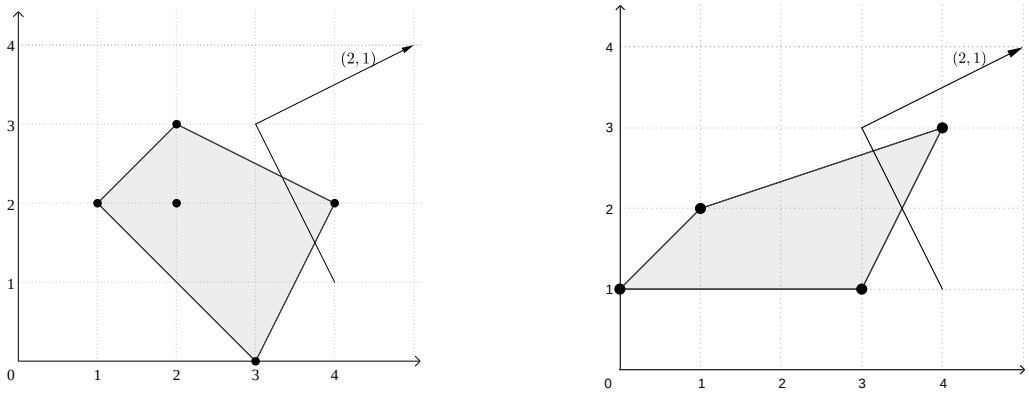


Figure 6.1: Newton polytope of $x^4y^2 - 3x^3 + x^2y^2 - 1000x^2y^3 - 9xy^2$ and $x^4y^3 - 555x^3y - 793xy^2 - y$

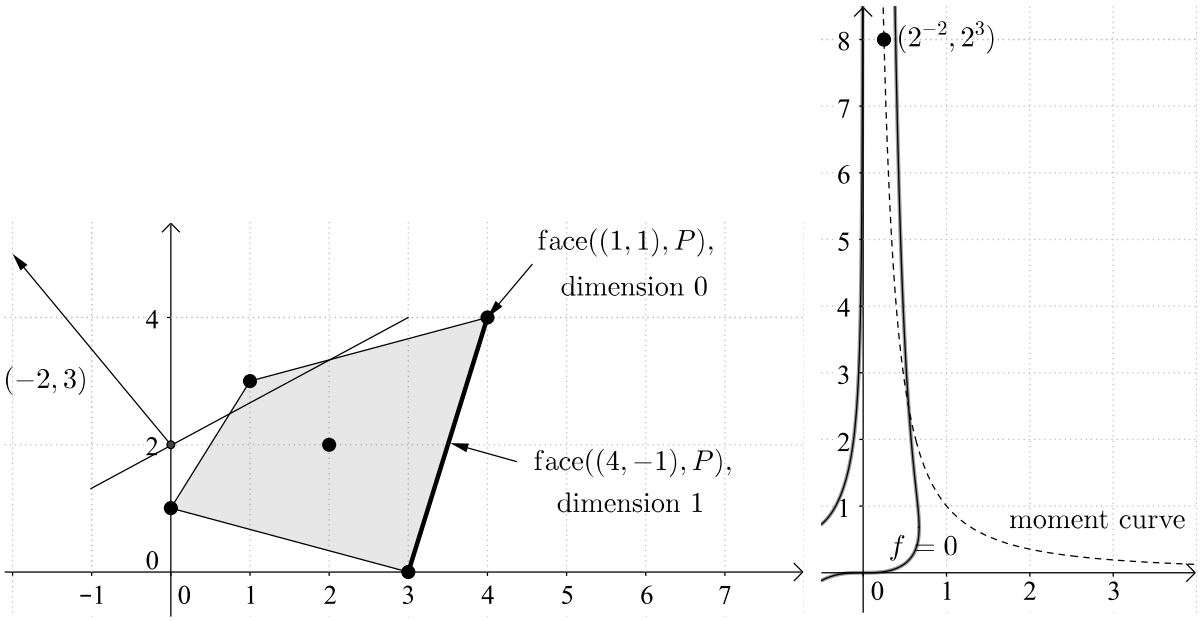
Example 30 *Consider* $f > 0 \wedge g > 0$ *where* $f := x^4y^2 - 3x^3 + x^2y^2 - 1000x^2y^3 - 9xy^2$ *and* $g = x^4y^3 - 555x^3y - 793xy^2 - y > 0$. *Figure 6.1 illustrates the Newton polytope of* f *and* g. *We might construct a solution for* $f > 0 \wedge g > o$ *as follows.*

- *Following example 15, since* $(4, 2)$ *is a vertex of the Newton polytope, the monomial* x^4y^2 *dominates* $-3x^3 + x^2y^2 - 1000x^2y^3 - 9xy^2$ *when* (x, y) *follows the curve* (a^2, a^1). *As a result, with a large enough* a, $f(a^2, a^1) > 0$.
- *Similarly* $(4, 3)$ *is a vertex of Newton polytope for* g, *the monomial* x^4y^3 *dominates others in* g *when* (x, y) *follows the curve* (a^2, a^1). *As a result, with a large enough* a, $f(a^2, a^1) > 0$.

As a consequence, with a large enough a, $f(a) > 0 \wedge g(a) > 0$.

In this example, the common normal vector $(2, 1)$ ensures the global solution for inequalities, which is the central idea of our extension.

Before describing the extension and some generalizations for the method, we briefly introduce basic facts and revisit the idea of subtropical real root finding [76].



(a) The frame and the Newton polytope P of f

(b) The variety of f and the moment curve (a^{-2}, a^3)

Figure 6.2: An illustration of Example 31, where $f = y + 2xy^3 - 3x^2y^2 - x^3 - 4x^4y^4$

6.1 Basic Facts about Newton Polytopes

For $a \in \mathbb{R}$, a vector $\mathbf{x} = (x_1, \ldots, x_d)$ of variables, and $\mathbf{p} = (p_1, \ldots, p_d) \in \mathbb{R}^d$ we use notations $a^{\mathbf{p}} = (a^{p_1}, \ldots, a^{p_d})$ and $\mathbf{x}^{\mathbf{p}} = (x_1^{p_1}, \ldots, x_d^{p_d})$. The *frame* F of a multivariate polynomial $f \in \mathbb{Z}[x_1, \ldots, x_d]$ in sparse distributive representation

$$f = \sum_{\mathbf{p} \in F} f_{\mathbf{p}} \mathbf{x}^{\mathbf{p}}, \quad f_{\mathbf{p}} \neq 0, \quad F \subset \mathbb{N}^d,$$

is uniquely determined, and written $\text{frame}(f)$. It can be partitioned into a positive and a negative frame, according to the sign of $f_{\mathbf{p}}$:

$$\text{frame}^+(f) = \{\, \mathbf{p} \in \text{frame}(f) \mid f_{\mathbf{p}} > 0 \,\}, \quad \text{frame}^-(f) = \{\, \mathbf{p} \in \text{frame}(f) \mid f_{\mathbf{p}} < 0 \,\}.$$

For $\mathbf{p}$, $\mathbf{q} \in \mathbb{R}^d$ we define $\overline{\mathbf{pq}} = \{\, \lambda\mathbf{p} + (1-\lambda)\mathbf{q} \in \mathbb{R}^n \mid \lambda \in [0,1] \,\}$. Recall that $S \subseteq \mathbb{R}^d$ is *convex* if $\overline{\mathbf{pq}} \subseteq S$ for all $\mathbf{p}$, $\mathbf{q} \in S$. Furthermore, given any $S \subseteq \mathbb{R}^d$, the *convex hull* $\text{conv}(S) \subseteq \mathbb{R}^d$ is the unique inclusion-minimal convex set containing S. The *Newton polytope* of a polynomial f is the convex hull of its frame, $\text{newton}(f) = \text{conv}(\text{frame}(f))$. Fig. 6.2a illustrates the Newton polytope of

$$y + 2xy^3 - 3x^2y^2 - x^3 - 4x^4y^4 \in \mathbb{Z}[x, y],$$

which is the convex hull of its frame $\{(0,1), (1,3), (2,2), (3,0), (4,4)\} \subset \mathbb{N}^2$. As a convex hull of a finite set of points, the Newton polytope is bounded and thus indeed a polytope [69].

The *face* [69] of a polytope $P \subseteq \mathbb{R}^d$ with respect to a vector $\mathbf{n} \in \mathbb{R}^d$ is

$$\text{face}(\mathbf{n}, P) = \{\, \mathbf{p} \in P \mid \mathbf{n}^T\mathbf{p} \geq \mathbf{n}^T\mathbf{q} \text{ for all } \mathbf{q} \in P \,\}.$$

Faces of dimension 0 are called *vertices*. We denote by $\mathrm{V}(P)$ the set of all vertices of P. We have $\mathbf{p} \in \mathrm{V}(P)$ if and only if there exists $\mathbf{n} \in \mathbb{R}^d$ such that $\mathbf{n}^T\mathbf{p} > \mathbf{n}^T\mathbf{q}$ for all $\mathbf{q} \in P \setminus \{\mathbf{p}\}$. In Fig.6.2a, $(4, 4)$ is a vertex of the Newton polytope with respect to $(1, 1)$.

It is easy to see that for finite $S \subset \mathbb{R}^d$ we have

$$\mathrm{V}(\mathrm{conv}(S)) \subseteq S \subseteq \mathrm{conv}(S). \tag{6.1}$$

The following lemma gives a characterization of $\mathrm{V}(\mathrm{conv}(S))$:

Lemma 2 *Let $S \subset \mathbb{R}^d$ be finite, and let $\mathbf{p} \in S$. The following are equivalent:*

(i) $\mathbf{p}$ *is a vertex of* $\mathrm{conv}(S)$ *with respect to* $\mathbf{n}$.

(ii) There exists a hyperplane $H : \mathbf{n}^T\mathbf{x} + c = 0$ that strictly separates $\mathbf{p}$ from $S \setminus \{\mathbf{p}\}$, and the normal vector $\mathbf{n}$ is directed from H towards $\mathbf{p}$.

Proof 3 *Assume (i). Then there exists $\mathbf{n} \in \mathbb{R}^d$ such that $\mathbf{n}^T\mathbf{p} > \mathbf{n}^T\mathbf{q}$ for all $\mathbf{q} \in S\setminus\{\mathbf{p}\} \subseteq \mathrm{conv}(S) \setminus \{\mathbf{p}\}$. Choose $\mathbf{q}_0 \in S \setminus \{\mathbf{p}\}$ such that $\mathbf{n}^T\mathbf{q}_0$ is maximal, and choose c such that $\mathbf{n}^T\mathbf{p} > -c > \mathbf{n}^T\mathbf{q}_0$. Then $\mathbf{n}^T\mathbf{p} + c > 0$ and $\mathbf{n}^T\mathbf{q} + c \leq \mathbf{n}^T\mathbf{q}_0 + c < 0$ for all $\mathbf{q} \in S \setminus \{\mathbf{p}\}$. Hence $H : \mathbf{n}^T\mathbf{p} + c = 0$ is the desired hyperplane.*

Assume (ii). It follows that $\mathbf{n}^T\mathbf{p}+c > 0 > \mathbf{n}^T\mathbf{q}+c$ for all $\mathbf{q} \in S \setminus \{\mathbf{p}\}$. If $\mathbf{q} \in S \setminus \{\mathbf{p}\}$, then $\mathbf{n}^T\mathbf{p} > \mathbf{n}^T\mathbf{q}$. If, in contrast, $\mathbf{q} \in (\mathrm{conv}(S)\setminus S)\setminus\{\mathbf{p}\} = \mathrm{conv}(S)\setminus S$, then $\mathbf{q} = \sum_{\mathbf{s}\in S} t_\mathbf{s}\mathbf{s}$, where $t_\mathbf{s} \in [0, 1]$, $\sum_{\mathbf{s}\in S} t_\mathbf{s} = 1$, and at least two $t_\mathbf{s}$ are greater than 0. It follows that

$$\mathbf{n}^T\mathbf{q} = \mathbf{n}^T \sum_{\mathbf{s}\in S} t_\mathbf{s}\mathbf{s} < \mathbf{n}^T\mathbf{p} \sum_{\mathbf{s}\in S} t_\mathbf{s} = \mathbf{n}^T\mathbf{p}. \quad (Q.E.D.)$$

Let $S_1, \ldots, S_m \subseteq \mathbb{R}^d$, and let $\mathbf{n} \in \mathbb{R}^d$. If there exist $\mathbf{p}_1 \in S_1, \ldots, \mathbf{p}_n \in S_m$ such that each $\mathbf{p}_i$ is a vertex of $\mathrm{conv}(S_i)$ with respect to $\mathbf{n}$, then the (unique) *vertex cluster* of $\{S_i\}_{i\in\{1,\ldots,m\}}$ with respect to $\mathbf{n}$ is defined as $(\mathbf{p}_1, \ldots, \mathbf{p}_m)$.

6.2 Subtropical Real Root Finding Revisited

This section improves on the original method described in [76]. It furthermore lays some theoretical foundations to better understand the limitations of the heuristic approach.

The method finds real zeros with all positive coordinates of a multivariate polynomial f in three steps:

1. Evaluate $f(1, \ldots, 1)$. If this is 0, we are done. If this is greater than 0, then consider $-f$ instead of f. We may now assume that we have found $f(1, \ldots, 1) < 0$.

2. Find $\mathbf{p}$ with all positive coordinates such that $f(\mathbf{p}) > 0$.

3. Use the Intermediate Value Theorem (a continuous function with positive and negative values has a zero) to construct a root of f on the line segment $\overline{\mathbf{1p}}$.

We focus here on Step 2. Our technique builds on [76, Lemma 4], which we are going to restate now in a slightly generalized form. While the original lemma required that $\mathbf{p} \in \mathrm{frame}(f) \setminus \{\mathbf{0}\}$, inspection of the proof shows that this limitation is not necessary:

Lemma 3 *Let f be a polynomial, and let $\mathbf{p} \in \text{frame}(f)$ be a vertex of $\text{newton}(f)$ with respect to $\mathbf{n} \in \mathbb{R}^d$. Then there exists $a_0 \in \mathbb{R}^+$ such that for all $a \in \mathbb{R}^+$ with $a \geq a_0$ the following holds:*

1. $|f_{\mathbf{p}}\ a^{\mathbf{n}^T\mathbf{p}}| > |\sum_{\mathbf{q}\in\text{frame}(f)\setminus\{\mathbf{p}\}} f_{\mathbf{q}}\ a^{\mathbf{n}^T\mathbf{q}}|$,
2. $\text{sign}(f(a^{\mathbf{n}})) = \text{sign}(f_{\mathbf{p}})$. *(Q.E.D.)*

In order to find a point with all positive coordinates where $f > 0$, the original method iteratively examines each $\mathbf{p} \in \text{frame}^+(f) \setminus \{\mathbf{0}\}$ to check if it is a vertex of $\text{newton}(f)$ with respect to some $\mathbf{n} \in \mathbb{R}^d$. In the positive case, Lemma 3 guarantees for large enough $a \in \mathbb{R}^+$ that $\text{sign}(f(a^{\mathbf{n}})) = \text{sign}(f_{\mathbf{p}}) = 1$, in other words, $f(a^{\mathbf{n}}) > 0$.

Example 31 *Consider $f = y + 2xy^3 - 3x^2y^2 - x^3 - 4x^4y^4$. Figure 6.2a illustrates the frame and the Newton polytope of f, of which $(1, 3)$ is a vertex with respect to $(-2, 3)$. Lemma 3 ensures that $f(a^{-2}, a^3)$ is strictly positive for sufficiently large positive a. For example, $f(2^{-2}, 2^3) = \frac{51193}{256}$. Figure 6.2b shows how the moment curve (a^{-2}, a^3) with $a \geq 2$ will not leave the sign invariant region of f that contains $(2^{-2}, 2^3)$.*

An exponent vector $\mathbf{0} \in \text{frame}(f)$ corresponds to an absolute summand $f_{\mathbf{0}}$ in f. Its above-mentioned explicit exclusion in [76, Lemma 4] originated from the false intuition that one cannot achieve $\text{sign}(f(a^{\mathbf{n}})) = \text{sign}(f_{\mathbf{0}})$ because the monomial $f_{\mathbf{0}}$ is invariant under the choice of a. However, the inclusion of $\mathbf{0}$ can yield a normal vector $\mathbf{n}$ which renders all other monomials small enough for $f_{\mathbf{0}}$ to dominate.

Given a finite set $S \subset \mathbb{R}^d$ and a point $\mathbf{p} \in S$, the original method uses linear programming to determine if $\mathbf{p}$ is a vertex of $\text{conv}(S)$ w.r.t. some vector $\mathbf{n} \in \mathbb{R}^d$. Indeed, from Lemma 2, the problem can be reduced to finding a hyperplane $H : \mathbf{n}^T\mathbf{x} + c = 0$ that strictly separates $\mathbf{p}$ from $S \setminus \{\mathbf{p}\}$ with the normal vector $\mathbf{n}$ pointing from H to $\mathbf{p}$. This is equivalent to solving the following linear problem with $d + 1$ real variables $\mathbf{n}$ and c:

$$\varphi(\mathbf{p}, S, \mathbf{n}, c) \doteq \mathbf{n}^T\mathbf{p} + c > 0 \wedge \bigwedge_{\mathbf{q}\in S\setminus\{\mathbf{p}\}} \mathbf{n}^T\mathbf{q} + c < 0. \tag{6.2}$$

Notice that with the occurrence of a nonzero absolute summand the corresponding point $\mathbf{0}$ is generally a vertex of the Newton polytope with respect to $-\mathbf{1} = (-1, \ldots, -1)$. This raises the question whether there are other special points that are certainly vertices of the Newton polytope. In fact, $\mathbf{0}$ is a lexicographic minimum in $\text{frame}(f)$, and it is not hard to see that minima and maxima with respect to lexicographic orderings are generally vertices of the Newton polytope.

We are now going to generalize that observation. A *monotonic total preorder* $\preceq \subseteq \mathbb{Z}^d \times \mathbb{Z}^d$ is defined as follows:

(i) $\mathbf{x} \preceq \mathbf{x}$ (reflexivity)

(ii) $\mathbf{x} \preceq \mathbf{y} \wedge \mathbf{y} \preceq \mathbf{z} \longrightarrow \mathbf{x} \preceq \mathbf{z}$ (transitivity)

(iii) $\mathbf{x} \preceq \mathbf{y} \longrightarrow \mathbf{x} + \mathbf{z} \preceq \mathbf{y} + \mathbf{z}$ (monotonicity)

(iv) $\mathbf{x} \preceq \mathbf{y} \vee \mathbf{y} \preceq \mathbf{x}$ (totality).

The difference to a total order is the missing anti-symmetry. As an example in $\mathbb{Z}^2$ consider $(x_1, x_2) \preceq (y_1, y_2)$ if and only if $x_1 + x_2 \leq y_1 + y_2$. Then $-2 \preceq 2$ and $2 \preceq -2$ but $-2 \neq 2$. Our definition of $\preceq$ on the extended domain $\mathbb{Z}^d$ guarantees a cancellation law $\mathbf{x} + \mathbf{z} \preceq \mathbf{y} + \mathbf{z} \longrightarrow \mathbf{x} \preceq \mathbf{y}$ also on $\mathbb{N}^d$. The following lemma follows by induction using monotonicity and cancellation:

Lemma 4 *For $n \in \mathbb{N} \setminus \{0\}$ denote as usual the n-fold addition of $\mathbf{x}$ as $n \odot \mathbf{x}$. Then $\mathbf{x} \preceq \mathbf{y} \longleftrightarrow n \odot \mathbf{x} \preceq n \odot \mathbf{y}$.* *(Q.E.D.)*

Any monotonic preorder $\preceq$ on $\mathbb{Z}^d$ can be extended to $\mathbb{Q}^d$: Using a suitable principle denominator $n \in \mathbb{N} \setminus \{0\}$ define

$$\left(\frac{x_1}{n}, \ldots, \frac{x_d}{n}\right) \preceq \left(\frac{y_1}{n}, \ldots, \frac{y_d}{n}\right) \quad \text{if and only if} \quad (x_1, \ldots, x_d) \preceq (y_1, \ldots, y_d).$$

This is well-defined.

Given $\mathbf{x} \preceq \mathbf{y}$ we have either $\mathbf{y} \not\preceq \mathbf{x}$ or $\mathbf{y} \preceq \mathbf{x}$. In the former case, we say that $\mathbf{x}$ and $\mathbf{y}$ are *strictly* preordered and write $\mathbf{x} \prec \mathbf{y}$. In the latter case, they are *not* strictly preordered, i.e., $\mathbf{x} \not\prec \mathbf{y}$ although we might have $\mathbf{x} \neq \mathbf{y}$. In particular, reflexivity yields $\mathbf{x} \preceq \mathbf{x}$ and hence certainly $\mathbf{x} \not\prec \mathbf{x}$.

Example 32 *Lexicographic orders are monotonic total orders and thus monotonic total preorders. Hence our notion covers our discussion of the absolute summand above. Here are some further examples: For $i \in \{1, \ldots, d\}$ we define $\mathbf{x} \preceq_i \mathbf{y}$ if and only if $\pi_i(\mathbf{x}) \leq \pi_i(\mathbf{y})$, where π_i denotes the i-th projection. Similarly, $\mathbf{x} \succeq_i \mathbf{y}$ if and only if $\pi_i(\mathbf{x}) \geq \pi_i(\mathbf{y})$. Next, $\mathbf{x} \preceq_\Sigma \mathbf{y}$ if and only if $\sum_i x_i \leq \sum_i y_i$. Our last example is going to be instrumental with the proof of the next theorem: Fix $\mathbf{n} \in \mathbb{R}^d$, and define for $\mathbf{p}$, $\mathbf{p}' \in \mathbb{Z}^d$ that $\mathbf{p} \preceq_\mathbf{n} \mathbf{p}'$ if and only if $\mathbf{n}^T\mathbf{p} \leq \mathbf{n}^T\mathbf{p}'$.*

Theorem 4 *Let $f \in \mathbb{Z}[x_1, \ldots, x_d]$, and let $\mathbf{p} \in \mathrm{frame}(f)$. Then the following are equivalent:*

(i) $\mathbf{p} \in \mathrm{V}(\mathrm{newton}(f))$

(ii) There exists a monotonic total preorder $\preceq$ on $\mathbb{Z}^d$ such that

$$\mathbf{p} = \max\nolimits_\prec(\mathrm{frame}(f)).$$

Proof 4 *Let $\mathbf{p}$ be a vertex of $\mathrm{newton}(f)$ specifically with respect to $\mathbf{n}$. By our definition of a vertex in Sect. 6.1, $\mathbf{p}$ is the maximum of $\mathrm{frame}(f)$ with respect to $\prec_\mathbf{n}$.*

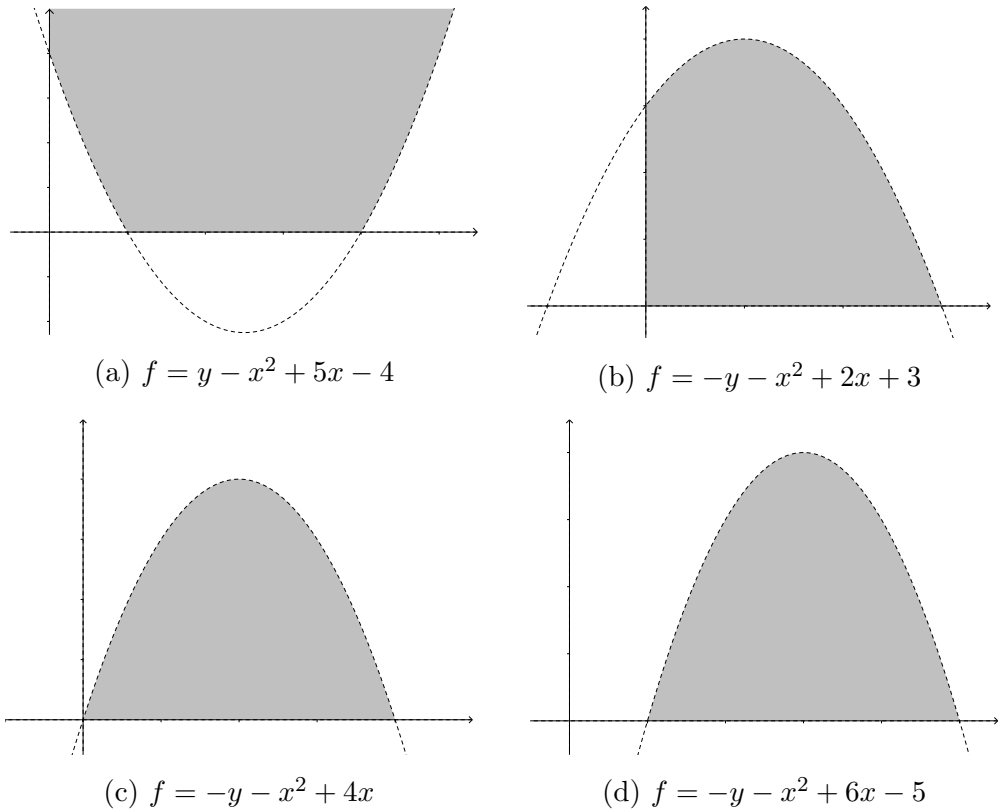
Let, vice versa, $\preceq$ be a monotonic total preorder on $\mathbb{Z}^d$, and let $\mathbf{p} = \max_\prec(\mathrm{frame}(f))$. Shortly denote $V = \mathrm{V}(\mathrm{newton}(f))$, and assume for a contradiction that $\mathbf{p} \notin V$. Since $\mathbf{p} \in \mathrm{frame}(f) \subseteq \mathrm{newton}(f)$, we have

$$\mathbf{p} = \sum_{\mathbf{s} \in V} t_\mathbf{s}\mathbf{s}, \quad \textit{where} \quad t_\mathbf{s} \in [0, 1] \quad \textit{and} \quad \sum_{\mathbf{s} \in V} t_\mathbf{s} = 1.$$

According to (6.1) in Sect. 6.1 we know that $V \subseteq \mathrm{frame}(f) \subseteq \mathrm{newton}(f)$. It follows that $\mathbf{s} \prec \mathbf{p}$ for all $\mathbf{s} \in V$, and using monotony we obtain

$$\mathbf{p} \prec \sum_{\mathbf{s} \in V} t_\mathbf{s}\mathbf{p} = \left(\sum_{\mathbf{s} \in V} t_\mathbf{s}\right)\mathbf{p} = \mathbf{p}.$$

On the other hand, we know that generally $\mathbf{p} \not\prec \mathbf{p}$, a contradiction. *(Q.E.D.)*



(a) $f = y - x^2 + 5x - 4$ (b) $f = -y - x^2 + 2x + 3$

(c) $f = -y - x^2 + 4x$ (d) $f = -y - x^2 + 6x - 5$

Figure 6.3: Four scenarios of polynomials for the subtropical method. The shaded regions show $\Pi(f)$.

In Fig. 6.2a we have $(0,1) = \max_{\succeq_1}(\text{frame}(f))$, $(3,0) = \max_{\succeq_2}(\text{frame}(f))$, and $(4,4) = \max_{\preceq_1}(\text{frame}(f)) = \max_{\preceq_2}(\text{frame}(f))$. This shows that, besides contributing to our theoretical understanding, the theorem can be used to substantiate the efficient treatment of certain special cases in combination with other methods for identifying vertices of the Newton polytope.

Corollary 1 *Let $f \in \mathbb{Z}[x_1, \ldots, x_d]$, and let $\mathbf{p} \in \text{frame}(f)$. If $p = \max(\text{frame}(f))$ or $p = \min(\text{frame}(f))$ with respect to an admissible term order in the sense of Grbner Basis theory (see Sec. 3.1.2), then $p \in \text{V}(\text{newton}(f))$.* *(Q.E.D.)*

It is one of our research goals to identify and characterize those polynomials where the subtropical heuristic succeeds in finding positive points. We are now going to give a necessary criterion. Let $f \in \mathbb{Z}[x_1, \ldots, x_d]$, define $\Pi(f) = \{\, \mathbf{r} \in]0, \infty[^d \mid f(\mathbf{r}) > 0 \,\}$, and denote by $\overline{\Pi(f)}$ its closure with respect to the natural topology. In Lemma 3, when a tends to ∞, $a^{\mathbf{n}}$ will tend to some $\mathbf{r} \in \{0, \infty\}^d$. If $\mathbf{r} = \mathbf{0}$, then $\mathbf{0} \in \overline{\Pi(f)}$. Otherwise, $\Pi(f)$ is unbounded. Consequently, for the method to succeed, Π must have at least one of those two properties. Figure 6.3 illustrates four scenarios: the subtropical method succeeds in the first three cases while it fails to find a point in $\Pi(f)$ in the last one. The first sub-figure presents a case where $\Pi(f)$ is unbounded. The second and third sub-figures illustrate cases where the closure of $\Pi(f)$ contains $(0,0)$. In the fourth sub-figure where neither $\Pi(f)$ is unbounded nor its closure contains $(0,0)$, the method cannot find any positive value of the variables for f to be positive.

6.3 From One Polynomial to Multiple Ones

The subtropical method as presented in [76] finds zeros with all positive coordinates of one single multivariate polynomial. This requires to find a corresponding point with a positive value of the polynomial. In the sequel we restrict ourselves to this sub-task. This will allow us to generalize from one polynomial to simultaneous positive values of finitely many polynomials.

A Sufficient Condition

With a single polynomial, the existence of a positive vertex of the Newton polytope guarantees the existence of positive real choices for the variables with a positive value of that polynomial. For several polynomials we introduce a more general notion: A sequence $(\mathbf{p}_1, \ldots, \mathbf{p}_m)$ is a *positive vertex cluster* of $\{f_i\}_{i\in\{1,\ldots,m\}}$ with respect to $\mathbf{n} \in \mathbb{R}^d$ if it is a vertex cluster of $\{\text{frame}(f_i)\}_{i\in\{1,\ldots,m\}}$ with respect to $\mathbf{n}$ and $\mathbf{p}_i \in \text{frame}^+(f_i)$ for all $i \in \{1, \ldots, m\}$. The existence of a positive vertex cluster will guarantee the existence of positive real choices of the variables such that all polynomials f_1, ..., f_m are simultaneously positive. The following lemma is a corresponding generalization of Lemma 3:

Lemma 5 *If there exists a vertex cluster $(\mathbf{p}_1, \ldots, \mathbf{p}_m)$ of $\{\text{frame}(f_i)\}_{i\in\{1,\ldots,m\}}$ with respect to $\mathbf{n} \in \mathbb{R}^n$, then there exists $a_0 \in \mathbb{R}^+$ such that the following holds for all $a \in \mathbb{R}^+$ with $a \geq a_0$ and all $i \in \{1, \ldots, m\}$:*

1. $|(f_i)_{\mathbf{p}_i}\ a^{\mathbf{n}^T\mathbf{p}_i}| > |\sum_{\mathbf{q}\in\text{frame}(f_i)\setminus\{\mathbf{p}_i\}}(f_i)_{\mathbf{q}}\ a^{\mathbf{n}^T\mathbf{q}}|$,
2. $\text{sign}(f_i(a^{\mathbf{n}})) = \text{sign}((f_i)_{\mathbf{p}_i})$.

Proof 5 *From [76, Lemma 4], for each $i \in \{1, \ldots, m\}$, there exist $a_{0,i} \in \mathbb{R}^+$ such that for all $a \in \mathbb{R}^+$ with $a \geq a_{0,i}$ the following holds:*

1. $|(f_i)_{\mathbf{p}_i}\ a^{\mathbf{n}^T\mathbf{p}_i}| > |\sum_{\mathbf{q}\in\text{frame}(f_i)\setminus\{\mathbf{p}_i\}}(f_i)_{\mathbf{q}}\ a^{\mathbf{n}^T\mathbf{q}}|$,
2. $\text{sign}(f_i(a^n)) = \text{sign}((f_i)_{\mathbf{p}_i})$.

It now suffices to take $a_0 = \max\{a_{0,i} \mid 1 \leq i \leq m\}$. *(Q.E.D.)*

Similarly to the case of one polynomial, the following Proposition provides a sufficient condition for the existence of a common point with a positive value for multiple polynomials.

Proposition 1 *If there exists a positive vertex cluster $(\mathbf{p}_1, \ldots, \mathbf{p}_m)$ of the polynomials $\{f_i\}_{i\in\{1,\ldots,m\}}$ with respect to a vector $\mathbf{n} \in \mathbb{R}^d$, then there exists $a_0 \in \mathbb{R}^+$ such that for all $a \in \mathbb{R}^+$ with $a \geq a_0$ the following holds:*

$$\bigwedge_{i=1}^{m} f_i(a^{\mathbf{n}}) > 0.$$

Proof 6 *For $i \in \{1, \ldots, m\}$, since $\mathbf{p}_i \in \text{frame}^+(f_i)$, Lemma 5 implies $f_i(a^{\mathbf{n}}) > 0$. (Q.E.D.)*

Example 33 *Consider* $f_1 = 2 - xy^2z + x^2yz^3$, $f_2 = 3 - xy^2z^4 - x^2z - x^4y^3z^3$, *and* $f_3 = 4 - z - y - x + 4$. *The exponent vector* $\mathbf{0}$ *is a vertex of newton*(f_1), newton(f_2), *and* newton(f_3) *with respect to* $(-1,-1,-1)$. *Choose* $a_0 = 2 \in \mathbb{R}^+$. *Then for all* $a \in \mathbb{R}$ *with* $a \geq a_0$ *we have* $f_1(a^{-1},a^{-1},a^{-1}) > 0 \wedge f_2(a^{-1},a^{-1},a^{-1}) > 0 \wedge f_3(a^{-1},a^{-1},a^{-1}) > 0$. *(Q.E.D.)*

Existence of Positive Vertex Clusters

Given polynomials $f_1, \ldots, f_m$, Proposition 1 provides a sufficient condition, i.e. the existence of a positive vertex cluster of $\{f_i\}_{i\in\{1,\ldots,m\}}$, for the satisfiability of $\bigwedge_{i=1}^m f_i > 0$. A straightforward method to decide the existence of such a cluster is to verify whether each $(\mathbf{p}_1,\ldots,\mathbf{p}_m) \in \text{frame}^+(f_1) \times \cdots \times \text{frame}^+(f_m)$ is a positive vertex cluster by checking the satisfiability of the formula

$$\bigwedge_{i\in\{1,\ldots,m\}} \varphi(\mathbf{p}_i, \text{frame}(f_i), \mathbf{n}, c_i),$$

where φ is defined as in (6.2) on p.38. This is a linear problem with $d+m$ variables $\mathbf{n}$, c_1, $\ldots$, c_m. Since frame(f_1), $\ldots$, frame(f_m) are finite, checking all m-tuples $(\mathbf{p}_1,\ldots,\mathbf{p}_m)$ will terminate, provided we rely on a complete algorithm for linear programming, such as the Simplex algorithm [17], the ellipsoid method [45], or the interior point method [44]. This provides a decision procedure for the existence of a positive vertex cluster of $\{f_i\}_{i\in\{1,\ldots,m\}}$. However, this requires checking all candidates in $\text{frame}^+(f_1) \times \cdots \times \text{frame}^+(f_m)$.

We propose to use instead state-of-the-art SMT solving techniques over linear real arithmetic to examine whether or not $\{f_i\}_{i\in\{1,\ldots,m\}}$ has a positive vertex cluster with respect to some $\mathbf{n} \in \mathbb{R}^d$. In the positive case, a solution for $\bigwedge_{i=1}^m f_i > 0$ can be constructed as $a^{\mathbf{n}}$ with a sufficiently large $a \in \mathbb{R}^+$.

To start with, we provide a characterization for the positive frame of a single polynomial to contain a vertex of the Newton polytope.

Lemma 6 *Let* $f \in \mathbb{Z}[\mathbf{x}]$. *The following are equivalent:*

(i) There exists a vertex $\mathbf{p} \in \text{frame}^+(f)$ *of* $\text{newton}(f) = \text{conv}(\text{frame}(f))$ *with respect to* $\mathbf{n} \in \mathbb{R}^d$.

(ii) There exists a vertex $\mathbf{p}' \in \text{frame}^+(f)$ *such that* $\mathbf{p}'$ *is also a vertex of* $\text{conv}(\text{frame}^-(f) \cup \{\mathbf{p}'\})$ *with respect to* $\mathbf{n}' \in \mathbb{R}^d$.

Proof 7 *Assume (i). Take* $\mathbf{p}' = \mathbf{p}$ *and* $\mathbf{n}' = \mathbf{n}$. *Since* $\mathbf{p}$ *is a vertex of newton*(f) *with respect to* $\mathbf{n}$, $\mathbf{n}^T\mathbf{p} > \mathbf{n}^T\mathbf{p}_1$ *for all* $\mathbf{p}_1 \in \text{frame}(f) \setminus \{\mathbf{p}\}$. *This implies that* $\mathbf{n}^T\mathbf{p} > \mathbf{n}^T\mathbf{p}_1$ *for all* $\mathbf{p}_1 \in \text{frame}^-(f) \setminus \{\mathbf{p}\} = \big(\text{frame}^-(f) \cup \{\mathbf{p}\}\big) \setminus \{\mathbf{p}\}$. *In other words,* $\mathbf{p}$ *is a vertex of* $\text{conv}(\text{frame}^-(f) \cup \{\mathbf{p}\})$ *with respect to* $\mathbf{n}$.

Assume (ii). Suppose $V = \text{V}(\text{newton}(f)) \subseteq \text{frame}^-(f)$. *Then,* $\mathbf{p}' = \sum_{\mathbf{s}\in V} t_{\mathbf{s}}\mathbf{s}$ *where* $t_{\mathbf{s}} \in [0,1]$, $\sum_{\mathbf{s}\in V} t_{\mathbf{s}} = 1$. *It follows that*

$$\mathbf{n}'^T\mathbf{p}' = \sum_{\mathbf{s}\in V} t_{\mathbf{s}}\mathbf{n}'^T\mathbf{s} < \sum_{\mathbf{s}\in V} t_{\mathbf{s}}\mathbf{n}'^T\mathbf{p}' = \mathbf{n}'^T\mathbf{p}' \sum_{\mathbf{s}\in V} t_{\mathbf{s}} = \mathbf{n}'^T\mathbf{p}',$$

which is a contradiction. As a result, there must be some $\mathbf{p} \in \text{frame}^+(f)$ *which is a vertex of* $\text{newton}(f)$ *with respect to some* $\mathbf{n} \in \mathbb{R}^d$. *(Q.E.D.)*

Thus some $\mathbf{p} \in \text{frame}^+(f)$ is a vertex of the Newton polytope of a polynomial f if and only if the following formula is satisfiable:

$$\begin{aligned}\psi(f, \mathbf{n}', c) &\doteq \bigvee_{\mathbf{p}\in\text{frame}^+(f)} \varphi\left(\mathbf{p}, \text{frame}^-(f) \cup \{\mathbf{p}\}, \mathbf{n}', c\right) \\ &\equiv \bigvee_{\mathbf{p}\in\text{frame}^+(f)} \left[\mathbf{n}'^T\mathbf{p} + c > 0 \wedge \bigwedge_{\mathbf{q}\in\text{frame}^-(f)} \mathbf{n}'^T\mathbf{q} + c < 0\right] \\ &\equiv \left[\bigvee_{\mathbf{p}\in\text{frame}^+(f)} \mathbf{n}'^T\mathbf{p} + c > 0\right] \wedge \left[\bigwedge_{\mathbf{p}\in\text{frame}^-(f)} \mathbf{n}'^T\mathbf{p} + c < 0\right].\end{aligned}$$

For the case of several polynomials, the following theorem is a direct consequence of Lemma 6.

Theorem 5 *Polynomials $\{f_i\}_{i\in\{1,\ldots,m\}}$ have a positive vertex cluster with respect to $\mathbf{n} \in \mathbb{R}^d$ if and only if $\bigwedge_{i=1}^m \psi(f_i, \mathbf{n}, c_i)$ is satisfiable.* *(Q.E.D.)*

The formula $\bigwedge_{i=1}^m \psi(f_i, \mathbf{n}, c_i)$ can be checked for satisfiability using combinations of linear programming techniques and DPLL(T) procedures [25, 31], i.e., satisfiability modulo linear arithmetic on reals. Any SMT solver supporting the QF_LRA logic is suitable. In the satisfiable case $\{f_i\}_{i\in\{1,\ldots,m\}}$ has a positive vertex cluster and we can construct a solution for $\bigwedge_{i=1}^m f_i > 0$ as discussed earlier.

Example 34 *Consider $f_1 = -12 + 2x^{12}y^{25}z^{49} - 31x^{13}y^{22}z^{110} - 11x^{1000}y^{500}z^{89}$ and $f_2 = -23 + 5xy^{22}z^{110} - 21x^{15}y^{20}z^{1000} + 2x^{100}y^2z^{49}$. With $\mathbf{n} = (n_1, n_2, n_3)$ this yields*

$$\begin{aligned}\psi(f_1, \mathbf{n}, c_1) &= 12n_1 + 25n_2 + 49n_3 + c_1 > 0 \wedge 13n_1 + 22n_2 + 110n_3 + c_1 < 0 \\ &\quad \wedge 1000n_1 + 500n_2 + 89n_3 + c_1 < 0 \wedge c_1 < 0, \\ \psi(f_2, \mathbf{n}, c_2) &= (n_1 + 22n_2 + 110n_3 + c_2 > 0 \vee 100n_1 + 2n_2 + 49n_3 + c_2 > 0) \\ &\quad \wedge 15n_1 + 20n_2 + 1000n_3 + c_2 < 0 \wedge c_2 < 0.\end{aligned}$$

The conjunction $\psi(f_1, \mathbf{n}, c_1) \wedge \psi(f_2, \mathbf{n}, c_2)$ is satisfiable. The SMT solver CVC4 computes $\mathbf{n} = (-\frac{238834}{120461}, \frac{2672460}{1325071}, -\frac{368561}{1325071})$ and $c_1 = c_2 = -1$ as a model. Theorem 5 and Proposition 1 guarantee that there exists a large enough $a \in \mathbb{R}^+$ such that $f_1(a^{\mathbf{n}}) > 0 \wedge f_2(a^{\mathbf{n}}) > 0$. Indeed, $a = 2$ already yields $f_1(a^{\mathbf{n}}) \approx 16371.99$ and $f_2(a^{\mathbf{n}}) \approx 17707.27$. *(Q.E.D.)*

6.4 More Generalization

So far all variables were assumed to be strictly positive, i.e., only solutions $\mathbf{x} \in]0, \infty[^d$ were considered. This section proposes a method for searching over $\mathbb{R}^d$ by encoding sign conditions along with the condition in Theorem 5 as a quantifier-free formula over linear real arithmetic.

Let $V = \{x_1, \ldots, x_d\}$ be the set of variables. We define a *sign variant* of V as a function $\tau : V \mapsto V \cup \{-x \mid x \in V\}$ such that for each $x \in V$, $\tau(x) \in \{x, -x\}$. We write $\tau(f)$ to denote the substitution $f(\tau(x_1), \ldots, \tau(x_d))$ of τ into a polynomial f. Furthermore, $\tau(a)$ denotes $\left(\frac{\tau(x_1)}{x_1}a, \ldots, \frac{\tau(x_d)}{x_d}a\right)$ for $a \in \mathbb{R}$. A sequence $(\mathbf{p}_1, \ldots, \mathbf{p}_m)$ is a *variant positive*

vertex cluster of $\{f_i\}_{i\in\{1,\ldots,m\}}$ with respect to a vector $\mathbf{n} \in \mathbb{R}^d$ and a sign variant τ if $(\mathbf{p}_1, \ldots, \mathbf{p}_m)$ is a positive vertex cluster of $\{\tau(f_i)\}_{i\in\{1,\ldots,m\}}$. Note that the substitution of τ into a polynomial f does not change the exponent vectors in f in terms of their exponents values, but only possibly changes signs of monomials. Given $\mathbf{p} = (p_1, \ldots, p_d) \in \mathbb{N}^d$ and a sign variant τ, we define a formula $\vartheta(\mathbf{p}, \tau)$ such that it is TRUE if and only if the sign of the monomial associated with $\mathbf{p}$ is changed after applying the substitution defined by τ:

$$\vartheta(\mathbf{p}, \tau) \doteq \bigoplus_{i=1}^{d} \big(\tau(x_i) = -x_i \wedge (p_i \bmod 2 = 1)\big).$$

Note that this xor expression becomes TRUE if and only if an odd number of its operands are TRUE. Furthermore, a variable can change the sign of a monomial only when its exponent in that monomial is odd. As a result, if $\vartheta(\mathbf{p}, \tau)$ is TRUE, then applying the substitution defined by τ will change the sign of the monomial associated with $\mathbf{p}$. In conclusion, some $\mathbf{p} \in \text{frame}(f)$ is in the positive frame of $\tau(f)$ if and only if one of the following mutually exclusive conditions holds:

(i) $\mathbf{p} \in \text{frame}^+(f)$ and $\vartheta(\mathbf{p}, \tau) = \text{FALSE}$

(ii) $\mathbf{p} \in \text{frame}^-(f)$ and $\vartheta(\mathbf{p}, \tau) = \text{TRUE}$.

In other words, $\mathbf{p}$ is in the positive frame of $\tau(f)$ if and only if the formula $\Theta(\mathbf{p}, f, \tau) \doteq \big(f_{\mathbf{p}} > 0 \wedge \neg\vartheta(\mathbf{p}, \tau)\big) \vee \big(f_{\mathbf{p}} < 0 \wedge \vartheta(\mathbf{p}, \tau)\big)$ holds. Then, the positive and negative frames of $\tau(f)$ parameterized by τ are defined as

$$\begin{aligned} \text{frame}^+(\tau(f)) &= \{\, \mathbf{p} \in \text{frame}(f) \mid \Theta(\mathbf{p}, f, \tau) \,\}, \\ \text{frame}^-(\tau(f)) &= \{\, \mathbf{p} \in \text{frame}(f) \mid \neg\Theta(\mathbf{p}, f, \tau) \,\}, \end{aligned}$$

respectively. The next lemma provides a sufficient condition for the existence of a solution in $\mathbb{R}^d$ of $\bigwedge_{i=1}^m f_i > 0$.

Lemma 7 *If there exists a variant positive vertex cluster of $\{f_i\}_{i\in\{1,\ldots,m\}}$ with respect to $\mathbf{n} \in \mathbb{R}^d$ and a sign variant τ, then there exists $a_0 \in \mathbb{R}^+$ such that for all $a \in \mathbb{R}^+$ with $a \geq a_0$ the following holds:*

$$\bigwedge_{i=1}^{m} f_i\big(\tau(a)^{\mathbf{n}}\big) > 0.$$

Proof 8 *Since $\{\tau(f_i)\}_{i\in\{1,\ldots,m\}}$ has a positive vertex cluster with respect to $\mathbf{n}$, Proposition 1 guarantees that there exists $a_0 \in \mathbb{R}$ such that for all $a \in \mathbb{R}$ with $a \geq a_0$, we have $\bigwedge_{i=1}^m \tau(f_i)(a^{\mathbf{n}}) > 0$, or $\bigwedge_{i=1}^m f_i\big(\tau(a)^{\mathbf{n}}\big) > 0$.* *(Q.E.D.)*

A variant positive vertex cluster exists if and only if there exist $\mathbf{n} \in \mathbb{R}^d$, $c_1, \ldots, c_m \in \mathbb{R}$, and a sign variant τ such that the following formula becomes TRUE:

$$\Psi(f_1, \ldots, f_m, \mathbf{n}, c_1, \ldots, c_m, \tau) \doteq \bigwedge_{i=1}^{m} \psi\big(\tau(f_i), \mathbf{n}, c_i\big),$$

where for $i \in \{1, \ldots, m\}$:

$$\begin{aligned}
\psi\big(\tau(f_i), \mathbf{n}, c_i\big) &\equiv \left[\bigvee_{\mathbf{p} \in \text{frame}^+(\tau(f_i))} \mathbf{n}^T\mathbf{p} + c_i > 0\right] \wedge \left[\bigwedge_{\mathbf{p} \in \text{frame}^-(\tau(f_i))} \mathbf{n}^T\mathbf{p} + c_i < 0\right] \\
&\equiv \left[\bigvee_{\mathbf{p} \in \text{frame}(f_i)} \Theta(\mathbf{p}, f_i, \tau) \wedge \mathbf{n}^T\mathbf{p} + c_i > 0\right] \\
&\quad \wedge \left[\bigwedge_{\mathbf{p} \in \text{frame}(f_i)} \Theta(\mathbf{p}, f_i, \tau) \vee \mathbf{n}^T\mathbf{p} + c_i < 0\right].
\end{aligned}$$

The sign variant τ can be encoded as d Boolean variables $b_1, \ldots, b_d$ such that b_i is TRUE if and only if $\tau(x_i) = -x_i$ for all $i \in \{1, \ldots, d\}$. Then, the formula $\Psi(f_1, \ldots, f_m, \mathbf{n}, c_1, \ldots, c_m, \tau)$ can be checked for satisfiability using an SMT solver for quantifier-free logic with linear real arithmetic.

Chapter 7

The Intermediate Value Theorem for Solving Equations

As mentioned, using ICP alone is almost impossible to prove satisfiable equations. The authors in [46] proposed a combination of interval arithmetic (IA) and the intermediate value theorem (IVT) to show satisfiability of combinations between inequalities and equations. This chapter extends this idea to also combine testing results with IA and IVT to do the job. Before that, we revisit the generalized intermediate value theorem.

7.1 The Generalized Intermediate Value Theorem

Let us recall related definitions and the *generalized IVT*, i.e., Theorem 5.3.7 in [58]. $\mathcal{P}(E)$ denote the set of all subsets of a set E. A set $C \in \mathcal{P}(\mathbb{R}^n)$ is said to connect two sets $A, B \in \mathcal{P}(\mathbb{R}^n)$ if there is a continuum (i.e. a compact and connected set) $C_0 \subseteq C$ such that $A \cap C_0$ and $B \cap C_0$ are nonempty.

Definition 27 *For a set-valued function $G : E \subseteq \mathbb{R}^n \to \mathcal{P}(\mathbb{R}^m)$, G is* connection-continuous *(*c-continuous *for short) in E, if, for every continuum $C \subseteq E$ and $t, t' \in C$,*

$$graph(G|C) = \{(t, u) \mid t \in C, u \in G(t)\}$$

is compact and connects $\{t\} \times \mathbb{R}^m$ and $\{t'\} \times \mathbb{R}^m$.

Example 35 *Consider the function $G : \mathbb{R} \to \mathcal{P}(\mathbb{R})$ defined by $G(x) = \{x^2, x + 1\}$ for $x \in \mathbb{R}$. Consider any continuum $C \subseteq \mathbb{R}$, e.g. $C = [0.5, 1.5]$, it is easy to see that the graph $graph(G|C)$ consisting of the red line AB and the green curve CD in Fig. 7.1 connects $\{t\} \times \mathbb{R}$ and $\{t'\} \times \mathbb{R}$ for any $t, t' \in C$.*

Theorem 6 ([58, Theorem 5.3.7]) *Let $G : D \times E \subseteq \mathbb{R}^n \times \mathbb{R}^p \to \mathcal{P}(\mathbb{R}^n)$ be c-continuous in $D \times E$. Let $B = [l_1, h_1] \times \cdots \times [l_n, h_n] \subseteq D$ and suppose that, for $i = 1, \cdots, n$ and all $t \in E$,*

$$\begin{array}{ll} sup(G_i(x, t)) \leq 0 & \text{if } x = (r_1, \cdots, r_n) \in B \text{ and } r_i = l_i \\ inf(G_i(x, t)) \geq 0 & \text{if } x = (r_1, \cdots, r_n) \in B \text{ and } r_i = h_i. \end{array}$$

If E is closed and convex, then the set-valued function $H : E \to \mathcal{P}(\mathbb{R}^n)$ defined by:

$$H(t) = \{x \in B \mid (\underbrace{0, \cdots, 0}_{n}) \in G(x, t)\}$$

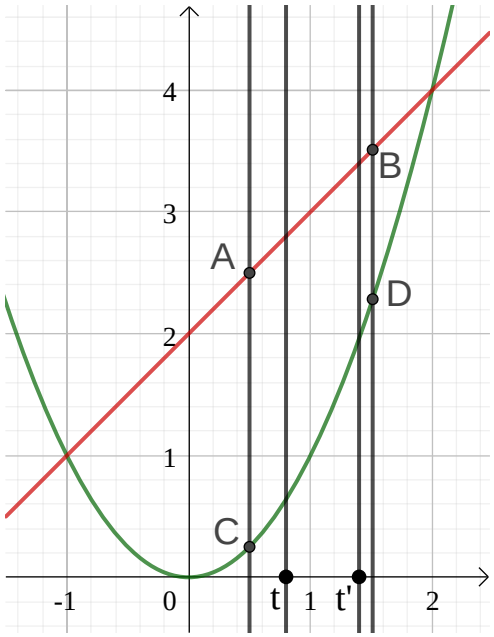


Figure 7.1: Example of a c-continuous function

is c-continuous. In particular, $H(t)$ is nonempty for all $t \in E$.

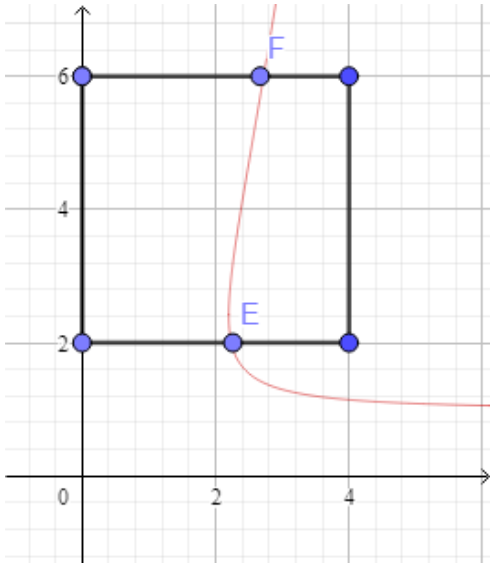


Figure 7.2: Example for the generalized intermediate value theorem

Example 36 *Consider the function $G : [0,4] \times [2,6] \subset \mathbb{R} \times \mathbb{R} \to \mathcal{P}(\mathbb{R})$ defined by $G(x,y) = \{x^2y - x^2 - y^2 - 1\}$. Let $B = [0,4]$. It is easy to see that for all $t \in [2,6]$, we have*

$$sup(G_1(0,t)) = sup(\{-t^2 - 1\}) = -t^2 - 1 \leq 0$$
$$inf(G_1(4,t)) = inf(\{16t - 16 - t^2 - 1\}) = 16t - 16 - t^2 - 1 \geq 0.$$

Since $[2,6]$ is closed and convex, by Theorem 6, the function $H : [2,6] \to \mathcal{P}(\mathbb{R})$ defined by $H(t) = \{x \in [0,4] \mid 0 \in G(x,t)\}$ is c-continuous. In fact,

$$H(t) = \{x \in [0,4] \mid x^2t - x^2 - t^2 - 1 = 0\}.$$

Consider any continuum $C \subseteq [2,6]$, without loss of generality, let $C = [2,6]$. In addition,

$$graph(H \mid C) = \{(t,x) \mid t \in C, x \in H(t)\} = \{(t,x) \mid t \in C, x \in [0,4], x^2t - x^2 - t^2 - 1 = 0\}.$$

Fig. 7.2 illustrates that $graph(H \mid C)$ (the curve FE) is compact and connects $\{t\} \times \mathbb{R}$ and $\{t'\} \times \mathbb{R}$ for any $t, t' \in C$. Note that we use the vertical axis for t and we interpret each point in the plane as (t,x) instead of the casual one (i.e. (x,t)).

It should be noted that in Theorem 6, the dimension of D is n which is equal to the dimension of $g(x) \in G(x)$ for any $x \in D \times E$. In other words, intuitively for our problem setting, the number of variables for applying the IVT is equal to the number of polynomials. First, we provide a theorem extending Theorem 6 so that we can allow the number of variables to be greater than or equal to the number of polynomials. In order to simplify the formulas, we define some notations. Let $B = [l_1, h_1] \times \cdots [l_n, h_n]$ be a box for variables $(x_1, \cdots, x_n)$. The set of variables is denoted as $V = \{x_1, \cdots, x_n\}$, and let $V' = \{x_{i_1}, \cdots, x_{i_k}\}$ be a subset of V. We write:

$$\begin{aligned} B{\downarrow_{V'}} &= \{(r_1, \cdots, r_n) \in B \mid r_j = l_j \text{ for } j = i_1, ..., i_k\} \text{ and} \\ B{\uparrow_{V'}} &= \{(r_1, \cdots, r_n) \in B \mid r_j = h_j \text{ for } j = i_1, ..., i_k\}. \end{aligned}$$

Example 37 *Consider a box $B = [-3,1] \times [-4,2]$ for variables sequence (x,y), (i.e., $V = \{x,y\}$), and $V' = \{y\}$. We have $B{\downarrow_{V'}} = [-3,1] \times [\mathbf{-4}, \mathbf{-4}]$ and $B{\uparrow_{V'}} = [-3,1] \times [2,2]$.*

Furthermore, we denote $B(i) = \frac{h_i - l_i}{\sum_{i=1}^{n}(h_i - l_i)}$ for $i = 1, \ldots, n$. It is easy to the property that for any $t \in \mathbb{R}$, $\sum_{i=1}^{n} B(i)t = t$.

Example 38 *For $B = [2,5] \times [1,8]$, we have $B(1) = \frac{3}{10}$, $B(2) = \frac{7}{10}$, and for any $t \in \mathbb{R}$, $B(1)t + B(2)t = t$.*

We are ready to give the extension of Theorem 6.

Theorem 7 *Let $G : D_1 \times \cdots \times D_n \times E \subseteq \mathbb{R}^{m_1} \times \cdots \times \mathbb{R}^{m_n} \times \mathbb{R}^p \to \mathcal{P}(\mathbb{R}^n)$ be c-continuous in $D_1 \times \cdots \times D_n \times E$ where $m_i > 0$ for $i = 1, \ldots, n$. Let $B = B_1 \times \cdots \times B_n \subseteq D_1 \times \cdots \times D_n$ be a box in $D_1 \times \cdots \times D_n$ and suppose that, for $i = 1, \cdots, n$ and all $t \in E$,*

$$\begin{aligned} sup(G_i(x,t)) \leq 0 \qquad & \text{if } x \in B_1 \times \ldots B_i{\downarrow_{V_i}} \cdots \times B_n \\ inf(G_i(x,t)) \geq 0 \qquad & \text{if } x \in B_1 \times \ldots B_i{\uparrow_{V_i}} \cdots \times B_n \end{aligned}$$

where V_i is the set of variables corresponding to $\mathbb{R}^{m_i}$ in the definition of G. If E is closed and convex, then the set-valued function $H : E \to \mathcal{P}(\mathbb{R}^{\sum_{i=1}^{n} m_i})$ defined by:

$$H(t) = \{x \in B \mid (\underbrace{0, \cdots, 0}_{n}) \in G(x,t)\}$$

is c-continuous. In particular, $H(t)$ is nonempty for all $t \in E$.

Proof 9 *Denote $B_i = [l_{i,1}, h_{i,1}] \times \cdots \times [l_{i,m_i}, h_{i,m_i}]$ and let $V_i = \{x_{i,j} \mid j = 1, \ldots, m_i\}$ for $i = 1, \ldots, n$.*

Define the function $G' : D'_1 \times \cdots \times D'_n \times E \subseteq \mathbb{R}^{m_1} \times \cdots \times \mathbb{R}^{m_n} \times \mathbb{R}^p \to \mathcal{P}(\mathbb{R}^n)$ such that

$$G'(x_{1,1}, \ldots, x_{n,m_n}, t) = G(x_{1,1} + l_{1,1}, \ldots, x_{n,m_n} + l_{n,m_n}, t)$$

where $D'_i = \{(r_1 - l_{i,1}, \ldots, r_{m_i} - l) \mid (r_1, \ldots, r_{m_i}) \in D_i\}$ *for* $i = 1, \ldots, n$. *Since* $G(x)$ *is c-continuous in* $D_1 \times \cdots \times D_n \times E$, $G'(x)$ *is c-continuous in* $D'_1 \times \cdots \times D'_n \times E$.

Consider $B' = B'_1 \times \cdots \times B'_n \subseteq D'_1 \times \cdots \times D'_n$ *such that* $B'_i = [0, h_{i,1} - l_{i,1}] \times \cdots \times [0, h_{i,1} - l_{i,1}] \times \cdots \times [0, h_{i,m_i} - l_{i,m_i}]$ *for* $i = 1, \ldots, n$. *From the hypothesis of the theorem, we can derive that for* $i = 1, \ldots, n$ *and all* $t \in E$,

$$\begin{array}{ll} sup(G'_i(x,t)) \leq 0 & \text{if } x \in B'_1 \times \ldots B'_i{\downarrow_{V_i}} \cdots \times B'_n \\ inf(G_i(x,t)) \geq 0 & \text{if } x \in B'_1 \times \ldots B'_i{\uparrow_{V_i}} \cdots \times B'_n. \end{array}$$

Consider another function $G'' : D''_1 \times \cdots \times D''_n \times E \subseteq \mathbb{R}^n \times \mathbb{R}^p \to \mathcal{P}(\mathbb{R}^n)$ *defined by*

$$G''(t_1, \ldots, t_n, t) = G'(B'_1(1)t_1, \ldots, B'_1(m_1)t_1, \ldots, B'_n(m_n)t_n, t)$$

where $D''_i = \{\sum_{j=0}^{m_i} r_j \mid (r_1, \ldots, r_{m_i}) \in D'_i\}$ *for* $i = 1, \ldots, n$.

Let $B'' = B''_1 \times \cdots \times B''_n \subseteq D''_1 \times \cdots \times D''_n$ *such that* $B''_i = [0, \sum_{j=1}^{m_i}(h_{i,j} - l_{i,j})]$ *for* $i = 1, \ldots, n$. *By the construction of* G'', *we have* $i = 1, \ldots, n$ *and all* $t \in E$,

$$\begin{array}{ll} sup(G''_i(x,t)) \leq 0 & \text{if } x = (r_1, \cdots, r_n) \in B'' \text{ and } r_i = 0 \\ inf(G''_i(x,t)) \geq 0 & \text{if } x = (r_1, \cdots, r_n) \in B'' \text{ and } r_i = \sum_{j=1}^{m_i}(h_{i,j} - l_{i,j}). \end{array}$$

Since E *is closed and convex, Theorem 6 asserts that the set-valued function* $H'' : E \to \mathcal{P}(\mathbb{R}^n)$ *defined by:*

$$H''(t) = \{(t_1, \ldots, t_n) \in B'' \mid (\underbrace{0, \cdots, 0}_{n}) \in G''(t_1, \ldots, t_n, t)\}$$

is c-continuous. In particular, $H''(t)$ *is nonempty for all* $t \in E$. *Since the function* $H_0 : B'' \subseteq \mathbb{R}^n \to \mathcal{P}(\mathbb{R}^{\sum_{i=1}^n m_i})$ *defined by*

$$H_0(t_1, \ldots, t_n) = \{(B'_1(1)t_1 + l_{1,1}, \ldots, B'_1(m_1)t_1 + l_{1,m_1}, \ldots, B'_n(m_n)t_n + l_{n,m_n})\}$$

is c-continuous, the function composition $H_1 : E \to \mathcal{P}(\mathbb{R}^{\sum_{i=1}^n m_i})$ *such that*

$$H_1(t) = H_0(H''(t)) := \cup\{H_0(\tilde{t}) \mid \tilde{t} \in H''(t)\}$$

is also c-continuous. Since $H_1(t) \subseteq H(t)$ *for every* $t \in E$, $H(t)$ *is also c-continuous and clearly non-empty.* *(Q.E.D.)*

Example 39 *Consider the function* $G : D_1 \times D_2 \times E \in \mathbb{R} \times \mathbb{R}^2 \times \mathbb{R} \to \mathcal{P}(\mathbb{R}^2)$ *defined by*

$$G(x, y, z, t) = \{(-x + z + 2, yz - xt)\}$$

where $D_1 = [9.75, 10]$, $D_2 = [4.875, 10.25] \times [7.875, 8]$, *and* $E = [4, 8]$. *Consider the box* $B = B_1 \times B_2$ *where* $B_1 = D_1$ *and* $B_2 = D_2$. *Since CI estimates the range of* $-x + z + 2$ *as* $[-0.25, -0.125]$ *in the box* $B_1{\downarrow_{\{x\}}} \times B_2 \times E$ *and as* $[0, 0.125]$ *in the box* $B_1{\uparrow_{\{x\}}} \times B_2 \times E$, *we have for for all* t *in* E,

$$\begin{array}{ll} sup(G_1(x',t)) \leq 0 & \text{if } x' \in B_1{\downarrow_{\{x\}}} \times B_2 \\ inf(G_1(x',t)) \geq 0 & \text{if } x' \in B_1{\uparrow_{\{x\}}} \times B_2. \end{array}$$

Similarly, CI estimates the range of $yz - xt$ *as* $[-15.619327, -13.917259]$ *in the box* $B_1 \times B_2{\downarrow}_{\{y,z\}} \times E$ *and as* $[27.990048, 29.692116]$ *in the box* $B_1 \times B_2{\uparrow}_{\{y,z\}} \times E$,

$$\begin{array}{ll} sup(G_2(x',t)) \leq 0 & \text{if } x' \in B_1 \times B_2{\downarrow}_{\{y,z\}} \\ inf(G_2(x',t)) \geq 0 & \text{if } x' \in B_1 \times B_2{\uparrow}_{\{y,z\}}. \end{array}$$

Theorem 7 asserts that the function $H : E \to \mathcal{P}(\mathbb{R}^3)$ *defined by:*

$$H(t) = \{x' \in B \mid (0,0) \in G(x',t)\}$$

is c-continuous and $H(t)$ *is nonempty for all* $t \in E$. *In other words, for all* $t \in E$, *there exists at least one* $x' \in B$ *such that* $(-x + z + 2, yz - xt) = (0,0)$.

Let us illustrate how the proof of Theorem 7 works by continuing the previous example.

Example 40 *The function* $G' : D_1' \times D_2' \times E \to \mathcal{P}(\mathbb{R}^2)$ *in the proof for this example is defined by*

$$G'(x,y,z,t) = \{(-(x + 9.75) + (z + 7.875) + 2, (y + 4.875)(z + 7.875) - (x + 9.75)t)\}$$

where $D_1' = [0, 0.25]$ *and* $D_2' = [0, 5.375] \times [0, 0.125]$. *We have* $G'(x,y,z,t)$ *is c-continuous. The box* B' *in the proof for this example is* $D_1' \times D_2'$. *Since (see Example 39)*

$$\begin{array}{ll} sup(G_1(x',t)) \leq 0 & \text{if } x' \in B_1{\downarrow}_{\{x\}} \times B_2 \\ inf(G_1(x',t)) \geq 0 & \text{if } x' \in B_1{\uparrow}_{\{x\}} \times B_2, \end{array}$$

we have

$$\begin{array}{ll} sup(G_1'(x',t)) \leq 0 & \text{if } x' \in B_1'{\downarrow}_{\{x\}} \times B_2' \\ inf(G_1'(x',t)) \geq 0 & \text{if } x' \in B_1'{\uparrow}_{\{x\}} \times B_2'. \end{array}$$

Similarly,

$$\begin{array}{ll} sup(G_2'(x',t)) \leq 0 & \text{if } x' \in B_1' \times B_2'{\downarrow}_{\{y,z\}} \\ inf(G_2'(x',t)) \geq 0 & \text{if } x' \in B_1' \times B_2'{\uparrow}_{\{y,z\}}. \end{array}$$

Next, the function $G'' : D_1'' \times D_2'' \times E \subseteq \mathbb{R}^2 \times \mathbb{R} \to \mathcal{P}(\mathbb{R}^2)$ *is defined by*

$$G''(t_1, t_2, t) = G'(t_1, \frac{5.375}{5.5} t_2, \frac{0.125}{5.5} t_2, t)$$

where $D_1'' = [0, 0.25]$ *and* $D_2'' = [0, 5.5]$.

The box B'' *in the proof for this example is* $D_1'' \times D_2''$. *We have for all* $t \in E$, *and if* $r_2 \in [0, 5.5]$

$$sup(G_1''(0, r_2, t)) \qquad = sup(G_1'(0, \tfrac{5.375}{5.5} r_2, \tfrac{0.125}{5.5} r_2, t)) \leq 0.$$

In other words, for all $t \in E$,

$$sup(G_1''(r_1, r_2, t)) \leq 0 \qquad \text{if } x = (r_1, r_2) \in B'' \text{ and } r_1 = 0.$$

Similarly we have for all $t \in E$,

$$\begin{array}{ll} sup(G_1''(r_1, r_2, t)) \geq 0 & \text{if } x = (r_1, r_2) \in B'' \text{ and } r_1 = 0.25 \\ sup(G_2''(r_1, r_2, t)) \leq 0 & \text{if } x = (r_1, r_2) \in B'' \text{ and } r_2 = 0 \\ sup(G_2''(r_1, r_2, t)) \geq 0 & \text{if } x = (r_1, r_2) \in B'' \text{ and } r_2 = 5.5. \end{array}$$

By Theorem 6, the function $H'' : E \to \mathcal{P}(\mathbb{R}^2)$ defined by

$$H''(t) = \{(t_1, t_2) \in B'' \mid (0,0) \in G''(t_1, t_2, t)\}$$

is c-continuous and $H''(t)$ is nonempty for any $t \in E$. Since the function $H_0 : B'' \subseteq \mathbb{R}^n \to \mathcal{P}(\mathbb{R}^3)$ defined by

$$H_0(t_1, t_2) = \{(t_1 + 9.75, \frac{5.375}{5.5}t_2 + 4.875, \frac{0.125}{5.5}t_2 + 7.875)\}$$

is c-continuous, the function composition $H_1 : E \to \mathcal{P}(\mathbb{R}^3)$ defined by

$$\begin{aligned} H_1(t) = H_0(H''(t)) \qquad &:= \cup\{H_0(\tilde{t}) \mid \tilde{t} \in H''(t)\} \\ &= \{(t_1 + 9.75, \tfrac{5.375}{5.5}t_2 + 4.875, \tfrac{0.125}{5.5}t_2 + 7.875) \mid (t_1, t_2) \in B'' \\ &\quad \textit{and } (0,0) \in G''(t_1, t_2, t)\} \end{aligned}$$

is also c-continuous. Since $H_1(t) \subseteq H(t)$ for every $t \in E$, $H(t)$ is also c-continuous and non-empty.

Intuitively, Theorem 7 requires that the function G has negative values at lower bounds of considering variables and positive values at upper bounds. The following lemma extends the theorem to allow the function to have different signs on bounds of variables in order to apply the intermediate value theorem.

Lemma 8 *Let $G : D_1 \times \cdots \times D_n \times E \subseteq \mathbb{R}^{m_1} \times \cdots \times \mathbb{R}^{m_n} \times \mathbb{R}^p \to \mathcal{P}(\mathbb{R}^n)$ be c-continuous in $D_1 \times \cdots \times D_n \times E$ where $m_i > 0$ for $i = 1, \ldots, n$. Let $B = B_1 \times \cdots \times B_n \subseteq D_1 \times \cdots \times D_n$ be a box in $D_1 \times \cdots \times D_n$ and suppose that, for $i = 1, \cdots, n$ and all $t \in E$,*

$$\begin{aligned} &sup(G_i(x,t)) \leq 0 \qquad \textit{if } x \in B_1 \times \ldots B_i{\downarrow_{V_i}} \cdots \times B_n \\ &inf(G_i(x,t)) \geq 0 \qquad \textit{if } x \in B_1 \times \ldots B_i{\uparrow_{V_i}} \cdots \times B_n \end{aligned}$$

or,

$$\begin{aligned} &sup(G_i(x,t)) \geq 0 \qquad \textit{if } x \in B_1 \times \ldots B_i{\downarrow_{V_i}} \cdots \times B_n \\ &inf(G_i(x,t)) \leq 0 \qquad \textit{if } x \in B_1 \times \ldots B_i{\uparrow_{V_i}} \cdots \times B_n \end{aligned}$$

where V_i is the set of variables corresponding to $\mathbb{R}^{m_i}$ in the definition of G. If E is closed and convex, then the set-valued function $H : E \to \mathcal{P}(\mathbb{R}^{\sum_{i=1}^n m_i})$ defined by:

$$H(t) = \{x \in B \mid (\underbrace{0, \cdots, 0}_{n}) \in G(x,t)\}$$

is c-continuous. In particular, $H(t)$ is nonempty for all $t \in E$.

Proof 10 *Define the new function $G' : D_1 \times \cdots \times D_n \times E \subseteq \mathbb{R}^{m_1} \times \cdots \times \mathbb{R}^{m_n} \times \mathbb{R}^p \to \mathcal{P}(\mathbb{R}^n)$ such that for all $i = 1, \ldots, n$, and all $t \in E$, if*

$$\begin{aligned} &sup(G_i(x,t)) \leq 0 \qquad \textit{if } x \in B_1 \times \ldots B_i{\downarrow_{V_i}} \cdots \times B_n \\ &inf(G_i(x,t)) \geq 0 \qquad \textit{if } x \in B_1 \times \ldots B_i{\uparrow_{V_i}} \cdots \times B_n \end{aligned}$$

then $G'_i(x,t) = G_i(x,t)$, otherwise $G'_i(x,t) = -G_i(x,t)$. Then, G' is also c-continuous. Theorem 7 now can be applied to prove the lemma.

7.2 Combining Interval Arithmetic, Testing, and the IVT to Show Satisfiability of Combinations of Inequalities and Equations

This section proposes a combination of interval arithmetic, testing, and the IVT aiming at showing satisfiability combinations of inequalities and equations. Our idea is based on Lemma 8 where if $g_1, \ldots, g_m$ are polynomials, then the function $G : R^n \to \mathcal{P}(\mathbb{R}^m)$ defined by $G(x) = \{(g_1(x), \ldots, g_m(x)\}$ is c-continuous.

First, we simplify the conditions in Lemma 8 by defining the notion of *check basis* for equations $\bigwedge_{j=1}^{m} g_j = 0$, to describe support for a satisfiable assignment of $\bigwedge_{j=1}^{m} g_j = 0$.

Definition 28 *Let $\bigwedge_{j=1}^{m} g_j = 0$ be a conjunction of equations over V. A sequence $(V_1, \cdots, V_m)$ is a* check basis *of $(g_1, \cdots, g_m)$ in B, if, for each $j, j' \leq m$,*

1. *$\emptyset \neq V_j \subseteq var(g_j)$,*
2. *$V_j \cap V_{j'} = \emptyset$ if $j \neq j'$, and*
3. *either $g_j \leq 0$ on $B{\uparrow}_{V_j}$ and $g_j \geq 0$ on $B{\downarrow}_{V_j}$, or $g_j \leq 0$ on $B{\uparrow}_{V_j}$ and $g_j \geq 0$ on $B{\downarrow}_{V_j}$.*

Example 41 *Consider a conjunction of equations $g_1 = 0 \wedge g_2 = 0$ where $g_1 = x - z - 2$, and $g_2 = yz - xt$, a box $B = [9.75, 10] \times [4.875, 10.25] \times [7.875, 8] \times [4, 8]$ for variables sequence (x, y, z, t). Let us examine if $(\{x\}, \{y, z\})$ is a check basis of (g_1, g_2) or not. The first two conditions in Def. 28 are satisfied. The third one is also satisfied since*

$$G_1(B{\uparrow}_{\{x\}}) = [-0.25, -0.125] \text{ implies that } g_1 \leq 0 \text{ on } B{\uparrow}_{\{x\}},$$

$$G_1(B{\downarrow}_{\{x\}}) = [0, 0.125] \text{ implies that } g_1 \geq 0 \text{ on } B{\downarrow}_{\{x\}},$$

$$G_2(B{\uparrow}_{\{y,z\}}) = [27.990048, 29.692116] \text{ implies that } g_2 \geq 0 \text{ on } B{\uparrow}_{\{y,z\}},$$

$$G_2(B{\downarrow}_{\{y,z\}}) = [-15.619327, -13.917259] \text{ implies that } g_2 \leq 0 \text{ on } B{\downarrow}_{\{y,z\}}.$$

Because of the first condition, the existence of a check basis requires that the number of variables in V must be greater than or equal to the number of equations m.

Given a test case $\theta : V' \mapsto \mathbb{R}$ in B, we write

$$B|_\theta = \{(r_1, \cdots, r_n) \in B \mid r_i = \theta(x_i) \text{ if } x_i \in V'\}.$$

Example 42 *Given a box $B = [-3, 1] \times [-4, 2]$ for variables sequence (x, y) and a test case $\theta = \{(x, -2)\}$, then $B|_\theta = [\mathbf{-2}, \mathbf{-2}] \times [-4, 2]$.*

The next theorem is a corollary of Lemma 8 which combines interval arithmetic, testing, and the application of IVT to show the satisfiability of a combination of inequalities and equations.

Theorem 8 *For a conjunction φ of polynomial inequalities and equations*

$$\bigwedge_{j=1}^{m} g_j > 0 \wedge \bigwedge_{j=m+1}^{m'} g_j = 0$$

and $B = [l_1, h_1] \times \cdots [l_n, h_n]$, assume that the following holds.

1. *For a decomposition $\varphi_1 \wedge \varphi_2$ of $\bigwedge_{j=1}^{m} g_j > 0$, φ_1 is* IA-VALID *in B and φ_2 is Test-*SAT *in B with a test case θ_{φ_2} on the set $var(\varphi_2)$ and in B.*
2. *A* check basis *$(V_{m+1}, \cdots, V_{m'})$ over $V \setminus var(\varphi_2)$ of $(g_{m+1}, \cdots, g_{m'})$ in $B|_{\theta_{\varphi_2}}$ exists.*

Then, φ has a SAT *instance in B.*

Proof 11 *Consider the function $G : D_1 \times \cdots \times D_{m'-m} \times E \subseteq \mathbb{R}^{|V_{m+1}|} \times \cdots \times \mathbb{R}^{|V_{m'}|} \times \mathbb{R}^{n-m'+m} \to \mathbb{R}^{m'-m}$ defined by $G(x) = \{(g_{m+1}, \ldots, g_{m'})\}$ with the assumption that variables are in an appropriate order and $D_1 \times \cdots \times D_{m'-m} \times E = B$. With the existence of a check basis, Lemma 8 asserts that the function $H : E \to \mathcal{P}(\mathbb{R}^{m'-m})$ defined by*

$$\begin{aligned} H(t) &= \{x \in D_1 \times \cdots \times D_{m'-m} \mid (\underbrace{0, \cdots, 0}_{m'-m}) \in G(x,t)\} \\ &= \{x \in D_1 \times \cdots \times D_{m'-m} \mid \bigwedge_{j=m+1}^{m'} g_j(x,t) = 0\} \end{aligned}$$

is nonempty for any $t \in E$. By taking any point $t_0 \in E|_{\theta_{\varphi_2}}$, we have the set $H(t_0) = \{x \in D_1 \times \cdots \times D_{m'-m} \mid \bigwedge_{j=m+1}^{m'} g_j = 0\}$ is nonempty. Take $x_0 \in H(t_0)$. Consider the point $r = (x_0, t_0)$ which is inside B. The conjunction φ is satisfiable at r because of the following reasons.

- *Since φ_1 is* IA-VALID *in B and $r \in B$, it is satisfiable at r (Lemma 1).*
- *Since φ_2 is Test-*SAT *with the test case θ_{φ_2}, it is satisfiable at point t_0 and thus at point r.*
- *By the construction of x_0, t_0, we have $\bigwedge_{j=m+1}^{m'} g_j(x_0, t_0) = 0$.*

(Q.E.D.)

Example 43 illustrates Theorem 8 for $V = \{x, y\}$ with $m = 0$ and $m' = n = 2$.

Example 43 *(Fig. 7.3) Let $g_1(x, y) = 0$ and $g_2(x, y) = 0$ be two equations. Assume that there exists a box $B = [c_1, d_1] \times [c_2, d_2]$ such that:*

- *$g_1(c_1, y) < 0$ for all $y \in [c_2, d_2]$, $g_1(d_1, y) > 0$ for all $y \in [c_2, d_2]$, and*
- *$g_2(x, c_2) < 0$ for all $x \in [c_1, d_1]$, $g_2(x, d_2) > 0$ for all $x \in [c_1, d_1]$.*

Thus, $g_1(x, y) = 0$ and $g_2(x, y) = 0$ share a common root in B.

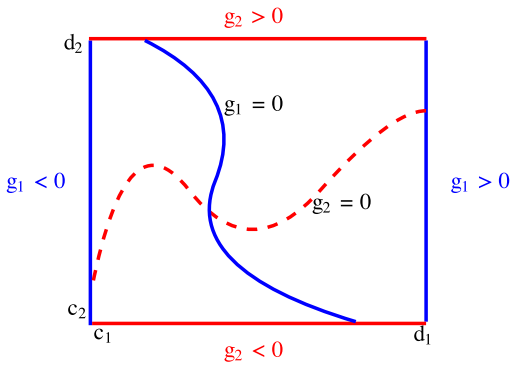


Figure 7.3: Example of applying IVT

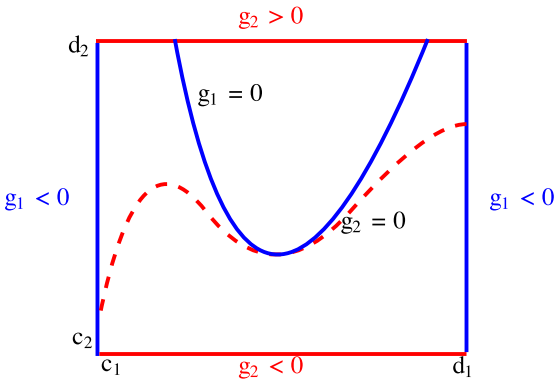


Figure 7.4: Failure of applying IVT

There are two limitations to apply Theorem 8.

- The number of variables (dimensions) must be greater than or equal to the number of equations.
- Trajectories of equations must cross. For instance, even with iterative box decompositions, there is no hope of detecting satisfiability if two equations $g_1 = 0$ and $g_2 = 0$ are touching (Fig. 7.4).

The complexity of finding a check basis is given below.

Lemma 9 *For m polynomials $(g_1, \cdots, g_m)$, a set V of n variables, and a box B, the complexity to find a check basis in B is bounded by $S(n,m)m!T(B)$, where $S(n,m)$ is the Stirling number of the second kind [67] and $T(B)$ is the complexity of verifying whether a sequence $(V_1, \cdots, V_m)$ is a check basis of $(g_1, \cdots, g_m)$ over B.*

Proof 12 *Finding a check basis can be done by enumerating all possible candidates and verifying each of them. The candidates are formed as follows.*

- *Partition n variables into m non-empty subsets. The number of ways to partition is the Stirling number of the second kind [67].*
- *For each partition, check whether each of its permutations is a check basis. There are $m!$ permutations for each partition.* *(Q.E.D.)*

Since the third condition in Definition 28 is the interpretation of a quantified formula over reals, a complete method to check whether $(V_1, \cdots, V_m)$ is a check basis of $(g_1, \ldots, g_m)$ is doubly exponential [19] which is the upper bound of $T(B)$. We here propose an incomplete but terminating and efficient method using interval arithmetic for verifying a check basis.

The *intermediate value theorem* (IVT) detects satisfiability of a single equation $g(x) = 0$ if there exist t_1 and t_2 such that $g(t_1) > 0$ and $g(t_2) < 0$. Then, $g = 0$ holds in between. For multi-variant equations, we apply the *generalized IVT* [58, Theorem 5.3.7], and its usage is summarized in Theorem 8.

We borrow notations φ, φ_1, φ_2, θ_{φ_2}, and B from Theorem 8.

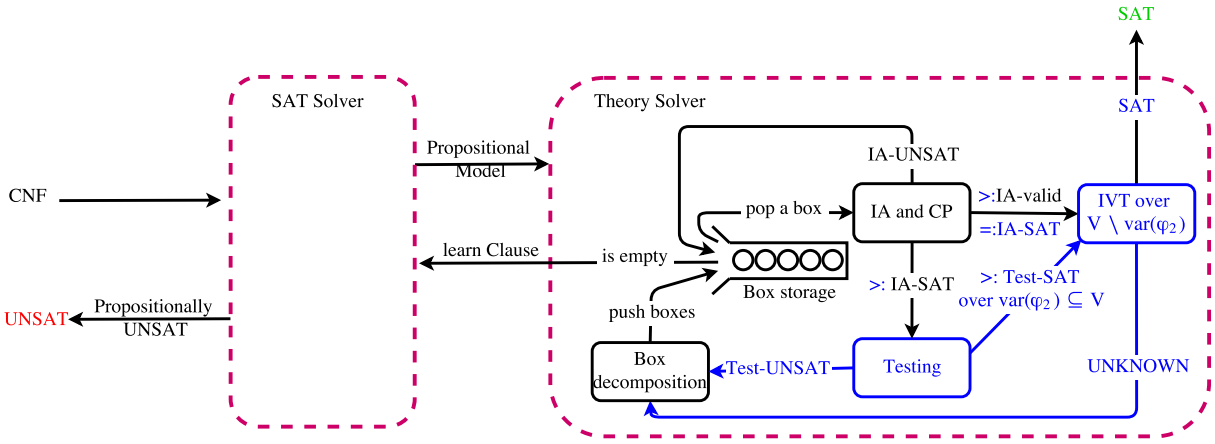


Figure 7.5: raSAT loop with the application of the IVT

Fig. 7.5 illustrates the application of Theorem 8, which is added to the **raSAT** loop. The label "$>$: IA-VALID" means that the conjunction of inequalities in φ is IA-VALID. This is similarly the case for "$=$: IA-SAT" and "$>$: Test-SAT".

The label "$>$: Test-SAT over $var(\varphi_2) \subseteq V$" means that a test case on $var(\varphi_2)$ concludes the Test-SAT of φ_2 and the generalized IVT is applied over $V \setminus var(\varphi_2)$ in the box $B|_{\theta_{\varphi_2}}$ (described by "IVT over $V \setminus var(\varphi_2)$").

Algorithm 5 Solving multiple equations $\bigwedge_{i=1}^{m} g_i = 0$ for a box $B = [l_1, h_1] \times \cdots [l_n, h_n]$ on variables $x_1, \cdots, x_n$

```
1: EQUATIONSPROVER(⋀_{i=1}^{m} g_i = 0, B, var(φ2))
2: function EQUATIONSPROVER(⋀_{i=j}^{m} g_i = 0, B, V0)
3:     if j > m then                          ▷ All equations are checked
4:         return SAT
5:     end if
6:     for V_j ∈ P(var(g_j)) do               ▷ P(var(g_j)) is the power-set of var(g_j)
7:         if V_j ∩ V0 = ∅ and IVT(V_j, g_j, B) then
8:             V0 ← V0 ∪ V_j
9:             if EQUATIONSPROVER(⋀_{i=j+1}^{m} g_i = 0, B, V0) = SAT then
10:                return SAT
11:            end if
12:        end if
13:    end for
14:    return UNKNOWN
15: end function
```

Algorithm 5 explains the procedure in the box labeled "IVT over $V \setminus var(\varphi_2)$" in Fig. 7.5. For the set of equations $\bigwedge_{i=1}^{m} g_i = 0$, the function examines all the check basis

candidates in $V \setminus var(\varphi_2)$. The function call IVT$(V_j, g_j, B)$ at line 7, which is described in Algorithm 6, checks the third condition in the Definition 28. Since Algorithm 5 naïvely searches a check basis, its complexity reaches the upper bound mentioned in Lemma 9 with $T(B)$ is polynomial instead of doubly exponential. If one of two limitations to apply Theorem 8 mentioned in Sec. 7.1 occurs, Algorithm 5 terminates and returns UNKNOWN.

Algorithm 6 Showing the satisfiability of an equation $g = 0$ inside a box B by applying IVT at bounds of variables V

```
 1: function IVT(V, g, B)
 2:     B_1 ← B↑_V
 3:     B_2 ← B↓_V
 4:     if g(B_1) ≤ 0 ∧ g(B_2) ≥ 0 then
 5:         return TRUE
 6:     end if
 7:     if g(B_1) ≥ 0 ∧ g(B_2) ≤ 0 then
 8:         return TRUE
 9:     end if
10:     return FALSE
11: end function
```

Example 44 *Suppose* φ *is* $g_1 > 0 \wedge g_2 = 0 \wedge g_3 = 0$*, where* $g_1 = cd - d, g_2 = a - c - 2$*, and* $g_3 = bc - ad - 2$*. Initially the box storage consists of a single box* $B = [-2, 3.5] \times [-5, 0] \times [0, 1.5] \times [-5, -0.5]$ *for* (a, b, c, d)*.*

Fig. 7.6 shows the flow for the **raSAT** *loop with the application of the generalized IVT, where a label* $[...], B$ *is a pair of a box storage and a currently exploring box* B*, and* θ *denotes a test case. The interval contraction and the constraint propagation reduce* B*,* B_1*, and* B_3 *to* B'*,* B'_1*, and* B'_3*, respectively.*

We end this section by a pseudo-code for the addition of the IVT into RASATLOOP(φ) in the Algorithm 7.

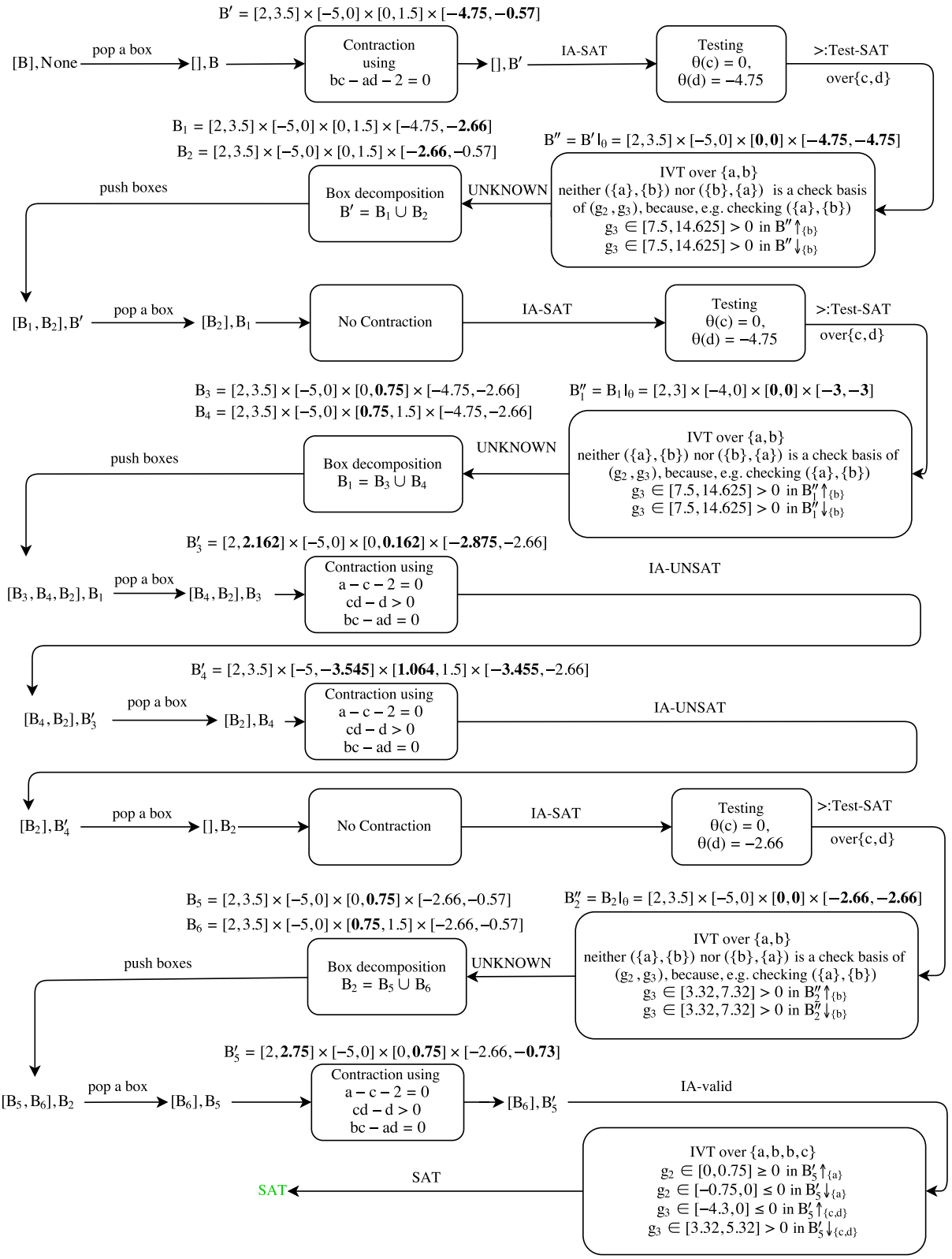


Figure 7.6: An example execution of raSAT

Algorithm 7 Application of the IVT into **raSAT** loop for a set of polynomial constraints φ

function RASATLOOP-IVT(φ)
  $S \leftarrow \{\,]-\infty, \infty[^n\}$
  while $S \neq \emptyset$ **do**
    choose $B \in S$
    $S \leftarrow S \setminus \{B\}$
    IA-RESULT, $B' \leftarrow$ IA_CP(φ, B)
    if IA-RESULT = IA-UNSAT **then**
      continue
    else if IA-RESULT = IA-VALID **then**
      return SAT
    end if
    $\varphi_{\text{IA-SAT}} \leftarrow \{\varphi_{\text{APC}} \mid \varphi_{\text{APC}} \in \varphi$ and φ_{APC} is IA-SAT in $B'\}$ ▷ exclude IA-VALID APCs
    TEST-RESULT $\leftarrow$ TESTING($\varphi_{\text{IA-SAT}}, B', \emptyset$)
    if TEST-RESULT =Test-SAT **then**
      return SAT
    end if
    $\varphi_2 \leftarrow \{\varphi_{\text{APC}} | \varphi_{\text{APC}} \in \varphi_{\text{IA-SAT}}$ and φ_{APC} is Test-SAT$\}$
    $\varphi_3 \leftarrow \varphi_{\text{IA-SAT}} \setminus \varphi_2$
    if φ_3 are equations and EQUATIONSPROVER($\varphi_3, B', var(\varphi_2)$) = SAT **then**
      return SAT
    end if
    $B_1, B_2 \leftarrow$ decompose B'
    $S \leftarrow S \cup \{B_1, B_2\}$
  end while
  return UNSAT
end function

Chapter 8

Combining Procedures

Our idea is to combine efficient incomplete methods with a complete procedure to produce an efficient complete framework for solving non-linear arithmetic. As discussed in Chapter 3, CAD and its variants provide a complete algorithm for solving polynomial constraints. In our combination, re-implementing CAD would require lots of efforts and expertise. As a result, we would like to reuse the existing implementation, specifically the one provided by Redlog/Reduce.

8.1 Utilizing Redlog/Reduce for Completeness

As a complete method, we use Quantifier Elimination techniques (CAD as the core with virtual substitution as a heuristic) and utilize a computer algebra system, namely Reduce, and more precisely, the package Redlog [21] which implements interpreted first-order logic on top of computer algebra. Besides many other domains, Redlog provides decision procedures for real closed fields. For real arithmetic, Redlog combines two quantifier elimination techniques, virtual substitution [87, 88, 47] with partial cylindrical algebraic decomposition [13, 70]. For a survey of real applications of Redlog, see [77]. Further theories supported by Redlog include discretely valued fields [75], term algebras [78], QBF [71, 79], Presburger Arithmetic [49] and fragments of non-linear integer arithmetic [48].

To embed a decision procedure in SMT, one property is mandatory: the procedure should feature small unsat core production. From an unsatisfiable set of APCs, it should produce an unsatisfiable subset containing few necessary APCs, optimally a minimal unsatisfiable subset, i.e. such that all proper subsets are satisfiable. With small unsat core production, the SMT infrastructure goes considerably beyond an SAT solver enumerating all possible models of the Boolean abstraction of the input formula, the theory reasoner refuting them one at a time. Recently a method using linear optimization to compute conflict sets for cylindrical algebraic decomposition and virtual substitution was presented in [42] . Virtual substitution and cylindrical algebraic decomposition share the same basic idea of finding a finite set of test points that suffice to determine the unsatisfiability of a set of constraints S. If the set S is unsatisfiable, each of these test points falsifies at least one constraint in S. Finding a conflict set reduces to finding a subset of the constraints that contains, for each test point, at least one unsatisfied constraint. The idea was implemented in the Quantifier Elimination algorithms within Redlog. Redlog/Reduce now furthermore provides an interface for SMT solver so that the software can be used

as a theory reasoner with little effort. This reasoner is used in our portfolio of tools and guarantees the completeness of the combination of reasoners on real closed fields.

8.2 Lazy Combination

We have previously discussed subtropical satisfiability, extensions of interval constraint propagation, and quantifier elimination. Our aim is to combine these in a complete and efficient framework for solving polynomial constraints. Recall that subtropical satisfiability is only an incomplete heuristic, and the ICP does not even guarantee termination; it might loop forever, for instance in the case of touching spheres. Combining the ICP sequentially with other procedures thus requires heuristics for its termination to ensure fairness and to let the other algorithms also work on the constraints. Before decomposing a box into smaller ones (line 12 in Algorithm 2), our algorithm will check if bounded boxes have been generated, all the bounded boxes are smaller than a chosen value ε in all dimensions. If so, the algorithm gives up and returns UNKNOWN along with the box resulting from the contraction of $]-\infty,\infty[^n$ from constraints; the algorithm returns the empty box in case of unsatisfiability and the valid box in case of satisfiability. We furthermore require that boxes are handled in a chronological order, that is, S in Algorithm 2 becomes a queue, where the first box (the oldest one) is chosen and removed, and newer boxes are added at the end. Boxes are decomposed only along axes with lengths greater than ε. The procedure ensures unbounded boxes are not decomposed infinitely. The algorithm terminates either when it detects satisfiability (or unsatisfiability) or all bounded boxes have all dimensions smaller than ε.

A lazy combining approach considers the procedures as black boxes and invokes them sequentially to check the satisfiability of the constraints. The fastest procedures to terminate are ordered first, and the complete procedure is called last. In Algorithm 8, STROPSAT(φ) returns SAT if and only if subtropical satisfiability succeeds in finding a model for φ; remember that subtropical satisfiability is indeed essentially a model finding method. The complete decision procedure (in our case, quantifier elimination methods as implemented in Redlog/Reduce) is called in line 9, and returns either SAT (UNSAT) if the input is satisfiable (resp. unsatisfiable). Notice that, when calling the complete decision procedure, the set of constraints φ is complemented with a box B found by the ICP. Actually (this is not shown in Algorithm 8), φ is furthermore cleaned of the constraints that are valid in the box B.

Algorithm 8 Lazy combination of procedures

```
 1: function LAZY(φ)
 2:     if SUBTROP(φ) = SAT then
 3:         return SAT
 4:     end if
 5:     (result, B) ← RASATLOOP-IVT(φ)
 6:     if result ≠ UNKNOWN then
 7:         return result
 8:     end if
 9:     return reduce(φ ∧ B)
10: end function
```

8.3 Less Lazy Combination

Instead of making a sequential combination of ICP and the complete framework, it is able to make the integration tighter. Algorithm 9 illustrates such a less lazy combination.

Algorithm 9 Less lazy combination of procedures

1: **function** LESS_LAZY(φ, ϵ)
2: **if** STROPSAT(φ) **then**
3: **return** SAT
4: **end if**
5: $S \leftarrow \{\,]-\infty, \infty[^n\}$
6: **while** $S \neq \emptyset$ **do**
7: **choose** $B \in S$
8: $S \leftarrow S \setminus \{B\}$
9: IA-RESULT, $B' \leftarrow$ IA_CP(φ, B)
10: **if** IA-RESULT = IA-UNSAT **then**
11: **continue**
12: **else if** IA-RESULT = IA-VALID **then**
13: **return** SAT
14: **end if**
15: $\varphi_{\text{IA-SAT}} \leftarrow \{\varphi_{\text{APC}} \mid \varphi_{\text{APC}} \in \varphi$ and φ_{APC} is IA-SAT in $B'\}$ ▷ exclude IA-VALID APCs
16: TEST-RESULT $\leftarrow$ TESTING($\varphi_{\text{IA-SAT}}, B', \emptyset$)
17: **if** TEST-RESULT =Test-SAT **then**
18: **return** SAT
19: **end if**
20: $\varphi_2 \leftarrow \{\varphi_{\text{APC}} | \varphi_{\text{APC}} \in \varphi_{\text{IA-SAT}}$ and φ_{APC} is Test-SAT$\}$
21: $\varphi_3 \leftarrow \varphi_{\text{IA-SAT}} \setminus \varphi_2$
22: **if** φ_3 are equations and EQUATIONSPROVER($\varphi_3, B', var(\varphi_2)$) = SAT **then**
23: **return** SAT
24: **end if**
25: **if** $|B'| < \epsilon$ **then**
26: COMPLETE-RESULT = COMPLETE($\varphi_{\text{IA-SAT}} \wedge B'$)
27: **if** COMPLETE-RESULT = SAT **then**
28: **return** SAT
29: **else**
30: Add unsat cores computed by COMPLETE($\varphi_{\text{IA-SAT}} \wedge B'$) to *unsat_cores*
31: **end if**
32: **else**
33: $B_1, B_2 \leftarrow$ decompose B'
34: $S \leftarrow S \cup \{B_1, B_2\}$
35: **end if**
36: **end while**
37: **return** UNSAT
38: **end function**

The combination is similar to ideas from [15] where when each box of ICP becomes

smaller than a threshold ϵ, the complete framework is provoked to solve the remaining unknown constraints over such a small box. The small box is also necessarily passed to the complete framework (at line 26 of Algorithm 9, the box B' is complemented to the IA-SAT (UNKNOWN) constraints $\varphi_{\text{IA-SAT}}$ before being passed to the complete framework). While the small box B' helps the complete framework to early prune their searching space, in the case of unsatisfiable boxes the complete framework has to exhaustively check each of those boxes which turns to be a bottleneck of the less lazy combination. When the complete framework detects unsatisfiability, we also ask it for the unsat core and add it to the global variable *unsat_cores* which we mentioned previously. Note that we only extract the APC from the returned unsat core by the complete framework since the cause (the APCs causing contraction) of the box B' is already stored in the variable *unsat_cores* by the IA_CP(_, _) algorithm.

Chapter 9

Experiments on SMT-LIB Benchmarks

Experiments in this chapter were done on SMT-LIB benchmarks which have had little changes over years. Since the development of tools happened in different years, the experiments were also done on different versions of the SMT-LIB. We explicitly state the version in each section.

9.1 Performance of raSAT

All experiments in this section were done on SMT-LIB 2015. The ideas in Chapters 5 and 7 were implemented as a theory solver. The implementation consists of 6000 lines of Ocaml code of which one-third account for implementing various kinds of interval arithmetic. The theory solver is combined with the SAT solver miniSAT to create the SMT solver **raSAT**.

Effects of Testing and Heuristics

This section clarifies the effects of testing and proposed heuristics in the **raSAT** framework. The experiments were executed over 192 MPI processes on a SGI UV3000 system. Each process was distributed one CPU core and ran one pair of solver-benchmark at a time. The CPU specification is Intel Xeon E5-4655v3. The timeout and the maximum memory for each task (a pair of solver-benchmark) were set to 2500 seconds and 8G of RAM respectively.

Table 9.1, Figure 9.1, and Figure 9.2 present experimental results which clarify effects of testing and proposed heuristics within **raSAT** framework.

raSAT with testing was generally faster than without testing in detecting satisfiability. Moreover, testing made **raSAT** detect 524 SAT benchmarks and 2 UNSAT ones that were not solved without testing. The 2 UNSAT benchmarks difference is due to testing, **raSAT** decomposed intervals of variables in Test-UNSAT APCs. On the other hand, without testing, it decomposed those of variables from IA-SAT APCs. Different box decompositions led to different results on the same benchmark. For the same reason, **raSAT** without testing solved 9 UNSAT benchmarks which were not solved by **raSAT**.

473 benchmarks out of above 524 ones contain equations. This indicates that testing

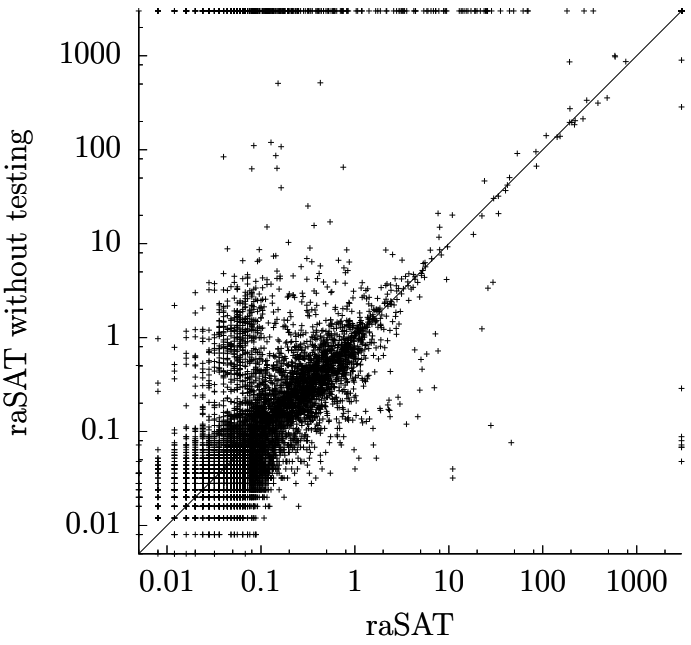


Figure 9.1: Comparing **raSAT** and **raSAT** without testing on QF_NRA

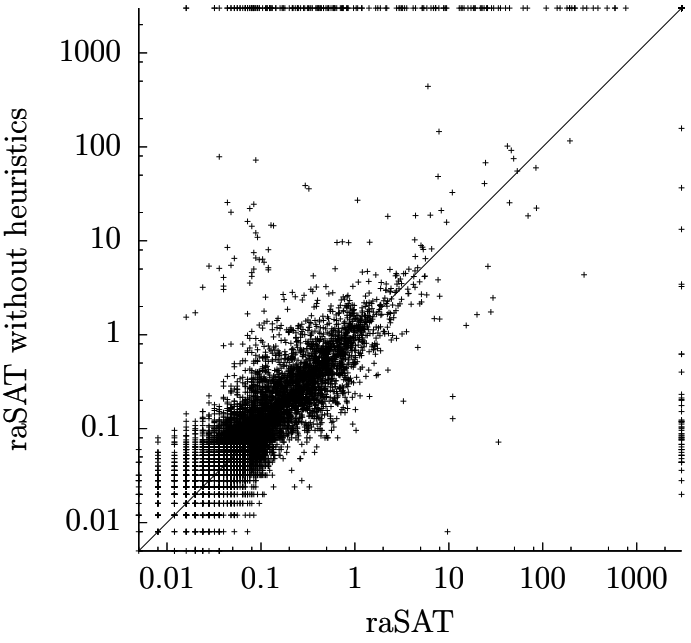


Figure 9.2: Comparing **raSAT** and **raSAT** without heuristics on QF_NRA

	raSAT without testing		**raSAT**	
Benchmarks	Number	Time(s)	Number	Time(s)
Commonly solved	8576	11136.76	8576	**7334.36**
Uniquely solved	9	1184.56	**526**	**1667.368**

	raSAT without heuristics		**raSAT**	
Benchmarks	Number	Time(s)	Number	Time(s)
Commonly solved	8924	3867.80	8924	**2843.58**
Uniquely solved	52	219.552	**178**	**6158.148**

Table 9.1: Effects of Testing and Heuristics on QF_NRA

improved satisfiability detection of not only inequalities but also combinations of inequalities and equations. Consider the following simple example.

Example 45 *Consider the constraint* $\varphi := x^2 - 2 < 0 \wedge xy = 0$ *and a box* $B := [-2, 3.5] \times [-5, 5]$. *First, "IA and CP" contracts* B *to* $B' = [-\sqrt{2}, \overline{\sqrt{2}}] \times [-5, 5]$ *and it fails to conclude that* $x^2 - 2 < 0$ *is* IA-VALID *in* B'. *As a result, if we only combine IA and the IVT, we cannot show satisfiability of* φ. *However, with testing to find the test case* $x = 1.23$ *we have* $x^2 - 2 < 0$ *is Test-*SAT *and then we can check the values of* xy *at the boundaries,*

- *for* $y = -5$, $xy = 1.23 * -5 = -6.15 < 0$, *and*
- *for* $y = 5$, $xy = 1.23 * 5 = 6.15 > 0$.

Here the IVT concludes that $xy = 0$ *has a solution* $(1.23, y_0)$ *in* B' *for some* $y_0 \in [-5, 5]$.

In this example, validity of $x^2 - 2 < 0$ can be achieved by iteratively decomposing $x \in [-\underline{\sqrt{2}}, \overline{\sqrt{2}}]$ to create a new interval, e.g. $x \in [0, 1]$, in which the APC is IA-VALID. However, in general, since Test-SAT is weaker than IA-VALID, testing essentially improves satisfiability detection on some combinations of inequalities and equations. It should be noted that Theorem 8 combines IA-VALID, Test-SAT, and IVT to show satisfiability of a combination between inequalities and equations.

In terms of effects of proposed heuristics, we found that they were especially effective on large benchmarks (large number of APCs and/or variables, e.g. benchmarks in QF_NRA/zankl/Matrix2∼5). In addition, the experiments showed that the incremental testing was useful in quickly detecting Test-USAT while it is still effective in finding satisfying assignments.

Comparison with other SMT Solvers

Our comparison involved two main views: (1) ICP-based solvers, e.g., **iSAT3** and **dReal**, and (2) other SMT-solvers from the 2016 SMT competition, viz., **Z3**, **SMT-RAT 2.0**, **Yices**, and **CVC4**, which can be downloaded from the execution Website[1] of the competition. They were compared on the division QF_NRA of the SMT-LIB benchmarks 2015-06-01[2] with a timeout of 2500 seconds (both CPU and wall time) on an Intel Xeon E5-4655v3 processor with 8GB of RAM. Note that:

[1]https://www.starexec.org/starexec/secure/explore/spaces.jsp?id=170422
[2]http://smtlib.cs.uiowa.edu/benchmarks.shtml

- **iSAT3** requires bounded intervals, and the bound was set to $[-1000, 1000]$. For other tools (including **raSAT**), it was set to $(-\infty, \infty)$.
- **dReal** decides δ-SAT, instead of SAT, which allows δ-deviation on the evaluation of polynomials for some $\delta > 0$. Generally, δ-SAT does not imply SAT and δ is set to 0.001 by default.

Table 9.2 lists the numbers of solved problems in each family of benchmarks in the QF_NRA category of SMT-LIB. The "Time" row indicates the cumulative running time of successful cases (SAT or UNSAT). In the "Family" column, the numbers of SAT/UNSAT problems are associated if there are already known. Here, "*" means δ-SAT.

Family	**raSAT**	**dReal**	**iSAT3**	**CVC4**	**SMT-RAT**	**Yices**	**Z3**
zankl (SAT)	36	53*	16	0	10	49	**53**
zankl (UNSAT)	10	0	12	0	15	**28**	23
meti-tarski (SAT) (3220)	3203	3220*	2774	0	3028	3194	**3220**
meti-tarski (UNSAT) (1526)	1139	1227	1242	245	1452	1496	**1522**
hong (UNSAT) (20)	**20**	**20**	**20**	0	**20**	9	9
LassoRanker (SAT)	0	0*	0	0	0	0	0
LassoRanker (UNSAT)	0	0	0	0	0	0	**2**
Total	4408	1247	4064	245	4525	4776	4829
Time(s)	22406.82	11740.17	1823.83	21.72	27222.17	10146.97	8289.79

Family	**raSAT**	**dReal**	**iSAT3**	**CVC4**	**SMT-RAT**	**Yices**	**Z3**
zankl (SAT) (11)	**11**	11*	0	0	**11**	**11**	**11**
zankl (UNSAT) (4)	**4**	**4**	**4**	1	**4**	**4**	**4**
meti-tarski (SAT) (1805)	1645	1805*	303	0	1765	**1805**	**1805**
meti-tarski (UNSAT) (1162)	1011	910	1121	346	1143	1159	**1162**
kissing (SAT) (42)	15	36*	0	0	9	10	**36**
kissing (UNSAT) (3)	0	**1**	0	0	0	0	0
hycomp (SAT)	0	256*	0	0	98	226	**256**
hycomp (UNSAT)	1940	2130	**2279**	2102	1651	2184	2209
LassoRanker (SAT)	0	145*	16	0	0	142	**145**
LassoRanker (UNSAT)	0	0	27	0	0	**249**	137
Total	4626	3045	3750	2449	4681	5790	5765
Time(s)	22319.20	31781.54	4522.84	319.71	34741.72	109771.08	48787.60

Table 9.2: Comparison of SMT solvers on SMT-LIB benchmark ($* = \delta$-SAT)

In the followings, we present the scatter plots indicating the tendency on the execution time on benchmarks between **raSAT** and other tools.

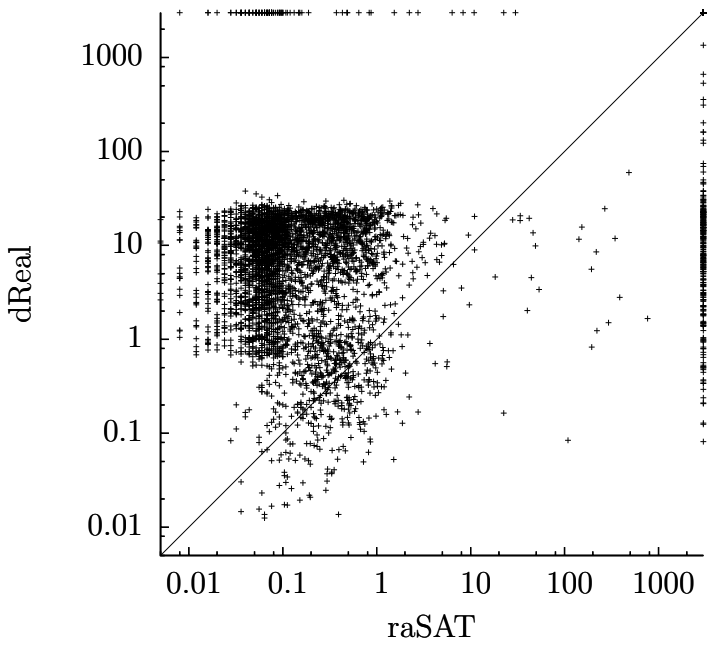


Figure 9.3: Comparing **raSAT** and **dReal** on QF_NRA

Figure 9.3 compares the performances of **raSAT** with **dReal** on unsatisfiable benchmarks since **dReal** does not answer SAT. While **dReal** uses **RealPaver** for ICP solving, **raSAT** implements its own version of ICP which allows combining different kinds of interval arithmetic and optimizations in interval arithmetic as mentioned previously. As a result, **raSAT** is expected to spend less time on interval arithmetic with a more refined approximation.

In terms of comparison with **iSAT3**, because **iSAT3** requires bounds on variables, we only compare on satisfiable benchmarks (the second sub-figure of Figure 9.4). Most of the time when both solvers solved the same benchmark, **iSAT3** was often faster than **raSAT** was. This is because **iSAT3** has an efficient mechanism for conflict analysis (which is discussed later in Chapter 10) to early prune boxes in which constraints are unsatisfiable, leading to a quick detection of satisfiability on relatively small benchmarks on which both solvers terminated after 0.1 seconds. There were a number of large benchmarks on which **raSAT** with the help of testing detected satisfiability quicker than **iSAT3** did.

Figures 9.5 to 9.8 illustrates that raSAT complements to other methodologies implemented in state-of-the-art solvers, and the running time of raSAT on solved problems is comparable with other solvers.

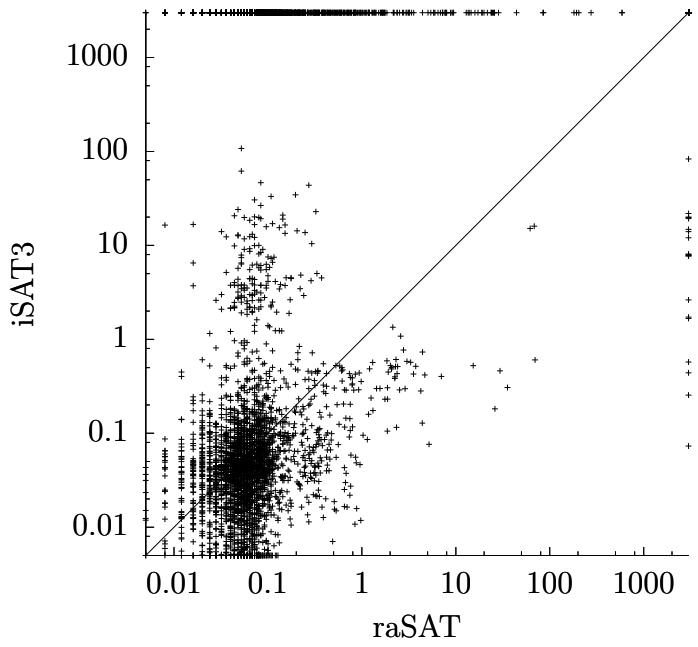


Figure 9.4: Comparing **raSAT** and **iSAT3** on QF_NRA

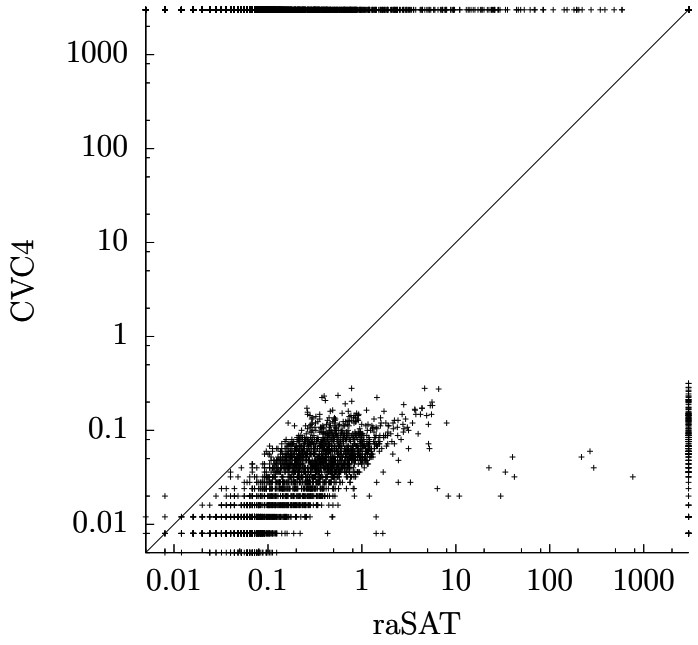


Figure 9.5: Comparing **raSAT** and **CVC4** on QF_NRA

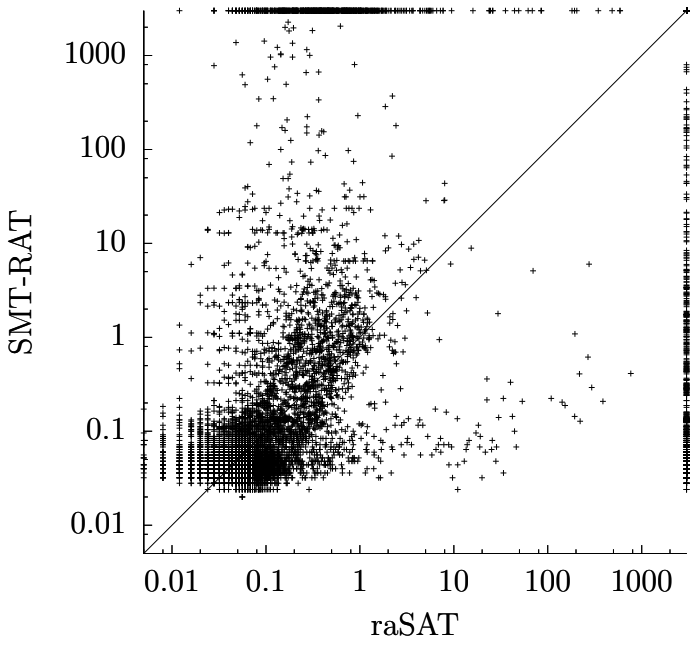


Figure 9.6: Comparing **raSAT** and **SMT-RAT** on QF_NRA

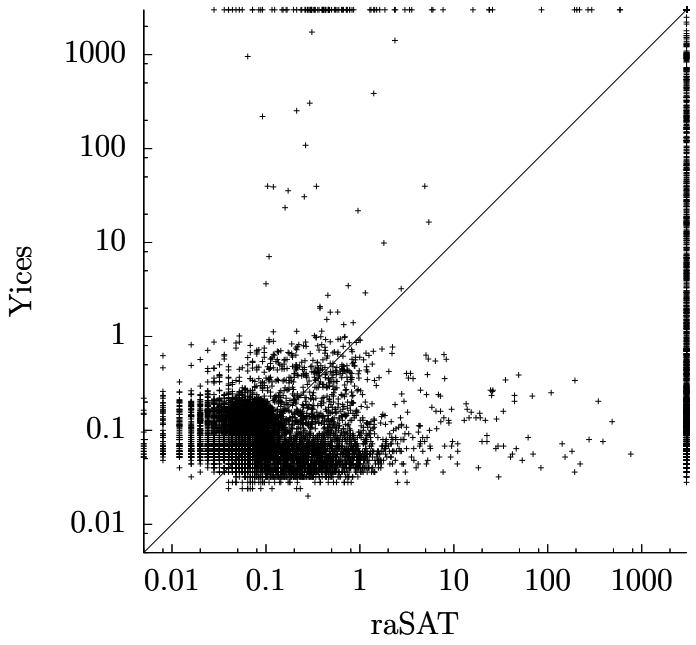


Figure 9.7: Comparing **raSAT** and **Yices** on QF_NRA

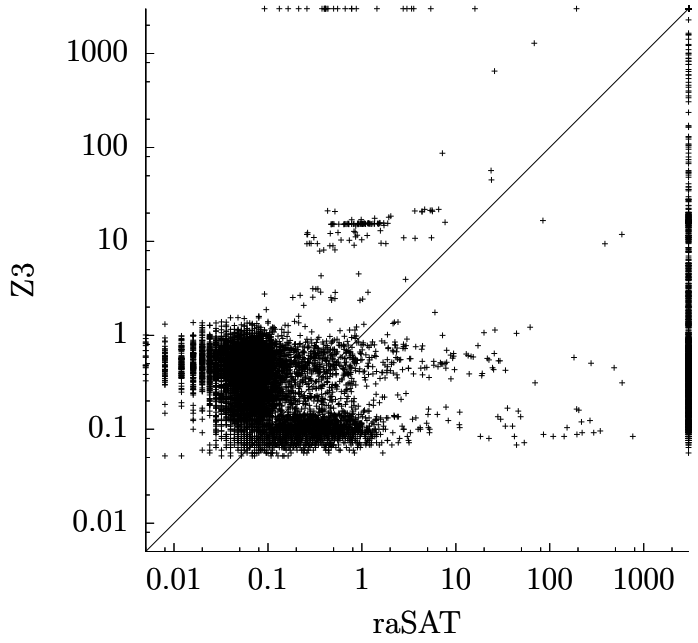


Figure 9.8: Comparing **raSAT** and **Z3** on QF_NRA

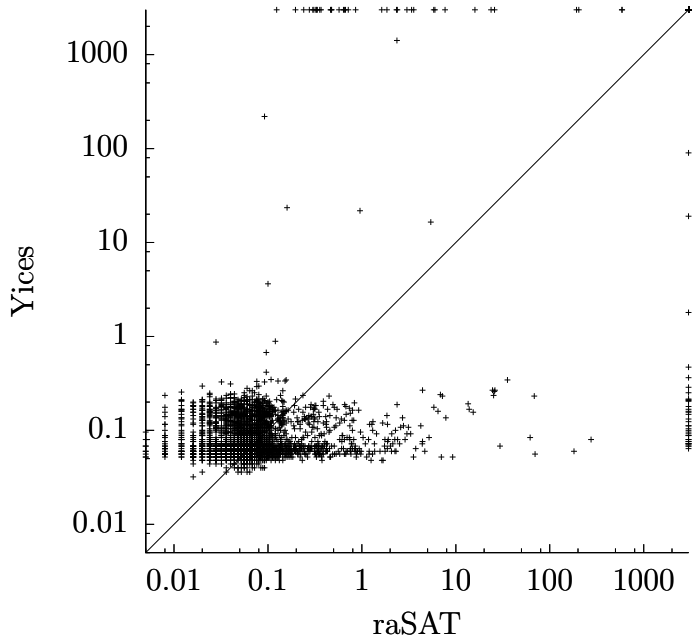


Figure 9.9: Comparing **raSAT** and **Yices** on satisfiable inequalities of QF_NRA

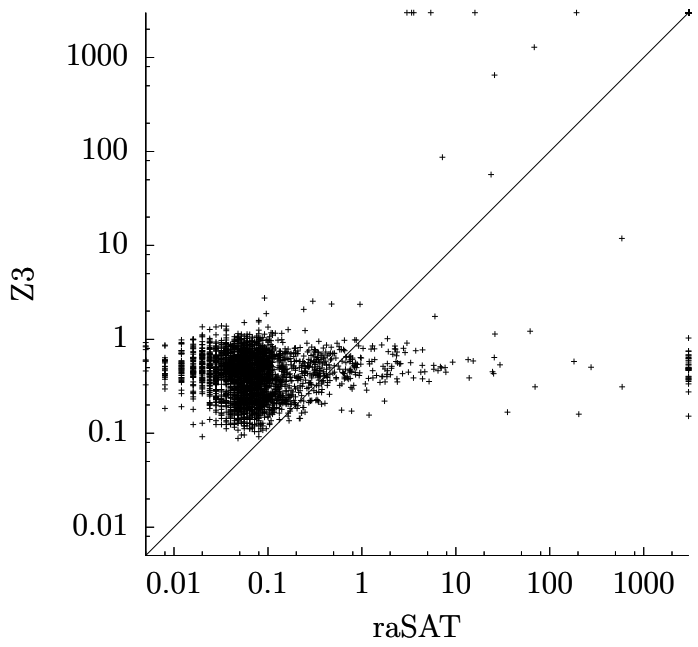


Figure 9.10: Comparing **raSAT** and **Z3** on satisfiable inequalities o QF_NRA

Figures 9.9 and 9.10 compares **raSAT** with CAD-based solvers, viz. **Yices** and **Z3** on satisfiable inequalities. It can be seen from the figure that **raSAT** was comparable to these two solvers, sometimes it even outperformed them.

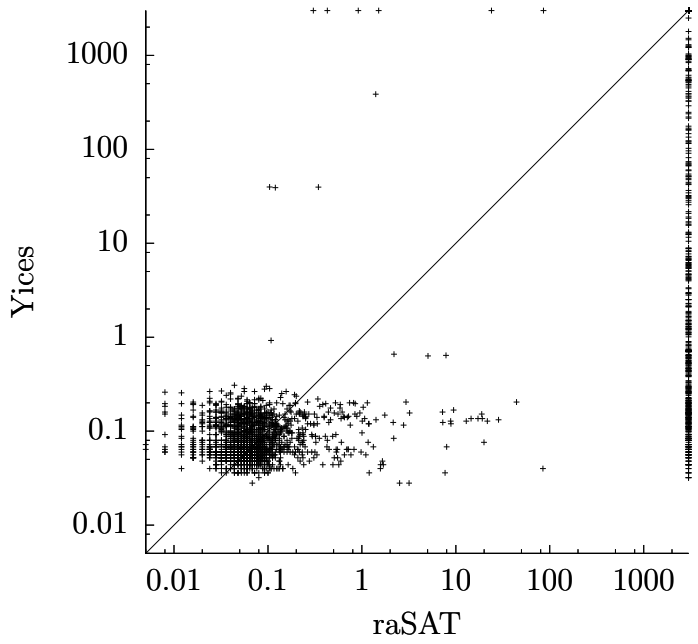


Figure 9.11: Comparing **raSAT** and **Yices** on satisfiable constraints containing equations of QF_NRA

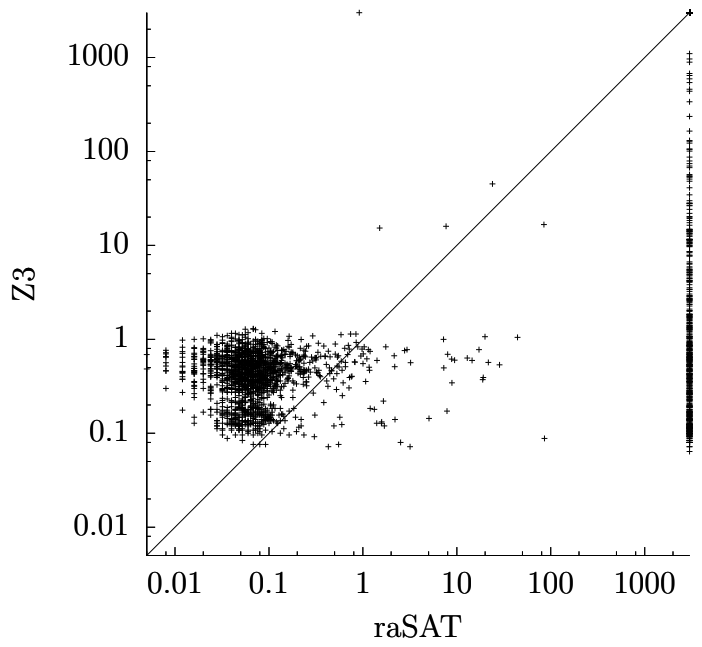


Figure 9.12: Comparing **raSAT** and **Z3** on satisfiable constraints containing equations of QF_NRA

Figures 9.11 and 9.12 compares **raSAT** with **Yices** and **Z3** on satisfiable constraints containing equations. Although **Yices** and **Z3** solved a larger number of benchmarks than **raSAT** did, **raSAT** was often faster when it solved a problem. Observing the benchmarks that were not solved by **raSAT**, it was often the case that there are many variables in the equations, which was a bottle-neck for finding a check basis when applying IVT. Heuristics for efficiently finding a check basis is one of our future works.

In addition to the completeness, efficient detection of UNSAT remains as an obstruction to **raSAT**. We will briefly address this as a future work in Chapter 11.

Unknown Problems in SMT-LIB

In the SMT-LIB, many problems are marked "*unknown*". Table 9.3 summarizes the number of solved problems by SMT solvers, categorized by inequalities and equations.

Unknown Inequalities	**raSAT**	**dReal**	**iSAT3**	**CVC4**	**SMT-RAT**	**Yices**	**Z3**
SAT	12	63*	4	0	3	11	**13**
UNSAT	10	0	12	0	15	**28**	25

Unknown Equations	**raSAT**	**dReal**	**iSAT3**	**CVC4**	**SMT-RAT**	**Yices**	**Z3**
SAT	0	72*	10	0	98	**302**	276
UNSAT	3	**524**	232	36	38	327	169

Table 9.3: Number of unknown problems solved by SMT solvers

For unknown problems, SAT results of **raSAT** are straightforward to justify. Because **raSAT** returns SAT with either small intervals or an assignment for the variables (or a

combination of both), the verification can be implemented by first adding the following two assertions to the original benchmark.

- For a small interval $x \in [l, h]$, add (assert (and (>= x l) (<= x h))).
- For a test value $x = r$, add (assert (= x r)).

Thereafter, we can run other SMT solvers against the newly generated benchmarks. **raSAT** showed SAT of 12 unknown problems which were verified by other SMT solvers as illustrated in the execution job[3] on starexec.org. Table 9.4 lists these 12 unknown problems, in which "Num of Vars", "Max Vars among APCs", and "Num of APCs" are acronyms for "Number of variables in the benchmark", "Maximum number of variables among APCs", and "Number of APCs in the benchmark" respectively.

Unknown Benchmarks	Num of Vars	Max Vars among APCs	Num of APCs	Also solved by
QF_NRA/zankl/matrix-1-all-15.smt2	10	7	28	SMT-RAT, Yices, Z3, iSAT3
QF_NRA/zankl/matrix-1-all-18.smt2	6	5	21	SMT-RAT, Yices, Z3, iSAT3
QF_NRA/zankl/matrix-1-all-24.smt2	11	5	29	SMT-RAT, Yices, Z3 , iSAT3
QF_NRA/zankl/matrix-2-all-11.smt2	17	16	36	Yices, iSAT3
QF_NRA/zankl/matrix-2-all-6.smt2	17	16	30	Z3, iSAT3
QF_NRA/zankl/matrix-4-all-2.smt2	119	42	275	Z3 , iSAT3
QF_NRA/zankl/matrix-4-all-3.smt2	139	73	257	iSAT3
QF_NRA/zankl/matrix-4-all-7.smt2	178	97	279	
QF_NRA/zankl/matrix-4-all-9.smt2	193	61	292	Yices, iSAT3
QF_NRA/zankl/matrix-5-all-1.smt2	132	47	284	
QF_NRA/zankl/matrix-5-all-2.smt2	173	57	388	Z3 , iSAT3
QF_NRA/zankl/matrix-5-all-5.smt2	173	57	394	Z3 , iSAT3

Table 9.4: Unknown benchmarks that have been proven to be SAT by **raSAT**

9.2 Performance of STROPSAT

A library STROPSAT implementing Subtropical Satisfiability, is available on our web page[4] which consists of 2000 lines of C code. It should be noted that the idea of converting the subtropical conditions into linear constraints can be implemented with a very few lines of code. However, the library needs to implements preprocessing phase, i.e. converting polynomials into sparse distributive representation (sum of monomials)l, which accounts for the most amount of the code. The library is integrated into veriT [5] as an incomplete theory solver for non-linear arithmetic benchmarks. We experimented on the QF_NRA category of the SMT-LIB 2016 on all benchmarks consisting of only inequalities, that is 4917 formulas out of 11601 in the whole category. The experiments thus focus on those 4917 benchmarks, comprising 3265 SAT-annotated ones, 106 UNKNOWNs, and 1546 UNSAT benchmarks. We used the SMT solver CVC4 to handle the generated linear real arithmetic formulas $\Psi(f_1, \ldots, f_m, \mathbf{n}, c_1, \ldots, c_m, \tau)$, and we ran veriT (with STROPSAT as the theory solver) against the clear winner of the SMT-COMP 2016 on the QF_NRA category, i.e., Z3 (implementing nlsat [43]), on a CX250 Cluster with Intel Xeon E5-2680v2 2.80GHz CPUs. Each pair of benchmark and solver was run on one CPU with a timeout of 2500 seconds and 20 GB memory. The experimental data and the library are also available on Zenodo[5].

[3] https://www.starexec.org/starexec/secure/explore/spaces.jsp?id=184545

[4] `http://www.jaist.ac.jp/~s1520002/STROPSAT/`

[5] `http://doi.org/10.5281/zenodo.817615`

Since our method focuses on showing satisfiability, only brief statistics on UNSAT benchmarks are provided. Among the 1546 UNSAT benchmarks, 200 benchmarks are found unsatisfiable already by the linear arithmetic theory reasoning in veriT. For each of the remaining ones, the method quickly returns UNKNOWN within 0.002 to 0.096 seconds, with a total cumulative time of 18.45 seconds (0.014 seconds on average). This clearly shows that the method can be applied with a very small overhead, upfront of another, complete or less incomplete procedure to check for unsatisfiability.

Table 9.5: Comparison between STROPSAT and Z3 (times in seconds)

Family	STROPSAT				Z3			
	SAT	Time	UNKOWN	Time	SAT	Time	UNSAT	Time
meti-tarski (SAT - 3220)	2359	32.37	861	10.22	**3220**	88.55	0	0
zankl (SAT - 45)	29	3.77	16	0.59	**42**	2974.35	0	0
zankl (UNKNOWN - 106)	**15**	2859.44	76	6291.33	14	1713.16	23	1.06

Table 9.5 provides the experimental results on benchmarks with SAT or UNKNOWN status, and the cumulative times. The meti-tarski family consists of small benchmarks (most of them contain 3 to 4 variables and 1 to 23 polynomials with degrees between 1 and 4). Those are proof obligations extracted from the meti-tarski project [1], where the polynomials represent approximations of elementary real functions; all of them have defined statuses. The zankl family consists of large benchmarks (large numbers of variables and polynomials but small degrees) stemming from termination proofs for term-rewriting systems [29].

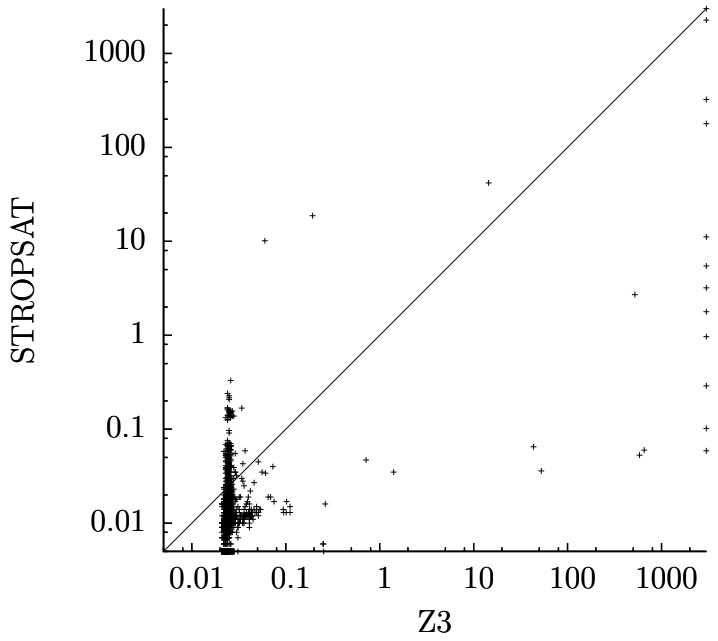


Figure 9.13: STROPSAT returns SAT or timeout (2418 benchmarks, times in seconds)

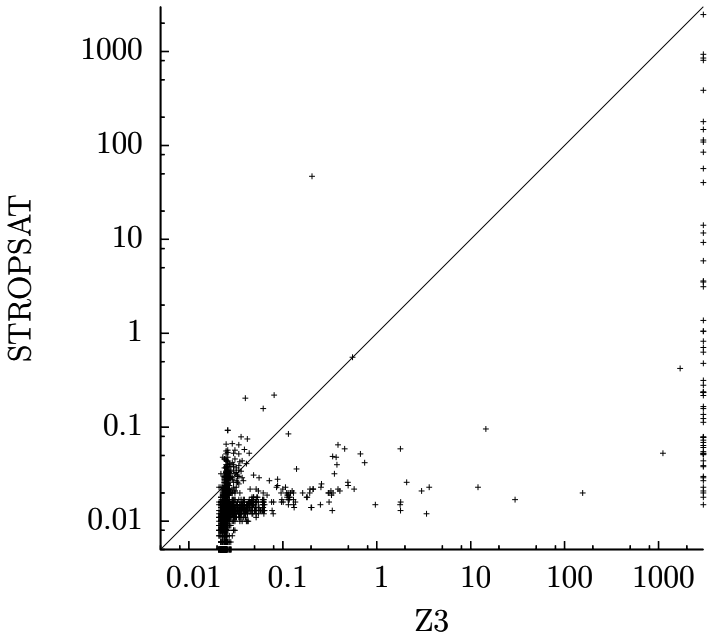


Figure 9.14: STROPSAT returns UNKNOWN (2299 benchmarks, times in seconds)

Although Z3 clearly outperforms STROPSAT in the number of solved benchmarks, the results also clearly show that our method is a useful complementing heuristic with little drawbacks, to be used either upfront or in a portfolio with other approaches. As already said, it returns UNKNOWN quickly on UNSAT benchmarks. In particular, on all benchmarks solved by Z3 only, STROPSAT returns UNKNOWN quickly (see Figure 9.14).

When both solvers can solve the same benchmark, the running time of STROPSAT is comparable with Z3 (Figure 9.13). There are 11 large benchmarks (9 of them have the UNKNOWN status) that are solved by STROPSAT but time out with Z3. STROPSAT times out for only 15 problems, on which Z3 times out as well. STROPSAT provides a model for 15 UNKNOWN benchmarks, whereas Z3 times out on 9 of them. The virtual best solver (i.e. running Z3 and STROPSAT in parallel and using the quickest answer) decreases the execution time for the meti-tarski problems to 54.43 seconds, solves all satisfiable zankl problems in 1120 seconds, and 24 of the unknown ones in 4502 seconds.

Since the exponents of the polynomials become coefficients in the linear formulas, high degrees do not hurt our method significantly. As the SMT-LIB does not currently contain any inequality benchmarks with high degrees, our experimental results above do not demonstrate this claim. However, formulas like in Example 34 are totally within reach of our method (STROPSAT returned SAT within a second) while Z3 runs out of memory (20 GB) after 30 seconds for the constraint $f_1 > 0 \wedge f_2 > 0$.

9.3 Performance of the Efficient Complete Framework

All experiments in this section were conducted on an Intel Xeon E5-2680v2 at 2.80GHz running GNU/Linux CentOS 6.4 with kernel 2.6.32. Each solver ran on the 11354 benchmarks of the QF_NRA category (quantifier-free Non-linear Real Arithmetic) of the SMT-

LIB 2017 (4963 are labeled as satisfiable, 5296 unsatisfiable, 1095 unknown) with a memory limit of 8 GB memory and a time out of 2500 seconds for each benchmark.[6]

In our implementation, raSAT [46, 83] serves as the ICP-based solver, the computer algebra system Redlog/Reduce [21] for quantifier elimination, and STROPSAT as the implementation of subtropical satisfiability. The interface for the three tools within veriT is 900 lines of C code. Notice that CVC4 is used inside STROPSAT to solve linear constraints. The framework for solving polynomial constraints in the SMT solver is called lazily when the underlying SAT solver has produced a full model and if the corresponding set of literals is not shown unsatisfiable by the linear arithmetic decision procedure in veriT.

9.3.1 Comparing Individual Components with the Combined One

Table 9.6 presents the experimental performances of **raSAT**, **STROPSAT**, **Redlog**, and their combinations.

Table 9.6: Numbers of benchmarks solved by component procedures

Benchmarks	**STROPSAT**	**raSAT**	**Redlog**	**Lazy**	**Lazy** w/o **STROPSAT**	**Less Lazy**
SAT	1936	4302	4400	**4450**	4433	4445
UNSAT	2530	4472	4959	**5012**	**5012**	4984
Total (11354)	4466	8774	9359	**9462**	9445	9429
Total time (s)	4744	18835	67945	50632	44815	75676

Redlog alone already solves many problems, but combining the procedures (the lazy combination) brings improvements both in running time and in the number of solved problems. Let us use scatter plots to look at more closely on performances of the tools.

Firstly, Figure 9.15 illustrates that **raSAT** and **Redlog** are complementary and on most of the problems that were commonly solved by both, **raSAT** is often faster than **Redlog** to solve them. However, due to limitations (on touching and convergent cases) of the ICP-based method, there was a significant number of benchmarks on which **Redlog** can solve but **raSAT** cannot. We expect to achieve both efficiency (of **raSAT**) and completeness (of **Redlog**) when combining two procedures implemented inside those two tools.

Figure 9.16 compares **raSAT** alone with the lazy combination without **STROPSAT** (i.e. Algorithm 8 without lines 2 to 4). The Figure in addition to Figure 9.15 illustrates that the lazy combination was able to take advantages of the efficiency of **raSAT** and it also was able to complement **raSAT** in completeness on a significant number of benchmarks.

Figure 9.17 compares the performances of **Redlog** and the lazy combination without **STROPSAT** (i.e. Algorithm 8 without lines 2 to 4). While it again shows that combination was able to take advantages of the efficiency of **raSAT**, it also clearly says

[6]See `http://www.jaist.ac.jp/~s1520002/veriT+STROPSAT+raSAT+Redlog/` for full results and the tool with the source code. Note to reviewers: this will soon be on Zenodo.

that there are a large number of benchmarks which are solved by **Redlog** but not by the combination. This leaves a room for our future work for improvements.

Figure 9.18 presents the effects of STROPSAT in the lazy combination. STROPSAT increases the number of solved satisfiable problems and improves times for several satisfiable problems, without significantly impacting negatively for the problems solved by the other methods: indeed, **STROPSAT** actually takes around 2000 additional seconds for six difficult problems also solved by the combination of raSAT and **Redlog**, and uses 4000 seconds to solve 17 more problems. In addition, **STROPSAT** does not affect the solved UNSAT problems since it quickly terminates and returns UNKNOWN on them. The scatter plots for comparing less lazy with less lazy without **STROPSAT** is similar to 9.18 and is omitted here[7]. Those evidences illustrate our previous claim that subtropical satisfiability establishes a useful complementing heuristic with little drawback, to be used either upfront or in portfolio with other approaches to deal with non-linear constraints.

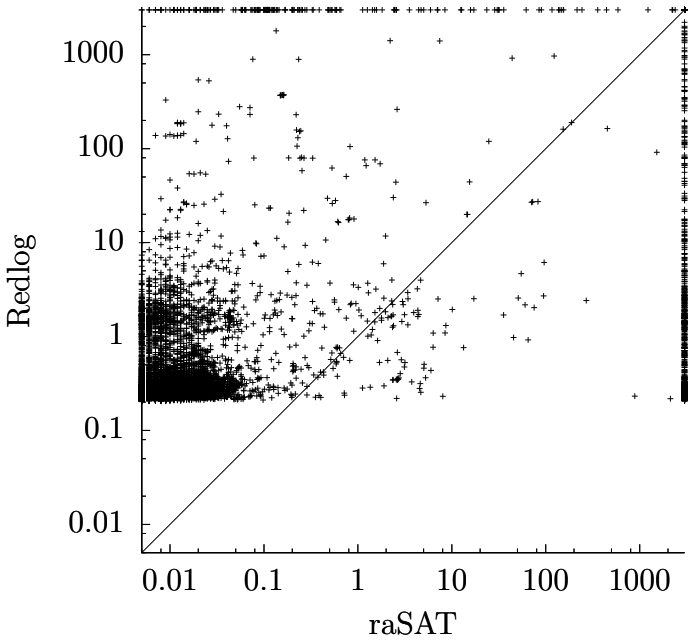


Figure 9.15: **raSAT** vs **Redlog**

[7]See `http://www.jaist.ac.jp/~s1520002/veriT+STROPSAT+raSAT+Redlog/` for full results)

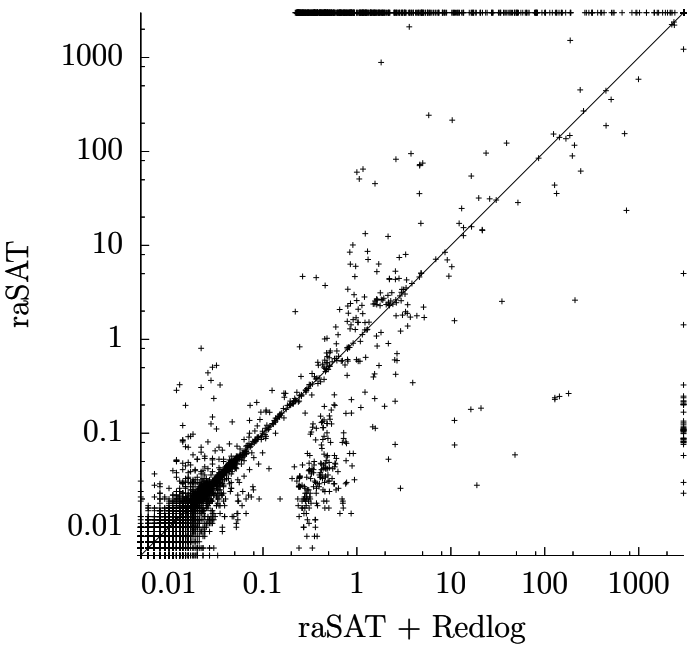


Figure 9.16: **raSAT** vs **Lazy** without **STROPSAT**

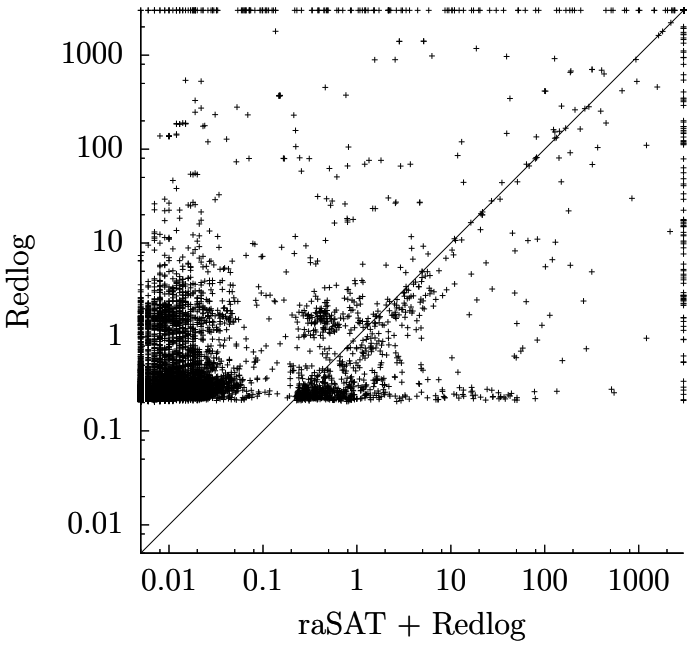


Figure 9.17: **Redlog** vs **Lazy** without **STROPSAT**

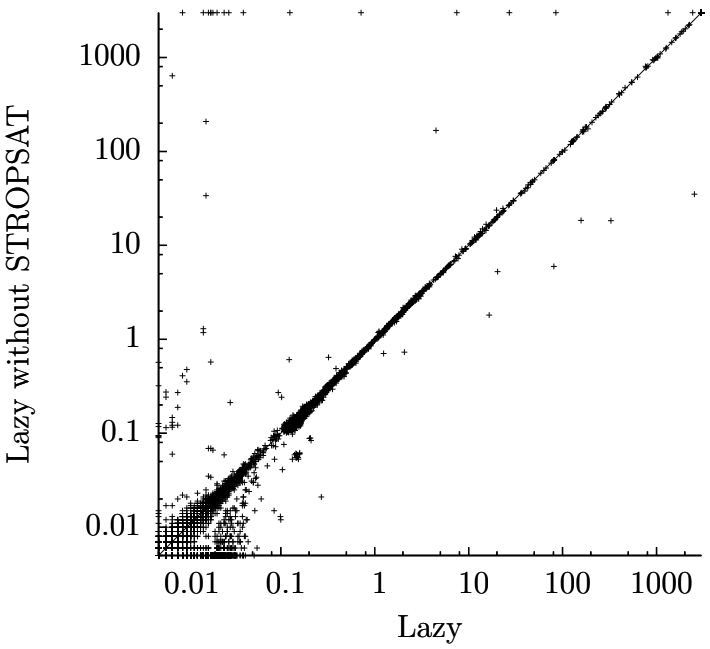


Figure 9.18: Lazy vs lazy without STROPSAT

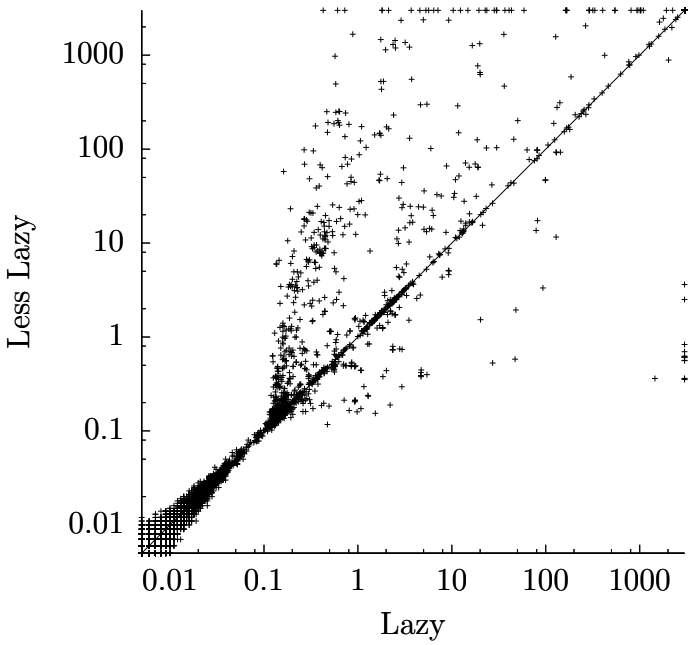


Figure 9.19: **Lazy vs Less Lazy**

Figure 9.19 compares two types of combination, namely **lazy** and **less lazy** approach. From the detailed experimental data, we observe that the less lazy combination suffers from repeatedly checking unsatisfiable boxes generated by the ICP framework. As one of our future work, we will investigate how to discuss multiple unsatisfiable boxes when the

complete framework detects UNSAT over one particular box which is expected to improve the performance of the less lazy combination.

9.3.2 Comparing with State-of-the-art SMT Solvers

This section compares the lazy combination of our procedures with other state-of-the-art SMT solvers. Since the lazy combination is implemented as a theory solver inside the SMT solver **veriT**, in this section, we simply refer the combination as **veriT**.

Table 9.7: Performance of state-of-the-art SMT solvers on QF_NRA

Benchmarks	**CVC4**	**SMT-RAT**	**Z3**	**Yices**	**veriT**	veriT only	Virtual best
SAT	2929	4398	**4905**	4845	4450	18	5183
UNSAT	**5324**	4425	5038	5120	5012	1	5744
Total (11354)	8253	8823	9943	**9965**	9462	19	10927
Total time (s)	146154	57787	37740	132137	50632	11706	119998

Table 9.7 presents experimental data of SMT solvers supporting non-linear arithmetic. SMT-RAT implements a less lazy combination (as mentioned in the previous section) between interval constraint propagation and cylindrical algebraic decomposition [50], Yices and Z3 implement the nlsat procedure [43]. CVC4 uses context-dependent simplification [68] and incremental linearization [10]. Our results validate the main point of the paper: a combination of simple heuristics (ICP and subtropical satisfiability) with quantifier elimination as implemented in a computer algebra system (Redlog/Reduce) slightly tuned to fit the SMT infrastructure is an efficient decision procedure to solve non-linear arithmetic SMT problems. Figures 9.20 to 9.23 closely compares **veriT** with each of state-of-the-art solvers and shows that our method is complementary to ones implemented in those other solvers. Table 9.7 also clearly exhibits that the virtual best solver — a portfolio of all mentioned solvers running in parallel — is much better than each individual solver; we attribute this to the variety of techniques used in the solvers. It should be noted that our theory solver requires very little implementation efforts. As already mentioned, tt took 6000 lines of Ocaml code for **raSAT**, 200 lines of C code for **STROPSAT**, and 900 lines of C code for the combination.

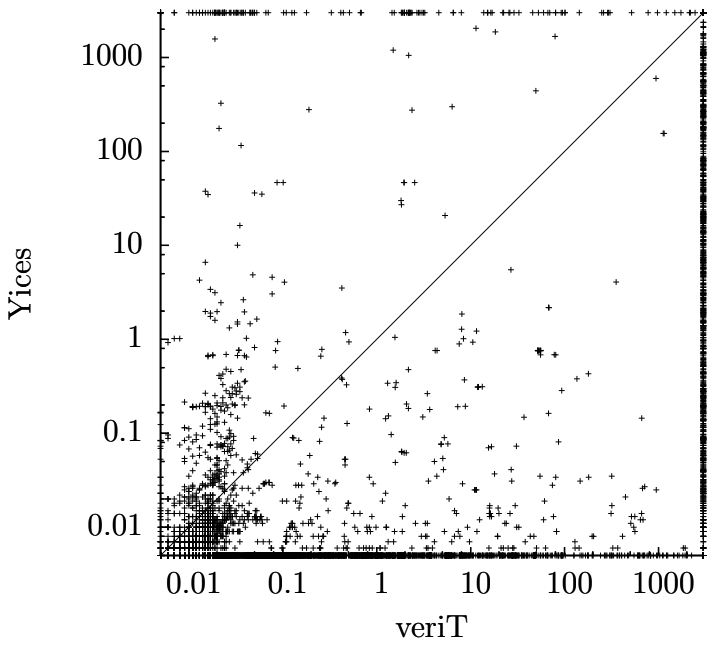


Figure 9.20: **veriT vs Yices**

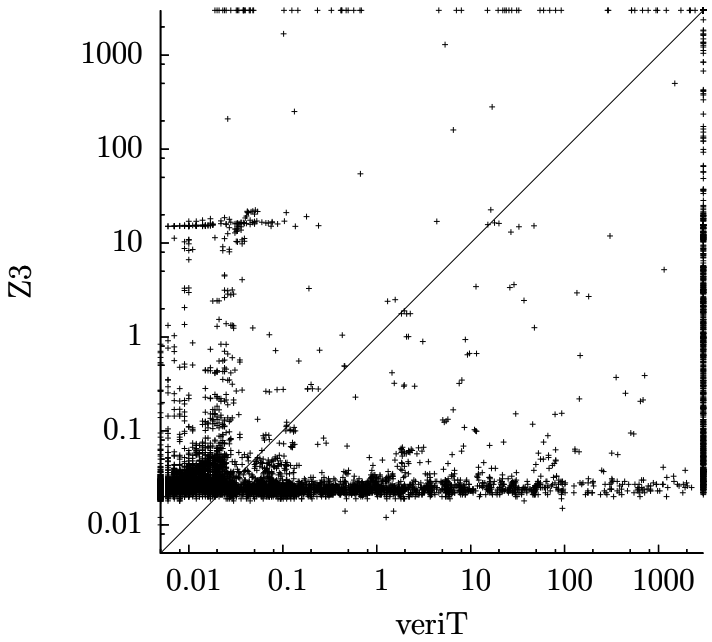


Figure 9.21: **veriT vs Z3**

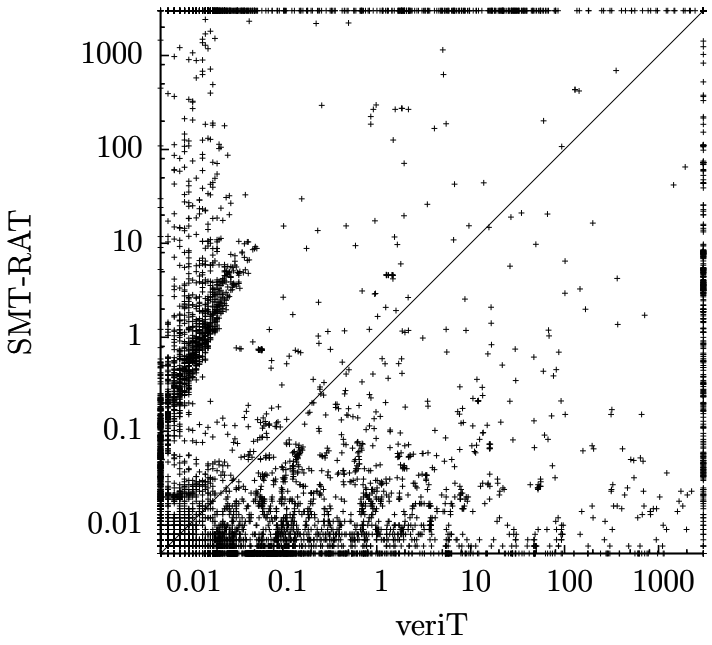


Figure 9.22: **veriT vs SMT-RAT**

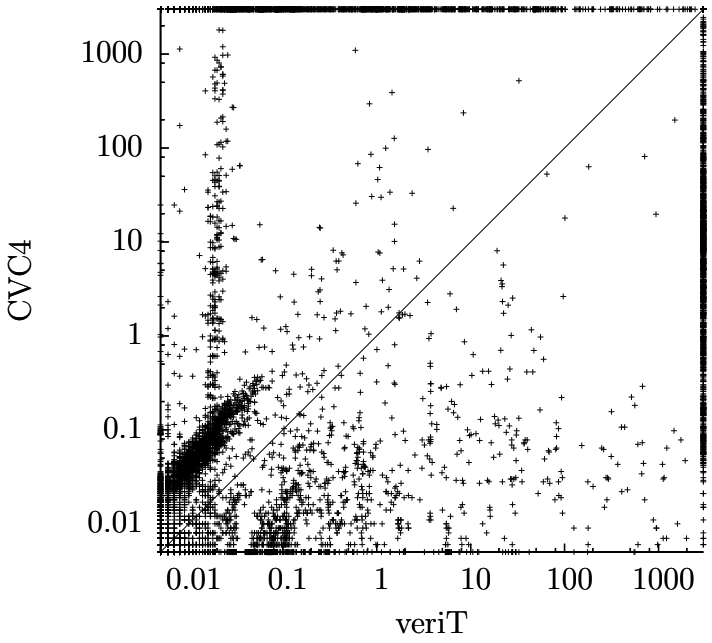


Figure 9.23: **veriT vs CVC4**

9.4 Annual SMT Competitions

Since 2014, we have participated to the annual SMT competitions. The detailed results of each year is as follows. We present the results of sequential performances while the

parallel performances do not differ too much from the sequential ones. Details of results can be found at the associated links. In the tables, [**S**] means that the solver **S** is non-competitive. The first row lists the solvers that participated in the competition, the second one presents the number of solved benchmarks, and the last row shows the total running time on solved benchmarks in seconds.

SMT-COMP 2014

raSAT was ranked third in the category of QF_NRA consisting of 10121 benchmarks[8].

Table 9.8: SMT-COMP 2014 result on QF_NRA

[Z3]	CVC3	CVC4	raSAT
9927	3543	2657	88
21428	33536	129	1

SMT-COMP 2015

raSAT was ranked third and second in the category of QF_NRA (10184 benchmarks) and QF_NIA (8475 benchmarks) respectively[9].

Table 9.9: SMT-COMP 2015 result on QF_NRA

[Z3]	Yices2-NL	SMT-RAT	raSAT	CVC3	CVC4(exp)	CVC4
10000	9854	8759	7952	3575	2694	2694
458921	884238	3451838	4969779	1220253	1021	**1030**

Table 9.10: SMT-COMP 2015 result on QF_NIA

[Z3]	AProVE	raSAT	SMT-RAT (parallel)	SMT-RAT	CVC3	CVC4	CVC4 (exp)
8459	8270	7917	7435	7309	191	76	8277 (1 error)
54050	482573	1290125	2496421	2806065	180124	4862	284689

SMT-COMP 2016

raSAT was ranked second and fifth in the category of QF_NRA (10245 benchmarks) and QF_NIA (8593 benchmarks) respectively[10].

[8] http://smtcomp.sourceforge.net/2014/results-summary.shtml?v=1403902163
[9] http://smtcomp.sourceforge.net/2015/results-summary.shtml?v=1446209369
[10] hhttp://smtcomp.sourceforge.net/2016/results-summary.shtml?v=1467876482

Table 9.11: SMT-COMP 2016 result on QF_NRA

[Z3]	**Yices2**	**raSAT 0.4**	**raSAT 0.3**	**CVC4**	SMT-RAT
10056	10019	9024	8431	2694	9026 (4 errors)
24785	61990	11176	13577	150	51053

Table 9.12: SMT-COMP 2016 result on QF_NIA

[Z3]	**Yices2**	**CVC4**	**SMT-RAT**	**AProVE**	**raSAT 0.4**	**raSAT 0.3**	**ProB**
8566	8451	8231	8443	8273	8017	7544	7557
27718	8523	161418	6235	8528	159248	70229	13586

SMT-COMP 2017

veriT+raSAT+Redlog was ranked second and fourth in the category of QF_NRA (11354 benchmarks) and QF_UFNRA (36 benchmarks) respectively[11].

Table 9.13: SMT-COMP 2017 result on QF_NRA

Yices	**[Z3]**	**veriT+raSAT+Redlog**	**SMTRAT**	**CVC4**
9570	9591	9102	8492	8005
20801	9868	20722	27058	83250

Table 9.14: SMT-COMP 2017 result on QF_UFNRA

[Z3]	**Yices**	**CVC4**	**veriT+raSAT+Redlog**
30	24	12	10
1705	1154	7	0

[11] http://smtcomp.sourceforge.net/2017/results-summary.shtml?v=1500632282

Chapter 10

Comparing raSAT with other ICP-based SMT Solvers

10.1 iSAT3

SMT Architecture

iSAT3 is a newer version of HySAT-II [28] which implements the iSAT algorithm [28]. Instead of lazily combining DPLL procedure with the theory solver ICP, **iSAT3** does it in an eager way.

- Each interval in a box is treated as a special APC.
- In the "Unit Propagation", **iSAT3** finds clauses that have all but one APC being IA-UNSAT with the current box, then the IA-SAT APC will be selected to be processed next.
- When an APC is selected by the "Unit Propagation", ICP is used to contract the current box. New APCs, which represents contracted intervals, are generated and added into the set of selected APCs.
- When both the "Unit Propagation" and contraction cannot be performed, "Box decomposition" is implemented to split one interval of the current box into smaller ones.
- The "Decide" step of the DPLL procedure selects one of the decomposed intervals to add into the set of selected APCs.

Another notable difference between **iSAT3** and **raSAT** loop is that **iSAT3** maintains an "implication graph" to keep track of implications between selected APCs, i.e. which APCs were selected under the implication from which APCs. In the case that the selected APCs are IA-UNSAT, **iSAT3** traverses the implication graph to derive a reason for the conflict and add this reason in negated form to the input formula. In order to do this effectively, the iSAT algorithm decomposes each polynomial into the three address forms: $x \diamond c$ and $x = y \circ z$, by introducing a number of fresh variables and equations. This transformation reduces the applicability of testing since it introduces many new equations and converts any original inequality $g > 0$ to the form $x > 0$. As a result, we did not adopt the methodology of the iSAT algorithm into our framework.

Handling equations

As mentioned, the iSAT algorithm decomposes each polynomial into the three address forms: $x \diamond c$ and $x = y \circ z$, by introducing a number of fresh variables and equations.

Example 46 *For a constraint $x^2 \geq 2 \wedge x \geq 2$, **iSAT** translates it into the following equi-satisfiable constraint.*

$$y = x^2 \wedge y \geq 2 \wedge x \geq 2$$

ICP will infer the intervals of variables as $x \in [2, \infty]$ and $y \in [4, \infty]$. The IA estimates the value of $y - x^2$ as $[-\infty, \infty]$ and concludes that $y = x^2$ is IA-SAT.

Although equations are often harmful with SAT detection by ICP, the variable y in the above example is, in fact, an auxiliary one, and it occurs exactly once on the left-hand side of the equation. Thus, if a box B does not conflict with the equation $y = x^2$, in the sense that $x^2 \in B(y)$ holds for each $x \in B(x)$, then the constraint $y = x^2$ is SAT.

iSAT generalizes this idea with the notion of *strong satisfiability* (SS) [28]. For a conjunction $\varphi = \{\psi_1, \cdots, \psi_m\}$, a box B *strongly satisfies* φ if and only if the following two conditions hold:

1. If ψ_i is an equation $x = y \circ z$, then either $B(x)$ is a point (i.e., $|B(x)| = 1$), or x occurs neither in ψ_j with $j > i$ nor on the right-hand side of ψ_i (i.e., $x \neq y$ and $x \neq z$).

2. The box B does not conflict with any APC $\psi \in \varphi$ in the following sense.

$$\psi = (x \diamond c) \text{ implies } B(x) \subseteq \{u \mid u \in \mathbb{R}, u \diamond c\}$$
$$\psi = (x = y \circ z) \text{ implies } B(x) \supseteq B(y) \circ B(z)$$

   Here, we abuse the notion $\circ$ in $B(y) \circ B(z)$ to denote the interval extension.

Theorem 9 ([28, Lemma 4]) *If there exists a box B that strongly satisfies φ, φ is satisfiable.* *(Q.E.D.)*

If φ is strongly satisfiable in a box B, then **raSAT loop** with the generalized IVT (Fig. 7.5) detects SAT (Theorem 10), but not vice versa (Example 47). The proof of Theorem 10 is presented in Appendix B.

Theorem 10 *If a box B strongly satisfies a conjunction φ, then the extension of ICP with the generalized IVT detects* SAT *of φ over B.* *(Q.E.D.)*

Example 47 *Consider a conjunction $y = x^2 \wedge y = z^2$ with the box $B = [0, 4] \times [2, 3] \times [1, 5]$ for variables (x, y, z).*

- *The first condition of SS does not hold.*
- *The application of IVT succeeds because $(\{x\}, \{z\})$ is a check basis of $(y - x^2, y - z^2)$ in B.* *(Q.E.D.)*

Strong satisfiability often works in proving equations that are either generated from the transformation of input formulas into the three-address forms or stem from translating an assignment associated to transitions of a hybrid system into constraints. However, it is often difficult to have SS, if the input formula contains equations. The example below describes such a situation.

Example 48 *Consider a simple constraint $x^2 = 2 \wedge x > 0$, the transformation into three-address forms gives $y = x^2 \wedge y = 2 \wedge x > 0$. The ICP will infer the box B with $B(x) = [\underline{\sqrt{2}}, \overline{\sqrt{2}}]$ and $B(y) = [2, 2]$. Although the first condition of SS is satisfied, the second condition is not.*

10.2 dReal

Instead of showing satisfiability/unsatisfiability of the polynomial constraints φ over the real numbers, **dReal** proves that either

- φ is unsatisfiable, or
- φ^δ is satisfiable.

Here, φ^δ is the δ-weakening of φ with $\delta > 0$. For instance, the δ-weakening of $x = 0$ is $|x| \leq \delta$. Any constraint with operators in $\{<, \leq, >, \geq, =, \neq\}$ can be converted into constraints that contains only $=$ by the following transformations.

- **Removing $\neq$**: Each formula of the form $f \neq 0$ is transformed into $f > 0 \vee f < 0$.
- **Removing $<$ and $\leq$**: Each formula of the form $f < 0$ or $f \leq 0$ is transformed into $-f \geq 0$ or $-f > 0$ respectively.
- **Removing $>$ and $\geq$**: Each formula of the form $f > 0$ or $f \geq 0$ is transformed into $f - x = 0$ by introducing an auxiliary variable x that has bound $[0, m]$ or $(0, m]$ respectively. Here, m is any rational number which is greater than the maximum of f over intervals of variables.

Though both **dReal** and **raSAT** pursue the lazy combination between DPLL and ICP, **dReal** applies the theory solver incrementally to perform theory propagation. Such an incremental combination can be done in **raSAT** by replacing the **miniSAT** solver by that of a general SMT solver such as **openSMT**[1] or **veriT**[2].

10.3 Handling Floating-point Arithmetic in SMT Solvers

There are several works on interval arithmetic based SMT solvers.

Abstract conflict-driven clause learning (ACDCL) [22, 6] is implemented in **MathSAT** [37], aiming at the closer interaction between the SAT solver and the ICP theory solver. Instead of boolean values, an implication graph in SAT solving is extended to a

[1] `http://verify.inf.usi.ch/opensmt`
[2] `http://www.verit-solver.org/`

lattice. The *bottom-everywhere problem* over a lattice decides whether a function f always has the bottom value. The lattice describes an over-approximation, e.g., an interval abstraction [23], similar to the round up/down techniques in the floating-point arithmetic. Then, the unsatisfiability problem is encoded as a bottom-everywhere problem of a reductive function f (i.e., $f(x) \subseteq x$), which refines the over-approximation. The bottom-everywhere problem is checked by computing the greatest fixed-point of f. During this procedure, the unit propagation in SAT solving of DPLL(T) is defined as taking the meet, and the implication graph gives an edge when the ordering of the lattice holds, instead of boolean implication. Current **raSAT** intends the portability of the backend theory solver for potential amalgamation with other SMT solvers, e.g., **veriT**, and has shallow communication with the SAT solver.

The method in [2] focuses on detection of floating-point exceptions, e.g., zero division caused by roundoff errors and overflow errors. It applies a linearization technique to nonlinear constraints, which is implemented as *Z3-ARIADNE*, incorporated with a symbolic execution engine *KLEE*.

Not precisely from the context of SMT solving, [41] introduces a new branch-and-prune heuristics in numerical programming to reduce the search in the projection problem of a hybrid system. The projection problem is to evaluate the projection on particular parameters of the trajectory of the conjunction of equations and inequalities, which include not only polynomials but also elementary functions. The new heuristic tunes a variable selection and a split-point. It improves the original branch-and-prune strategy in [35], and can be considered to try on future **raSAT**.

Chapter 11

Future Work

This chapter addresses some potential future work that might bring performance improvement for our framework.

11.1 Tighter Interaction between ICP and Algebraic Methods

In the less lazy combination of ICP and the complete procedure, we observed that generally the latter often has to repeatedly check multiple unsatisfiable boxes which makes the combination performing poorly. An interesting question is how to discard such multiple unsatisfiable boxes from the ICP when the complete procedure detects one.

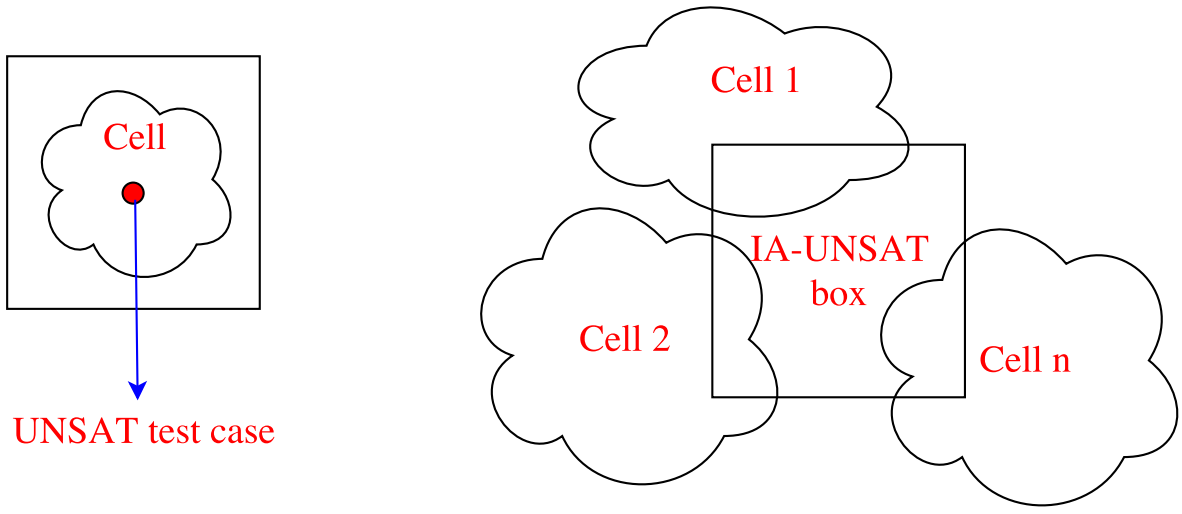


Figure 11.1: Ideas on efficient UNSAT detection by removing UNSAT CAD cells from the searching space

In order to improve the capability of proving UNSAT, **raSAT** can benefit from the idea in [43] in two aspects. First, when an APC is Test-UNSAT by a test case t, the cell containing the test case t (illustrated in the left sub-figure of Fig. 11.1) is eliminated from the search space. Second, when IA-UNSAT is detected over a box B, by applying CAD to

polynomials in *unsat_cores*, all the cells that have non-empty intersections with the box B (as illustrated in the right sub-figure of Fig. 11.1) are excluded from the search space.

In another direction, with exponential complexity, Gröbner bases provide an incomplete but efficient approach for solving equations. Since the limitations of ICP often appear with multiple roots (touching cases) and/or 0-dimensional ideals (the application of IVT fails), the Gröbner basis would be a promising direction to handle these situations. We expect to use the implementation of Gröbner basis in Redlog/Reduce.

11.2 Subtropical Satisfiability: From Vertices to Faces, and to Subset of Frame

To improve the completeness of subtropical satisfiability, it could be helpful to not only consider vertices of Newton polytopes but also faces. Then, the value of the coefficients and not only their sign would matter. Consider $\{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3\} = \text{face}(\mathbf{n}, \text{newton}(f))$, then we have $\mathbf{n}^T\mathbf{p}_1 = \mathbf{n}^T\mathbf{p}_2 = \mathbf{n}^T\mathbf{p}_3$. It is easy to see that $f_{\mathbf{p}_1}\mathbf{x}^{\mathbf{p}_1} + f_{\mathbf{p}_2}\mathbf{x}^{\mathbf{p}_2} + f_{\mathbf{p}_3}\mathbf{x}^{\mathbf{p}_3}$ will dominate the other monomials in the direction of $\mathbf{n}$. In other words, there exists $a_0 \in \mathbb{R}$ such that for all $a \in \mathbb{R}$ with $a \geq a_0$, $\text{sign}(f(a^{\mathbf{n}})) = \text{sign}(f_{\mathbf{p}_1} + f_{\mathbf{p}_2} + f_{\mathbf{p}_3})$. We leave for future work the encoding of the condition for the existence of such a face into linear formulas.

The idea can even be generalized to find any set of exponent vectors that make the corresponding monomials dominate others when variables follow a certain polynomial curve. Let us formally specify the extension. First, we define the notion of the *cluster of a normal value*.

Definition 29 *The set* $C = \{\mathbf{p} \in S \mid \mathbf{n}^T\mathbf{p} = c\}$ *is the cluster of normal value* c *in* $S \subset \mathbb{R}^d$ *with respect to* $\mathbf{n} \in \mathbb{R}^d$.

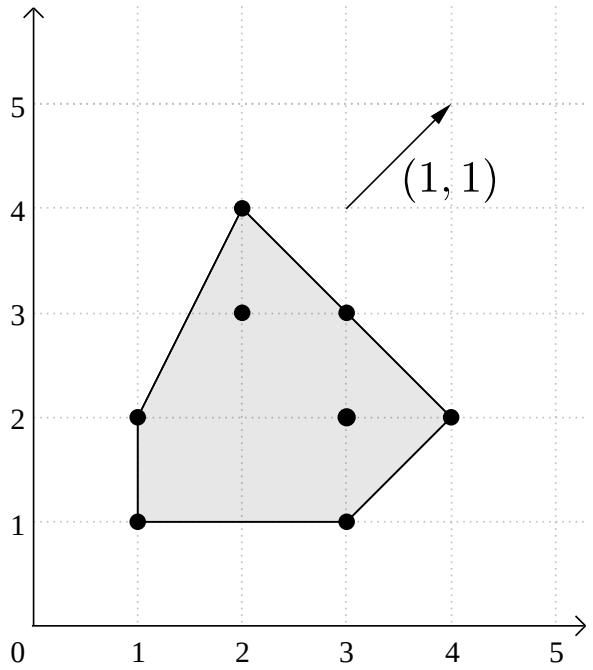


Figure 11.2: A generalization of subtropical satisfiability

Example 49 *(Figure 11.2) Consider the set S as follows*

$$S = \{(1,1),(1,2),(3,1),(2,3),(3,2),(2,4),(4,2),(3,3)\}$$

and a vector $\mathbf{n} = (1,1)$, then we can easily see that $\{(2,3),(3,2)\}$ is the cluster of normal value 5 in S with respect to $\mathbf{n}$.

Definition 30 *A cluster sequence of S with respect to $\mathbf{n}$ is the list of all clusters in S with respect to $\mathbf{n}$ decreasingly sorted by the normal values of the clusters.*

Example 50 *Continue Example 49, the cluster sequence of S with respect to $\mathbf{n} = (1,1)$ is*

$$(\{(4,2),(3,3),(2,4)\},\{(2,3),(3,2)\},\{(3,1)\},\{(1,2)\},\{(1,1)\})$$

with the corresponding normal values as

$$(6,5,4,3,2).$$

Given a polynomial f, the first element in the cluster sequence of frame(f) with respect to $\mathbf{n}$ is the face of newton(f) with respect to $\mathbf{n}$.

Example 51 *Consider the polynomial f as follows*

$$f = -xy - xy^2 - x^3y + 5x^3y^2 - 4x^2y^3 - 100x^4y^2 + 150x^3y^3 - 50x^2y^4$$

and the frame(f) *along with it Newton Polytope are illustrated in Figure 11.2. From the cluster sequence computed in Example 50, we can see that $\{(4,2),(3,3),(2,4)\}$ is the face of the Newton Polytope with respect to $\mathbf{n} = (1,1)$.*

We call C a reduced face of newton(f) if it is the first element in the cluster sequence of frame(f) with respect to $\mathbf{n}$ such that $\sum_{\mathbf{p}\in C} f_{\mathbf{p}} \neq 0$, or an empty set if such an element does not exist.

Example 52 *Continue Example 51, we see that for the cluster $\{(4,2),(3,3),(2,4)\}$, the sum of the corresponding coefficients is $-100+150-50=0$, but that for the next cluster $\{(2,3),(3,2)\}$ is $5-4=1$. As a result, $\{(2,3),(3,2)\}$ is a reduced face with respect to $\mathbf{n} = (1,1)$ of* newton(f).

Lemma 10 *Let f be a polynomial, and let $C \subset$ frame(f) be a reduced face of* newton(f) *with respect to $\mathbf{n} \in \mathbb{R}^d$. Then there exists $a_0 \in \mathbb{R}^+$ such that for all $a \in \mathbb{R}^+$ with $a \geq a_0$ the following holds:*

1. $|\sum_{\mathbf{q}\in C} f_{\mathbf{q}}\ a^{\mathbf{n}^T\mathbf{q}}| > |\sum_{\mathbf{q}\in \text{frame}(f)\setminus C} f_{\mathbf{q}}\ a^{\mathbf{n}^T\mathbf{q}}|$,
2. $\text{sign}(f(a^{\mathbf{n}})) = \text{sign}(\sum_{\mathbf{q}\in C} f_{\mathbf{q}})$.

Proof 13 *The proof is similar to Lemma 4 in [76].*

Example 53 *Continuing Example 52, the sum $5x^3y^2 - 4x^2y^3$ will determine the sign of f when (x,y) reach the limit of (a,a), i.e. (∞,∞). From Figure 11.2, the points $(2,3)$ and $(3,2)$ stays in the interior region of* newton(f) *but their corresponding monomials still dominate the others in f when (x,y) reaches the limit of the polynomial curve (a,a).*

Chapter 12

Conclusion

We developed two heuristic procedures for handling existential polynomial constraints in the context of SMT solving, namely extensions of the ICP with testing and the application of IVT, and subtropical satisfiability.

While the addition of testing is to make the ICP framework quickly detect satisfiable inequalities, the addition of the IVT aims at showing satisfiability of equations. We also proposed efficiency heuristics for improving the performance of the framework. In addition, a combination of testing, IA, and the IVT is presented to show satisfiability of combinations of inequalities and equations.

Subtropical satisfiability is essentially a model finding technique for inequalities. It abstracts polynomials as sets of exponent vectors and determines in each polynomial a monomial (with appropriate sign) dominating others when variables follow some polynomial curve. Such a monomial will decide the sign of the polynomial at the limit of the curve.

Experimental data shows that we were successful in achieving efficiency with the two proposed procedures. We further integrated those two procedures with the quantifier elimination methods implemented in the computer algebra system Redlog/Reduce to produce an efficient complete framework for solving non-linear arithmetic. Experiments on SMT-LIB benchmarks show that the resulting framework is efficient and is complementary to methods implemented in state-of-the-art SMT solvers. It contributes to the variety of methodologies for solving non-linear arithmetic.

To conclude the thesis, we would like to address some future prospects of nonlinear SMT solving.

First of all, once the solver successfully proves the unsatisfiability of a constraint, it is desirable to produce the proof which can be used to verify the algorithm or generating Craig interpolant [11]. The proof for the UNSAT answer of ICP framework is straightforward generated while following the operations of IA and constraint propagation as in Fig. 4.2. For CAD and VS, the generation of the proof is not simple and need further investigation.

Second, after generation of a proof, the next step would be to extract Craig interpolant which can be applied in abstraction refinement [53]. While the interpolant generation has been well studied for linear arithmetic and EUF [52], it is still at the early stage for nonlinear arithmetic to generate good interpolant which can be applied well in abstraction refinement.

Third, combining nonlinear reasoning with that of other theories especially linear

arithmetic can bring performance improvements of solving nonlinear constraints. Currently the combination is somewhat lazy where the non-linear reasoner is called when the linear one failed. An interesting future direction is to investigate interleaving decision procedures to take efficiency advantages of linear reasoning inside non-linear reasoner.

Last but not least, while recently machine learning (ML) has made great improvements, connecting ML techniques with formal verification (FV) has been still limited. One of the difficulties for the connection is that soundness, a critical requirement of FV, is however not guaranteed by ML algorithms. On the other hands, a lot of algorithms in FV can gain efficiency by heuristics. It is often the case that programs implementing the same algorithm shows complementary performances due to different heuristics when they have to make decisions. Given those programs and benchmarks where they perform well, ML can help by learning heuristic to choose possibly the best choice when needed, e.g. choosing variables order to eliminate in CAD and VS.

Bibliography

[1] Akbarpour, B., Paulson, L.C.: MetiTarski: An automatic theorem prover for real-valued special functions. Journal of Automated Reasoning **44**(3) (2010) 175–205

[2] Barr, E.T., Vo, T., Le, V., Su, Z.: Automatic detection of floating-point exceptions. In: Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. POPL '13, New York, ACM (2013) 549–560

[3] Benhamou, F., Granvilliers, L.: Continuous and interval constraints. In: Handbook of Constraint Programming. Elsevier, New York (2006) 571–604

[4] Borosh, I., Treybig, L.B.: Bounds on positive integral solutions of linear diophantine equations. Proceedings of the American Mathematical Society **55**(2) (1976) 299–304

[5] Bouton, T., Caminha B. De Oliveira, D., Déharbe, D., Fontaine, P.: veriT: An open, trustable and efficient SMT-Solver. In: Proceedings of the 22nd International Conference on Automated Deduction. CADE-22, Springer (2009) 151–156

[6] Brain, M., D'Silva, V., Griggio, A., Haller, L., Kroening, D.: Deciding floating-point logic with abstract conflict driven clause learning. Formal Methods in System Design **45** (2014) 213–245

[7] Buchberger, B.: Bruno buchbergers phd thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. Journal of Symbolic Computation **41**(3) (2006) 475 – 511 Logic, Mathematics and Computer Science: Interactions in honor of Bruno Buchberger (60th birthday).

[8] Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Invariant checking of NRA transition systems via incremental reduction to LRA with EUF. In: Tools and Algorithms for the Construction and Analysis of Systems: 23rd International Conference, TACAS 2017. Springer (2017) 58–75

[9] Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Satisfiability modulo transcendental functions via incremental linearization. In de Moura, L., ed.: Automated Deduction – CADE 26, Cham, Springer International Publishing (2017) 95–113

[10] Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Satisfiability modulo transcendental functions via incremental linearization. In de Moura, L., ed.: CADE 2017. Volume 10395 of LNCS., Springer (2017) 95–113

[11] Cimatti, A., Griggio, A., Sebastiani, R.: Efficient interpolant generation in satisfiability modulo theories. In Ramakrishnan, C.R., Rehof, J., eds.: Tools and Algorithms for the Construction and Analysis of Systems, Berlin, Heidelberg, Springer Berlin Heidelberg (2008) 397–412

[12] Collins, G.E. In: Quantifier elimination for real closed fields by cylindrical algebraic decompostion. Springer Berlin Heidelberg, Berlin, Heidelberg (1975) 134–183

[13] Collins, G.E., Hong, H.: Partial cylindrical algebraic decomposition for quantifier elimination. J. Symb. Comput. **12**(3) (September 1991) 299–328

[14] Comba, J.L.D., Stolfi, J.: Affine arithmetic and its applications to computer graphics. In: SIBGRAPI'93. (1993) 9–18

[15] Corzilius, F., Loup, U., Junges, S., Ábrahám, E.: SMT-RAT: An SMT-compliant nonlinear real arithmetic toolbox. In Alessandro, C., Roberto, S., eds.: Theory and Applications of Satisfiability Testing – SAT 2012. Springer (2012) 442–448

[16] Dantzig, G.B., Eaves, B.C.: Fourier-motzkin elimination and its dual. Journal of Combinatorial Theory, Series A **14**(3) (1973) 288 – 297

[17] Dantzig, G.B.: Linear programming and extensions. Prentice University Press, Princeton, NJ (1963)

[18] Davenport, J.H., Heintz, J.: Real quantifier elimination is doubly exponential. J. Symb. Comput. **5**(1-2) (February 1988) 29–35

[19] Davenport, J.H., Heintz, J.: Real quantifier elimination is doubly exponential. Journal of Symbolic Computation **5**(1) (1988) 29 – 35

[20] de Oliveira, D.C.B., Dharbe, D., Fontaine, P.: Combining decision procedures by (model-)equality propagation. Electronic Notes in Theoretical Computer Science **240** (2009) 113 – 128 Proceedings of the Eleventh Brazilian Symposium on Formal Methods (SBMF 2008).

[21] Dolzmann, A., Sturm, T.: Redlog: Computer algebra meets computer logic. SIGSAM Bull. **31**(2) (June 1997) 2–9

[22] D'Silva, V., Haller, L., Kroening, D.: Abstract conflict driven learning. In: Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. POPL '13, New York, ACM (2013) 143–154

[23] D'Silva, V., Haller, L., Kroening, D., Tautschnig, M.: Numeric bounds analysis with conflict-driven learning. In Flanagan, C., König, B., eds.: Tools and Algorithms for the Construction and Analysis of Systems. Springer, Berlin (2012) 48–63

[24] Dutertre, B.: Yices2.2. In Biere, A., Bloem, R., eds.: Computer Aided Verification, Cham, Springer International Publishing (2014) 737–744

[25] Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In Ball, T., Jones, R.B., eds.: Computer Aided Verification. Springer (2006) 81–94

[26] Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In Ball, T., Jones, R.B., eds.: Computer Aided Verification. Springerg, Berlin (2006) 81–94

[27] Fontaine, P., Ogawa, M., Sturm, T., Vu, X.T.: Subtropical satisfiability. In Dixon, C., Finger, M., eds.: Frontiers of Combining Systems: 11th International Symposium, FroCoS 2017. Springer International Publishing, Cham (2017) 189–206

[28] Fränzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure. Journal on Satisfiability, Boolean Modeling and Computation **1** (2007) 209–236

[29] Fuhs, C., Giesl, J., Middeldorp, A., Schneider-Kamp, P., Thiemann, R., Zankl, H.: SAT solving for termination analysis with polynomial interpretations. In Marques-Silva, J., Sakallah, K.A., eds.: Theory and Applications of Satisfiability Testing – SAT 2007. Springer (2007) 340–354

[30] Ganai, M., Ivancic, F.: Efficient decision procedure for non-linear arithmetic constraints using CORDIC. In: Formal Methods in Computer-Aided Design, 2009. FMCAD 2009. (2009) 61–68

[31] Ganzinger, H., Hagen, G., Nieuwenhuis, R., Oliveras, A., Tinelli, C.: DPLL(T): Fast decision procedures. In Alur, R., Peled, D.A., eds.: Computer Aided Verification. Springer (2004) 175–188

[32] Gao, S., Kong, S., Clarke, E.M.: Satisfiability modulo ODEs. In: Formal Methods in Computer-Aided Design (FMCAD), 2013. (2013) 105–112

[33] Gao, S., Avigad, J., Clarke, E.M.: δ-complete decision procedures for satisfiability over the reals. In: Proceedings of the 6th International Joint Conference on Automated Reasoning. IJCAR'12, Berlin, Heidelberg, Springer-Verlag (2012) 286–300

[34] Gao, S., Kong, S., Clarke, E.: dReal: An SMT solver for nonlinear theories over the reals. In Bonacina, M., ed.: Automated Deduction – CADE-24. Springer (2013) 208–214

[35] Goldsztejn, A.: A branch and prune algorithm for the approximation of non-linear aesolution sets. In: Proceedings of the 2006 ACM Symposium on Applied Computing, New York, USA, ACM (2006) 1650–1654

[36] Granvilliers, L., Benhamou, F.: RealPaver: An interval solver using constraint satisfaction techniques. ACM Transactions on Mathematical Software **32** (2006) 138–156

[37] Haller, L., Griggio, A., Brain, M., Kroening, D.: Deciding floating-point logic with systematic abstraction. In: 2012 Formal Methods in Computer-Aided Design (FMCAD). (Oct 2012) 131–140

[38] Harrison, J. In: Verifying Nonlinear Real Formulas Via Sums of Squares. Springer Berlin Heidelberg, Berlin, Heidelberg (2007) 102–118

[39] Hong, H.: An improvement of the projection operator in cylindrical algebraic decomposition. In: Proceedings of the International Symposium on Symbolic and Algebraic Computation. ISSAC '90, New York, NY, USA, ACM (1990) 261–264

[40] Hong, H.: Simple solution formula construction in cylindrical algebraic decomposition based quantifier elimination. In: Papers from the International Symposium on Symbolic and Algebraic Computation. ISSAC '92, New York, NY, USA, ACM (1992) 177–188

[41] Ishii, D., Goldsztejn, A., Jermann, C.: Interval-based projection method for under-constrained numerical systems. Constraints **17** (2012) 432–460

[42] Jaroschek, M., Dobal, P.F., Fontaine, P.: Adapting real quantifier elimination methods for conflict set computation. In Lutz, C., Ranise, S., eds.: FroCoS 2015. Springer (2015) 151–166

[43] Jovanović, D., de Moura, L.: Solving non-linear arithmetic. In Gramlich, B., Miller, D., Sattler, U., eds.: Automated Reasoning - 6th International Joint Conference, IJCAR 2012. Springer (2012) 339–354

[44] Karmarkar, N.: A new polynomial-time algorithm for linear programming. Combinatorica **4**(4) (1984) 373–395

[45] Khachiyan, L.: Polynomial algorithms in linear programming. USSR Computational Mathematics and Mathematical Physics **20**(1) (1980) 53–72

[46] Khanh, T.V., Ogawa, M.: SMT for polynomial constraints on real numbers. Electronic Notes in Theoretical Computer Science **289** (2012) 27 – 40 Third Workshop on Tools for Automatic Program Analysis (TAPAS' 2012).

[47] Košta, M.: New Concepts for Real Quantifier Elimination by Virtual Substitution. Doctoral dissertation, Saarland University, Germany (December 2016)

[48] Lasaruk, A., Sturm, T.: Weak integer quantifier elimination beyond the linear case. In: CASC 2007. Volume 4770 of LNCS. Springer (2007)

[49] Lasaruk, A., Sturm, T.: Weak quantifier elimination for the full linear theory of the integers. A uniform generalization of Presburger arithmetic. Appl. Algebra Eng. Commun. Comput. **18**(6) (December 2007) 545–574

[50] Loup, U., Scheibler, K., Corzilius, F., Ábrahám, E., Becker, B.: A symbiosis of interval constraint propagation and cylindrical algebraic decomposition. In Bonacina, M.P., ed.: CADE 24. Springer (2013) 193–207

[51] McCallum, S.: An improved projection operation for cylindrical algebraic decomposition of three-dimensional space. J. Symb. Comput. **5**(1-2) (February 1988) 141–161

[52] McMillan, K.L.: An interpolating theorem prover. In Jensen, K., Podelski, A., eds.: Tools and Algorithms for the Construction and Analysis of Systems, Berlin, Heidelberg, Springer Berlin Heidelberg (2004) 16–30

[53] McMillan, K.L.: Lazy abstraction with interpolants. In Ball, T., Jones, R.B., eds.: Computer Aided Verification, Berlin, Heidelberg, Springer (2006) 123–136

[54] Messine, F.: Extentions of affine arithmetic: Application to unconstrained global optimization. Journal of Universal Computer Science **8** (2002) 992–1015

[55] Mishra, B.: Algorithmic Algebra. Springer-Verlag New York, Inc., New York, NY, USA (1993)

[56] Miyajima, S., Miyata, T., Kashiwagi, M.: A new dividing method in affine arithmetic. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences **E86-A** (9 2003) 2192–2196

[57] Moore, R.: Interval analysis. Prentice-Hall, Englewood Cliffs (1966)

[58] Neumaier, A.: Interval Methods for Systems of Equations. Cambridge Middle East Library. Cambridge University Press (1990)

[59] Ngoc, D.T.B., Ogawa, M.: Overflow and roundoff error analysis via model checking. In: Proceedings of the 2009 Seventh IEEE International Conference on Software Engineering and Formal Methods. SEFM '09, Washington, IEEE Computer Society (2009) 105–114

[60] Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT Modulo Theories: From an Abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL(T). Journal of the ACM **53** (November 2006) 937–977

[61] Parrilo, P.A.: Semidefinite programming relaxations for semialgebraic problems. Mathematical Programming **96**(2) (May 2003) 293–320

[62] Passmore, G.O.: Combined decision procedures for nonlinear arithmetics, real and complex. Dissertation, School of Informatics, University of Edinburgh (2011)

[63] Passmore, G.O., Jackson, P.B.: Combined decision techniques for the existential theory of the reals. In Carette, J., Dixon, L., Coen, C.S., Watt, S.M., eds.: Intelligent Computer Mathematics, Springer (2009) 122–137

[64] Platzer, A., Quesel, J.D., Rümmer, P. In: Real World Verification. Springer Berlin Heidelberg, Berlin, Heidelberg (2009) 485–501

[65] Ratschan, S.: Efficient solving of quantified inequality constraints over the real numbers. ACM Transactions on Computational Logic **7** (2006) 723–748

[66] Ratschan, S.: Applications of quantified constraint solving over the reals - bibliography. CoRR **abs/1205.5571** (2012)

[67] Rennie, B., Dobson, A.: On stirling numbers of the second kind. Journal of Combinatorial Theory **7** (1969) 116 – 121

[68] Reynolds, A., Tinelli, C., Jovanovic, D., Barrett, C.: Designing theory solvers with extensions. In Dixon, C., Finger, M., eds.: FroCoS 2017. Volume 10483 of LNCS., Springer (2017) 22–40

[69] Schrijver, A.: Theory of Linear and Integer Programming. John Wiley & Sons, Inc., New York, NY (1986)

[70] Seidl, A.: Cylindrical Decomposition Under Application-Oriented Paradigms. PhD thesis, Universität Passau, 94030 Passau, Germany (March 2006)

[71] Seidl, A.M., Sturm, T.: Boolean quantification in a first-order context. In Ganzha, V.G., Mayr, E.W., Vorozhtsov, E.V., eds.: CASC 2003. Institut für Informatik, Technische Universität München, München, Germany (2003) 329–345

[72] Stengle, G.: A nullstellensatz and a positivstellensatz in semialgebraic geometry. Mathematische Annalen **207** (1974) 87–98

[73] Stolfi, J., Figueiredo, L.H.D.: Self-validated numerical methods and applications. In: Monograph for 21st Brazilian Mathematics Colloquium. (1997)

[74] Strichman, O.: On solving presburger and linear arithmetic with sat. In Aagaard, M.D., O'Leary, J.W., eds.: Formal Methods in Computer-Aided Design, Berlin, Heidelberg, Springer Berlin Heidelberg (2002) 160–170

[75] Sturm, T.: Linear problems in valued fields. J. Symb. Comput. **30**(2) (August 2000) 207–219

[76] Sturm, T.: Subtropical real root finding. In: Proceedings of the ISSAC 2015. ACM (2015) 347–354

[77] Sturm, T.: A survey of some methods for real quantifier elimination, decision, and satisfiability and their applications. Math. Comput. Sci. **11**(3–4) (December 2017) 483–502

[78] Sturm, T., Weispfenning, V.: Quantifier elimination in term algebras. The case of finite languages. 285–300

[79] Sturm, T., Zengler, C.: Parametric quantified SAT solving. In: ISSAC 2010. 77–84

[80] Tarski, A.: A decision method for elementary algebra and geometry. Technical Report R-109, Rand Corporation (1951)

[81] Truong, A., Hung, D.V., Dang, D., Vu, X.: A type system for counting logs of multi-threaded nested transactional programs. In: Distributed Computing and Internet Technology - 12th International Conference, ICDCIT 2016. (2016) 157–168

[82] Volder, J.E.: The cordic trigonometric computing technique. IRE Transactions on Electronic Computers **EC-8**(3) (Sept 1959) 330–334

[83] Vu, X.T., Khanh, T.V., Ogawa, M.: raSAT: an SMT solver for polynomial constraints. Formal Methods in System Design **51**(3) (2017) 462–499

[84] Vu, X.T., Nguyen, L.M., Hoang, D.T.: Semantic parsing for vietnamese question answering system. In: 2015 Seventh International Conference on Knowledge and Systems Engineering (KSE). (Oct 2015) 332–335

[85] Vu, X.T., Van Khanh, T., Ogawa, M.: raSAT: An SMT solver for polynomial constraints. In Olivetti, N., Tiwari, A., eds.: Automated Reasoning - 8th International Joint Conference, IJCAR 2016. Springer (2016) 228–237

[86] Weispfenning, V.: Quantifier elimination for real algebra — the quadratic case and beyond. Applicable Algebra in Engineering, Communication and Computing **8**(2) (Jan 1997) 85–101

[87] Weispfenning, V.: The complexity of linear problems in fields. J. Symb. Comput. **5**(1–2) (February–April 1988) 3–27

[88] Weispfenning, V.: Quantifier elimination for real algebra—the quadratic case and beyond. Appl. Algebra Eng. Commun. Comput. **8**(2) (February 1997) 85–101

[89] Zankl, H., Middeldorp, A.: Satisfiability of non-linear (ir)rational arithmetic. In Clarke, E.M., Voronkov, A., eds.: Logic for Programming, Artificial Intelligence, and Reasoning. Springer (2010) 481–500

Publications

Related to this Thesis

[1] Vu, X.T., Van Khanh, T., Ogawa, M.: raSAT: An SMT solver for polynomial constraints. In Olivetti, N., Tiwari, A., eds.: Automated Reasoning - 8th International Joint Conference, IJCAR 2016. Springer (2016) 228–237

[2] Fontaine, P., Ogawa, M., Sturm, T., Vu, X.T.: Subtropical satisfiability. In Dixon, C., Finger, M., eds.: Frontiers of Combining Systems: 11th International Symposium, FroCoS 2017. Springer International Publishing, Cham (2017) 189–206

[3] Vu, X.T., Khanh, T.V., Ogawa, M.: raSAT: an SMT solver for polynomial constraints. Formal Methods in System Design **51**(3) (2017) 462–499

Others

[4] Vu, X.T., Nguyen, L.M., Hoang, D.T.: Semantic parsing for vietnamese question answering system. In: 2015 Seventh International Conference on Knowledge and Systems Engineering (KSE). (Oct 2015) 332–335

[5] Truong, A., Hung, D.V., Dang, D., Vu, X.: A type system for counting logs of multithreaded nested transactional programs. In: Distributed Computing and Internet Technology - 12th International Conference, ICDCIT 2016. (2016) 157–168

Appendix A

Definition of CAI

Given CAI1 forms: $\mathring{x} = \bar{a}_0 + \sum_{i=1}^{n} \bar{a}_i\epsilon_i + \sum_{i=1}^{n} \bar{a}_{i+n}\epsilon_{i+n} + \bar{a}_{2n+1}\epsilon_{\pm}$, and $\mathring{y} = \bar{b}_0 + \sum_{i=1}^{n} \bar{b}_i\epsilon_i + \sum_{i=1}^{n} \bar{b}_{i+n}\epsilon_{i+n} + \bar{b}_{2n+1}\epsilon_{\pm}$, and $\bar{c} = [-1, 1]$. Arithmetic operations $\{\mathring{\bar{+}}, \mathring{\bar{-}}, \mathring{\bar{\times}}\}$ of `CAI1 arithmetic` are defined as follows *(for simplicity we denote $\bar{a}\bar{b}$ for $\bar{a}\bar{\times}\bar{b}$)*:

- $\mathring{x}\mathring{\bar{+}}\mathring{y} = (\bar{a}_0\bar{+}\bar{b}_0) + \sum_{i=1}^{2n}(\bar{a}_i\bar{+}\bar{b}_i)\epsilon_i + (\bar{c}\bar{a}_{2n+1}\bar{+}\bar{c}\bar{b}_{2n+1})\epsilon_{\pm}$
- $\mathring{x}\mathring{\bar{-}}\mathring{y} = (\bar{a}_0\bar{-}\bar{b}_0) + \sum_{i=1}^{2n}(\bar{a}_i\bar{-}\bar{b}_i)\epsilon_i + (\bar{c}\bar{a}_{2n+1}\bar{+}\bar{c}\bar{b}_{2n+1})\epsilon_{\pm}$
- $\mathring{x}\mathring{\bar{\times}}\mathring{y} = K_0 + \sum_{i=1}^{n}(\bar{a}_0\bar{b}_i\bar{+}\bar{a}_i\bar{b}_0\bar{+}\bar{a}_i\bar{b}_{i+n}\bar{+}\bar{a}_{i+n}\bar{b}_i)\epsilon_i\bar{+}$
  $\sum_{i=1}^{n}(\bar{a}_0\bar{b}_{i+n}\bar{+}\bar{a}_{i+n}\bar{b}_0\bar{+}\bar{a}_i\bar{b}_i\bar{+}\bar{a}_{i+n}\bar{b}_{i+n})\epsilon_{i+n} + K\epsilon_{\pm}$,

where $\{\bar{+}, \bar{-}, \bar{\times}\}$ are CI arithmetic, and

- $K_0 = \bar{a}_0\bar{b}_0\bar{+}\sum_{i=1}^{n}(\bar{a}_i\bar{b}_i[-\frac{1}{4}, 0]\bar{+}\bar{a}_i\bar{b}_{i+n}[-\frac{1}{4}, \frac{1}{4}]\bar{+}\bar{b}_i\bar{a}_{i+n}[-\frac{1}{4}, \frac{1}{4}]\bar{+}\bar{a}_{i+n}\bar{b}_{i+n}[-\frac{1}{4}, 0])$
- $K = (\bar{c}\bar{a}_0\bar{b}_{2n+1}\bar{+}\bar{c}\bar{b}_0\bar{a}_{2n+1})\bar{+}\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\bar{c}\bar{a}_i\bar{b}_j\bar{+}\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\bar{c}\bar{a}_i\bar{b}_{j+n}$
  $\bar{+}\sum_{i=1}^{n}\bar{c}\bar{a}_i\bar{b}_{2n+1}\bar{+}\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\bar{c}\bar{a}_{i+n}\bar{b}_j\bar{+}\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\bar{c}\bar{a}_{i+n}\bar{b}_{j+n}$
  $\bar{+}\sum_{i=1}^{n}\bar{c}\bar{a}_{i+n}\bar{b}_{2n+1} + \bar{c}\bar{a}_{2n+1}\bar{b}_{2n+1}$

Note that since $\epsilon_{\pm}$ is propagated from *unknown* sources, its coefficient is propagated by applying multiplication other coefficients with $\bar{c} = [-1, 1]$.

Remark 1 *Introduction of Chebyshev approximation is not new. For instance, authors in [73] proposed it based on the mean-value theorem, as in the left of Fig. 5.3. Authors in [56] applied not only for products of the same noise symbols but also those of different noise symbols. However, their estimation on* x^2 *is only in the positive interval using the fact* $x - \frac{1}{4} \leq x^2 \leq x$ *for* $x \in [0, 1]$*. We newly introduce noise symbols for absolute values. The advantage is, coefficients are half compared to them, which reduce the effect of the offset* $[-\frac{1}{4}, 0]$*. Currently, we only focus on products of the same noise symbols, which is useful for computation like in Taylor expansion.*

Appendix B

Proof of Theorem 10

Let $\varphi = \{\psi_1, \cdots, \psi_n\}$ be a conjunction of *APCs* $\psi_1, \cdots, \psi_n$.

Theorem 10 If a box B strongly satisfies a conjunction φ, then the extension of ICP with the generalized IVT detects SAT of φ over B.

Proof 14 *We are going to prove that φ can be divided into two conjunctions φ_1 and φ_2 such that*

- $\varphi_1 \cup \varphi_2 = \varphi$,
- $\varphi_1 \cap \varphi_2 = \emptyset$,
- φ_1 *is IA-valid in* B,
- φ_2 *contains only equations,*
- φ_2 *and* φ_2' *are equivalent, and*
- equationsProver$(\varphi_2', B, \emptyset)$ *(Algorithm 5) terminates and returns SAT.*

Initially, $\varphi_1 = \emptyset$, $\varphi_2 = \emptyset$, and $\varphi_2' = \emptyset$. We use an auxiliary variable $C = \emptyset$ for storing a list of variables sets. We assume that the elements in φ_2' and C are ordered by the order pf the insertion. During the construction, we preserve the invariant that C is a check basis for $\{g \mid g = 0 \in \varphi_2'\}$ in B.

For $\varphi = \{\psi_1, \cdots, \psi_n\}$, we apply the case analysis below from $i = m$ to $i = 1$ in the decreasing order. For $i = m$ down to 1, there are three cases of ψ_i.

1. *$\psi_i = (x \diamond c)$. From the second condition of SS, $B(x) \subseteq \{x \mid u \in \mathbb{R}, u \diamond c\}$. Thus, $x - c \diamond 0$ is IA-valid, and add ψ_i to φ_1 (by conjunction).*
2. *$\psi_i = (x = y \circ z)$ and $|B(x)| = 1$. From the second condition of SS, $x - (y \circ z)$ is IA-valid, and add ψ_i to φ_1 (by conjunction).*
3. *$\psi_i = (x = y \circ z)$ such that x does not appear in $\varphi_1 \wedge \varphi_2$, $x \neq y$, and $x \neq z$. Add ψ_i, $x - (y \circ z) = 0$, and x to φ_2, φ_2', and C, respectively. From the second condition of SS,* ivt$(\{x\}, x - (y \circ z), B)$ *(Algorithm 6) terminates and return* True. *Thus, C remains as a check basis for polynomials in φ_2'.*

As the invariant of the procedure, φ_1 is IA-valid in B, and φ_2 and φ_2' are equivalent. It remains to show that EQUATIONSPROVER$(\varphi_2', B, \emptyset)$ *terminates and returns SAT. Since C is a check basis, a brute-force search eventually satisfies the assumptions of Theorem 8. (Q.E.D.)*

Note that the order of APCs in φ_2' does not matter for the function EQUATIONSPROVER$(\varphi_2', B, \emptyset)$. We maintain the order (used in the definition of SS) in the proof in order to show the existence of a check basis.

Appendix C

Experiments on strategy combinations

We evaluate 18 choices of SAT intended heuristics by experiments on Zankl and Meti-Tarski families in the QF_NRA division of SMT-LIB benchmark. Our choices of strategies are,

Selection of test-UNSAT APC	Selection of box (to explore):	Selection of variable:
(1) Least SAT-likelyhood.	(3) Largest number of SAT APCs.	(8) Largest sensitivity.
(2) Largest SAT-likelyhood.	(4) Least number of SAT APCs.	
	(5) Largest SAT-likelyhood.	
	(6) Least SAT-likelyhood.	
(10) Random.	(7) Random.	(9) Random.

Table C.1 shows the experimental results of the above mentioned combination. The timeout is set to 500sec, and the time is the total execution time of successful cases.

Note that (10)-(7)-(9) means all random selections. Generally, the combination (1)-(5)-(8) outperforms in terms of the number of solved problems and the running time.

Benchmark	(1)-(5)-(8)		(1)-(5)-(9)		(1)-(6)-(8)		(1)-(6)-(9)		(10)-(5)-(8)		(10)-(6)-(8)	
Matrix-1 (SAT)	20	132.72 (s)	21	21.48	19	526.76	18	562.19	21	462.57	19	155.77
Matrix-1 (UNSAT)	2	0.00	**3**	0.00	**3**	0.00	**3**	0.00	**3**	0.00	**3**	0.00
Matrix-2,3,4,5 (SAT)	**11**	632.37	1	4.83	0	0.00	1	22.50	9	943.08	1	30.48
Matrix-2,3,4,5 (UNSAT)	8	0.37	8	0.39	8	0.37	8	0.38	8	0.38	8	0.38

Benchmark	(2)-(5)-(8)		(2)-(5)-(9)		(2)-(6)-(8)		(2)-(6)-(9)		(2)-(7)-(8)		(10)-(7)-(9)	
Matrix-1 (SAT)	**22**	163.47 (s)	19	736.17	20	324.97	18	1068.40	21	799.79	21	933.39
Matrix-1 (UNSAT)	2	0.00	2	0.00	2	0.00	2	0.00	2	0.00	2	0.00
Matrix-2,3,4,5 (SAT)	5	202.37	1	350.84	1	138.86	0	0.00	0	0.00	0	0.00
Matrix-2,3,4,5 (UNSAT)	8	0.43	8	0.37	8	0.40	8	0.38	8	0.37	8	0.38

Benchmark	(1)-(3)-(8)		(1)-(4)-(8)		(2)-(3)-(8)		(2)-(4)-(8)		(10)-(3)-(8)		(10)-(4)-(8)	
Matrix-1 (SAT)	20	738.26 (s)	21	1537.9	18	479.60	21	867.99	20	588.78	19	196.21
Matrix-1 (UNSAT)	2	0.00	2	0.00	2	0.00	2	0.00	2	0.00	2	0.00
Matrix-2,3,4,5 (SAT)	0	0.00	2	289.17	1	467.12	1	328.03	1	195.18	2	354.94
Matrix-2,3,4,5 (UNSAT)	8	0.36	8	0.36	8	0.34	8	0.37	8	0.37	8	0.39

Benchmark	(1)-(5)-(8)		(1)-(5)-(9)		(10)-(5)-(8)		(10)-(7)-(9)	
Meti-Tarski (SAT, 3528)	3322	369.60 (s)	3303	425.37	**3325**	653.87	3322	642.04
Meti-Tarski (UNSAT, 1573)	1052	383.40	1064	1141.67	**1100**	842.73	1076	829.43

Table C.1: Numbers of solved problems by combinations of the strategies in QF_NRA/Zankl,Meti-Tarski benchmark