

SMT for Polynomial Constraints and Its Applications

by

TO VAN KHANH

submitted to
Japan Advanced Institute of Science and Technology
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Supervisor: Professor Mizuhito Ogawa

*School of Information Science
Japan Advanced Institute of Science and Technology*

June, 2013

Abstract

Solving polynomial constraints plays an important role in program verification, e.g., checking roundoff/overflow error with fixed point or floating point arithmetic, measures for proving termination, and loop invariant generation. Tarski proved that polynomial constraints over real numbers (algebraic numbers) are decidable, and later Collins proposed Quantifier Elimination by Cylindrical Algebraic Decomposition, which is nowadays implemented in Mathematica, Maple/SynRac, Reduce/Redlog, and QEPCAD. However, it is DEXPTIME with regard to the number of variables, and works fine in practice up to 5 variables and lower degrees. For instance, 8 variables with degree 10 already require 20–30 hours by supercomputer.

Motivated from numerous applications of polynomial constraint solving, this thesis aims to develop and build an *SMT solver for solving polynomial constraints*. We first focus on *polynomial inequality constraints* and reasons for our focus are,

- (a) in constructive analysis, for real numbers, solving equality constraints are undecidable (decidable only for algebraic numbers), whereas solving inequality constraints are decidable, i.e., $a > b$ is computable but $a = b$ is not computable,
- (b) it is possible to use approximation for solving inequality constraints, i.e., inequality constraints on real numbers are properly approximated on rational numbers, and
- (c) by approximation in the domains of rational numbers, roundoff/overflow errors can be avoided by using exact rational number arithmetic packages.

Our approaches and contributions in the thesis are summarized as follows:

- (i) We propose an approach of *iterative approximation refinement* for solving constraints, which is described as an *abstract DPLL(T) procedure* for *over/under-approximations* and *refinements* under a background theory T . For proving and disproving in the background theory T , over and under-approximations are used.

When they neither prove nor disprove, *refinements* are applied to decompose an atomic formula of an input formula, i.e., ψ to $\psi_1 \vee \psi_2$ such that $\psi \Leftrightarrow \psi_1 \vee \psi_2$. The proposed DPLL(T) procedure for over/under-approximations and refinements is sound and almost complete for solving polynomial inequality constraints.

- (ii) We instantiate *interval arithmetic* to over-approximation and *testing* to under-approximation. A new form of affine interval, called *Chebyshev Affine Interval*, is proposed. Chebyshev Affine Interval has an advantage over current *affine intervals* such that it can keep sources of computation by noise symbols of variables for high degrees, which would be useful for guiding refinements from *sensitivity* of variables.
- (iii) We propose UNSAT cores of polynomials that can improve efficiency in theory propagation. Computation of UNSAT cores in polynomials allows inferring other unsatisfiable domains when a particular domain is detected as unsatisfiability. When performing a large number of test data (i.e., a large number of variables), a proposed approach for incremental test data generation would be useful in testing.
- (iv) We propose strategies for refinements such that choices of intervals to decompose and methods to decompose an interval to smaller intervals. These strategies are guided from interval arithmetic, i.e., *sensitivity* of variables, from testing results, and test data (*tick points*) for *tick decomposition*, and from monotonical effects of variables on a polynomial (*monotonic decomposition*).
- (v) The proposed approach is implemented as the SMT solver **raSAT**, which applies MiniSat2.2 [13] as backend SAT solver, *interval arithmetic* (over-approximation theory, aiming to decide unsatisfiability), *testing* (under-approximation theory, aiming to decide satisfiability), and proposed strategies for *refinements*.
- (vi) The proposed approach is also extended for *greater-than-or-equal* (\geq) constraints, i.e., $\bigwedge_i f_i \geq 0$ is transformed to $\bigwedge_i f_i > 0$ for proving satisfiability and for proving unsatisfiability $\bigwedge_i f_i \geq 0$ is transformed to $\bigwedge_i f_i > -\delta_i$ for $\delta_i > 0$.
- (vii) We propose a non-constructive method for solving polynomial constraints including *equalities* based on *intermediate value theorem*.

Key words: interval arithmetic, affine arithmetic, SMT, polynomial constraints, testing, abstract DPLL.

To my wife, Nguyen Kieu Phuong, and my son, To Van Nhat Minh.

Acknowledgements

I would like to express my gratitude to all those who gave me the possibility to complete this thesis. In the first place, I would like to express my sincere gratitude to my supervisor, Professor Mizuhito Ogawa for his supervision, advice, assistance and guidance during the whole period of my doctoral course. Professor Mizuhito Ogawa always encouraged me to have a deep view on doing research. This helped me a lot to improve skills. He also provided me kind encouragements and supports not only in my research and but also in my life.

I would like to thank Associate Professor Nao Hirokawa for his guidance and advice during my doctoral course. He is very zealous in discussion and sharing ideas. I received a lot of valuable suggestion from him for my work.

I would like to acknowledge the Ministry of Education and Training - Vietnam (MOET) for their finance support in my study time in Japan. Without the support of MOET, I could not complete doctoral programs in Japan.

My wife, Ms. Nguyen Kieu Phuong always encouraged me to focus on research and take care about my living life. I also want to thank my little son To Van Nhat Minh. Seen him and looking his growing up everyday make me feel happy and release stress in research and life.

Last but not least, I would like to give my special thanks to my family members, especially my parents whose encouragements and sharing enabled me to complete this work. Without my family's support, I would not have finished the degree.

Contents

Abstract	i
Acknowledgements	v
1 Introduction	1
1.1 Polynomial Constraint Solving	1
1.2 Existing Approaches	2
1.3 The Proposed Approach and Contributions	3
1.4 Thesis Outline	5
2 Abstract DPLL	6
2.1 The Abstract DPLL Procedure	6
2.2 Abstract DPLL Modulo Theories	9
3 Abstract DPLL for Approximation Refinement	12
3.1 Approximation Theory	12
3.2 DPLL(T) Procedure with Over-Approximation and Under-Approximation	13
3.3 DPLL(T) Procedure with Refinements and Heuristics	14
4 Soundness and Completeness of Approximation Theories on Polynomial Inequality	16
4.1 Polynomial Inequality Constraints	16
4.2 Open Boxex as Topological Basis	17
4.3 Soundness and Completeness	18
5 Over and Under-Approximation for Intervals	21

5.1	Interval Arithmetic	21
5.1.1	Classical Interval	21
5.1.2	Affine Interval	23
	The AF form	24
	Conversion between CI and AF	24
	The AF ₁ form	26
	The AF ₂ form	27
5.1.3	Chebyshev Approximation Interval	29
5.2	Over and Under Approximation for Intervals	34
5.2.1	Interval Arithmetic as Over Approximation	34
5.2.2	Testing as Under Approximation	35
6	Strategy for Over and Under-Approximation for Intervals	38
6.1	UNSAT Core in a Polynomial Inequality	38
6.2	Incremental Test Data Generation	40
7	Strategies for Refinement	43
7.1	Selecting Intervals to Decompose	43
7.2	Interval Decomposition	45
	7.2.1 Balanced Decomposition.	45
	7.2.2 Monotonic Decomposition.	45
	7.2.3 Tick Decomposition.	47
8	SMT Framework	51
8.1	Design of SMT Framework	51
8.2	Examples	52
9	Experiments	56
9.1	Experiments on Different Measures	56
9.2	Experiments on Benchmarks of SMT-LIB	61
10	Extensions to Polynomial Equality	64
10.1	Greater-than-or-Equal Handling	64
10.2	Polynomial Equality Handling	65

10.2.1 Polynomial Equality by Intermediate Value Theorem	65
10.2.2 Extension to Multiple Equalities	67
11 Conclusions	69
11.1 Summary of the Thesis	69
11.2 Future Directions	71
11.3 Applications	72
Publications	76

List of Figures

5.1	Chebyshev approximation	30
5.2	Chebyshev approximation of x^2 and $x x $	30
5.3	Strategy for random generation of test data	36
8.1	Framework of raSAT	52
8.2	Interval decompositions by raSAT for Example 8.2.1	53
9.1	The choices of r are far from the threshold $\sqrt[n]{\frac{1}{k}}$	57
9.2	The choices of r are close to the threshold $\sqrt[n]{\frac{1}{k}}$	57
10.1	Solving an equality by intermediate value theorem	66
10.2	Solving multiple equalities	67

List of Tables

6.1	Experimental results with and without UNSAT core	40
7.1	Experimental results for selecting of intervals to decompose	44
7.2	Experimental results for different strategies of interval decomposition . . .	50
9.1	Experimental results for $\psi = x_1^n + x_2^n < 1 \wedge (x_1 - r)^n + (x_2 - r)^n < 1$. . .	58
9.2	Experimental results for $\psi = \sum_{i=1}^k x_i^n < 1 \wedge \sum_{i=1}^k (x_i - r)^n < 1$	59
9.3	Experimental results for $\psi = \psi_1 \wedge \psi_2$	61
9.4	Experimental results for Hong and Zankl family	63
10.1	Experimental results for 15 equality problems of Zankl family	66

Chapter 1

Introduction

1.1 Polynomial Constraint Solving

Many problems in hardware/software verification and analysis can be reduced to polynomial constraint solving, e.g., checking over-flow/roundoff error [25], measures for proving termination [19], loop invariant generation [8], analyzing reachability/discovering inductive invariants in hybrid systems [30], and control design problems concerning with stationary and stability [1].

Solving polynomial constraints on real numbers is decidable [32], and quantifier elimination by cylindrical algebraic decomposition (QE-CAD) [7] is known to be wide use. QE-CAD is implemented in Mathematica, Maple/SynRac, Reduce/Redlog, QEPCAD, and recently nlSAT [16]. However, it is DEXPTIME with regard to the number of variables, and in practice, it works fine up to 5 variables and lower degrees. For instance, solving a problem with 8 variables and degree 10 requires over 20 hours on a supercomputer.

An example of polynomial constraint solving is,

Example 1.1.1. Given a polynomial constraint $F = (1 < x < 3) \wedge (2 < y < 4) \wedge (x^3 - x^2y - 4 > 0)$. Is F satisfiable (SAT) or unsatisfiable (UNSAT)?

In this example, F is SAT and $x = 2.8, y = 2.1$ is a SAT instance.

1.2 Existing Approaches

Currently decision procedures for solving polynomial constraints can be considered to fall into one of five categories or combinations among them:

1. **QE-CAD.** RAHD [28] is based on the core computation of QE-CAD proposed by Tarski. It applies different versions of QE-CAD implementations such as QEPCAD-B, Reduce/Redlog. Because QE-CAD is DEXPTIME complexity in the number of variables, solving problems with a lot of variables seems to be a challenge for QE-CAD variants.
2. **Interval constraint propagation (ICP).** ICP applies interval arithmetic as an over approximation for propagating conflict in background theories. Many decision procedures are based on ICP such as RSOLVER [29] and iSAT [14], which apply classical interval (CI). To remove unsatisfiable elements, while RSOLVER develops a pruning algorithm, iSAT apply a tight interaction of SAT solver and eager theory propagation.
3. **Bit-blasting.** In this category, problems are reduced to SAT solving problems. Input formulas are bit-blasting to propositional formulas, which are then solved by SAT solver. MiniSmt [34] applies bounded bit encoding to represent rational numbers and then extend representations for some fragments of real numbers. MiniSmt can show satisfiability quickly, but due to the bound on representation, it cannot conclude unsatisfiability. UCLID [6] represents an input formula by a bit-vector formula on a given finite width of bits. UCLID also applies both under and over approximation to refine each other. Reducing number of bits to represent an input formula is regarded an under approximation, and constructing an over-approximation formula by removing some clauses from the original formula. UCLID aims at finite-precision integer arithmetic then unsatisfiable problems cannot be detected when applying for real numbers.
4. **Linearization.** Several decision procedures apply linearization for polynomial constraints and then call an SMT solver for linear constraints to solve constraints obtained from linearization. Barcelogic [4] linearizes polynomial constraints by applying case analysis which instantiates one of arguments in multiplication with finitely

possible integers in a given-bounded range. Barcelogic applies for integer domain with finite input ranges. CORD [15] uses another technique for linearization, called CORDIC (COordinate Rotation DIgital Computer). Both Barcelogic and CORD apply Yices as an SMT solver for solving linear constraints.

5. **Virtual substitution (VS).** VS method [33] is adapted for SMT solving. In SMT-RAT toolbox [10, 11], combinations of VS method and incremental fashion for SMT, less lazy and eager theory propagation, are implemented. Due to restriction in degree of variables, required degree 2 (or at most degree 4), solving constraints with higher degrees seems a challenge. Z3 [21], the winner in the category QF_NRA (Quantifier Free of Nonlinear Real Arithmetic) of SMT competition in 2011, also applies VS in combinations with ICP and decision procedures for linear arithmetic.

1.3 The Proposed Approach and Contributions

Because of high time complexity of QE-CAD and incompleteness of existing approaches, the aim of the research is to propose an *complete approach* and develop an *SMT solver for solving polynomial constraints*. We first focus on *polynomial inequality constraints* and reasons for our focus are,

- (a) in constructive analysis, for real numbers, solving equality constraints are undecidable (decidable only for algebraic numbers), whereas solving inequality constraints are decidable, i.e., $a > b$ is computable but $a = b$ is not computable,
- (b) it is possible to use approximation for solving inequality constraints, i.e., inequality constraints on real numbers are properly approximated on rational numbers, and
- (c) by approximation in the domains of rational numbers, roundoff/overflow errors can be avoided by using exact rational number arithmetic packages, e.g., the numerical packages of Ocaml.

Our approach and contributions in the thesis are summarized as follows:

- (i) We propose an approach of *iterative approximation refinement* for solving constraints, which is described as an *abstract DPLL(T) procedure* for *over/under-approximations* and *refinements* under a background theory T . For proving and

disproving in the background theory T , over and under-approximations are used. When they neither prove nor disprove, *refinements* are applied to decompose an atomic formula of the input formula, i.e., ψ to $\psi_1 \vee \psi_2$ such that $\psi \Leftrightarrow \psi_1 \vee \psi_2$. The proposed DPLL(T) procedure for over/under-approximations and refinements is sound and almost complete for solving polynomial inequality constraints.

- (ii) We instantiate *interval arithmetic* to over-approximation and *testing* to under-approximation. A new form of affine interval, called *Chebyshev Affine Interval* (CAI_1), is proposed. CAI_1 has an advantage over current *affine interval* (i.e., AF, AF_1, AF_2) such that it can keep sources of computation by noise symbols of variables (i.e., ϵ and $|\epsilon|$) for high degrees, which would be useful for guiding refinements from *sensitivity* of variables.
- (iii) We propose UNSAT cores of polynomials that can improve efficiency in theory propagation, i.e., number of clauses for learning is reduced. Computation of UNSAT cores in polynomials allows inferring other UNSAT domains when a particular domain is detected as UNSAT. When performing a large number of test data (i.e., a large number of variables), the proposed approach for incremental test data generation would be useful in testing.
- (iv) We propose strategies for refinements such that choices of interval to decompose and methods to decompose an interval to smaller intervals. These strategies are guided from interval arithmetic, i.e., *sensitivity* of variables, from testing results, and test data (*tick points*) for *tick decomposition*, and from monotonical affects of variables on a polynomial (*monotonic decomposition*).
- (v) The proposed approach is implemented as the SMT solver **raSAT**, which applies MiniSat2.2 [13] as backend SAT solver, *interval arithmetic* (over-approximation theory, aiming to decide unsatisfiability), *testing* (under-approximation theory, aiming to decide satisfiability), and proposed strategies for *refinements*.
- (vi) The proposed approach is also extended for *greater-than-or-equal* (\geq) constraints, i.e., $\bigwedge_i f_i \geq 0$ is transformed to $\bigwedge_i f_i > 0$ for proving SAT and for proving UNSAT $\bigwedge_i f_i \geq 0$ is transformed to $\bigwedge_i f_i > -\delta_i$ for $\delta_i > 0$.

- (vii) We propose a non-constructive method for solving polynomial constraints including *equalities* based on *intermediate value theorem*.

1.4 Thesis Outline

The structure of the thesis is organized as follows:

- Chapter 2 introduces the abstract DPLL procedure and abstract DPLL modulo theories (DPLL(T)).
- Chapter 3 proposes the abstract DPLL(T) for sandwiching by over/under-approximations theory and refinements.
- Soundness and completeness are discussed in Chapter 4.
- Chapter 5 presents *interval arithmetic* as over-approximation theory and *testing* as under-approximation theory. A new form of affine interval called CAI_1 is proposed in this chapter.
- Chapter 6 proposes UNSAT cores of a polynomial, which allow inferring domains of UNSAT, and an approach for incremental test data generation.
- Chapter 7 proposes strategies applied for refinements.
- Chapter 8 shows the design framework of **raSAT** and demonstrates how **raSAT** works in an example by different strategies of interval decomposition.
- Chapter 9 shows experimental results of **raSAT** on different measures and problems of the benchmarks SMT-LIB.
- We extend our approach for greater-than-or-equal constraints and constraints including equalities in Chapter 10.
- Finally, conclusions, future directions, and applications are given in Chapter 11.

Chapter 2

Abstract DPLL

2.1 The Abstract DPLL Procedure

In this section, we introduce the DPLL procedure [27] applied for searching a satisfying truth assignment for a given *conjunctive normal form* (CNF) formula F of propositional logic. The assignment is incrementally built step by step. At each step, a next assignment is deduced from a current assignment and F (*boolean constraint propagation*), or by a non-deterministic guess (*decision*) on the truth value of one of the remaining undefined variables.

As notational convention, for a finite set of atoms (propositional symbols) A , an atom $a \in A$ is a *positive literal* and $\neg a$ is a *negative literal*. The negation of a literal l , written $\neg l$, denotes $\neg a$ if l is a , and a if l is $\neg a$. A *clause* is denoted C which is a set of literals, and a *CNF formula* F is a set of clauses (it is also regarded as conjunctions of clauses $F = C_1 \wedge \dots \wedge C_n$). M is a (partial truth) *assignment*, which is sequences of literals (it is also regarded as a set of literals) such that $\{a, \neg a\} \subseteq M$ for no a . If $l \in M$ then l is *true* in M , if $\neg l \in M$ then l is *false* in M and *undefined* otherwise. M is a *full assignment* if no literal of F is undefined. The *empty assignment* is denoted \emptyset . A clause C is true in M if $C \cap M \neq \emptyset$, denoted $M \models C$, is false in M , denoted $M \models \neg C$, if all its literals are false in M , and is undefined otherwise. We write $M \models F$, if all clauses of F are true in M , which is called a *model* (a propositional model) of F . If F has no model then it is *unsatisfiable*. We write $F \models C$ if the clause C is true in all models of F . We denote $C \vee l$ for the clause including literal l and all literals of C .

A *binary relation* over *states* is denoted \Longrightarrow , where states are either *fail* or pairs of an assignment M and a CNF F , denoted $M \parallel F$. The DPLL procedure consists of rules describing *transitions* from a state to another state.

Followings are four basic DPLL procedure rules,

- **UnitPropagate:**

$$M \parallel F \wedge (C \vee l) \Longrightarrow Ml \parallel F \wedge (C \vee l) \quad \text{if} \quad \begin{cases} M \models \neg C \text{ and} \\ l \text{ is undefined in } M \end{cases}$$

- **Decide:**

$$M \parallel F \Longrightarrow Ml^d \parallel F \quad \text{if} \quad \begin{cases} l \text{ or } \neg l \text{ occurs in a clause of } F \text{ and} \\ l \text{ is undefined in } M \end{cases}$$

- **Fail:**

$$M \parallel F \wedge C \Longrightarrow \text{fail} \quad \text{if} \quad \begin{cases} M \models \neg C \text{ and} \\ M \text{ contains no decision literals} \end{cases}$$

- **Backjump:**

$$Ml^d M_1 \parallel F \Longrightarrow Ml' \parallel F \quad \text{if} \quad \begin{cases} \text{there is some clause } C \vee l' \text{ such that} \\ F \models C \vee l', \\ M \models \neg C, \\ l' \text{ is undefined in } M \text{ and} \\ l' \text{ or } \neg l' \text{ occurs in a clause of } F \end{cases}$$

The DPLL procedure takes an input CNF formula F and computes an finite sequences of the form $\emptyset \parallel F \Longrightarrow \cdots \Longrightarrow S$. It starts from the initial state $\emptyset \parallel F$ and terminates when it reaches to a final state S , which is either *fail* when F is unsatisfiable, or in the form of $M \parallel F'$, where $M \models F'$, when F is satisfiable. Note that the CNF formula F' in a final state can be different from the input F by adding some clauses during the DPLL procedure.

UnitPropagate rule (*unit propagation*) is applied for deducing the next assignment from the current assignment M and the current CNF formula F . *Decide* rule is applied for a non-deterministic guess on the truth value of an undefined literal. Note that a literal l is chosen by decide rule written by l^d . *Fail* and *backjump* rules are applied when there is a *conflict*, i.e., a clause in the current CNF formula F that is false in the current assignment M . If the CNF formula F is unsatisfiable, *fail* rule is applied when there are no decision literals in M . Otherwise *backjump* rule is applied.

Example 2.1.1. This is an example applying the basic DPLL rules for finding a model of the input CNF formula $F = (l_1 \vee \neg l_2) \wedge (\neg l_4 \vee \neg l_3) \wedge (\neg l_4 \vee \neg l_1 \vee l_5)$.

$$\begin{aligned}
\emptyset \parallel (l_1 \vee \neg l_2) \wedge (\neg l_4 \vee \neg l_3) \wedge (\neg l_4 \vee \neg l_1 \vee l_5) &\implies (Decide) \\
l_2^d \parallel (l_1 \vee \neg l_2) \wedge (\neg l_4 \vee \neg l_3) \wedge (\neg l_4 \vee \neg l_1 \vee l_5) &\implies (UnitPropagate) \\
l_2^d l_1 \parallel (l_1 \vee \neg l_2) \wedge (\neg l_4 \vee \neg l_3) \wedge (\neg l_4 \vee \neg l_1 \vee l_5) &\implies (Decide) \\
l_2^d l_1 l_4^d \parallel (l_1 \vee \neg l_2) \wedge (\neg l_4 \vee \neg l_3) \wedge (\neg l_4 \vee \neg l_1 \vee l_5) &\implies (UnitPropagate) \\
l_2^d l_1 l_4^d \neg l_3 \parallel (l_1 \vee \neg l_2) \wedge (\neg l_4 \vee \neg l_3) \wedge (\neg l_4 \vee \neg l_1 \vee l_5) &\implies (UnitPropagate) \\
l_2^d l_1 l_4^d \neg l_3 l_5 \parallel (l_1 \vee \neg l_2) \wedge (\neg l_4 \vee \neg l_3) \wedge (\neg l_4 \vee \neg l_1 \vee l_5) &\quad \text{Final state: model found}
\end{aligned}$$

In addition to basic rules, the DPLL procedure with clause learning consists of two additional rules, which are *learn* and *forget* rules.

- **Learn:**

$$M \parallel F \implies M \parallel F \wedge C \quad \text{if} \quad \begin{cases} F \models C \text{ and} \\ \text{all atoms of } C \text{ occur in } F \end{cases}$$

- **Forget:**

$$M \parallel F \wedge C \implies M \parallel F \quad \text{if} \quad F \models C$$

In these two rules, the clause C is said to be learned and forgotten, respectively. In backjump rule, the clause $C \vee l'$, called a *conflict clause*, is discovered by *implication graph* [12], which is applied for finding causes of a conflict. Then the conflict clause is learned by *learn* rule to avoid producing the same conflict. *Forget* rule is applied for free memory

by removing a clause C with low *activity* (i.e., the number of times C causes some conflict or some unit propagation) [24].

Example 2.1.2. This is an example applying *backjump* rule and *learn* rule for finding a model of the input CNF formula $F = (\neg l_1 \vee l_2) \wedge (\neg l_3 \vee \neg l_2 \vee l_4) \wedge (\neg l_3 \vee \neg l_4)$.

$$\begin{array}{ll}
\emptyset & \parallel (\neg l_1 \vee l_2) \wedge (\neg l_3 \vee \neg l_2 \vee l_4) \wedge (\neg l_3 \vee \neg l_4) \implies (Decide) \\
l_1^d & \parallel (\neg l_1 \vee l_2) \wedge (\neg l_3 \vee \neg l_2 \vee l_4) \wedge (\neg l_3 \vee \neg l_4) \implies (UnitPropagate) \\
l_1^d l_2 & \parallel (\neg l_1 \vee l_2) \wedge (\neg l_3 \vee \neg l_2 \vee l_4) \wedge (\neg l_3 \vee \neg l_4) \implies (Decide) \\
l_1^d l_2 l_3^d & \parallel (\neg l_1 \vee l_2) \wedge (\neg l_3 \vee \neg l_2 \vee l_4) \wedge (\neg l_3 \vee \neg l_4) \implies (UnitPropagate) \\
l_1^d l_2 l_3^d l_4 & \parallel (\neg l_1 \vee l_2) \wedge (\neg l_3 \vee \neg l_2 \vee l_4) \wedge (\neg l_3 \vee \neg l_4) \implies (Backjump) \\
l_1^d l_2 & \parallel (\neg l_1 \vee l_2) \wedge (\neg l_3 \vee \neg l_2 \vee l_4) \wedge (\neg l_3 \vee \neg l_4) \implies (Learn) \\
l_1^d l_2 & \parallel (\neg l_1 \vee l_2) \wedge (\neg l_3 \vee \neg l_2 \vee l_4) \wedge (\neg l_3 \vee \neg l_4) \wedge (\neg l_1 \vee \neg l_3) \implies (UnitPropagate) \\
l_1^d l_2 \neg l_3 & \parallel (\neg l_1 \vee l_2) \wedge (\neg l_3 \vee \neg l_2 \vee l_4) \wedge (\neg l_3 \vee \neg l_4) \wedge (\neg l_1 \vee \neg l_3) \quad \text{Final state: model found}
\end{array}$$

2.2 Abstract DPLL Modulo Theories

Satisfiability Modulo Theories (SMT) is a procedure to detect satisfiable instances under a background theory. Unlike SAT solving, which focuses on satisfiability (SAT) of propositional formulas, SMT aims to detect satisfiable instances in more expressive logics. SMT separates case analysis as SAT solving and a decision procedure (DP_T) for a background theory, which is applied for checking *consistency* of atoms given from SAT solving.

For example, a formula in the theory of non-linear arithmetic is $F = (x > 0 \vee x < -2) \wedge (y > 0) \wedge (x^2 + y < 4)$. Atoms of F are $x > 0$, $x < -2$, $y > 0$ and $x^2 + y < 4$. If SAT solving gives a model for F as $M = (x < -2), (y > 0), (x^2 + y < 4)$, then conjunction of literals in M is checked by DP_T . In this case, DP_T decides the satisfiability of $(x < -2) \wedge (y > 0) \wedge (x^2 + y < 4)$. If it is satisfiable, it is *T-consistent*, otherwise it is *T-inconsistent*. In this example, $(x < -2) \wedge (y > 0) \wedge (x^2 + y < 4)$ is T-inconsistent and $(x > 0) \wedge (y > 0) \wedge (x^2 + y < 4)$ is T-consistent.

We write $F \models_T \neg G$ if $F \wedge G$ is T-inconsistent.

Interaction between SAT solving and theory has *very lazy theory learning*, *less lazy*

theory learning and *eager theory propagation*, which are described below as *abstract DPLL modulo theories* [27].

- **Very lazy theory learning** interacts with theory T when a full assignment is obtained from a SAT solver. If the theory T disproves the full assignment, the SAT solver learns the clause $\neg l_1 \vee \dots \vee \neg l_n \vee \neg l$ and is started again.

$$MlM_1 \parallel F \implies \emptyset \parallel F \wedge (\neg l_1 \vee \dots \vee \neg l_n \vee \neg l) \quad \text{if} \quad \begin{cases} MlM_1 \models F, \\ \{l_1, \dots, l_n\} \subseteq M, \text{ and} \\ l_1 \wedge \dots \wedge l_n \models_T \neg l. \end{cases}$$

Note that MlM_1 is a full assignment.

- **Less lazy theory learning** interacts with theory T when a (partial) assignment is obtained from a SAT solver and the theory T refutes $l_1 \wedge \dots \wedge l_n \wedge l$, it will learn the clause $\neg l_1 \vee \dots \vee \neg l_n \vee \neg l$.

$$MlM_1 \parallel F \implies MlM_1 \parallel F \wedge (\neg l_1 \vee \dots \vee \neg l_n \vee \neg l) \quad \text{if} \quad \begin{cases} \{l_1, \dots, l_n\} \subseteq M, \\ l_1 \wedge \dots \wedge l_n \models_T \neg l, \text{ and} \\ \neg l_1 \vee \dots \vee \neg l_n \vee \neg l \notin F. \end{cases}$$

Note that MlM_1 is possibly a partial assignment and SAT solver does not need to restart when it learns a clause.

- **Eager theory propagation** interacts with theory T during the DPLL procedure of SAT solving, and the DPLL procedure continues when the theory admits the current decisions.

$$M \parallel F \implies Ml \parallel F \quad \text{if} \quad \begin{cases} M \models_T l, \\ l \text{ is undefined in } M, \text{ and} \\ l \text{ or } \neg l \text{ occurs in } F. \end{cases}$$

In our SMT solver **raSAT** we apply *very lazy theory learning* when interacting with MiniSat2.2. For proving and disproving full assignments from MiniSat, we apply *interval*

arithmetic and *testing* which are presented in Chapter 5. Certain combination with *eager theory propagation* would improve efficiency. However, it is left for future work, since it requires tighter interaction between SAT solving and background theory, which needs internal modification of MiniSat.

Chapter 3

Abstract DPLL for Approximation Refinement

In this chapter, we introduce abstract DPLL for sandwiching by over/under-approximation theories and their refinements. The DPLL(T) procedure applies over and under-approximation for proving and disproving full truth assignments in the background theory T , respectively. Refinements in the DPLL(T) procedure lead to better approximations.

3.1 Approximation Theory

We start with a general framework, and assume that a target constraint F is a CNF formula (of first-order logic). In Section 4.1, we will instantiate F as a conjunction of I and P , where I is an interval constraint, $x_1 \in (a_1, b_1) \wedge \cdots \wedge x_n \in (a_n, b_n)$, and P is a conjunction of polynomial inequalities, $\bigwedge_{i=1}^m f_i(x_1, \dots, x_n) > 0$.

In the *very lazy theory learning*, a background theory T proves or disproves full truth assignments (obtained from a SAT solver). We abuse the symbol \models_T in the following way. As notational convention, m (the lower case) denotes an instance (m is aimed at variable assignments), and M (the upper case) denotes a (full) truth assignment. We regard M as a conjunction of literals (in a background theory T).

- If an instance m of all variables appearing in F satisfies F , we denote $m \models_T F$.
- For a truth assignment, M if an instance m satisfies M we denote $m \in M$. If m satisfies F for each instance $m \in M$, we denote $M \models_T F$.

Definition 3.1.1. For a constraint F and a truth assignment M , we say that F is

- T -valid if $M \models_T F$,
- T -satisfiable (T -SAT) if $m \models_T F$ for some $m \in M$, and
- T -unsatisfiable (T -UNSAT) if $M \models_T \neg F$.

If T is clear from the context, we simply say valid, satisfiable, and unsatisfiable.

Definition 3.1.2. We say that,

- $O.T$ is an *over-approximation theory* if $O.T$ -UNSAT implies T -UNSAT, and
- $U.T$ is an *under-approximation theory* if $U.T$ -SAT implies T -SAT.

We further assume that $O.T$ -valid implies T -valid.

Later in Chapter 5, we will instantiate $O.T$ and $U.T$ with interval arithmetic and testing, respectively.

3.2 DPLL(T) Procedure with Over-Approximation and Under-Approximation

When solving a constraint F by SMT, we first project each literal (of first-order logic) in F to a boolean variable (i.e., a literal of propositional logic), and we obtain $proj(F)$, which is a CNF of propositional formula. SAT solver will give a boolean instance (if satisfiable).

We present the DPLL(T) procedure in very lazy theory learning, denoted as \implies_{VL} , and use $\models_{O.T}$ and $\models_{U.T}$ for proving and disproving a full truth assignment given from SAT solver. It works as,

- (i) M will be chosen by SAT solver and will be evaluated by $\models_{O.T}$ and $\models_{U.T}$.
- (ii) If either $\models_{O.T}$ or $\models_{U.T}$ proves SAT (*SAT rule* is applied), the DPLL(T) procedure terminates and outputs SAT.
- (iii) If $\models_{O.T}$ disproves (*very lazy theory learning rule* is applied), SAT solver will detect another M .

(iv) When $\models_{O.T}$ and $\models_{U.T}$ neither prove nor disprove, a *refinement rule* decomposes an atomic formula ψ in F to $\psi_1 \vee \psi_2$ such that they are mutually exclusive.

Assume that M is a conjunction of literals in F such that $proj(M) \models proj(F)$ (i.e., $proj(M)$ is a truth assignment returned by a SAT solver).

- **SAT rule** is applied when either $M \models_{O.T} F$, or $m \models_{U.T} F$ for some $m \in M$.

$$M \parallel F \implies_{VL} SAT \quad \text{if} \quad \begin{cases} M \models_{O.T} F, \text{ or} \\ m \models_{U.T} F \text{ for some } m \in M. \end{cases}$$

- **Fail rule** is applied when SAT solver returns UNSAT.

$$\emptyset \parallel F \implies_{VL} fail \quad \text{if } proj(F) \text{ is UNSAT}$$

- **Very lazy theory learning rule** is applied when $M \models_{O.T} \neg F$.

$$M \parallel F \implies_{VL} \emptyset \parallel (\neg l_1 \vee \dots \vee \neg l_n) \wedge F \quad \text{if} \quad \begin{cases} \{l_1, \dots, l_n\} \subseteq M \text{ and} \\ l_1 \wedge \dots \wedge l_n \models_{O.T} \neg F. \end{cases}$$

Whenever either SAT rule or fail rule are applied, the DPLL(T) procedure terminates and informs SAT or UNSAT, respectively. Note that in *very lazy theory learning rule*, if $l_1 \wedge \dots \wedge l_n$ is chosen to be minimal, it is an *UNSAT core*, which causes unsatisfiability. Finding an UNSAT core improves efficiency in theory propagation (learning) of the *DPLL(T)* procedure.

3.3 DPLL(T) Procedure with Refinements and Heuristics

Definition 3.3.1. For a constraint F , ψ is an atomic formula in F if $prof(\psi)$ is a propositional literal of $prof(F)$.

When $\models_{O.T}$ and $\models_{U.T}$ neither prove nor disprove M , a *refinement rule* decomposes an atomic formula ψ in F to $\psi_1 \vee \psi_2$ such that theory are mutually exclusive ($\psi \Leftrightarrow \psi_1 \vee \psi_2$).

- **Refinement rule** is applied when $\models_{O.T}$ and $\models_{U.T}$ neither prove nor disprove M , and F is refined to $F' = (\psi \Leftrightarrow \psi_1 \vee \dots \vee \psi_n) \wedge F$.

$$M \parallel F \Longrightarrow_{VL} \emptyset \parallel (\psi \Leftrightarrow \psi_1 \vee \dots \vee \psi_n) \wedge F \quad \begin{array}{l} \text{if } \psi \text{ is an atomic formula in } F \\ \text{and } \psi \Leftrightarrow \psi_1 \vee \dots \vee \psi_n. \end{array}$$

An example of the refinement rule is described below for a target constraint F including constraints for variable ranges, which will be presented in Section 4.1 as interval constraints, i.e., $x \in (a, b)$ represented for $a < x < b$.

Example 3.3.2. Assume that $x \in (0, 2)$ is an atomic formula in F , it is refined into $x \in (0, 2) \Leftrightarrow x \in (0, 1) \vee (x = 1) \vee x \in (1, 2)$.

$$M \parallel F \Longrightarrow_{VL} \emptyset \parallel (x \in (0, 2) \Leftrightarrow x \in (0, 1) \vee (x = 1) \vee x \in (1, 2)) \wedge F$$

A *heuristic rule* is to halt the DPLL(T) procedure by setting *termination condition isHalt*, i.e., refined too much, etc. In Section 4.1, we will give definition of *isHalt* when boxes, represented for interval constraints, are small enough, i.e., less than a given threshold.

- **Heuristic rule** learns a clause $\neg M$ when M holds *isHalt*(M)

$$M \parallel F \Longrightarrow_{VL} \emptyset \parallel \neg M \wedge F \quad \text{if } isHalt(M)$$

If heuristic rules are applied in the DPLL(T) procedure, the fail rule becomes $\emptyset \parallel F \Longrightarrow_{VL} \text{unknown}$ if *proj*(F) is UNSAT, and the DPLL (T) procedure cannot conclude UNSAT, though SAT rules correctly conclude SAT. Heuristic rule focuses on detecting SAT, when the execution time becomes too long.

Chapter 4

Soundness and Completeness of Approximation Theories on Polynomial Inequality

4.1 Polynomial Inequality Constraints

We focus on *polynomial inequality constraints* with input ranges as open boxes which is described in Definition 4.1.1.

Definition 4.1.1. A polynomial inequality constraint $F = I \wedge P$ consists of

- an interval constraint $I = x_1 \in (a_1, b_1) \wedge \cdots \wedge x_n \in (a_n, b_n)$, and
- a polynomial constraint $P = \bigwedge_{i=1}^m f_i(x_1, \dots, x_n) > 0$

for $a_i, b_i \in \mathbb{R}$ and a polynomial $f_i(x_1, \dots, x_n)$ over variables x_1, \dots, x_n . F is satisfiable if there is an instance satisfying $I \wedge P$. $f_i(x_1, \dots, x_n) > 0$ is called an atomic polynomial constraint.

By regarding I to be a truth assignment (i.e., $x_i \in (a_i, b_i)$ is regarded as a literal), we denote $I \models P$ if P is I -valid.

Example 4.1.2.

$$F = x \in (-1, 3) \wedge y \in (2, 4) \text{ wedge } (x^3y - y^4 > 0) \wedge (y^3 - xy > 0)$$

4.2 Open Boxes as Topological Basis

For mathematical simplicity, we prepare terminology of topology [5, 18].

Definition 4.2.1. Let X be a set and \mathbb{T} be a set of subsets on X . If \mathbb{T} satisfies the following conditions:

- (i) $\emptyset \in \mathbb{T}$ and $X \in \mathbb{T}$
- (ii) If $U_1 \in \mathbb{T}$ and $U_2 \in \mathbb{T}$, then $U_1 \cap U_2 \in \mathbb{T}$
- (iii) If $\mathbb{T}' \subset \mathbb{T}$ then $\cup \mathbb{T}' \in \mathbb{T}$,

\mathbb{T} is a topology on X and X is called a topological space. A member of \mathbb{T} is called an *open set* of X . A *closed set* of X is the complement of an open set. The intersection of all closed sets of X containing A is the *closure* of A , and the union of all open subsets of A is the *interior* of A .

Example 4.2.2. Let \mathbb{T} be a set of all open intervals, $\{(a, b) \mid a, b \in \mathbb{R}, a < b\}$. \mathbb{T} is a topology on \mathbb{R} and the pair (\mathbb{T}, \mathbb{R}) is a topological space. An open interval (a, b) is an open set of \mathbb{R} and a closed interval $[a, b]$ is a closed set of \mathbb{R} . The closure of A is the smallest closed set of X containing A , for instance, $\text{closure}((1, 4)) = [1, 4]$, $\text{closure}(\{(x, y) \mid x^2 - xy > 0\}) = \{(x, y) \mid x^2 - xy \geq 0\}$.

Definition 4.2.3. An *open box* of dimension n is a set $(a_1, b_1) \times \cdots \times (a_n, b_n)$ where $a_i, b_i \in \mathbb{R}, a_i < b_i$. For $\mathbf{a} = (a_1, \cdots, a_n)$ and $\mathbf{b} = (b_1, \cdots, b_n)$, we denote $(a_1, b_1) \times \cdots \times (a_n, b_n)$ by (\mathbf{a}, \mathbf{b}) .

The set of all open boxes of dimension n is a topology on \mathbb{R}^n . We will consider covering by open boxes only, i.e., for a subset U of X , a covering is a set $\{B_\lambda\}$ of open boxes such that $U \subseteq \cup B_\lambda$. A set U is *compact*, if, for each covering of U , there exists a finite subset that is a covering of U . In Euclidian space, a set U is compact if, and only if, U is a bounded closed set.

Definition 4.2.4. Let X, Y be topological spaces and $f : X \mapsto Y$ be a map from X into Y . A map f is *continuous*, if, for each open set U of Y , $f^{-1}(U)$ is an open set of X .

4.3 Soundness and Completeness

In this section we discuss about soundness and completeness of the DPLL(T) procedure, presented in Chapter 3, for polynomial inequality constraints defined in Definition 4.1.1.

Definition 4.3.1. Let F be polynomial inequality constraint, $\mathbb{S}(F) = \{x \in \mathbb{R}^n \mid F \text{ holds}\}$.

Since a polynomial is a continuous function, $\mathbb{S}(\bigwedge_{i=1}^m f_i > 0)$ is an open set. Since \mathbb{Q} is dense in \mathbb{R} , next lemma is immediate.

Lemma 4.3.2. $\exists x \in \mathbb{R}^n. F(x) \iff \exists x \in \mathbb{Q}^n. F(x)$

Lemma 4.3.2 says that proving SAT of F among real numbers is reduced to that among rational numbers.

Lemma 4.3.3. *Suppose that $a_j < b_j$ for $1 \leq j \leq n$ and f_i are polynomials. Assume $a_k < c < b_k$ for $1 \leq k \leq n$. Then, $x \in (a_1, b_1) \times \cdots \times (a_n, b_n) \wedge \bigwedge_{i=1}^m f_i > 0$ is SAT (resp. UNSAT) if, and only if, $(x \in (a_1, b_1) \times \cdots \times (a_k, c) \cdots \times (a_n, b_n) \vee x \in (a_1, b_1) \times \cdots \times (c, b_k) \cdots \times (a_n, b_n)) \wedge \bigwedge_{i=1}^m f_i > 0$ is SAT (resp. UNSAT).*

Proof. We show for the SAT case. If-part is obvious. For Only-if-part, since $\mathbb{S}(\bigwedge_{i=1}^m f_i > 0)$ is an open set, if $y \in (a_1, b_1) \times \cdots \times \{c\} \cdots \times (a_n, b_n)$ satisfies $\bigwedge_{i=1}^m f_i > 0$, there exists $x \in (a_1, b_1) \times \cdots \times (a_k, c) \cdots \times (a_n, b_n)$ (also $x \in (a_1, b_1) \times \cdots \times (c, b_k) \cdots \times (a_n, b_n)$) that satisfies $\bigwedge_{i=1}^m f_i > 0$. \square

For a polynomial inequality constraint $F = I \wedge P$, when an atomic formula $x \in (a, b)$ is refined into $x \in (a, b) \iff x \in (a, c) \vee x \in (c, b)$ (instead of $x \in (a, b) \iff x \in (a, c) \vee (x = c) \vee x \in (c, b)$), it does not change SAT (resp. UNSAT) from Lemma 4.3.3. We apply this interval decomposition as a refinement rule in abstract DPLL. Note that initially, I and P have conjunctions only. By refinements, I becomes a CNF, though P has conjunctions only. Thus, during abstract DPLL only $proj(I)$ is sent to a SAT solver, and then a full truth assignment M is in the form of $M = x_1 \in (l_1, h_1) \wedge \cdots \wedge x_n \in (l_n, h_n)$. If P is a CNF, $prof(I \wedge P)$ is sent to a SAT solver, instead.

Let us recall Example 4.1.2 for refinement rules3.

Example 4.3.4. Let $M = x \in (-1, 3) \wedge y \in (2, 4)$, $x \in (-1, 3)$ and $y \in (2, 4)$ are refined to smaller intervals.

$$M \parallel F \implies_{VL} \emptyset \parallel (x \in (-1, 3) \Leftrightarrow x \in (-1, 1) \vee x \in (1, 3)) \wedge (y \in (2, 4) \Leftrightarrow y \in (2, 3) \vee y \in (3, 4)) \wedge F$$

For a polynomial inequality constraint, termination condition *isHalt* is defined based on length of intervals, i.e., less than a given threshold.

Definition 4.3.5. For $M = x_1 \in (l_1, h_1) \wedge \dots \wedge x_n \in (l_n, h_n)$ and a bound $\delta \in \mathbb{R}$, $isHalt(M) = (h_1 - l_1 < \delta) \wedge \dots \wedge (h_n - l_n < \delta)$.

For a polynomial inequality, we fix $DPLL(T)_{A.R}$ for the DPLL(T) procedure that applies the *SAT*, *Fail*, *Very Lazy Theory Learning*, *Refinement*, and *isHalt* rules. If we set the threshold for *isHalt* rule enough small, we can conclude soundness and (restricted) completeness of $DPLL(T)_{A.R}$. Note that such threshold is not easy to compute (if we use QE-CAD algorithm, we may be able to compute), and in our **raSAT**, it is left as a heuristics.

Definition 4.3.6. Let $F = I \wedge P$ with $P = \bigwedge_{i=1}^m f_i > 0$ be a polynomial inequality constraint such that I is bounded. An over-approximation theory $O.T$ is *complete* (wrt F) if, for each $\delta > 0$ and $c = (c_1, \dots, c_n)$ satisfying I , there exists $\gamma > 0$ such that $\bigwedge_{j=1}^n x_j \in (c_j - \gamma, c_j + \gamma) \models_{O.T} \bigwedge_{i=1}^m (f_i(x) > f_i(c) - \delta \wedge f_i(x) < f_i(c) + \delta)$.

Definition 4.3.7. Let $I = \bigwedge_{j=1}^n I_j$ for $I_j = x_j \in (a_j, b_j)$ and $F = I \wedge \bigwedge_{i=1}^m f_i(x_1, \dots, x_n) > 0$. An interval decomposition strategy is *fair*, if, for each $c_j \in (a_j, b_j)$ and $\gamma > 0$, an interval decomposition for x_j for each j eventually occurs in $(c_j - \gamma, c_j + \gamma)$ (as long as neither $I' \models P$ nor $I' \models \neg P$, where I' is a decomposed box).

Theorem 4.3.8. Let $I = \bigwedge_{j=1}^n I_j$ for $I_j = x_j \in (a_j, b_j)$, $F = I \wedge \bigwedge_{i=1}^m f_i(x_1, \dots, x_n) > 0$ and $\mathbb{S}(f_i) = \{(x_1, \dots, x_n) \mid f_i(x_1, \dots, x_n) > 0\}$. Assume that an over-approximation theory $O.T$ is complete (wrt F). If the threshold for *isHalt* rule is enough small and an interval decomposition strategy is fair, the followings hold.

- **Soundness:** If $DPLL(T)_{A.R}$ reports SAT (UNSAT), F is really SAT (UNSAT).
- **Completeness:**

- If F is SAT, $DPLL(T)_{A.R}$ eventually find SAT instances
- If $\cap closure(S(f_j)) = \emptyset$ and $closure(I)$ is compact, $DPLL(T)_{A.R}$ eventually detects UNSAT.

Proof. Soundness: it is obvious from the definitions of O.T and U.T (the SAT rule).

Completeness: If F is SAT, $\cap S(f_j) \neq \emptyset$, and there exists an openbox in it with the size $\delta > 0$. If $\cap closure(S(f_j)) = \emptyset$ and $closure(I)$ is compact, let $\delta(f_i)(x) = \max\{|f_i(x) - f_1(x)|, \dots, |f_i(x) - f_m(x)|\}$. Since $\cap closure(S(f_j)) = \emptyset$, $\delta(f_i)(x) > 0$ for each i . Since $\delta(f_i)$ is continuous and $closure(I)$ is compact, $\delta(f_i)(x)$ has the minimal value for $x \in closure(I)$. Thus, $\delta_i = \min\{\delta(f_i)(x) \mid x \in I\} > 0$. We set $\delta = \min\{\delta_i\}$, and $\delta > 0$.

In either case, since $O.T$ is complete, there exists $\gamma > 0$ satisfying Definition 4.3.6. We set γ to be the threshold of *isHalt*. Since an interval decomposition is fair, decomposed boxes detect either SAT or UNSAT, respectively. \square

Note that Theorem requires only $O.T$ to be complete, since $O.T$ -valid works as $U.T$ -SAT. Later in Section 5, we apply an interval arithmetic as $O.T$ and testing as $U.T$. It is not difficult to see that an interval arithmetic is complete, and the aims of $U.T$ are,

- to obtain practical efficiency, and
- to guide interval decomposition (like “First test-UNSAT” in Section 7.1).

Chapter 5

Over and Under-Approximation for Intervals

We present *interval interval* as an over-approximation theory, $\models_{O.T}$, and *testing* as an under-approximation theory, $\models_{U.T}$, in this chapter.

5.1 Interval Arithmetic

In this section, we first show interval arithmetic in the presentations of *classical interval* (CI) [23] and *affine arithmetic* (AF, AF_1, AF_2) [20], and we propose a new form of affine interval, called Chebyshev Affine Interval (CAI_1) [17].

5.1.1 Classical Interval

A popular example of IA is Classical Interval (CI), introduced in 1960s by Moore [23], which keeps a lower bound and an upper bound for representing a range of a variable.

Definition 5.1.1. CI arithmetic consisting of $\{+, -, \bar{\times}, \bar{\div}\}$ are defined as follows:

- $(a, b) \bar{+}(c, d) = (a + c, b + d)$
- $(a, b) \bar{-}(c, d) = (a - d, b - c)$
- $(a, b) \bar{\times}(c, d) = (\min(ac, ad, bc, bd), \max(ac, ad, bc, bd))$
- $(a, b) \bar{\div}(c, d) = (a, b) \bar{\times}(\frac{1}{d}, \frac{1}{c})$ if $0 \notin (c, d)$

Followings are examples of CI arithmetic.

Example 5.1.2. Let $x \in \bar{x} = (-2, 5)$ and $y \in \bar{y} = (4, 6)$. By using CI arithmetic, the bounds of $z = x \odot y$ ($\odot \in \{+, -, \times, \div\}$) are,

- addition $z = x + y$:

$$\begin{aligned}\bar{z} &= \bar{x} \bar{+} \bar{y} \\ &= (-2, 5) \bar{+} (4, 6) \\ &= (-2 + 4, 5 + 6) \\ &= (2, 11)\end{aligned}$$

We can conclude that $z \in (2, 11)$.

- subtraction $z = x - y$:

$$\begin{aligned}\bar{z} &= \bar{x} \bar{-} \bar{y} \\ &= (-2, 5) \bar{-} (4, 6) \\ &= (-2 - 6, 5 - 4) \\ &= (-8, 1)\end{aligned}$$

then $z \in (-8, 1)$.

- multiplication $z = x \times y$:

$$\begin{aligned}\bar{z} &= \bar{x} \bar{\times} \bar{y} \\ &= (-2, 5) \bar{\times} (4, 6) \\ &= (\min(-2 \times 4, -2 \times 6, 5 \times 4, 5 \times 6), \max(-2 \times 4, -2 \times 6, 5 \times 4, 5 \times 6)) \\ &= (\min(-8, -12, 20, 30), \max(-8, -12, 20, 30)) \\ &= (-12, 30)\end{aligned}$$

then $z \in (-12, 30)$.

- division $z = x \div y$:

$$\begin{aligned}
\bar{z} &= \bar{x} \bar{\div} \bar{y} \\
&= (-2, 5) \bar{\div} (4, 6) \\
&= (-2, 5) \bar{\times} \left(\frac{1}{6}, \frac{1}{4}\right) \\
&= \left(\min\left(-\frac{2}{6}, -\frac{2}{4}, \frac{5}{6}, \frac{5}{4}\right), \max\left(-\frac{2}{6}, -\frac{2}{4}, \frac{5}{6}, \frac{5}{4}\right)\right) \\
&= \left(-\frac{1}{2}, \frac{5}{4}\right)
\end{aligned}$$

We can conclude that

$$z \in \left(-\frac{1}{2}, \frac{5}{4}\right).$$

The weakness of CI is loss of dependency among values. For instance, if $x \in \bar{x} = (2, 4)$ then, $x - x \in \bar{x} \bar{-} \bar{x} = (2, 4) \bar{-} (2, 4) = (-2, 2)$.

5.1.2 Affine Interval

Affine Interval [9, 20] introduces *noise symbols* ϵ , which are interpreted as values in $(-1, 1)$. For instance, $x \in (2, 4)$ is represented as $x = 3 + \epsilon$, and $x - x = (3 + \epsilon) - (3 + \epsilon)$ is safely evaluated to 0.

Forms of affine intervals vary by choices how to estimate multiplications. For instance, let $x \in (0, 2)$ and $y \in (1, 3)$, the affine form of x is $1 + \epsilon_1$ and the affine form of y is $2 + \epsilon_2$. Thus,

$$\begin{aligned}
x^2 - x \times y &= (1 + \epsilon_1)^2 - (1 + \epsilon_1)(2 + \epsilon_2) \\
&= (1 + 2\epsilon_1 + \epsilon_1\epsilon_1) - (2 + \epsilon_2 + 2\epsilon_1 + \epsilon_1\epsilon_2) \\
&= -1 - \epsilon_2 + \epsilon_1\epsilon_1 - \epsilon_1\epsilon_2.
\end{aligned}$$

Choices are,

- (i) $\epsilon_1\epsilon_2$ is replaced with a fresh noise symbol (AF) [9, 31],
- (ii) $\epsilon_1\epsilon_2$ is pushed into the fixed error noise symbol ϵ_{\pm} (AF_1 and AF_2) [20],
- (iii) $\epsilon_1\epsilon_2$ is replaced by $(-1, 1)\epsilon_1$ or $(-1, 1)\epsilon_2$ (EAI) [25],

(iv) $\epsilon_1\epsilon_1$ is replaced by the fixed noise symbols ϵ_+ or ϵ_- (AF_2) [20]

Followings are the affine forms of AF , AF_1 , AF_2 and their arithmetic.

The AF form

Definition 5.1.3. An AF of x is a formula of the form:

$$\ddot{x} = a_0 + \sum_{i=1}^n a_i \epsilon_i$$

where $x \in (a_0 - \sum_{i=1}^n |a_i|, a_0 + \sum_{i=1}^n |a_i|)$ and $\epsilon_i \in (-1, 1)$ is a *noise symbol*.

For AF arithmetic, linear operations (i.e., addition and subtraction) are straightforward operations and nonlinear operations such that multiplication is applied (i) for approximating non-linear parts.

Definition 5.1.4. Let $\ddot{x} = a_0 + \sum_{i=1}^n a_i \epsilon_i$ and $\ddot{y} = b_0 + \sum_{i=1}^n b_i \epsilon_i$. AF arithmetic consisting of $\{\ddot{+}, \ddot{-}, \ddot{\times}, \ddot{\div}\}$ are defined as follows:

- $\ddot{x} \ddot{+} \ddot{y} = (a_0 + b_0) + \sum_{i=1}^n (a_i + b_i) \epsilon_i$
- $\ddot{x} \ddot{-} \ddot{y} = (a_0 - b_0) + \sum_{i=1}^n (a_i - b_i) \epsilon_i$
- $\ddot{x} \ddot{\times} \ddot{y} = (a_0 b_0) + \sum_{i=1}^n (a_0 b_i + b_0 a_i) \epsilon_i + (\sum_{i=1}^n |a_i|)(\sum_{i=1}^n |b_i|) \epsilon_{n+1}$
- $\ddot{x} \ddot{\div} \ddot{y} = \ddot{x} \ddot{\times} \frac{1}{\ddot{y}}$ if $0 \notin (a_0 - \sum_{i=1}^n |b_i|, a_0 + \sum_{i=1}^n |b_i|)$

where ϵ_{n+1} is a fresh noise symbol, interpreted as a value in $(-1, 1)$, and $\frac{1}{\ddot{y}}$ is computed by Chebyshev approximation [31].

Conversion between CI and AF

- *CI to AF*: For a CI $\bar{x} = (l, h)$, a corresponding AF form of x is $\ddot{x} = \frac{h+l}{2} + \frac{h-l}{2} \epsilon_x$.
- *AF to CF*: For a given AF $\ddot{x} = a_0 + \sum_{i=1}^n a_i \epsilon_i$, the CI form of \ddot{x} is $\bar{x} = (a_0 - \sum_{i=1}^n |a_i|, a_0 + \sum_{i=1}^n |a_i|)$.

Followings are examples of AF arithmetic.

Example 5.1.5. Let $x \in (1, 5)$ and $y \in (-1, 3)$. The AF form of x and y are,

- $\ddot{x} = 3 + 2\epsilon_1$
- $\ddot{y} = 1 + 2\epsilon_2$

By using AF arithmetic, the bounds of $z = x \odot y$ ($\odot \in \{+, -, \times, \div\}$) are,

- addition $z = x + y$:

$$\begin{aligned}\ddot{z} &= \ddot{x} \dot{+} \ddot{y} \\ &= 3 + 2\epsilon_1 + 1 + 2\epsilon_2 \\ &= 4 + 2\epsilon_1 + 2\epsilon_2\end{aligned}$$

The AF projection of \ddot{z} is $(4 - 2 - 2, 4 + 2 + 2) = (0, 8)$. Then we can conclude that $z \in (0, 8)$.

- subtraction $z = x - y$:

$$\begin{aligned}\ddot{z} &= \ddot{x} \dot{-} \ddot{y} \\ &= 3 + 2\epsilon_1 - 1 - 2\epsilon_2 \\ &= 2 + 2\epsilon_1 - 2\epsilon_2\end{aligned}$$

The AF projection of \ddot{z} is $(2 - 2 - 2, 2 + 2 + 2) = (-2, 6)$, then $z \in (-2, 6)$.

- multiplication $z = x \times y$:

$$\begin{aligned}\ddot{z} &= \ddot{x} \dot{\times} \ddot{y} \\ &= (3 + 2\epsilon_1) \times (1 + 2\epsilon_2) \\ &= 3 + 6\epsilon_2 + 2\epsilon_1 + 4\epsilon_1\epsilon_2 \\ &= 3 + 4\epsilon_1 + 6\epsilon_2 + 4\epsilon_3\end{aligned}$$

The AF projection of \ddot{z} is $(3 - 4 - 6 - 4, 3 + 4 + 6 + 4) = (-11, 17)$. Then we can conclude that $z \in (-11, 17)$. Note that ϵ_3 is a new fresh noise symbol which is created by a multiplication from $\ddot{x} \dot{\times} \ddot{y}$.

- division $z = x \div y$, we can not compute the bounds of z because $0 \in (-1, 3)$.

The drawback of AF is increasing of fresh noise symbols when a number of non-linear operations is large.

The AF_1 form

Definition 5.1.6. An AF_1 of x is a formula of the form:

$$\hat{x} = a_0 + \sum_{i=1}^n a_i \epsilon_i + a_{n+1} \epsilon_{\pm}$$

where $x \in (a_0 - \sum_{i=1}^n |a_i| - a_{n+1}, a_0 + \sum_{i=1}^n |a_i| + a_{n+1})$, $\epsilon_i \in (-1, 1)$ is a *noise symbol*, $\epsilon_{\pm} \in (-1, 1)$ is the fixed error noise symbol and $a_{n+1} \geq 0$.

For AF_1 arithmetic, linear operations (i.e., addition and subtraction) are straightforward operations and nonlinear operations such that multiplication is applied (ii) for approximating non-linear parts.

Definition 5.1.7. Let $\hat{x} = a_0 + \sum_{i=1}^n a_i \epsilon_i + a_{n+1} \epsilon_{\pm}$ and $\hat{y} = b_0 + \sum_{i=1}^n b_i \epsilon_i + b_{n+1} \epsilon_{\pm}$. AF_1 arithmetic consisting of $\{\hat{+}, \hat{-}, \hat{\times}\}$ are defined as follows:

- $\hat{x} \hat{+} \hat{y} = (a_0 + b_0) + \sum_{i=1}^n (a_i + b_i) \epsilon_i + (a_{n+1} + b_{n+1}) \epsilon_{\pm}$.
- $\hat{x} \hat{-} \hat{y} = (a_0 - b_0) + \sum_{i=1}^n (a_i - b_i) \epsilon_i + (a_{n+1} + b_{n+1}) \epsilon_{\pm}$.
- $\hat{x} \hat{\times} \hat{y} = (a_0 b_0) + \sum_{i=1}^n (a_0 b_i + b_0 a_i) \epsilon_i + (|a_0| b_{n+1} + |b_0| a_{n+1}) + (\sum_{i=1}^n |a_i|)(\sum_{i=1}^n |b_i|) \epsilon_{\pm}$

Note that $a_{n+1}, b_{n+1} \geq 0$ and coefficient of ϵ_{\pm} is $(a_{n+1} + b_{n+1})$ for $\hat{-}$.

Followings are examples of AF_1 arithmetic.

Example 5.1.8. Recall from Example 5.1.5, the AF_1 form of x and y are,

- $\hat{x} = 3 + 2\epsilon_1$
- $\hat{y} = 1 + 2\epsilon_2$

By using AF_1 arithmetic, the bounds of $z = x \odot y$ ($\odot \in \{+, -, \times\}$) are,

- addition $z = x + y$ and subtraction $z = x - y$ are the same as AF .
- multiplication $z = x \times y$:

$$\begin{aligned}\hat{z} &= \hat{x} \hat{\times} \hat{y} \\ &= (3 + 2\epsilon_1) \times (1 + 2\epsilon_2) \\ &= 3 + 6\epsilon_2 + 2\epsilon_1 + 4\epsilon_1\epsilon_2 \\ &= 3 + 4\epsilon_1 + 6\epsilon_2 + 4\epsilon_{\pm}\end{aligned}$$

The AF_1 projection of \hat{z} is the same as AF ($z \in (-11, 17)$). Note that all of non-linear parts are pushed into ϵ_{\pm} .

AF_1 solves the problem of increasing fresh noise symbols in AF by approximating non-linear parts in the unique error noise symbol ϵ_{\pm} .

The AF_2 form

The AF_2 applies (ii) and (iv) for approximating non-linear operations.

Definition 5.1.9. An AF_2 of x is a formula of the form:

$$\check{x} = a_0 + \sum_{i=1}^n a_i \epsilon_i + a_{n+1} \epsilon_+ + a_{n+2} \epsilon_- + a_{n+3} \epsilon_{\pm}$$

where $x \in (a_0 - \sum_{i=1}^n |a_i| - a_{n+2} - a_{n+3}, a_0 + \sum_{i=1}^n |a_i| + a_{n+1} + a_{n+3})$, $\epsilon_i \in (-1, 1)$ is a *noise symbol*, $\epsilon_+ \in (0, 1)$ is the positive noise symbol, $\epsilon_- \in (-1, 0)$ is the negative noise symbol, $\epsilon_{\pm} \in (-1, 1)$ is the fixed error noise symbol and $a_{n+1}, a_{n+2}, a_{n+3} \geq 0$.

Definition 5.1.10. Let $\check{x} = a_0 + \sum_{i=1}^n a_i \epsilon_i + a_{n+1} \epsilon_+ + a_{n+2} \epsilon_- + a_{n+3} \epsilon_{\pm}$ and $\check{y} = b_0 + \sum_{i=1}^n b_i \epsilon_i + b_{n+1} \epsilon_+ + b_{n+2} \epsilon_- + b_{n+3} \epsilon_{\pm}$. AF_2 arithmetic consisting of $\{\check{+}, \check{-}, \check{\times}\}$ are defined as follows:

- $\check{x} \check{+} \check{y} = (a_0 + b_0) + \sum_{i=1}^n (a_i + b_i) \epsilon_i + (a_{n+1} + b_{n+1}) \epsilon_+ + (a_{n+2} + b_{n+2}) \epsilon_- + (a_{n+3} + b_{n+3}) \epsilon_{\pm}$.
- $\check{x} \check{-} \check{y} = (a_0 - b_0) + \sum_{i=1}^n (a_i - b_i) \epsilon_i + (a_{n+1} + b_{n+2}) \epsilon_+ + (a_{n+2} + b_{n+1}) \epsilon_- + (a_{n+3} + b_{n+3}) \epsilon_{\pm}$.

- $\check{x} \check{\times} \check{y} = (a_0 b_0) + \sum_{i=1}^n (a_0 b_i + b_0 a_i) \epsilon_i + K_1 \epsilon_+ + K_2 \epsilon_- + K_3 \epsilon_{\pm}$

where:

$$K_1 = \sum_{i=1, a_i b_i > 0}^{n+2} a_i b_i + \begin{cases} a_0 b_{n+1} + b_0 a_{n+1} & \text{if } a_0, b_0 \geq 0 \\ a_0 b_{n+1} - b_0 a_{n+2} & \text{if } a_0 > 0, b_0 < 0 \\ -a_0 b_{n+2} + b_0 a_{n+1} & \text{if } a_0 < 0, b_0 > 0 \\ -a_0 b_{n+2} - b_0 a_{n+2} & \text{if } a_0, b_0 < 0 \end{cases}$$

$$K_2 = \sum_{i=1, a_i b_i < 0}^{n+2} a_i b_i + \begin{cases} a_0 b_{n+2} + b_0 a_{n+2} & \text{if } a_0, b_0 \geq 0 \\ a_0 b_{n+2} - b_0 a_{n+1} & \text{if } a_0 > 0, b_0 < 0 \\ -a_0 b_{n+1} + b_0 a_{n+2} & \text{if } a_0 < 0, b_0 > 0 \\ -a_0 b_{n+1} - b_0 a_{n+1} & \text{if } a_0, b_0 < 0 \end{cases}$$

$$K_3 = \sum_{i=1}^{n+3} \sum_{j=1, j \neq i}^{n+3} |a_i b_j| + (|a_0| b_{n+3} + |b_0| a_{n+3}) + a_{n+3} b_{n+3}$$

Followings are examples of AF_2 arithmetic.

Example 5.1.11. Let $x \in (0, 2)$ and $y = 2 - x$. The AF_2 form of x and y are,

- $\check{x} = 1 + \epsilon_1$
- $\check{y} = 2 - (1 + \epsilon_1) = 1 - \epsilon_1$

By using AF_2 arithmetic, the bounds of $z = x \odot y$ ($\odot \in \{+, -, \times\}$) are,

- addition $z = x + y$:

$$\begin{aligned} \check{z} &= \check{x} \check{+} \check{y} \\ &= (1 + \epsilon_1) + (1 - \epsilon_1) \\ &= 2 \end{aligned}$$

We can conclude that $z = 2$.

- subtraction $z = x - y$:

$$\begin{aligned}
\check{z} &= \check{x} \check{-} \check{y} \\
&= (1 + \epsilon_1) - (1 - \epsilon_1) \\
&= 2\epsilon_1
\end{aligned}$$

The AF_2 projection of \check{z} is $(-2, 2)$, then $z \in (-2, 2)$.

- multiplication $z = x \times y$:

$$\begin{aligned}
\check{z} &= \check{x} \check{\times} \check{y} \\
&= (1 + \epsilon_1) \times (1 - \epsilon_1) \\
&= 1 - \epsilon_1 + \epsilon_1 - \epsilon_1 \epsilon_1 \\
&= 1 + \epsilon_-
\end{aligned}$$

The AF_2 projection of \check{z} is $(0, 1)$ ($\epsilon_- \in (-1, 0)$), then $z \in (\mathbf{0}, \mathbf{1})$. By applying CI, AF and AF_1 , we have,

$$CI : z \in (0, 4)$$

$$AF : z \in (0, 2)$$

$$AF_1 : z \in (0, 2)$$

AF_2 can improve precision when approximating product of two same noise symbols, which are pushed into the positive or negative noise symbols (ϵ_+ or ϵ_-) depending on their coefficients.

5.1.3 Chebyshev Approximation Interval

In this section, we propose a new form of affine interval, called Chebyshev Approximation Interval (CAI_1), which is based on Chebyshev approximation.

Definition 5.1.12. An CAI_1 of x is a formula of the form:

$$\check{x} = \bar{a}_0 + \sum_{i=1}^n \bar{a}_i \epsilon_i + \sum_{i=1}^n \bar{a}_{i+n} \epsilon_{i+n} + \bar{a}_{2n+1} \epsilon_{\pm}$$

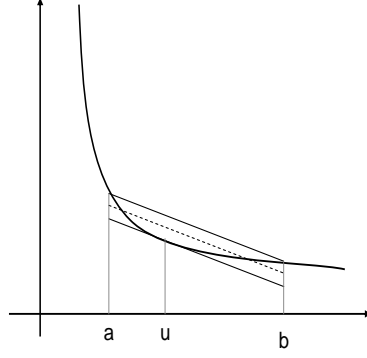


Figure 5.1: Chebyshev approximation

where $\epsilon_i \in (-1, 1)$ is a noise symbol, $\epsilon_{\pm} \in (-1, 1)$ is the fixed error noise symbol, $\epsilon_{i+n} \in (0, 1)$ represents for the absolute value $|\epsilon_i|$ of ϵ_i , and a coefficient \bar{a}_i represents for a CI.

Ideas behind are,

(v) introduction of noise symbols for absolute values and

(vi) Chebyshev approximation of x^2 with noise symbols for absolute values.

(vi) comes from the observation that, for $x \in (-1, 1)$,

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

which are explained in Figure 5.2. This observation leads symbolic manipulation on products of the same noise symbol ϵ as

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

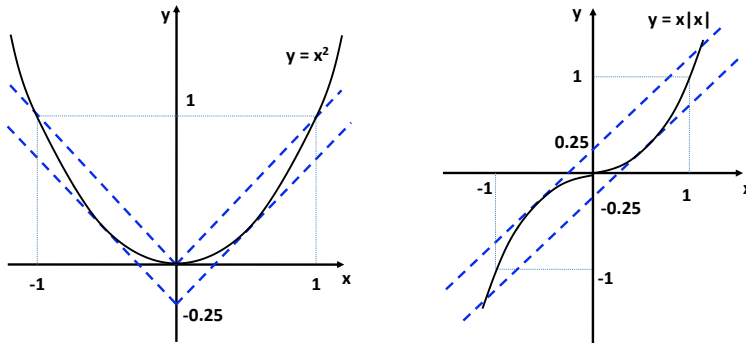


Figure 5.2: Chebyshev approximation of x^2 and $x|x|$

Definition 5.1.13. Let \mathring{x} and \mathring{y} be represented by CAI_1 form,

- $\mathring{x} = \bar{a}_0 + \sum_{i=1}^n \bar{a}_i \epsilon_i + \sum_{i=1}^n \bar{a}_{i+n} \epsilon_{i+n} + \bar{a}_{2n+1} \epsilon_{\pm}$
- $\mathring{y} = \bar{b}_0 + \sum_{i=1}^n \bar{b}_i \epsilon_i + \sum_{i=1}^n \bar{b}_{i+n} \epsilon_{i+n} + \bar{b}_{2n+1} \epsilon_{\pm}$

and $\bar{c} = (-1, 1)$. CAI_1 arithmetic consisting of $\{\overset{\circ}{+}, \overset{\circ}{-}, \overset{\circ}{\times}\}$ are defined as follows ($\bar{a}\bar{b}$ is denoted for $\bar{a}\bar{\times}\bar{b}$):

- $\mathring{x} \overset{\circ}{+} \mathring{y} = (\bar{a}_0 \bar{+} \bar{b}_0) + \sum_{i=1}^{2n} (\bar{a}_i \bar{+} \bar{b}_i) \epsilon_i + (\bar{c} \bar{a}_{2n+1} \bar{+} \bar{c} \bar{b}_{2n+1}) \epsilon_{\pm}$
- $\mathring{x} \overset{\circ}{-} \mathring{y} = (\bar{a}_0 \bar{-} \bar{b}_0) + \sum_{i=1}^{2n} (\bar{a}_i \bar{-} \bar{b}_i) \epsilon_i + (\bar{c} \bar{a}_{2n+1} \bar{+} \bar{c} \bar{b}_{2n+1}) \epsilon_{\pm}$
- $\mathring{x} \overset{\circ}{\times} \mathring{y} = K_0 + K_1 \epsilon_i + K_2 \epsilon_{i+n} + K \epsilon_{\pm}$, where $\{\bar{+}, \bar{-}, \bar{\times}\}$ are CI arithmetic, and
 - $K_0 = \bar{a}_0 \bar{b}_0 \bar{+} \sum_{i=1}^n (\bar{a}_i \bar{b}_i (-\frac{1}{4}, 0) \bar{+} \bar{a}_i \bar{b}_{i+n} (-\frac{1}{4}, \frac{1}{4}) \bar{+} \bar{b}_i \bar{a}_{i+n} (-\frac{1}{4}, \frac{1}{4}) \bar{+} \bar{a}_{i+n} \bar{b}_{i+n} (-\frac{1}{4}, 0))$
 - $K_1 = \sum_{i=1}^n (\bar{a}_0 \bar{b}_i \bar{+} \bar{a}_i \bar{b}_0 \bar{+} \bar{a}_i \bar{b}_{i+n} \bar{+} \bar{a}_{i+n} \bar{b}_i)$
 - $K_2 = \sum_{i=1}^n (\bar{a}_0 \bar{b}_{i+n} \bar{+} \bar{a}_{i+n} \bar{b}_0 \bar{+} \bar{a}_i \bar{b}_i \bar{+} \bar{a}_{i+n} \bar{b}_{i+n})$
 - $K = (\bar{c} \bar{a}_0 \bar{b}_{2n+1} \bar{+} \bar{c} \bar{b}_0 \bar{a}_{2n+1}) \bar{+} \sum_{i=1}^n \sum_{j=1, j \neq i}^n \bar{c} \bar{a}_i \bar{b}_j \bar{+} \sum_{i=1}^n \sum_{j=1, j \neq i}^n \bar{c} \bar{a}_i \bar{b}_{j+n} \bar{+} \sum_{i=1}^n \bar{c} \bar{a}_i \bar{b}_{2n+1} \bar{+}$
 $\sum_{i=1}^n \sum_{j=1, j \neq i}^n \bar{c} \bar{a}_{i+n} \bar{b}_j \bar{+} \sum_{i=1}^n \sum_{j=1, j \neq i}^n \bar{c} \bar{a}_{i+n} \bar{b}_{j+n} \bar{+} \sum_{i=1}^n \bar{c} \bar{a}_{i+n} \bar{b}_{2n+1} + \bar{c} \bar{a}_{2n+1} \bar{b}_{2n+1}$

Note that ϵ_{\pm} is propagated from *unknown* sources, then its coefficient is computed by applying multiplication other coefficients with $\bar{c} = (-1, 1)$.

Remark 5.1.14. Introduction of Chebyshev approximation is not new. For instance, Stolfi [31] proposed it based on the mean-value theorem, as in the Figure 5.1. Miyajima et al. [22] 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 of high degrees like in Taylor expansion.

Roughly speaking that CAI_1 applies (ii), (v) and (vi). Followings are examples of CAI_1 arithmetic.

Example 5.1.15. Let $f = x^3 - 3x + x^2$ with $x \in (-3, 1)$. The CAI_1 form of x is $\overset{\circ}{x} = -1 + 2\epsilon$.

$$\begin{aligned}
\overset{\circ}{f} &= \overset{\circ}{x}^3 - 3\overset{\circ}{x} + \overset{\circ}{x}^2 \\
&= (-1 + 2\epsilon) \overset{\circ}{\times} (-1 + 2\epsilon) \overset{\circ}{\times} (-1 + 2\epsilon) \overset{\circ}{-} (-3 + 6\epsilon) \overset{\circ}{+} (-1 + 2\epsilon) \overset{\circ}{\times} (-1 + 2\epsilon) \\
&= (1 - 4\epsilon + 4\epsilon^2) \overset{\circ}{\times} (-1 + 2\epsilon) \overset{\circ}{-} (-3 + 6\epsilon) \overset{\circ}{+} (1 - 4\epsilon + 4\epsilon^2) \\
&= (1 - 4\epsilon + 4(|\epsilon| + (-\frac{1}{4}, 0))) \overset{\circ}{\times} (-1 + 2\epsilon) \overset{\circ}{-} (-3 + 6\epsilon) \overset{\circ}{+} (1 - 4\epsilon + 4(|\epsilon| + (-\frac{1}{4}, 0))) \\
&= (1 - 4\epsilon + 4|\epsilon| + (-1, 0)) \overset{\circ}{\times} (-1 + 2\epsilon) \overset{\circ}{-} (-3 + 6\epsilon) \overset{\circ}{+} (1 - 4\epsilon + 4|\epsilon| + (-1, 0)) \\
&= ((0, 1) - 4\epsilon + 4|\epsilon|) \overset{\circ}{\times} (-1 + 2\epsilon) \overset{\circ}{-} (-3 + 6\epsilon) \overset{\circ}{+} ((0, 1) - 4\epsilon + 4|\epsilon|) \\
&= ((-1, 0) + (0, 2)\epsilon + 4\epsilon - 8\epsilon^2 - 4|\epsilon| + 8\epsilon|\epsilon|) \overset{\circ}{-} (-3 + 6\epsilon) \overset{\circ}{+} ((0, 1) - 4\epsilon + 4|\epsilon|) \\
&= ((-1, 0) + (0, 2)\epsilon + 4\epsilon - 8(|\epsilon| + (-\frac{1}{4}, 0)) - 4|\epsilon| + 8(\epsilon + (-\frac{1}{4}, \frac{1}{4}))) \overset{\circ}{-} (-3 + 6\epsilon) \overset{\circ}{+} ((0, 1) - 4\epsilon + 4|\epsilon|) \\
&= ((-1, 0) + (0, 2)\epsilon + 4\epsilon - 8|\epsilon| - (-2, 0) - 4|\epsilon| + 8\epsilon + (-2, 2)) \overset{\circ}{-} (-3 + 6\epsilon) \overset{\circ}{+} ((0, 1) - 4\epsilon + 4|\epsilon|) \\
&= ((-3, 4) + (12, 14)\epsilon - 12\epsilon) \overset{\circ}{-} (-3 + 6\epsilon) \overset{\circ}{+} ((0, 1) - 4\epsilon + 4|\epsilon|) \\
&= (0, 8) + (2, 4)\epsilon - 8|\epsilon|
\end{aligned}$$

Note that we write a real number r for a representation of a CI (r, r) (i.e., 4 for $(4, 4)$).

For CAI_1 projection, we apply *case analysis* for $|\epsilon|$, which are $\epsilon \in (0, 1)$ and $\epsilon \in (-1, 0)$,

- if $\epsilon \in (0, 1)$ then $\overset{\circ}{f} = (0, 8) + (2, 4)\epsilon - (8, 8)\epsilon = (0, 8) + (-6, -4)\epsilon$. The projection of $\overset{\circ}{f}$ is $(-6, 8)$.
- if $\epsilon \in (-1, 0)$ then $\overset{\circ}{f} = (0, 8) + (2, 4)\epsilon + (8, 8)\epsilon = (0, 8) + (10, 12)\epsilon$. The projection of $\overset{\circ}{f}$ is $(-12, 8)$.

For $\epsilon \in (-1, 1)$, the projection of $\overset{\circ}{f}$ is $(-6, 8) \cup (-12, 8) = (-12, 8)$, then $f \in \mathbf{(-12, 8)}$. In comparison with CI , AF_1 and AF_2 , we have the following results:

$$CI : f \in (-33, 27)$$

$$AF_1 : f \in (-25, 31)$$

$$AF_2 : f \in (-13, 19)$$

In Example 5.1.15, the affine form of x is $-1 + 2\epsilon$. When approximating for ϵ^3 , AF_2 pushes it into the fixed error noise symbol (ϵ_{\pm}) while CAI_1 estimates it by a form of ϵ , which allows to keep information about sources of computation. In comparison with AF_2 , CAI_1 can keep sources of computation for high degrees (i.e., degrees ≥ 3), whereas AF_2 can handle up to degree 2.

We present below two additional examples to compare the results of CAI_1 with CI , AF_1 and AF_2 . The first one is a polynomial of degrees 4 and the second one is a *Taylor expansion* of $\sin(x)$ function, which is expanded to degree 9. In both of them, CAI_1 gives the best bounds.

Example 5.1.16. Given $f = (x^2 - 2y^2 + 7)^2 + (3x + y - 5)^2$ with $x \in (-1, 1)$ and $y \in (-2, 0)$, the bounds of f computed by CI , AF_1 , AF_2 and CAI_1 are,

$$CI : (-12, 164)$$

$$AF_1 : (-98, 220)$$

$$AF_2 : (-53, 191)$$

$$CAI_1 : (-4.6875, 163.25)$$

Example 5.1.17. Given $\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!}$ with $x \in (0, 0.523598)$ (x is ranged from 0 to $\frac{\pi}{6}$), the bounds of $\sin(x)$ computed by CI , AF_1 , AF_2 and CAI_1 are,

$$CI : 10^{-6}(-23926.630584, 523925.958917)$$

$$AF_1 : 10^{-6}(-6290.490992, 523927.832027)$$

$$AF_2 : 10^{-6}(-6188.005805, 514955.797111)$$

$$CAI_1 : 10^{-6}(-1591.614677, 503782.471931)$$

When IA has a noise symbol ϵ , we define *sensitivity* [25] of a variable as the absolute value of the coefficient of corresponding ϵ . CAI_1 can keep information about sources of computation for high degrees. For instance, in CAI_1 form of Example 5.1.18, the coefficient 3 of $|\epsilon_1|$ has the largest sensitivity, which indicates x is the most influential.

Example 5.1.18. Let $f = x^3 - 2xy$ with $x \in (0, 2)$ ($x = 1 + \epsilon_1$) and $y \in (1, 3)$ ($y = 2 + \epsilon_2$), we have,

- by AF_2 , $\check{f} = -3 - \epsilon_1 - 2\epsilon_2 + 3\epsilon_+ + 3\epsilon_{\pm}$ and the bound of f is estimated as $(-9, 6)$,

- by CAI_1 , $\mathring{f} = (-4, -\frac{11}{4}) + (-\frac{1}{4}, 0)\epsilon_1 - 2\epsilon_2 + 3|\epsilon_1| + (-2, 2)\epsilon_{\pm}$ and the bound of f is estimated as **(-8,4.5)**.

5.2 Over and Under Approximation for Intervals

5.2.1 Interval Arithmetic as Over Approximation

Interval arithmetic (IA) is applied for estimating bounds of polynomials under a given input range, and we use it as an over-approximation theory. We instantiate IA to OT in Chapter 3, and obtain the definition below.

Definition 5.2.1. Given an interval constraint $I = x_1 \in (a_1, b_1) \wedge \cdots \wedge x_n \in (a_n, b_n)$, a polynomial constraint P of the form

$$\bigwedge_{i=1}^m f_i(x_1, \dots, x_n) > 0,$$

Let $f_i^l(x_1, \dots, x_n)$ and $f_i^u(x_1, \dots, x_n)$ be lower and upper bounds estimated by IA when x_i holds the interval constraint I . We say,

- P is *IA-VALID* under I , if IA evaluates $\forall i \in (1, m). f_i^l(x_1, \dots, x_n) > 0$,
- P is *IA-UNSAT* under I , $\exists i \in (1, m). f_i^u(x_1, \dots, x_n) \leq 0$, and
- P is *IA-SAT* under I , if $(\exists j \in (1, m). f_j^l(x_1, \dots, x_n) \leq 0) \wedge (\bigwedge_{i=1}^m f_i^u(x_1, \dots, x_n) > 0)$.

IA-VALID and IA-UNSAT safely reason satisfiability (SAT) and unsatisfiability (UNSAT), respectively. However, IA-SAT cannot conclude SAT. IA-SAT is regarded as *unknown* and shifted to under-approximation (testing) for finding SAT solution. For interval arithmetic, we apply affine intervals (AF, AF_1, AF_2) [20] and Chebyshev Affine Interval (CAI_1).

Example 5.2.2. Given an interval constraint $I = x \in (0, 2) \wedge y \in (1, 3)$ and three polynomial constraints,

- $(f_1 = x^5 - 3x^2y + y^3 + 20) > 0$
- $(f_2 = xy^2 - 2xy - y^3 - 7) > 0$
- $(f_3 = x^2y^3 - y^4) > 0$

applying CAI_1 arithmetic, the bounds of polynomials are

- $f_1 \in (0.9375, 67.75)$,
- $f_2 \in (-38.75, 0)$ and
- $f_3 \in (-131.5, 94.125)$

Then, we can conclude that

- $f_1 > 0$ is *IA-VALID* under I ,
- $f_2 > 0$ is *IA-UNSAT* under I and
- $f_3 > 0$ is *IA-SAT* under I .

5.2.2 Testing as Under Approximation

We instantiate testing to $U.T$ in Chapter 3, and obtain the definition below.

Definition 5.2.3. Given an interval constraint $I = x_1 \in (a_1, b_1) \wedge \cdots \wedge x_n \in (a_n, b_n)$, a polynomial constraint P of the form

$$\bigwedge_{i=1}^m f_i(x_1, \dots, x_n) > 0,$$

and a choice function $\theta : (\mathbb{R} \times \mathbb{R})^n \rightarrow \mathbb{R}^n$ with $\theta(I) \in (a_1, b_1) \times \cdots \times (a_n, b_n)$.

For a finite set Θ of choice functions, we say

- P is *Test-SAT* under I if P holds for some $\theta \in \Theta$, and
- P is *Test-UNSAT* under I if P never hold for each $\theta \in \Theta$.

The set Θ of choice functions in definition 5.2.3 is a set of test data. Note that Test-SAT implies SAT and Test-UNSAT does not imply UNSAT, which will be applied for refinements (narrowing ranges in an interval constraint).

There are two immediate strategies [26] to generate random test data.

Definition 5.2.4. For an interval (l, h) and $k \geq 1$,

- the **k -random ticks** are $\{c_1, \dots, c_k\}$, and
- the **k -periodic ticks** are $\{c, c + \Delta, \dots, c + (k - 1)\Delta\}$,

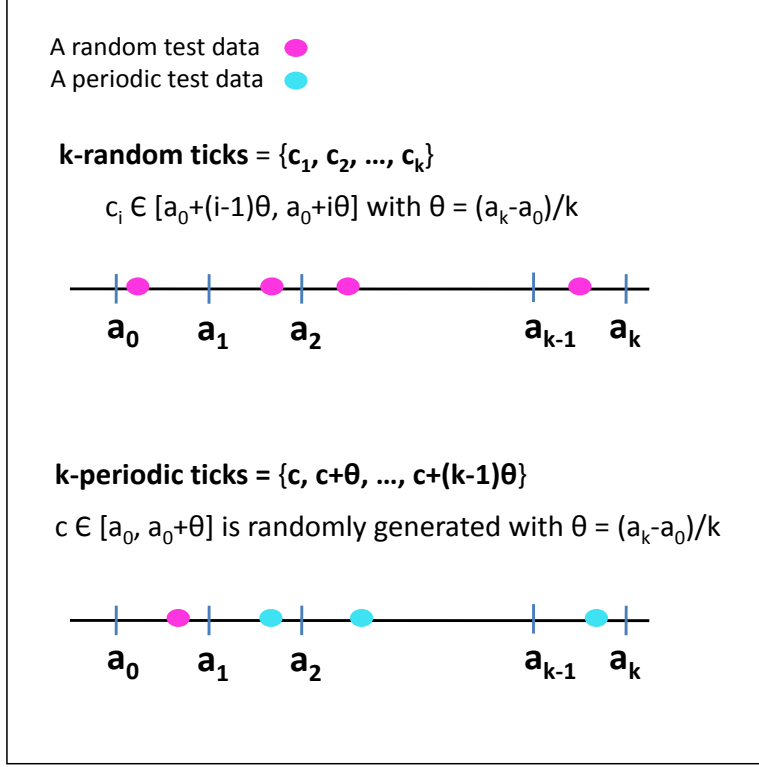


Figure 5.3: Strategy for random generation of test data

where $\Delta = \frac{h-l}{k}$, and $c \in (l, l + \Delta)$, $c_i \in (l + (i-1)\Delta, l + i\Delta)$ are randomly generated (with $i \in \{1, \dots, k\}$).

Example 5.2.5. Recall from Example 5.2.2, for an interval constraint $I = x \in (0, 2) \wedge y \in (1, 3)$ and a polynomial constraint $(f_3 = x^2y^3 - y^4) > 0$. Because $f_3 > 0$ is IA-SAT, it is shifted to testing. By applying 2-random ticks for generating test data,

- if the test data in Θ_1 is $\{x = 0.5, x = 1.2, y = 1.5, y = 2.5\}$

$$x = 0.5, y = 1.5 \quad f_3 = x^2y^3 - y^4 = -4.21875$$

$$x = 0.5, y = 2.5 \quad f_3 = x^2y^3 - y^4 = -35.15625$$

$$x = 1.2, y = 1.5 \quad f_3 = x^2y^3 - y^4 = -0.2025$$

$$x = 1.2, y = 2.5 \quad f_3 = x^2y^3 - y^4 = -16.5625$$

then $f_3 > 0$ is Test-UNSAT in Θ_1 .

- If the test data in Θ_2 is $\{x = 0.8, x = 1.7, y = 1.5, y = 2.5\}$

$$x = 0.8, y = 1.5 \quad f_3 = x^2y^3 - y^4 = -2.9025$$

$$x = 0.8, y = 2.5 \quad f_3 = x^2y^3 - y^4 = -29.0625$$

$$x = 1.5, y = 1.5 \quad f_3 = x^2y^3 - y^4 = \mathbf{2.53125}$$

$$x = 1.5, y = 2.5 \quad f_3 = x^2y^3 - y^4 = -3.90625$$

then $f_3 > 0$ is Test-SAT in Θ_2 by the test data $x = 1.5$ and $y = 1.5$. We can conclude that $x \in (0, 2) \wedge y \in (1, 3) \wedge (f_3 = x^2y^3 - y^4) > 0$ is SAT.

In our implementation of the SMT solver **raSAT**, we apply *k-random ticks* in Definition 5.2.4 for generating test data of each variable.

Chapter 6

Strategy for Over and Under-Approximation for Intervals

6.1 UNSAT Core in a Polynomial Inequality

The aim of *very lazy theory learning* is to remove UNSAT domains from searching domains. If we can infer other UNSAT domains from a particular UNSAT domain, it would be useful in theory propagation. UNSAT cores are applied as a way to infer UNSAT domains.

An UNSAT core is defined as a minimal set $M_0 = \{l_1, \dots, l_n\}$ that disproves P (w.r.t. $\models_{O.T.}$) in the very lazy theory learning rule (Section 3.2). To obtain a precise minimal M_0 is not easy. As a strategy to obtain smaller M_0 , we introduce an UNSAT core \hat{f} of a polynomial f based on the IA-UNSAT judgment. Then, M_0 is selected as literals in M corresponding to variables in \hat{f} .

Definition 6.1.1. \hat{f} is an UNSAT core of a polynomial f if IA-UNSAT of $\hat{f} > 0$ implies IA-UNSAT of $f > 0$.

Definition 6.1.1 says that, for an interval constraint I , if $\hat{f} > 0$ is IA-UNSAT under I , then $f > 0$ is IA-UNSAT under I .

To compute an UNSAT core \hat{f} of a polynomial f , we consider sub polynomials of f that cause unsatisfiability of the polynomial constraint $f > 0$. Ideas are,

- first, f is represented by two sub polynomials f_1 and f_2 from monomials of f such that $f = f_1 + f_2$,

- if $f_2 \leq 0$ for all values of its variables, then f_1 is an UNSAT core of f .

Example 6.1.2. For a polynomial inequality ($f = x^2 - xy - xz > 0$ with $x, y, z \in (0, \infty)$), f consists of 2 UNSAT cores, which are,

- $\hat{f}_1 = x^2 - xy$, because $f = \hat{f}_1 - xz$ and $-xz \leq 0$ for $x, z \in (0, \infty)$
- $\hat{f}_2 = x^2 - xz$, because $f = \hat{f}_2 - xy$ and $-xy \leq 0$ for $x, y \in (0, \infty)$

Finding UNSAT cores improves efficiency in theory propagation (learning) of the DPLL(T) procedure. It only learns minimal clauses, clauses with minimal numbers of literals. The example below demonstrates uses of applying UNSAT cores for the very lazy theory learning rule.

Example 6.1.3. For a polynomial constraint (taken from *Hong* benchmark of SMT-LIB [2]) $P = (1 - x_0^2 - x_1^2 - \dots - x_9^2 > 0) \wedge (x_0 x_1 \dots x_9 - 1 > 0)$. Interval constraints is presented in the form of $x \in (-\infty, -1) \vee x \in (-1, 1) \vee x \in (1, \infty)$ for a variable x .

Considering a polynomial $f = 1 - x_0^2 - x_1^2 - \dots - x_9^2$, UNSAT cores of f are,

$$\begin{array}{cccccc}
1 - x_0^2 & 1 - x_0^2 - x_1^2 & 1 - x_0^2 - x_2^2 & 1 - x_0^2 - x_1^2 - x_2^2 & \dots & \\
1 - x_1^2 & 1 - x_1^2 - x_2^2 & 1 - x_1^2 - x_3^2 & 1 - x_1^2 - x_2^2 - x_3^2 & \dots & \\
1 - x_2^2 & 1 - x_2^2 - x_3^2 & 1 - x_2^2 - x_4^2 & 1 - x_2^2 - x_3^2 - x_4^2 & \dots & \\
1 - x_3^2 & 1 - x_3^2 - x_4^2 & 1 - x_3^2 - x_5^2 & 1 - x_3^2 - x_4^2 - x_5^2 & \dots & \\
\dots & & & & & \\
1 - x_9^2 & & & & &
\end{array}$$

Assume that $M = (x_0 \in (1, \infty)) \wedge \dots \wedge (x_9 \in (1, \infty))$ is chosen from SAT solver. P is IA-UNSAT under M , then the very lazy learning rule learns the clause $\neg(x_0 \in (1, \infty)) \vee \dots \vee \neg(x_9 \in (1, \infty))$ as usual. By applying UNSAT cores, the very lazy theory learning rule learns only clauses with minimal number of literals, which are $\neg(1 - x_0^2)$, $\neg(1 - x_1^2)$, \dots and $\neg(1 - x_9^2)$. These clauses help to remove a large UNSAT domains from the searching domains. Note that we consider the UNSAT cores $1 - x_0^2, 1 - x_1^2, \dots$ rather than others, i.e., $1 - x_0^2 - x_1^2, 1 - x_0^2 - x_2^2, \dots$ because of its minimal variables are more useful in theory propagation.

UNSAT core: on/off	SAT		UNSAT	
	No. Problems	Time(s)	No. Problems	Time(s)
With UC	37	983.921	4	0.060
Without UC	38	1480.812	4	0.060

Table 6.1: Experimental results with and without UNSAT core

On the other hand, learning clauses from UNSAT cores avoids learning a large number of other clauses. For instance, learning the clause $\neg(1 - x_0^2)$ prevents 3^9 clauses from being learning as usual (i.e., for the remaining 9 variables and 3 choices $(-\infty, -1), (-1, 1), (1, \infty)$ for each, then their combinations are 3^9 clauses as total).

We compare efficiency of **raSAT** when applying and not applying UNSAT core, and their results are shown in Table 6.1. The problems in the experiment are taken from 151 inequality problems of *Zankl* family (in the division QF_NRA of SMT-LIB [2]). Among them, 41 problems are solved by UNSAT core and 42 for not applying UNSAT core. When applying UNSAT core, the number of detected problems is not improved, but we observed that running time is slightly improved (by examination of running time for each detected problem). At this moment, our implementation for UNSAT core is not well written. For identifying UNSAT cores of an UNSAT API we evaluate all sub polynomials of the API, and this process could take much time. Thus we plan to optimize for this step.

6.2 Incremental Test Data Generation

Performing a large number of test data generations affects efficiency. For instance, if we consider a polynomial constraint with 30 variables and we generate 2 test data for each variable, we have 2^{30} test data as total, which is intractable. The ideas for incremental test data generation are,

- (i) an atomic polynomial inequality (API)-wise test data generation with dynamic sorting of IA-SAT APIs, and
- (ii) thinning test data that does not satisfy an API.

Note that, during test data generation, an interval decomposition is fixed, and test data are generated for IA-SAT APIs only. Let $\{f_i > 0\}$ be the set of IA-SAT APIs, and let

$Var(f_i)$ be the set of variables appearing in an atomic polynomial inequality f_i .

For an API-wise test data generation, an ordering of testing of IA-SAT APIs affects the efficiency. Our ideas are,

- API with a smaller variable set,
- bottleneck API w.r.t. dependency ($Var(f_i) \subseteq Var(f_j)$), and
- API with a smaller additional test data generation

have priority. To formalize them, let $DEP_{f_i} = \{f_j \mid f_j \in P \wedge Var(f_i) \subseteq Var(f_j)\}$ and $dep_{f_i} = |DEP_{f_i}|$. Then, during an API-wise test data generation, $\{f_j\}$ is dynamically sorted at the choice of next API to hold

- (a) $Var(f_i) \subset Var(f_j)$ implies $i \leq j$,
- (b) dep_{f_1} is the largest, and
- (c) if, for some $j < m$, $Var(f_m) \subseteq \bigcup_{i=1}^j Var(f_i)$ and $\forall n. Var(f_n) \not\subseteq \bigcup_{i=1}^j Var(f_i)$, then $m \leq n$,

An API-wise test data generation requires storing previous test results of tested APIs. To reduce stored test results, test data refuting APIs are removed. When they become empty, it returns Test-UNSAT, and shifts the API that refutes all test data to the refinement.

Example 6.2.1. Let $P = (f_1 = 2x - y^2 - 2 > 0) \wedge (f_2 = x^2 - 1 > 0) \wedge (f_3 = xy - yz - zx > 0) \wedge (f_4 = u^2 - x^2y > 0) \wedge (f_5 = 2yv^2 - ux^2 - 1 > 0)$ with $x, y, z, u, v \in (0, 2)$ and 2-random ticks are applied for testing. We have, $dep_{f_1} = 3$, $dep_{f_2} = 5$, $dep_{f_3} = 2$, $dep_{f_4} = 2$ and $dep_{f_5} = 1$.

- First, $f_2 = x^2 - 1 > 0$ is chosen, since dep_{f_2} is the largest. Assume that generated test data are $\{x = 1.2, x = 0.5\}$. The satisfiable test set for $x^2 - 1 > 0$ is $\{x = 1.2\}$.
- Next $f_1 = 2x - y^2 - 2 > 0$ is chosen, since $\{y\}$ is a smaller set of additional variables and $Var(f_1) \subset Var(f_3), Var(f_4), Var(f_5)$. Assume that generated test data are $\{y = 1.4, y = 0.5\}$. The satisfiable test set $2x - y^2 - 2 > 0$ becomes $\{x = 1.2, y = 0.5\}$.

- The API $f_3 = xy - yz - zx > 0$ is chosen (we can choose either $f_3 = xy - yz - zx > 0$ or $f_4 = u^2 - x^2y > 0$ because $Var(f_3) \subset Var(f_5)$ and $Var(f_4) \subset Var(f_5)$). Assume that generated test data are $\{z = 0.8, z = 0.3\}$ and the satisfiable test set is $\{x = 1.2, y = 0.5, z = 0.3\}$.
- For $f_4 = u^2 - x^2y > 0$, assume that test data are $\{u = 1.05, u = 0.25\}$ and the satisfiable test set is $\{x = 1.2, y = 0.5, z = 0.3, u = 1.05\}$.
- Finally, for $f_5 = 2yv^2 - ux^2 - 1 > 0$, assume that generated test data are $\{v = 0.7, v = 1.3\}$. Neither satisfies it and Test-UNSAT is reported, and $f_5 = 2yv^2 - ux^2 - 1 > 0$ is shifted to interval decomposition. If generated test data are $\{v = 1.13, v = 1.77\}$, $\{x = 1.2, y = 0.5, z = 0.3, u = 1.05, v = 1.77\}$ satisfies it and reports SAT.

Example 6.2.1 shows that instead of generating 2^5 test data at beginning, number of test data kept in memory decreases much by incremental test data generation based on an ordering of IA-SAT APIs for testing.

Chapter 7

Strategies for Refinement

Similar to explosion of test data generation, interval decomposition may cause exponential explosion of boxes. We need to consider the choice of intervals to decompose, and how to decompose an interval.

7.1 Selecting Intervals to Decompose

The choice of intervals to decompose consists of two steps.

- (a) Choose an API such that its variables are candidates for interval decomposition.
- (b) Among variables of the selected API, choose influential ones.

(a) follows incremental test data generation in Section 6.2 where APIs are sorted based on their dependencies on variables. When an API $f_j > 0$, selected from the sorted APIs, refutes all generated test data, it reports Test-UNSAT. The API $f_j > 0$ is called *the first Test-UNSAT API*. Then, variables appearing in f_j are candidates for interval decompositions, since f_j is a direct cause of Test-UNSAT. In Example 6.2.1, Test-UNSAT of $2yv^2 - ux^2 - 1 > 0$ is reported with $\{v = 0.7, v = 1.3\}$, and $x, y, u, v \in (0, 2)$ (variables of the first Test-UNSAT API) become candidates for interval decompositions.

For (b), among variables in the selected API $f_j > 0$, we further filter variables that have sensitivity (Example 5.1.18) beyond a threshold, since they are expected to be more influential. Sensitivity is detected by previous IA-SAT detection phase.

Selecting Intervals to decompose	SAT		UNSAT	
	No. Problems	Time(s)	No. Problems	Time(s)
Random choice of an API	20	123.623	4	0.060
The first Test-UNSAT API	38	1480.812	4	0.060

Table 7.1: Experimental results for selecting of intervals to decompose

Among presented strategies, only (a) is implemented in current **raSAT** ((b) is not yet implemented), and we select all variables in the first Test-UNSAT API for interval decomposition.

Table 7.1 shows experimental results when we apply two different selections for selecting an API whose variables are candidate for intervals decomposition in **raSAT**. One is random choice of an API among IA-SAT APIs and the other is the first Test-UNSAT API. Then, all variables in the selected API are applied for interval decomposition. The problems chosen for the experiment are 151 inequality problems of *Zankl* family (in the division QF_NRA of SMT-LIB [2]). The number of solved problems and their total running time are indicated in SAT and UNSAT columns of Table 7.1.

Selecting the first Test-UNSAT API gives a better result than random choice of an API, i.e., 38 SAT problems are detected by selecting the first Test-UNSAT API, only 20 SAT problems are detected by random choice. The experiment shows that choice of variables for interval decomposition affects efficiency, and in **raSAT**, we apply a guidance from testing results (the first Test-UNSAT API), which would be useful because,

- an API that is more difficult to find SAT instances, i.e., a Test-UNSAT API, has high priority for examination. If the API is UNSAT, **raSAT** only applies interval decomposition for its variables without considering others. If it is SAT, **raSAT** quickly narrows its SAT domains.
- UNSAT domains are quickly removed because interval decomposition for variables of Test-UNSAT APIs helps to identify UNSAT domains.

iSAT [14], the SMT solver for polynomial constraints, applies both interval arithmetic (classical interval) and interval decomposition too, but it does not mention clearly about how it choose variables for decomposition. Choice of variables for interval decomposition is quite important when solving problems with a large number of variables, i.e., 30, 40,

> 100 variables like problems of Zankl family. Without testing we can not obtain such guidance.

7.2 Interval Decomposition

7.2.1 Balanced Decomposition.

Balanced decomposition decompose an interval into two intervals exactly half.

Definition 7.2.1. For an interval $x \in (a, b)$, a balanced decomposition is

$$D_b(x \in (a, b)) = \{x \in (a, \frac{a+b}{2}), x \in (\frac{a+b}{2}, b)\}$$

For example, $x \in (0, 2)$ is decomposed into $x \in (0, 1)$ and $x \in (1, 2)$ by balanced decomposition and we add the following clauses to the SAT solver,

$$\begin{aligned} & x \in (0, 2) \Rightarrow x \in (0, 1) \vee x \in (1, 2) \\ \wedge \quad & x \in (0, 1) \Rightarrow x \in (0, 2) \\ \wedge \quad & x \in (1, 2) \Rightarrow x \in (0, 2) \\ \wedge \quad & (\neg(x \in (0, 1)) \vee \neg(x \in (1, 2))) \end{aligned}$$

Note that \Rightarrow is denoted for *implication*.

7.2.2 Monotonic Decomposition.

Monotonic decomposition introduces bias δ to an interval decomposition, if a value of a corresponding variable monotonically affects on a value of a polynomial.

Definition 7.2.2. Let $f(x_1, \dots, x_k)$ be a polynomial, a variable x_i ($1 \leq i \leq k$) is monotonically increasing in f if $\forall x'_i \geq x''_i$ implies $f(x_1, \dots, x'_i, \dots, x_k) \geq f(x_1, \dots, x''_i, \dots, x_k)$, and is monotonically decreasing in f if $\forall x'_i \geq x''_i$ implies $f(x_1, \dots, x'_i, \dots, x_k) \leq f(x_1, \dots, x''_i, \dots, x_k)$. Pos_f and Neg_f denote the sets of monotonically increasing and decreasing variables of a polynomial f , respectively.

For a polynomial $f = 2yv^2 - ux^2 - 1$ where $x, y, u, v \in (0, \infty)$, $Pos_f = \{y, v\}$ and $Neg_f = \{x, u\}$.

Definition 7.2.3. Let $x \in (a, b)$ and $\delta < b - a$ for a bound δ . A monotonic decomposition is,

$$D_m(x \in (a, b)) = \begin{cases} \{x \in (a, b - \delta), x \in (b - \delta, b)\} & \text{if } x \in Pos_f \\ \{x \in (a, a + \delta), x \in (a + \delta, b)\} & \text{if } x \in Neg_f \\ \{x \in (a, \frac{a+b}{2}), x \in (\frac{a+b}{2}, b)\} & \text{otherwise} \end{cases}$$

Example 7.2.4. In Example 6.2.1, if the API $f = 2yv^2 - ux^2 - 1 > 0$ is Test-UNSAT under $I = x \in (0, 2) \wedge y \in (0, 2) \wedge u \in (0, 2) \wedge v \in (0, 2)$, the following clauses are added to the SAT solver for monotonic decomposition (with assumption $\delta = 0.25$),

- $x \in Neg_f$ then $x \in (0, 2)$ is decomposed into $x \in (0, 0.25)$ and $x \in (0.25, 2)$,

$$\begin{aligned} & x \in (0, 2) \Rightarrow x \in (0, 0.25) \vee x \in (0.25, 2) \\ \wedge & \quad x \in (0, 0.25) \Rightarrow x \in (0, 2) \\ \wedge & \quad x \in (0.25, 2) \Rightarrow x \in (0, 2) \\ \wedge & \quad (\neg(x \in (0, 0.25)) \vee \neg(x \in (0.25, 2))) \end{aligned}$$

- $y \in Pos_f$ then $y \in (0, 2)$ is decomposed into $y \in (0, 1.75)$ and $y \in (1.75, 2)$,

$$\begin{aligned} & y \in (0, 2) \Rightarrow y \in (0, 1.75) \vee y \in (1.75, 2) \\ \wedge & \quad y \in (0, 1.75) \Rightarrow y \in (0, 2) \\ \wedge & \quad y \in (1.75, 2) \Rightarrow y \in (0, 2) \\ \wedge & \quad (\neg(y \in (0, 1.75)) \vee \neg(y \in (1.75, 2))) \end{aligned}$$

- $u \in Neg_f$ then $u \in (0, 2)$ is decomposed into $u \in (0, 0.25)$ and $u \in (0.25, 2)$,

$$\begin{aligned}
& u \in (0, 2) \Rightarrow u \in (0, 0.25) \vee u \in (0.25, 2) \\
& \wedge \quad u \in (0, 0.25) \Rightarrow u \in (0, 2) \\
& \wedge \quad u \in (0.25, 2) \Rightarrow u \in (0, 2) \\
& \wedge \quad (\neg(u \in (0, 0.25)) \vee \neg(u \in (0.25, 2)))
\end{aligned}$$

- $v \in Pos_f$ then $v \in (0, 2)$ is decomposed into $v \in (0, 1.75)$ and $v \in (1.75, 2)$,

$$\begin{aligned}
& v \in (0, 2) \Rightarrow v \in (0, 1.75) \vee v \in (1.75, 2) \\
& \wedge \quad v \in (0, 1.75) \Rightarrow v \in (0, 2) \\
& \wedge \quad v \in (1.75, 2) \Rightarrow v \in (0, 2) \\
& \wedge \quad (\neg(v \in (0, 1.75)) \vee \neg(v \in (1.75, 2)))
\end{aligned}$$

We apply δ in Definition 7.2.3 (heuristic rule) as the bias δ , by regarding δ as a unit of searching. For a balanced decomposition, the SAT solver will choose an arbitrary combination of input ranges. However, for a monotonic decomposition, we would like to force the SAT solver to choose a narrower sub-interval (i.e., $(b - \delta, b)$ for Pos_f , and $(a, a + \delta)$ for Neg_f), which makes upper and lower bounds of the polynomial f increase, and provide more chances to lead f satisfiable. In Example 7.2.4, the choice is $x \in (0, 0.25) \wedge y \in (1.75, 2) \wedge u \in (0, 0.25) \wedge v \in (1.75, 2)$ for the next evaluation by IA and testing. MiniSat 2.2 chooses literals by the “activity” measure, and we manually increase the activity of literals corresponding to a narrower sub-interval.

7.2.3 Tick Decomposition.

Tick decomposition divides an interval into two or three sub intervals based on a given point (tick) inside the interval.

Definition 7.2.5. Let $x \in (a, b)$, $\delta < b - a$ for a bound δ and a tick $t \in (a, b)$, d_p , d_n and

d_{pn} are defined as follows:

$$\begin{aligned}
d_p(x \in (a, b)) &= \begin{cases} \{x \in (a, t) \vee x \in (t, t + \delta) \vee x \in (t + \delta, b)\} & \text{if } t + \delta < b \\ \{x \in (a, t) \vee x \in (t, b)\} & \text{otherwise} \end{cases} \\
d_n(x \in (a, b)) &= \begin{cases} \{x \in (a, t - \delta) \vee x \in (t - \delta, t) \vee x \in (t, b)\} & \text{if } t - \delta > a \\ \{x \in (a, t) \vee x \in (t, b)\} & \text{otherwise} \end{cases} \\
d_{pn}(x \in (a, b)) &= \begin{cases} \{x \in (a, t + 0.5\delta) \vee x \in (t + 0.5\delta, b)\} & \text{if } t - 0.5\delta \leq a \\ \{x \in (a, t - 0.5\delta) \vee x \in (t - 0.5\delta, b)\} & \text{if } t + 0.5\delta \geq b \\ \{x \in (a, t - 0.5\delta) \vee x \in (t - 0.5\delta, t + 0.5\delta) \vee x \in (t + 0.5\delta, b)\} & \text{otherwise} \end{cases}
\end{aligned}$$

Definition 7.2.6. Let $x \in (a, b)$, $\delta < b - a$ for a bound δ and a tick $t \in (a, b)$. A tick decomposition is,

$$D_t(x \in (a, b)) = \begin{cases} d_p & \text{if } x \in Pos_f \\ d_n & \text{if } x \in Neg_f \\ d_{pn} & \text{otherwise} \end{cases}$$

Example 7.2.7. In Example 6.2.1, assume that the API $f = 2yv^2 - ux^2 - 1 > 0$ is Test-UNSAT under $I = x \in (0, 2) \wedge y \in (0, 2) \wedge u \in (0, 2) \wedge v \in (0, 2)$, $\delta = 0.25$ and ticks are given by $t_x = 1.2, t_y = 0.5, t_u = 1.05$ and $t_v = 1.3$ (a test data in Example 6.2.1). Because $x, u \in Neg_f$ and $y, v \in Pos_f$, clauses for tick decomposition are,

- $x \in (0, 2)$ is decomposed into $x \in (0, 0.95)$, $x \in (0.95, 1.2)$ and $x \in (1.2, 2)$,

$$\begin{aligned}
& x \in (0, 2) \Rightarrow x \in (0, 0.95) \vee x \in (0.95, 1.2) \vee x \in (1.2, 2) \\
& \wedge \quad x \in (0, 0.95) \Rightarrow x \in (0, 2) \\
& \wedge \quad x \in (0.95, 1.2) \Rightarrow x \in (0, 2) \\
& \wedge \quad x \in (1.2, 2) \Rightarrow x \in (0, 2) \\
& \wedge \quad (\neg(x \in (0, 0.95)) \vee \neg(x \in (0.95, 1.2))) \\
& \wedge \quad (\neg(x \in (0, 0.95)) \vee \neg(x \in (1.2, 2))) \\
& \wedge \quad (\neg(x \in (0.95, 1.2)) \vee \neg(x \in (1.2, 2)))
\end{aligned}$$

- $y \in (0, 2)$ is decomposed into $y \in (0, 0.5)$, $y \in (0.5, 0.75)$ and $y \in (0.75, 2)$,

$$\begin{aligned}
& y \in (0, 2) \Rightarrow y \in (0, 0.5) \vee y \in (0.5, 0.75) \vee y \in (0.75, 2) \\
& \wedge \quad y \in (0, 0.5) \Rightarrow y \in (0, 2) \\
& \wedge \quad y \in (0.5, 0.75) \Rightarrow y \in (0, 2) \\
& \wedge \quad y \in (0.75, 2) \Rightarrow y \in (0, 2) \\
& \wedge \quad (\neg(y \in (0, 0.5)) \vee \neg(y \in (0.5, 0.75))) \\
& \wedge \quad (\neg(y \in (0, 0.5)) \vee \neg(y \in (0.75, 2))) \\
& \wedge \quad (\neg(y \in (0.5, 0.75)) \vee \neg(y \in (0.75, 2)))
\end{aligned}$$

- $u \in (0, 2)$ is decomposed into $u \in (0, 0.8)$, $u \in (0.8, 1.05)$ and $u \in (1.05, 2)$,

$$\begin{aligned}
& u \in (0, 2) \Rightarrow u \in (0, 0.8) \vee u \in (0.8, 1.05) \vee u \in (1.05, 2) \\
& \wedge \quad u \in (0, 0.8) \Rightarrow u \in (0, 2) \\
& \wedge \quad u \in (0.8, 1.05) \Rightarrow u \in (0, 2) \\
& \wedge \quad u \in (1.05, 2) \Rightarrow u \in (0, 2) \\
& \wedge \quad (\neg(u \in (0, 0.8)) \vee \neg(u \in (0.8, 1.05))) \\
& \wedge \quad (\neg(u \in (0, 0.8)) \vee \neg(u \in (1.05, 2))) \\
& \wedge \quad (\neg(u \in (0.8, 1.05)) \vee \neg(u \in (1.05, 2)))
\end{aligned}$$

- $v \in (0, 2)$ is decomposed into $v \in (0, 1.3)$, $v \in (1.3, 1.55)$ and $v \in (1.55, 2)$,

$$\begin{aligned}
& v \in (0, 2) \Rightarrow v \in (0, 1.3) \vee v \in (1.3, 1.55) \vee v \in (1.55, 2) \\
& \wedge \quad v \in (0, 1.3) \Rightarrow v \in (0, 2) \\
& \wedge \quad v \in (1.3, 1.55) \Rightarrow v \in (0, 2) \\
& \wedge \quad v \in (1.55, 2) \Rightarrow v \in (0, 2) \\
& \wedge \quad (\neg(v \in (0, 1.3)) \vee \neg(v \in (1.3, 1.55))) \\
& \wedge \quad (\neg(v \in (0, 1.3)) \vee \neg(v \in (1.55, 2))) \\
& \wedge \quad (\neg(v \in (1.3, 1.55)) \vee \neg(v \in (1.55, 2)))
\end{aligned}$$

Interval Decomposition	SAT		UNSAT	
	No. Problems	Time(s)	No. Problems	Time(s)
Balanced	35	1490.759	4	0.910
Monotonic	38	1480.812	4	0.060
Tick	24	431.475	4	0.060

Table 7.2: Experimental results for different strategies of interval decomposition

In implementation of **raSAT**, a given tick is the test data that makes the first Test-UNSAT API f_j be max value. Intuitionally, we would like to force the SAT solver to choose a narrower sub-interval around the max value test data (i.e., $(t, t + \delta)$ or (t, b) for Pos_f , $(t - \delta, t)$ or (a, t) for Neg_f). In Example 7.2.7, the choice is $x \in (0.95, 1.2) \wedge y \in (0.5, 0.75) \wedge u \in (0.8, 1.05) \wedge v \in (1.3, 1.55)$ for the next evaluation by IA and testing.

We also compare results when applying different strategies for interval decomposition. Table 7.2 shows experimental results for 151 problems of Zankl family. The first column indicates 3 strategies, which are *Balanced*, *Monotonic* and *Tick Decomposition*. The number of solved problems (SAT/UNSAT) and their total running time are shown in the remain columns. Though *Monotonic Decomposition* gives the best results, it is slightly better than *Balanced Decomposition*. We also plan to investigate other choices, i.e., adjusting length of smaller intervals, identifying UNSAT domains, etc.

Chapter 8

SMT Framework

8.1 Design of SMT Framework

We implement the SMT solver **raSAT** following to the DPLL(T) procedure in Chapter 3 by instantiating IA and testing as $O.T$ and $U.T$, respectively. Figure 8.1 shows its framework.

Initial input constraint are represented in the form of $F = I \wedge P$ where I is a CNF formula of an interval constraint, $I = (x_1 \in (a_1, b_1)) \wedge \dots \wedge (x_n \in (a_n, b_n))$ and polynomial constraint P is in the form of $\bigwedge_{i=1}^m f_i(x_1, \dots, x_n) > 0$.

We use **MiniSat2.2** as backend SAT solver. It chooses a combination of input ranges for all variables, which is first evaluated by IA. If IA informs IA-UNSAT, the combination of input ranges is sent to **UNSAT cores** for computing minimal combinations that make IA-UNSAT. The very lazy theory learning is applied for learning clauses of minimal combinations obtained from UNSAT cores. Then the SAT solver chooses another combination of input ranges. If IA informs IA-VALID, **raSAT** terminates and outputs SAT. Otherwise, testing is applied.

Testing generates test data from a given combination of input ranges. Testing will stop when it results Test-SAT, i.e., a SAT solution is found. Otherwise it finds the set of Test-UNSAT APIs. They are sent to *domain decomposition* and a refinement rule is applied.

If all input ranges become small enough, *heuristic rule* is applied. Once heuristic rules are used, **raSAT** concludes *unknown* when the SAT solver informs UNSAT. If they are

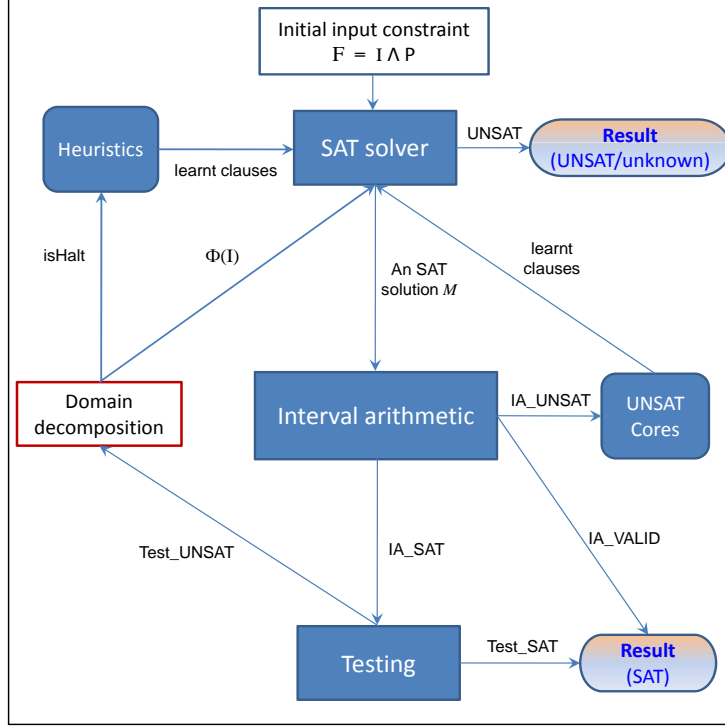


Figure 8.1: Framework of **raSAT**

never used, **raSAT** concludes UNSAT when the same occurs. We apply various *affine intervals* for IA and *k-random ticks* for testing.

8.2 Examples

Example 8.2.1. Figure 8.2 describes process of solving the polynomial constraint $F = (x \in (-1, 3) \wedge y \in (-1, 3)) \wedge (x^3 - x^2 + y - 1.99 > 0)$ with input ranges $x \in (-1, 3)$ and $y \in (-1, 3)$. **raSAT** initially searches on the box $x \in (-1, 3) \wedge y \in (-1, 3)$, and executes a refinement loop for searching on their sub-boxes.

We present process of **raSAT** for *balanced* and *monotonic* decomposition by the DPLL(T) rules for approximation refinement. P is denoted for the polynomial constraint $x^3 - x^2 + y - 1.99 > 0$.

Note that \Rightarrow_{VL} (SAT solver) is the rule that forces the SAT solver to choose a full assignment (a combination of input ranges). For an interval decomposition, i.e., $x \in (-1, 3)$ is decomposed into $x \in (-1, 1) \vee x \in (1, 3)$, in implementation we add four

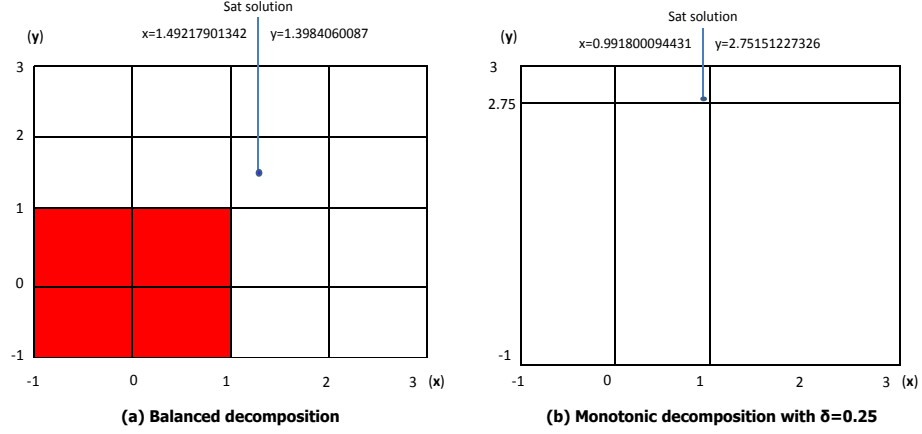


Figure 8.2: Interval decompositions by **raSAT** for Example 8.2.1

clauses below to the SAT solver,

$$\begin{aligned}
& x \in (-1, 3) \Rightarrow x \in (-1, 1) \vee x \in (1, 3) \\
& \wedge \quad x \in (-1, 1) \Rightarrow x \in (-1, 3) \\
& \wedge \quad x \in (1, 3) \Rightarrow x \in (-1, 3) \\
& \wedge \quad (\neg(x \in (-1, 1)) \vee \neg(x \in (1, 3)))
\end{aligned}$$

But for simplicity in presenting the DPLL(T) rules, we only add the clause $x \in (-1, 1) \vee x \in (1, 3)$ to the formula of interval constraints.

Balanced decomposition: Disjoin boxes applied balanced decomposition are shown in the left of Figure 8.2. The red boxes are detected as UNSAT (IA-UNSAT).

\emptyset	$\parallel (x \in (-1, 3)) \wedge (y \in (-1, 3)) \wedge P$	\Rightarrow_{VL} (SAT solver)
M_0	$\parallel (x \in (-1, 3)) \wedge (y \in (-1, 3)) \wedge P$	\Rightarrow_{VL} (refinement)
\emptyset	$\parallel \cdots \wedge (x \in (-1, 1) \vee x \in (1, 3)) \wedge (y \in (-1, 1) \vee y \in (1, 3)) \wedge P$	\Rightarrow_{VL} (SAT solver)
M_1	$\parallel \cdots \wedge (x \in (-1, 1) \vee x \in (1, 3)) \wedge (y \in (-1, 1) \vee y \in (1, 3)) \wedge P$	\Rightarrow_{VL} (refinement)
\emptyset	$\parallel \cdots \wedge (x \in (-1, 0) \vee x \in (0, 1)) \wedge (y \in (-1, 0) \vee y \in (0, 1)) \wedge P$	\Rightarrow_{VL} (SAT solver)
M_2	$\parallel \cdots \wedge (x \in (-1, 0) \vee x \in (0, 1)) \wedge (y \in (-1, 0) \vee y \in (0, 1)) \wedge P$	\Rightarrow_{VL} (Learning)
\emptyset	$\parallel \cdots \wedge (\neg(x \in (-1, 0)) \vee \neg(y \in (-1, 0))) \wedge P$	\Rightarrow_{VL} (SAT solver)
M_3	$\parallel \cdots \wedge (\neg(x \in (-1, 0)) \vee \neg(y \in (-1, 0))) \wedge P$	\Rightarrow_{VL} (Learning)
\emptyset	$\parallel \cdots \wedge (\neg(x \in (-1, 0)) \vee \neg(y \in (0, 1))) \wedge P$	\Rightarrow_{VL} (SAT solver)
M_4	$\parallel \cdots \wedge (\neg(x \in (-1, 0)) \vee \neg(y \in (0, 1))) \wedge P$	\Rightarrow_{VL} (Learning)
\emptyset	$\parallel \cdots \wedge (\neg(x \in (0, 1)) \vee \neg(y \in (-1, 0))) \wedge P$	\Rightarrow_{VL} (SAT solver)
M_5	$\parallel \cdots \wedge (\neg(x \in (0, 1)) \vee \neg(y \in (-1, 0))) \wedge P$	\Rightarrow_{VL} (Learning)
\emptyset	$\parallel \cdots \wedge (\neg(x \in (0, 1)) \vee \neg(y \in (0, 1))) \wedge P$	\Rightarrow_{VL} (SAT solver)
M_6	$\parallel \cdots \wedge (\neg(x \in (0, 1)) \vee \neg(y \in (0, 1))) \wedge P$	\Rightarrow_{VL} (refinement)
\emptyset	$\parallel \cdots \wedge (x \in (1, 2) \vee x \in (2, 3)) \wedge (y \in (1, 2) \vee y \in (2, 3)) \wedge P$	\Rightarrow_{VL} (SAT solver)
M_7	$\parallel \cdots \wedge (x \in (1, 2) \vee x \in (2, 3)) \wedge (y \in (1, 2) \vee y \in (2, 3)) \wedge P$	\Rightarrow_{VL} (SAT rule)
<i>SAT</i>		Model found

where M_i chosen by the SAT solver are,

- $M_0 = (x \in (-1, 3)) \wedge (y \in (-1, 3))$
- $M_1 = (x \in (-1, 1)) \wedge (y \in (-1, 1))$
- $M_2 = (x \in (-1, 0)) \wedge (y \in (-1, 0))$
- $M_3 = (x \in (-1, 0)) \wedge (y \in (0, 1))$
- $M_4 = (x \in (0, 1)) \wedge (y \in (-1, 0))$
- $M_5 = (x \in (0, 1)) \wedge (y \in (0, 1))$
- $M_6 = (x \in (1, 3)) \wedge (y \in (1, 3))$

- $M_7 = (x \in (1, 2)) \wedge (y \in (1, 2))$

When the SAT solver chooses $M_7 = (x \in (1, 2)) \wedge (y \in (1, 2))$, IA results IA-SAT, and testing finally finds a satisfiable test data $x = 1.49217901342$ and $y = 1.3984060087$ (Test-SAT). Then, **raSAT** returns SAT.

Monotonic decomposition: Similar process but with monotonic decomposition is described in the right of Figure 8.2 (with assumption $\delta = 0.25$).

$$\begin{array}{ll}
\emptyset & \parallel (x \in (-1, 3)) \wedge (y \in (-1, 3)) \wedge P \quad \Longrightarrow_{VL} \text{ (SAT solver)} \\
M_0 & \parallel (x \in (-1, 3)) \wedge (y \in (-1, 3)) \wedge P \quad \Longrightarrow_{VL} \text{ (refinement)} \\
\emptyset & \parallel \cdots \wedge (x \in (-1, 1) \vee x \in (1, 3)) \wedge (y \in (-1, 2.75) \vee y \in (2.75, 3)) \wedge P \quad \Longrightarrow_{VL} \text{ (SAT solver)} \\
M_1 & \parallel \cdots \wedge (x \in (-1, 1) \vee x \in (1, 3)) \wedge (y \in (-1, 2.75) \vee y \in (2.75, 3)) \wedge P \quad \Longrightarrow_{VL} \text{ (refinement)} \\
\emptyset & \parallel \cdots \wedge (y \in (-1, 2.75) \vee y \in (2.75, 3)) \wedge (x \in (-1, 0) \vee x \in (0, 1)) \wedge P \quad \Longrightarrow_{VL} \text{ (SAT solver)} \\
M_2 & \parallel \cdots \wedge (y \in (-1, 2.75) \vee y \in (2.75, 3)) \wedge (x \in (-1, 0) \vee x \in (0, 1)) \wedge P \quad \Longrightarrow_{VL} \text{ (SAT rule)} \\
SAT & \text{Model found}
\end{array}$$

where M_i chosen by the SAT solver are,

- $M_0 = (x \in (-1, 3)) \wedge (y \in (-1, 3))$
- $M_1 = (x \in (-1, 1)) \wedge (y \in (2.75, 3))$
- $M_2 = (x \in (0, 1)) \wedge (y \in (2.75, 3))$

The SAT solver chooses $M_2 = x \in (0, 1) \wedge y \in (2.75, 3)$ and testing finds a satisfiable test data with $x = 0.991800094431$ and $y = 2.75151227326$ (Test-SAT). With monotonic decomposition, **raSAT** finds a satisfiable instance with fewer decompositions.

Chapter 9

Experiments

In this chapter, we first apply different measures for experiments, i.e., high degrees, number of variables, number of APIs (atomic polynomial inequalities) and then we test on problems of the division QF_NRA (Quantifier Free of Nonlinear Real Arithmetic) in the benchmarks of SMT-LIB [2].

We compare our SMT solver **raSAT** with **Z3 4.3** and **CVC3 2.4.1**. Z3 (version 3.1) is the winner of SMT competition 2011 for QF_NRA and the latest version (**Z3 4.3**) is also called by another name **nlSAT** [16], which is known as a very strong SMT solver for non-linear arithmetic. CVC3 [3] participated in SMT competition 2010 and 2011 for the division QF_NRA.

In **raSAT**, affine intervals (AF_2 , CAI_1) for IA, *2-random ticks* for testing, *Test-UNSAT* of testing and *monotonic decomposition* for refinement are used. We do not apply UNSAT core computation in these experiments. All tests are run on a system with Intel Core Duo L7500 1.6 GHz and 2 GB of RAM.

9.1 Experiments on Different Measures

The first and the second measures apply for the problems of the form

$$\psi = \sum_{i=1}^k x_i^n < 1 \wedge \sum_{i=1}^k (x_i - r)^n < 1 \quad (9.1)$$

For experiments on the problems 9.1, we changed values of r based on the threshold

$\sqrt[n]{\frac{1}{k}}$, which is the threshold separating SAT and UNSAT problems when k and n are fixed. If r is close to the threshold, the problems are more difficult to decide and vice versa. Figure 9.1 and 9.2 demonstrate the choices of r . In our experiments we choose values of r for difficult SAT/UNSAT problems such that $|r - \sqrt[n]{\frac{1}{k}}| < 0.01$.

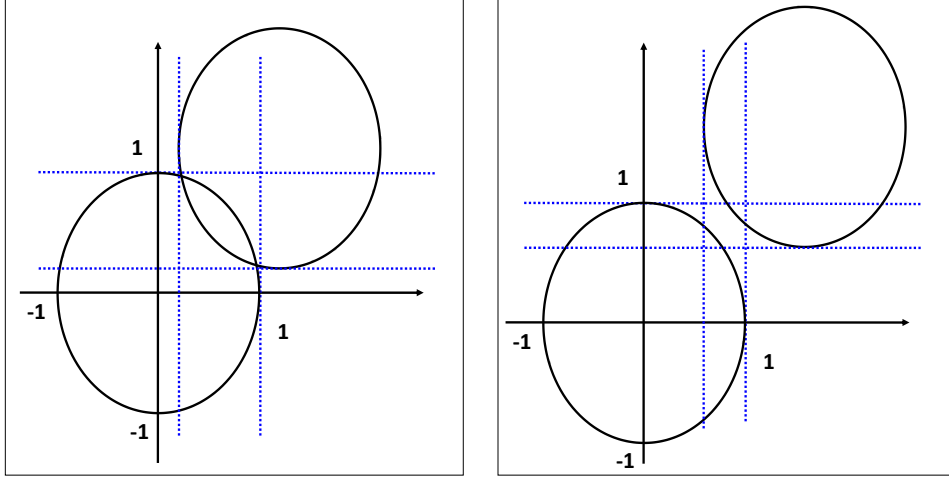


Figure 9.1: The choices of r are far from the threshold $\sqrt[n]{\frac{1}{k}}$

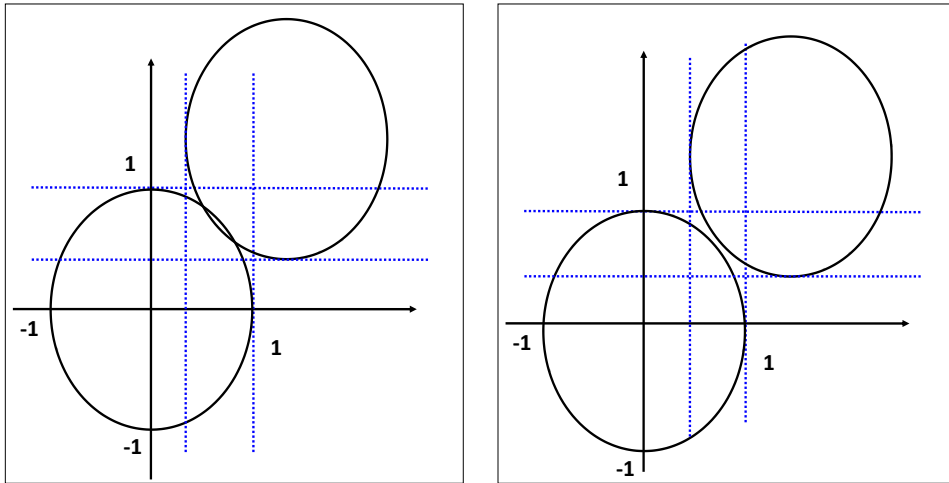


Figure 9.2: The choices of r are close to the threshold $\sqrt[n]{\frac{1}{k}}$

For settings of **raSAT**, the *unit searching* δ is set to 0.005 and initial interval constraints are presented as $\bigwedge_i x_i \in (-1, 1)$.

Note that **CVC3 2.4.1** reports *unknown* for all problems in these measures and we only show comparison of **raSAT** and **Z3 4.3**.

SAT/UNSAT	k	n	r	Time(s)	
				Z3 4.3	raSAT
SAT	2	8	1.83	1.330	0.265
UNSAT	2	8	1.84	0.580	0.328
SAT	2	10	1.86	4.530	0.140
UNSAT	2	10	1.87	125.000	0.796
SAT	2	12	1.88	0.360	0.140
UNSAT	2	12	1.89	40.280	1.390
SAT	2	18	1.92	289.110	0.562
UNSAT	2	18	1.93	391.670	0.765
SAT	2	20	1.93	1259.560	1.468
UNSAT	2	20	1.94	1650.860	0.921
SAT	2	22	1.93	? > 3600	0.437
UNSAT	2	22	1.94	? > 3600	3.203

Table 9.1: Experimental results for $\psi = x_1^n + x_2^n < 1 \wedge (x_1 - r)^n + (x_2 - r)^n < 1$

The first measure: degree increasing

We first show comparison of **raSAT** and **Z3 4.3** in Table 9.1 for the problems $\psi = x_1^n + x_2^n < 1 \wedge (x_1 - r)^n + (x_2 - r)^n < 1$, when we fix the number of variables to 2 ($k = 2$), increase degrees n (i.e., 8, 10, 12, 18, 20 and 22) and modify values of r .

In Table 9.1, the first column indicates problems that are SAT or UNSAT, the next are columns of k , n and r , respectively. We compare running time in seconds of **Z3 4.3** and **raSAT** for each problem in the last two columns. For the degree 22, the results of **Z3 4.3** are “? > 3600”, which means that **Z3 4.3** didn’t respond in 3600 seconds (note that we didn’t set timeout in the first experiment).

For the first experiment, we aim at seeing how **raSAT** and **Z3 4.3** are performed for problems with fewer variables (only 2) but high degrees. **raSAT** shows interesting results, which are,

- **raSAT** reports results very fast, i.e., its solving time are almost around 1 second and the longest running time is 3.203 seconds for the degree 22,
- **raSAT** does well for high degrees of 2 variables, while **Z3 4.3** suffers from degree increasing, i.e., 289.110 second for the degree 18 (SAT), 1259.560 seconds for the degree 20 (SAT) and no respond in 1 hour for the degree 22 (SAT).

SAT/UNSAT	k	n	r	Time(s)	
				Z3 4.3	raSAT
SAT	3	4	1.51	0.030	0.027
UNSAT	3	4	1.52	10.560	<i>timeout</i>
SAT	4	4	1.41	<i>timeout</i>	0.390
UNSAT	4	4	1.42	<i>timeout</i>	<i>timeout</i>
SAT	5	4	1.33	<i>timeout</i>	51.578
UNSAT	5	4	1.34	<i>timeout</i>	<i>timeout</i>
SAT	6	4	1.27	<i>timeout</i>	111.031
UNSAT	6	4	1.28	<i>timeout</i>	<i>timeout</i>
SAT	3	6	1.66	<i>timeout</i>	0.890
UNSAT	3	6	1.67	<i>timeout</i>	62.765
SAT	4	6	1.58	<i>timeout</i>	1.156
UNSAT	4	6	1.59	<i>timeout</i>	<i>timeout</i>
SAT	5	6	1.52	<i>timeout</i>	73.937
UNSAT	5	6	1.53	<i>timeout</i>	<i>timeout</i>
SAT	6	6	1.48	<i>timeout</i>	239.968
UNSAT	6	6	1.49	<i>timeout</i>	<i>timeout</i>
SAT	3	8	1.74	<i>timeout</i>	3.125
UNSAT	3	8	1.75	<i>timeout</i>	37.156
SAT	4	8	1.68	<i>timeout</i>	69.843
UNSAT	4	8	1.69	<i>timeout</i>	<i>timeout</i>

Table 9.2: Experimental results for $\psi = \sum_{i=1}^k x_i^n < 1 \wedge \sum_{i=1}^k (x_i - r)^n < 1$

- For each degree, we shift values of r a little from SAT to UNSAT (i.e., r is shifted from 1.86 to 1.87 for the degree 10, from 1.92 to 1.93 for the degree 18), **raSAT** can decide for both SAT and UNSAT problems in *short time*.

The second measure: number of variables

The problems in the second measure are taken from the formula (9.1) when we increase their number of variables (e.g., from 3 to 6) and their degrees (e.g., 4, 6, 8). We aim at seeing how **raSAT** and **Z3 4.3** work when both dimensions and degrees are enlarged. Results are shown in Table 9.2, which are,

- **raSAT** still outperforms than **Z3 4.3**, i.e., 12 problems are solved by **raSAT** and

only 2 problems are solved by **Z3 4.3** in total 20 problems,

- when dimension increasing, **raSAT** can detect for SAT problems but their running time increases much, i.e., from 0.390 seconds for 4 variables to 51.578 seconds for 5 variables of the degree 4.
- however, **raSAT** suffers from increasing number of variables (dimensions), especially for UNSAT problems. When dimension is increased, explosion on number of boxes occurs. For UNSAT problems, **raSAT** could extract all unit searching domains (small boxes decided by the unit searching $\delta = 0.005$) and proves them UNSAT by interval arithmetic.

In the second experiment, timeout is set to 600 seconds for each problem.

The third measure: number of APIs

The third measure is on increasing number of APIs (atomic polynomial inequalities) and its problems are taken from the formula $\psi = \psi_1 \wedge \psi_2$ where,

- $\psi_1 = x_0^n + x_1^n < 1 \wedge x_1^n + x_2^n < 1 \wedge \cdots \wedge x_k^n + x_0^n < 1$
- $\psi_2 = (x_0 - r)^n + (x_1 - r)^n < 1 \wedge (x_1 - r)^n + (x_2 - r)^n < 1 \wedge \cdots \wedge (x_k - r)^n + (x_0 - r)^n < 1$

The initial interval constraints are set as $\bigwedge_i x_i \in (-1, 1)$, timeout is set in 600 seconds and $|r - \sqrt[n]{\frac{1}{2}}| < 0.01$.

Comparison results of **Z3 4.3** and **raSAT** are shown in Table 9.3. The number of APIs is increased from 3 to 15, which is shown in the column k . The last 2 columns show running time of **Z3 4.3** and **raSAT**. While **Z3 4.3** reports timeout for 7 problems (among 14 as total and 600 seconds for timeout), **raSAT** can detect all of them. The longest running time is 6.640 seconds for the problems of 15 APIs and degree 6. By the measure on number of APIs, **raSAT** still outperforms than **Z3 4.3**.

Observation 1.

From the experiments on three measures, we have some observations that are,

- **Z3 4.3** meets difficulties for high degrees, i.e., problems of 2 variables with degrees 20, 22, and problems of 4, 5 variables with degrees 4, 6, 8 in our examples,

SAT/UNSAT	k	n	r	Time(s)	
				Z3 4.3	raSAT
SAT	3	6	1.78	<i>timeout</i>	0.171
UNSAT	3	6	1.79	0.280	0.796
SAT	5	6	1.78	<i>timeout</i>	0.375
UNSAT	5	6	1.79	0.280	0.640
SAT	7	6	1.78	<i>timeout</i>	0.765
UNSAT	7	6	1.79	0.250	0.734
SAT	9	6	1.78	<i>timeout</i>	2.671
UNSAT	9	6	1.79	0.300	1.921
SAT	11	6	1.78	<i>timeout</i>	3.328
UNSAT	11	6	1.79	0.220	1.343
SAT	13	6	1.78	<i>timeout</i>	4.460
UNSAT	13	6	1.79	0.300	1.875
SAT	15	6	1.78	<i>timeout</i>	6.640
UNSAT	15	6	1.79	0.300	2.265

Table 9.3: Experimental results for $\psi = \psi_1 \wedge \psi_2$

- **raSAT** outperforms than **Z3 4.3** in the experiment of three measures. It seems work quite well for high degrees and increasing number of APIs.
- However **raSAT** suffers from enlarging dimensions for both SAT and UNSAT problems. We need further comparison, investigation, and reasonable strategies for increasing dimensions because explosion of boxes occurs when intervals are decomposed in **raSAT**.

9.2 Experiments on Benchmarks of SMT-LIB

In the experiment, problems are taken from the division QF_NRA in the benchmarks of SMT-LIB [2]. Due to restriction on polynomial inequality, we extract them from the families *Hong* (20 among 20 problems) and *Zankl* (151 among 166 problems).

Table 9.4 shows the number of solved problems (either SAT or UNSAT) and their total running time (in seconds).

In Hong family, we present interval constraints as $x \in (-\infty, -1) \vee x \in (-1, 1) \vee x \in (1, \infty)$ for each variable x . In Zankl family, all variables of problems are originally given a

lower bound which is ≥ 0 . We first apply IA to estimate upper and lower bounds of these constraints with an input range $(0, \infty)$ for all variables (note that our implementation of IA allows to estimate infinite bound, i.e., $(0, \infty)$). If IA says IA-SAT (IA can not decide SAT or UNSAT for the input range $(0, \infty)$), upper bounds for all variables will be manually given and we evaluate almost these problems with a given range $(0, 2)$ (some problems are applied for a range $(0, 4)$). For infinite bounds, automatic choices of initial range decomposition will be considered, i.e., $(0, \infty)$ is initially represented as $(0, 2) \vee (2, \infty)$, however it is not yet implemented.

We set the *unit searching* δ to 0.25 and timeout is set to 600 seconds for each problem. Due to trade-off between precision and time consuming, we apply CAI_1 for problems with number of variables ≤ 15 and AF_2 for problems with number of variables > 15 .

Among 20 problems of Hong family, their degrees distribute from 1 to 20. **raSAT** solved all of them (all are UNSAT). **Z3 4.3** solved 8 problems, whose degrees are up to 8. **Z3 4.3** becomes timeout for other problems, which have higher degrees than 8. For Hong family, **CVC3 2.4** cannot solve any problems.

For Zankl family, **Z3 4.3** shows better performance than **raSAT** and **CVC3 2.4.1**. **Z3 4.3** runs very fast for problems that contain linear constraints combined with nonlinear constraints of lower degrees (e.g., degree 4). We observed that **raSAT** outperforms **Z3 4.3** for some problems that contain a long monomial (e.g., 60) with higher degrees (e.g., 6) and more variables (e.g., more than 14). For instance, **raSAT** can solve *matrix-2-all-5,8,11,12* (each less than 36 seconds), while these problem are timeout in **Z3 4.3**, and **raSAT** is quicker to obtain SAT (by testing) in *matrix-2-all-9,10* (3.8 and 2.3 seconds in **raSAT**; 93 and 436 seconds in **Z3 4.3**, respectively).

Zankl problems come from termination analysis of term rewriting and are regarded as standard problems for the division QF_NRA of SMT competition. In 2010, the first year of SMT competition for the division QF_NRA, MiniSmt [34] is the winner which can solve 44 among 60 problems $(44/60)^1$ (43 inequalities and 1 equality). However MiniSmt can not decide UNSAT problems because of bounded bit encoding. In 2011, the winner Z3 (version 3.1) solved 53/63 problems² (38 inequalities and 15 equalities). In 2012, there is no participant reporting comparative results for the division QF_NRA. Recently, there

¹<http://www.smtcomp.org/2010/>

²<http://www.smtcomp.org/2011/>

Solver	Hong (20)			Zankl (151)		
	SAT	UNSAT	time(s)	SAT	UNSAT	time(s)
CVC3 2.4.1	0	0	0	0	9	10.678
Z3 4.3	0	8	5.620	50	24	1144.320
raSAT	0	20	381.531	42	9	2417.931

Table 9.4: Experimental results for Hong and Zankl family

are some proposed approaches for polynomial constraint solving using Zankl problems in comparison with others, i.e., SMT-RAT (22/166) [11] and Z3 4.3 (89/166) [16] in comparison with iSAT (21/166), MiniSmt (46/166), Mathematica (50/166), QEPCAD (21/166).

Observation 2.

At the moment, **raSAT** is quite weak in Zankl family. However we feel that the results are encouraging. We judge that results of **raSAT** would be improved by solving linear fragment separated from non-linear constraints.

- The linear fragment indicates vertices of intersecting, which would be candidates of cutoff points for guiding interval decomposition.
- Solving the linear fragment can help to prove UNSAT directly, or quicker by narrowing domains of searching, i.e, remove UNSAT domains. For instance, the linear constraints $x_1 - x_2 > 0 \wedge x_2 - x_1 > 0$ are contained in a problem of Zankl family but **raSAT** can not prove it UNSAT by interval arithmetic and interval decomposition.

Chapter 10

Extensions to Polynomial Equality

We extend our approach for polynomial constraints including greater-than-or-equal constraints and equality constraints.

10.1 Greater-than-or-Equal Handling

We first give the definition of **strict-UNSAT** for greater-than-or-equal constraints (\geq).

Definition 10.1.1. The constraint $\bigwedge_j f_j \geq 0$ is said **strict-UNSAT** if $\bigwedge_j f_j > -\delta_j$ is UNSAT for $\delta_j > 0$.

Similar for IA-strict-UNSAT, $f > 0$ is said **IA-strict-UNSAT** if $f > -\delta$ is IA-UNSAT for $\delta > 0$. We have two lemmas as followings.

Lemma 10.1.2. *If the constraint $\bigwedge_j f_j \geq 0$ is **strict-UNSAT**, it is really UNSAT.*

Lemma 10.1.3. *If the constraint $\bigwedge_j f_j > 0$ is **SAT**, then the constraint $\bigwedge_j f_j \geq 0$ is **SAT**.*

For the greater-than-or-equal constraints $\bigwedge_j f_j \geq 0$, we apply Lemma 10.1.2 for proving UNSAT, and Lemma 10.1.3 for proving SAT. Note that if $\bigwedge_j f_j \geq 0$ is SAT but $\bigwedge_j f_j > 0$ is UNSAT (i.e., kissing situation), **raSAT** says **unknown**. In implementation of **raSAT**, strict-UNSAT is only applied for greater-than-or-equal constraints ($f_j \geq 0$) and IA-UNSAT is replaced by IA-strict-UNSAT.

10.2 Polynomial Equality Handling

10.2.1 Polynomial Equality by Intermediate Value Theorem

For solving polynomial constraints with an equality ($g = 0$), we apply intermediate value theorem. That is, if existing 2 test cases such that $g > 0$ and $g < 0$ then $g = 0$ is SAT.

Lemma 10.2.1. *For a polynomial constraint*

$$F = (x_1 \in (a_1, b_1) \wedge \cdots \wedge x_n \in (a_n, b_n)) \bigwedge_j^m f_j(x_1, \cdots, x_n) > 0 \wedge g(x_1, \cdots, x_n) = 0,$$

if

- (i) *existing a box $(l_1, h_1) \times \cdots \times (l_n, h_n)$ such that $(l_i, h_i) \subseteq (a_i, b_i)$ and $\bigwedge_j^m f_j(x_1, \cdots, x_n) > 0$ is IA-VALID in the box,*
- (ii) *existing two instances $(t_1, \cdots, t_n), (t'_1, \cdots, t'_n)$ in the box such that $g(t_1, \cdots, t_n) > 0$ and $g(t'_1, \cdots, t'_n) < 0$,*

then F is SAT.

We apply Lemma 10.2.1 for proving SAT of polynomial constraints containing an equality. In implementation, we first find an IA-VALID box by interval decomposition and interval arithmetic for inequality constraints (i), and then find 2 instances in the IA-VALID box by testing (ii). Figure 10.1 demonstrates our approach when applying intermediate value theorem.

In Table 10.1 we show preliminary experiment for 15 problems that contain polynomial equalities in Zankl family. **raSAT** works well for these SAT problems and it can detect all SAT problems (11 among 15). At the current implementation, raSAT reports *unknown* for UNSAT problems. The first 4 columns indicate *name of problems, the number of variables, the number of polynomial equalities and the number of inequalities* in each problem, respectively. The last 2 columns show comparison results of **Z3 4.3** and **raSAT**.

We also apply the same idea for multiple equalities $\bigwedge_i g_i = 0$ such that $Var(g_k) \cap Var(g_{k'}) = \emptyset$ where $Var(g_k)$ is denoted for the set of variables in the polynomial g_k . In the next section we will present idea for solving general cases of multiple equalities.

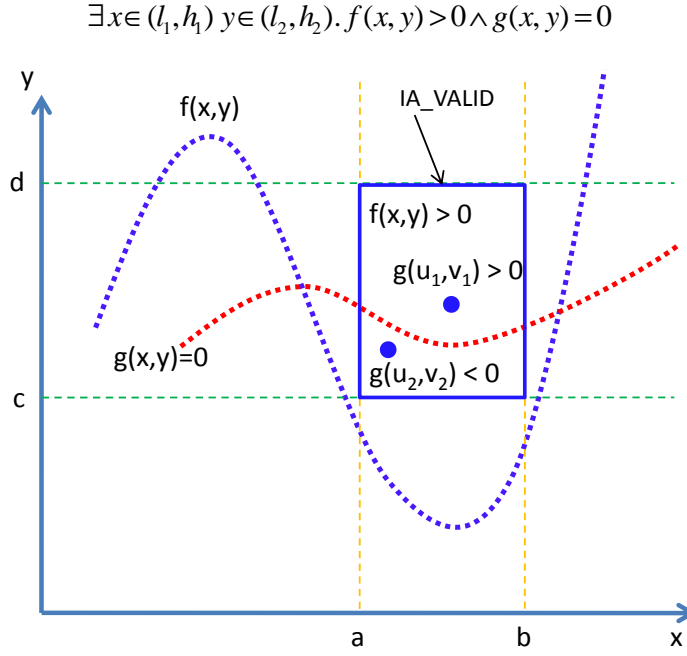


Figure 10.1: Solving an equality by intermediate value theorem

Problem Name	No. Variables	No. Equalities	No. Inequalities	Z3 4.3 (15/15)		raSAT (11/15)	
				Result	Time(s)	Result	Time(s)
gen-03	1	1	0	SAT	0.01	SAT	0.015
gen-04	1	1	0	SAT	0.01	SAT	0.015
gen-05	2	2	0	SAT	0.01	SAT	0.046
gen-06	2	2	1	SAT	0.01	SAT	0.062
gen-07	2	2	0	SAT	0.01	SAT	0.062
gen-08	2	2	1	SAT	0.01	SAT	0.062
gen-09	2	2	1	SAT	0.03	SAT	0.062
gen-10	1	1	0	SAT	0.02	SAT	0.031
gen-13	1	1	0	UNSAT	0.05	unknown	0.015
gen-14	1	1	0	UNSAT	0.01	unknown	0.015
gen-15	2	3	0	UNSAT	0.01	unknown	0.015
gen-16	2	2	1	SAT	0.01	SAT	0.062
gen-17	2	3	0	UNSAT	0.01	unknown	0.031
gen-18	2	2	1	SAT	0.01	SAT	0.078
gen-19	2	2	1	SAT	0.05	SAT	0.046

Table 10.1: Experimental results for 15 equality problems of Zankl family

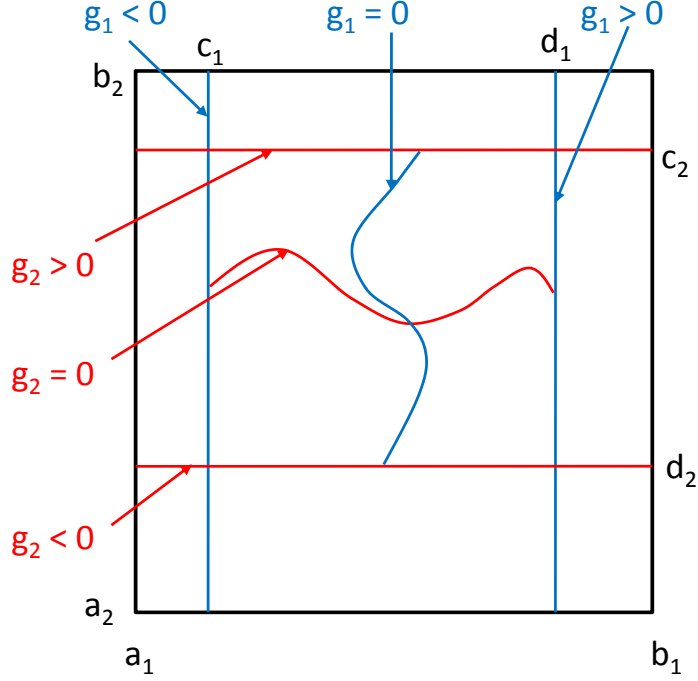


Figure 10.2: Solving multiple equalities

10.2.2 Extension to Multiple Equalities

We first present an approach for solving polynomial constraints with 2 equalities and then extend the same idea for general cases. We assume that constraints have two variables.

For a polynomial constraint $F = (x_1 \in (a_1, b_1) \wedge x_2 \in (a_2, b_2)) \bigwedge_j f_j > 0 \wedge g_1 = 0 \wedge g_2 = 0$, F can be proved as SAT by following steps:

- first, find a box such that $\bigwedge_j f_j > 0$ is IA-VALID, i.e., $(l_1, h_1) \times (l_2, h_2)$,
- find 2 instances $c_1, d_1 \in (l_1, h_1)$ such that $g_1 < 0$ on $\{c_1\} \times (l_2, h_2)$ and $g_1 > 0$ on $\{d_1\} \times (l_2, h_2)$ (values of g_1 are estimated by interval arithmetic),
- find 2 instances $c_2, d_2 \in (l_2, h_2)$ such that $g_2 < 0$ on $(l_1, h_1) \times \{c_2\}$ and $g_2 > 0$ on $(l_1, h_1) \times \{d_2\}$ (values of g_2 are also estimated by interval arithmetic).

Figure 10.2 demonstrates our approach when solving polynomial constraints with 2 equalities. The idea behind is finding a box such that IA-VALID for polynomial inequalities and proving that the line $g_1 = 0$ intersects the line $g_2 = 0$ inside the IA-VALID box.

We apply the idea above for general cases. For a polynomial constraint $F = (x_1 \in (a_1, b_1) \wedge \dots \wedge x_n \in (a_n, b_n)) \bigwedge_j f_j > 0 \bigwedge_{k=1}^m g_k = 0$, F can be proved as SAT by following steps:

- first, find a box such that $\bigwedge_j f_j > 0$ is IA-VALID, i.e., $(a_1, b_1) \times \dots \times (a_n, b_n)$,
- find m instances c_1, \dots, c_m in the IA-VALID box such that $g_i < 0$ is IA-VALID with $(a_1, b_1) \times \dots \times \{c_i\} \times \dots \times (a_n, b_n)$ (bounds of g_i are estimated by interval arithmetic),
- find m instances d_1, \dots, d_m in the IA-VALID rectangle such that $g_i > 0$ is IA-VALID with $(a_1, b_1) \times \dots \times \{d_i\} \times \dots \times (a_n, b_n)$ (bounds of g_i are estimated by interval arithmetic),

with a restriction on variables of equalities $\forall i_1, \dots, i_k. |\cup_{1 \leq l \leq k} Var(g_{il})| \geq k$. The restriction guarantees that $\forall k. g_k = 0$ have the same intersection point in the IA-VALID box.

For proving polynomial equality UNSAT, we transform equalities into inequality forms, i.e., for some $\delta > 0$, $\bigwedge_{k=1}^m g_k = 0$ are transformed into $\bigwedge_{k=1}^m -\delta < g_k < \delta$. We apply below lemma for proving UNSAT of polynomial inequalities.

Lemma 10.2.2. *For a polynomial constraint $F = (x_1 \in (a_1, b_1) \wedge \dots \wedge x_n \in (a_n, b_n)) \bigwedge_j f_j > 0 \bigwedge_{k=1}^m g_k = 0$, if $F' = (x_1 \in (a_1, b_1) \wedge \dots \wedge x_n \in (a_n, b_n)) \bigwedge_j f_j > 0 \bigwedge_{k=1}^m (-\delta < g_k < \delta)$ is UNSAT for some $\delta > 0$, then F is UNSAT.*

Chapter 11

Conclusions

11.1 Summary of the Thesis

The aim of research is to propose and develop an SMT for polynomial constraint solving. In the thesis, we propose an iterative approximation refinement scheme, which is applied for solving polynomial constraints on real numbers. The approximation scheme consists of *interval arithmetic* (over-approximation, aiming to decide unsatisfiability) and *testing* (under-approximation, aiming to decide satisfiability). If both of them fail to decide, *refinement* is applied that allows to focus searching on narrower domains. This scheme is described as an abstract DPLL(T) procedure and is implemented as the SMT solver **raSAT**. The DPLL(T) procedure performs as a refinement loop for interval arithmetic, testing and refinement, in which refinement is guided from interval arithmetic and testing.

Comparison with other SMT solvers, i.e., **Z3 4.3** and **CVC3 2.4.1**, was performed by experiments on the different measures and the benchmarks from SMT-LIB. In comparison with **Z3 4.3**, a very strong SMT solver for non-linear arithmetic, **raSAT** is a little weak for *Zankl* problems, however **raSAT** worked beyond **Z3 4.3** for the *Hong* problems and the three measures (i.e., high degrees, number of variables, number of APIs). The experimental results of **raSAT** seem encouraging and **raSAT** is still in improvements.

Our contributions are summarized as follows:

- In Chapter 3, we present *the DPLL(T) procedure* for over/under-approximation and refinement in very lazy approach, which is applied for constraint solving. For proving or disproving a full truth assignment obtained from a SAT solver, over

and under-approximation are used. *Refinement* is applied for sandwiching over-approximation and under-approximation, which decomposes an atomic formula of the input formula, i.e., ψ to $\psi_1 \vee \psi_2$ such that $\psi \Leftrightarrow \psi_1 \vee \psi_2$. Soundness and completeness of proposed approach for solving polynomial inequality constraints are given in Chapter 4.

- In Chapter 5, *interval arithmetic* is instantiated for over-approximation theory and *testing* for under-approximation theory. We propose a new form of affine interval, called *Chebyshev Affine Interval* (CAI_1). CAI_1 has an advantage over current *affine interval* (i.e., AF, AF_1, AF_2) such that it can keep sources of computation by noise symbols of variables (i.e., ϵ and $|\epsilon|$) for high degrees, which would be useful for guiding refinement from *sensitivity* of variables.
- In Chapter 6, We present UNSAT cores of polynomials that allow to infer other UNSAT domains when a particular domain is detected as IA-UNSAT. Theoretically, UNSAT core would improve efficiency in theory propagation, i.e., clauses for learning are reduced. Though current implementation for UNSAT core is not well developed and we need further improvement. When performing a large number of test data (i.e., a large number of variables), the proposed approach for incremental test data generation is useful for improving performance of testing.
- In Chapter 7, we present strategies for refinement such that choice of intervals to decompose and the ways to decompose an interval to sub intervals. These strategies are guided from interval arithmetic, i.e., *sensitivity* of variables (not yet implemented), from testing, i.e., Test-UNSAT APIs and a test data (*tick points*) for *tick decomposition*, and from monotonical affects of variables on a polynomial (*monotonic decomposition*). The experimental results in this chapter also show efficiency in one of proposed strategies (i.e., choice of intervals to decompose based on Test-UNSAT APIs).
- Solving polynomial constraints including equalities is extended in Chapter 9 by a non-constructive approach based on *intermediate value theorem*. Preliminary experiments of **raSAT** for polynomial equalities show that the extensions reasonably work and their results are comparable with **Z3 4.3** (i.e., 11 solved problems among

15 equalities of Zankl problems).

11.2 Future Directions

We are just in the beginning and have lots of future work that we are considering. Some of them are,

- **Separation of linear constraints:** Many benchmarks contain linear constraints. We expect improvement by separating them for existing SMT with Presburger arithmetic. From their results, vertices of intersecting linear constraints would be candidates of cutoff points for interval decompositions.

For example, if two linear constraints $y - 2x + 1 > 0$ and $y + x - 2 > 0$ are included in a polynomial inequality constraint, their intersecting vertex $x = 1, y = 1$ is applied for interval decomposition. Assume that the intervals $x \in (0, 5)$ and $y \in (-10, 2)$ are used for interval decomposition, choices are,

$$\begin{aligned}x \in (0, 5) &\Leftrightarrow x \in (0, 1) \vee x \in (1, 5) \\y \in (-10, 2) &\Leftrightarrow y \in (-10, 1) \vee y \in (1, 2)\end{aligned}$$

- **Incremental DPLL:** For interactions with the SAT solver, we currently apply the very lazy theory learning. Combination of *less lazy* and *eager* theory propagation would improve efficiency, in which we can propagate a conflict from a partial truth assignment instead of waiting for a full truth assignment obtained by SAT solver.
- **Error guaranteed floating point arithmetic:** Currently, standard floating point arithmetic is used for both interval arithmetic and testing. Although this has benefit on comparison between floating and fix point arithmetic behavior (e.g., round-off/overflow error analysis [25]), the result may not be sound for exact real arithmetic. We are planning to apply error guaranteed floating point arithmetic for SMT implementation by numerical packages of Ocaml.
- **Avoiding local optimality:** we plan to borrow the similar idea of *restart* in MiniSAT for escaping from hopeless local search (i.e., solution set is not dense or

empty). *Heuristics* is planned to apply, i.e., after a deep interval decomposition of a box or Test-UNSAT results are reported for smaller boxes of the same decomposition box.

- **Other directions:** we need further investigation for refinement strategies (both strategies and experiments), i.e., choice of variables for interval decomposition, how to decompose an interval into smaller intervals (i.e., based on sensitivity of variables) etc. In addition, polynomial equality with algebra methods, i.e., ideal based computation, would be combined.

11.3 Applications

We plan to apply our SMT solver **raSAT** for some problems of software/hardware verification, loop invariant generation, such as,

- **Checking overflow and roundoff error:** In the computers, the real numbers are represented by finite numbers (i.e., floating point numbers, fixed point numbers). Due to finite representation, the over-flow and roundoff errors (OREs) may occur. The OREs will be propagated through computations of the program. Further, the computations themselves also cause OREs because the arithmetic needs to round the result to fit the number format. Besides, OREs are also affected by types of statements, i.e., branch, loop, assignment statements. By symbolic execution, ORE constraints are propagated from a program and ORE problems are reduced to problems of solving ORE constraints for verifying whether OREs occur.
- **Loop invariant generation:** The problem of linear invariant generation can be reduced to the problem of non-linear constraint solving. Based on Farkas' Lemma [8], non-linear constraints on coefficients of the target linear invariant are generated and a satisfiable instance of these constraints is a candidate of the linear invariant.

Bibliography

- [1] ANAI, H. Algebraic methods for solving real polynomial constraints and their applications in biology. In *Algebraic Biology Computer Algebra in Biology* (2005), pp. 139–147.
- [2] BARRETT, C., STUMP, A., AND TINELLI, C. The satisfiability modulo theories library (SMT-LIB).
- [3] BARRETT, C., AND TINELLI, C. Cvc3. In *Proceedings of the 19th International Conference on Computer Aided Verification (CAV '07)* (jul 2007), W. Damm and H. Hermanns, Eds., vol. 4590 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 298–302.
- [4] BORRALLERAS, C., LUCAS, S., NAVARRO-MARSET, R., RODRÍGUEZ-CARBONELL, E., AND RUBIO, A. Solving non-linear polynomial arithmetic via sat modulo linear arithmetic. In *Proceedings of the 22nd International Conference on Automated Deduction* (2009), CADE-22, Springer-Verlag, pp. 294–305.
- [5] BOURBAKI, N. *General Topology*. Springer-Verlag, 1989.
- [6] BRYANT, R. E., KROENING, D., OUAKNINE, J., SESHIA, S. A., STRICHMAN, O., AND BRADY, B. Deciding bit-vector arithmetic with abstraction. In *Proceedings of the 13th international conference on Tools and algorithms for the construction and analysis of systems* (2007), TACAS'07, Springer-Verlag, pp. 358–372.
- [7] COLLINS, G. E. Quantifier elimination by cylindrical algebraic decomposition – twenty years of progress. In *Quantifier Elimination and Cylindrical Algebraic Decomposition* (1998), B. F. Caviness and J. R. Johnson, Eds., Springer-Verlag, pp. 8–23.
- [8] COLÓN, M., SANKARANARAYANAN, S., AND SIPMA, H. Linear invariant generation using non-linear constraint solving. In *CAV* (2003), vol. 2725 of *Lecture Notes in Computer Science*, Springer, pp. 420–432.
- [9] COMBA, J. L. D., AND STOLFI, J. Affine arithmetic and its applications to computer graphics. In *Proceedings of VI SIBGRAPI*. (1993), pp. 9–18.
- [10] CORZILIUS, F., AND ÁBRAHÁM, E. Virtual substitution for SMT-solving. In *Proceedings of the 18th international conference on Fundamentals of computation theory* (2011), FCT'11, Springer-Verlag, pp. 360–371.

- [11] CORZILIUS, F., LOUP, U., JUNGES, S., AND ÁBRAHÁM, E. SMT-RAT: an SMT-compliant nonlinear real arithmetic toolbox. In *Proceedings of the 15th international conference on Theory and Applications of Satisfiability Testing* (2012), SAT'12, Springer-Verlag, pp. 442–448.
- [12] DANIEL KROENING, O. S. *Decision Procedures: An Algorithmic Point of View*. Springer-Verlag, 2008.
- [13] EÉN, N., AND SÖRENSON, N. MiniSAT.
- [14] FRANZLE, M., HERDE, C., TEIGE, T., RATSCHAN, S., AND 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.
- [15] GANAI, M., AND IVANCIC, F. Efficient decision procedure for non-linear arithmetic constraints using cordic. In *Formal Methods in Computer-Aided Design, 2009. FMCAD 2009* (2009), pp. 61–68.
- [16] JOVANOVIĆ, D., AND DE MOURA, L. Solving non-linear arithmetic. In *Proceedings of the 6th international joint conference on Automated Reasoning* (2012), IJCAR'12, Springer-Verlag, pp. 339–354.
- [17] KHANH, T. V., AND OGAWA, M. SMT for polynomial constraints on real numbers. *Electr. Notes Theor. Comput. Sci.* 289 (2012), 27–40.
- [18] KLAAS PIETER HART, J.-I. N., AND VAUGHAN, J. E. *Encyclopedia of General Topology*. Elsevier, 2003.
- [19] LUCAS, S., AND NAVARRO-MARSET, R. Comparing csp and sat solvers for polynomial constraints in termination provers. *Electron. Notes Theor. Comput. Sci.* 206 (Apr. 2008), 75–90.
- [20] MESSINE, F. Extensions of affine arithmetic: Application to unconstrained global optimization. *Journal of Universal Computer Science* 8, 2 (2002).
- [21] MICROSOFT. Z3.
- [22] MIYAJIMA, S., T. M., AND KASHIWAGI, M. A new dividing method in affine arithmetic. *IEICE Transactions E86-A*, 9 (2003), 2192–2196.
- [23] MOORE, R. E. *Interval Analysis*. Prentice-Hall, 1966.
- [24] MOSKEWICZ, M. W., MADIGAN, C. F., ZHAO, Y., ZHANG, L., AND MALIK, S. Chaff: engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference* (2001), DAC '01, ACM, pp. 530–535.
- [25] NGOC, D. T. B., AND 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* (2009), SEFM '09, IEEE Computer Society, pp. 105–114.

- [26] NGOC, D. T. B., AND OGAWA, M. Checking roundoff errors using counterexample-guided narrowing. In *Proceedings of the IEEE/ACM international conference on Automated software engineering* (2010), ASE '10, ACM, pp. 301–304.
- [27] NIEUWENHUIS, R., OLIVERAS, A., AND TINELLI, C. Abstract DPLL and abstract DPLL modulo theories. In *Logic for Programming, Artificial Intelligence, and Reasoning*, F. Baader and A. Voronkov, Eds., vol. 3452 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005, pp. 36–50.
- [28] PASSMORE, G. O., AND JACKSON, P. B. Combined decision techniques for the existential theory of the reals. In *Proceedings of the 16th Symposium, 8th International Conference. Held as Part of CISM '09 on Intelligent Computer Mathematics* (2009), Calculemus '09/MKM '09, Springer-Verlag, pp. 122–137.
- [29] RATSCHAN, S. Efficient solving of quantified inequality constraints over the real numbers. *ACM Trans. Comput. Logic* 7, 4 (Oct. 2006), 723–748.
- [30] SANKARANARAYANAN, S., SIPMA, H. B., AND MANNA, Z. Constructing invariants for hybrid systems. *Form. Methods Syst. Des.* 32, 1 (2008), 25–55.
- [31] STOLFI, J. *Self-Validated Numerical Methods and Applications*. PhD thesis, PhD. Dissertation, Computer Science Department, Stanford University, 1997.
- [32] TARSKI, A. A decision method for elementary algebra and geometry. *Bulletin of the American Mathematical Society* 59 (1951).
- [33] WEISPFENNING, V. Quantifier elimination for real algebra - the quadratic case and beyond. *Applicable Algebra in Engineering, Communication and Computing* 8 (1997), 85–101.
- [34] ZANKL, H., AND MIDDELDORP, A. Satisfiability of non-linear (ir)rational arithmetic. In *Proceedings of the 16th international conference on Logic for programming, artificial intelligence, and reasoning* (2010), LPAR'10, Springer-Verlag, pp. 481–500.

Publications

- [1] To Van Khanh, Mizuhito Ogawa. *SMT for Polynomial Constraints on Real Numbers*. Third Workshop on Tools for Automatic Program Analysis (TAPAS' 2012). Electronic Notes in Theoretical Computer Science, vol. 289, Dec 2012, pages 27 – 40.