Equality handling and efficiency improvement of SMT for non-linear constraints over reals

By VU XUAN TUNG

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Professor Mizuhito Ogawa

March, 2015

Equality handling and efficiency improvement of SMT for non-linear constraints over reals

By VU XUAN TUNG (1310007)

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Professor Mizuhito Ogawa

and approved by
Associate Professor Nao Hirokawa
Professor Takuya Katayama
Associate Professor Katsuhiko Gondow

February, 2015 (Submitted)

Abstract

This thesis presents strategies for efficiency improvement and extensions of an SMT solver **raSAT** for polynomial constraints.

raSAT which initially focuses on polynomial inequalities over real numbers follows ICP methodology and adds testing to boost satisfiability detection [11]. In this work, in order to deal with exponential exploration of boxes, several heuristic measures, called *SAT likelyhood*, *sensitivity*, and the number of unsolved atomic polynomial constraints, are proposed. Extensions for handling equations and handling constraints over integer number are also presented.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Polynomial constraint solving

Polynomial constraint solving over real numbers is a task of computing an assignment of real values to variables that satisfies given polynomial inequalities/equations. If such an assignment exists, the constraint is said to be satisfiable (SAT) and the assignment is called SAT instance; otherwise we mention it as unsatisfiable (UNSAT).

Example 1 $x^2 + y^2 < 1 \wedge xy > 1$ *is an example of an UNSAT constraint. While the set of satisfiable points for the first inequality* ($x^2 + y^2 < 1$) *is the read circle in Figure 1.1, the set for the second one is the green area. Because these two areas do not intersect, the conjunction of two equalities is UNSAT.*

Example 2 *Figure 1.2 illustrates the satisfiability of the constraint:* $x^2+y^2 < 4 \wedge xy > 1$. *Any point in the purple area is a SAT instance of the constraint, e.g.* $(1.5, 1)$.

Solving polynomial constraints has many application in Software Verification, such as

- **Locating roundoff and overflow errors**, which is our motivation [14, 15]
- **Automatic termination proving**, which reduces termination detection to finding a suitable ordering [13], e.g., TTT2 [1], AProVE [2].
- **Loop invariant generation**. Farkas's lemma is a popular approach in linear loop invariant generation [4], and is reduced to degree 2 polynomials. Non-linear loop invariant [20] requires more complex polynomials.
- **Hybrid system**. SMT for QF_NRA are often used as backend engines [19].
- **Mechanical contrnol design**. PID control is simple but widely used, and designing parameters is reduced to polynomial constraints [1].

[1] `http://cl-informatik.uibk.ac.at/software/ttt2/`
[2] `http://aprove.informatik.rwth-aachen.de`

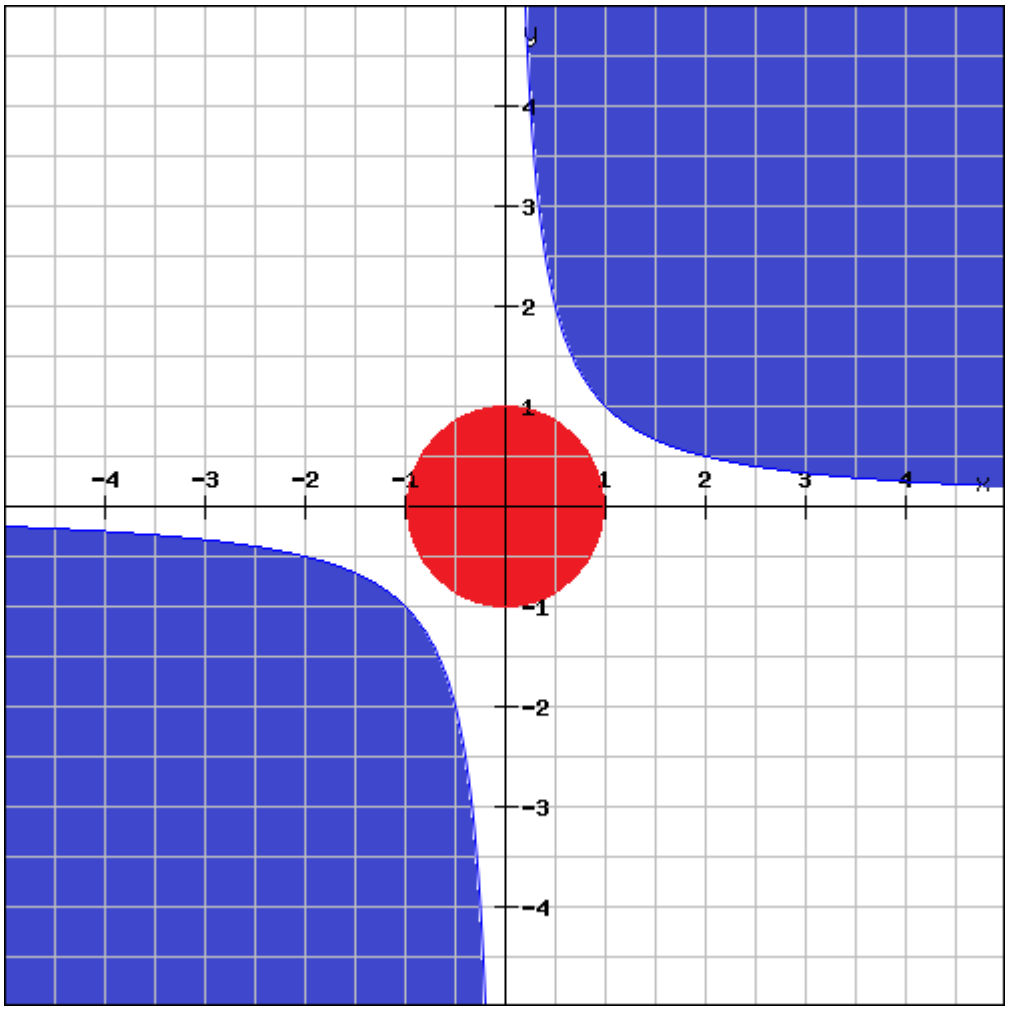


Figure 1.1: Example of UNSAT constraint

1.2 Existing approaches

Although solving polynomial constraints on real numbers is decidable [21], current methodologies have their own pros and cos. They can be classified into the following categories:

1. **Quantifier elimination by cylindrical algebraic decomposition (QE-CAD)** [3] is a complete technique, and is implemented in Mathematica, Maple/SynRac, Reduce/Redlog, QEPCAD-B, and recently in Z3 4.3 (which is referred as nlsat in [10]). Although QE-CAD is precise and detects beyond SAT instances (e.g., SAT regions), scalability is still challenging, since its complexity is doubly-exponential with respect to the number of variables.

2. **Virtual substitution** eliminates an existential quantifier by substituting the corresponding quantified variable with a very small value ($-\infty$), and either each root (with respect to that variable) of polynomials appearing in the constraint or each

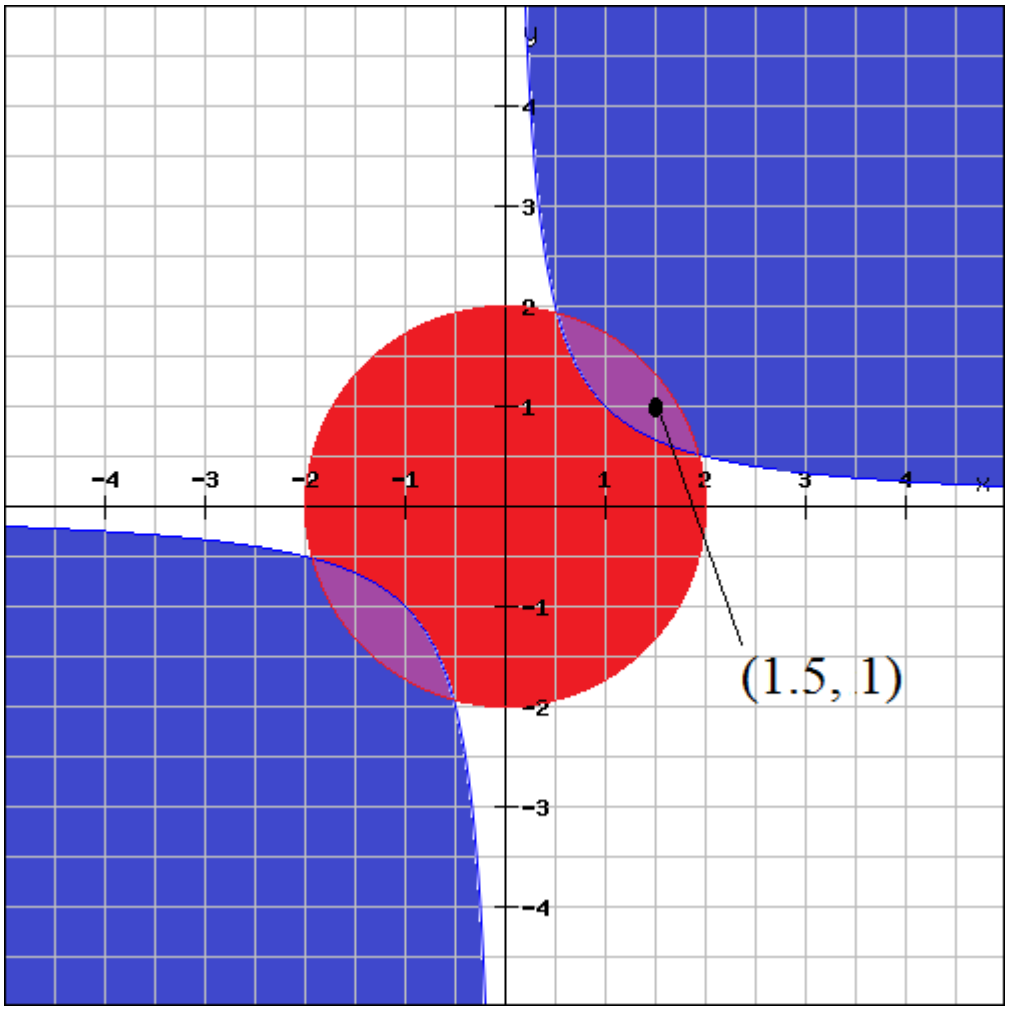


Figure 1.2: Example of SAT constraint

root plus an infinitesimal ϵ. Disjunction of constraints after substitutions is equivalent to the original constraint. Because VS needs the formula for roots of polynomials, its application is restricted to polynomials of degree up to 4. SMT-RAT and Z3 [17] applies VS.

3. **Bit-blasting**. In this category of methodology, numerical variables are represented by a sequence of binary variables. The given constraint is converted into another constraint over the boolean variables. SAT solver is then used to find a satisfiable instance of binary variables which can be used to calculate the values of numerical variables. MiniSmt [22], the winner of QF_NRA in SMT competition 2010, applies it for (ir)-rational numbers. It can show SAT quickly, but due to the bounded bit encoding, it cannot conclude UNSAT. In addition, high degree of polynomial results in large SAT formula which is an obstacle of bit-blasting.

4. **Linearization**. CORD [8] uses COrdinate Rotation DIgital Computer (CORDIC)

for real numbers to linearizes multiplications into a sequence of linear constraints. Each time one multiplication is linearized, a number of new constraints and new variables are introduced. As a consequence, high degree polynomials in the original constraint lead to large number of linear constraints.

5. **Interval Constraint Propagation (ICP)** which are used in SMT solver community, e.g., iSAT3 [7], dReal [9], and RSOLVER [18]. ICP combines over-approximation by interval arithmetics and constraint propagation to prune out the set of unsatisfiable points. When pruning does not work, decomposition (branching) on intervals is applied. ICP which is capable of solving "multiple thousand arithmetic constraints over some thousands of variables" [7] is practically often more efficient than algebraic computation.

1.3 Proposed Approach and Contributions

Our aim is an SMT solver for solving polynomial constraint. We first focus on strict inequalities because of the following reasons.

1. Satisfiable inequalities allow over-approximation. An over-approximation estimates the range of a polynomial f as $O.range(f)$ that covers all the possible values of f, i.e. $range(f) \subseteq O.range(f)$. For an inequality $f > 0$, if $O.range(f)$ stays in the positive side, it can be concluded as SAT. On the other hand, over-approximation cannot prove the satisfiability of SAT equations.

2. Satisfiable inequalities allow under-approximation. An under-approximation computes the range of the polynomial f as $U.range(f)$ such that $range(f) \supseteq U.range(f)$. If $U.range(f)$ is on the positive side, $f > 0$ can be said to be SAT. Due to the continuity of f, finding such an under-approximation for solving $f > 0$ is more feasible than that for $f = 0$.

   – If $f(\bar{x}) > 0$ has a real solution $\bar{x_0}$, there exist rational points near $\bar{x_0}$ which also satisfy the inequality. Solving inequalities over real numbers thus can be reduced to that over rational numbers.

   – The real solution of $f(\bar{x}) = 0$ cannot be approximated to any rational number.

For UNSAT constraint (both inequalities and equations) can be solved by over-approximation. Suppose $O.range(f)$ is the result of an over-approximation for a polynomial f.

1. If $O.range(f)$ resides on the negative side, $f > 0$ is UNSAT.

2. If $O.range(f)$ stays on either negative or positive side, $f = 0$ is UNSAT.

Our approach of "iterative approximation refinement" - **raSAT** loop for solving polynomial constraint was proposed and implemented as an SMT solver named raSAT in [11]. This work improves the efficiency of the tool and extend it to handle equations. The summary of the proposed method in [11] is:

1. Over-approximation is used for both disproving and proving polynomial inequalities. In addition, under-approximation is used for boosting SAT detection. When both of these methods cannot conclude the satisfiability, the input formula is refined so that the result of approximation become more precise.

2. Interval Arithmetic (IA) and Testing are instantiated as an over-approximation and an under-approximation respectively. While IA defines the computations over the intervals, e.g. $[1, 3] +_{IA} [3, 6] = [2, 9]$, Testing attempts to propose a number of assignments from variables to real numbers and check each assignment against the given constraint to find a SAT instance.

3. In refinement phase, intervals of variables are decomposed into smaller ones. For example, $x \in [0, 10]$ becomes $x \in [0, 4] \vee x \in [4, 10]$.

4. Khanh and Ogawa [11] also proposed a method for detecting satisfiability of equations using the Intermediate Value Theorem.

The contributions of this work are as follows:

1. Although the method of using IA is robust for large degrees of polynomial, the number of boxes (products of intervals) grows exponentially with respect to the number of variables during refinement (interval decomposition). As a result, strategies for selecting variables to decomposed and boxes to explore play a crucial role in efficiency. We introduce the following strategies:

   – A box with more possiblity to be SAT is selected to explore, which is estimated by several heuristic measures, called *SAT likelyhood*, and the number of unsolved atomic polynomial constraints.

   – A more influential variable is selected for multiple test cases and decomposition. This is estimated by *sensitivity* which is determined during the computation of IA.

2. Two schemes of incremental search are proposed for enhancing solving process:

   – **Incremental deepening**. raSAT follows the depth-first-search manner. In order to escape local optimums, it starts searching with a threshold that each interval will be decomposed no smaller than it. If neither SAT nor UNSAT is detected, a smaller threshold is taken and raSAT restarts.

   – **Incremental widening**. Starting with a small interval, if raSAT detects UNSAT, input intervals are enlarged and raSAT restarts. For SAT constraint, small (finite) interval allows sensitivity to be computed because Affine Interval [11] requires finite range of variables. As a consequence, our above strategies will take effects on finding SAT instance. For the UNSAT case, combination of small intervals and incremental deepening helps raSAT quickly determines the threshold in which unsatisfiability may be proved by IA.

3. SAT confirmation step by an error-bound guaranteed floating point package **iRRAM**[3], to avoid soundess bugs caused by roundoff errors.

4. This work also implement the idea of using Intermediate Value Theorem to show the satisfiability of equations which was suggested in [11].

5. We also extend raSAT to handle constraint over integer numbers. For this extension, we only generate the integer values for variables in testing phase. In addition, the threshold used for stopping decomposition is set to 1.

1.4 Thesis outline

Coming soon...

[3] `http://irram.uni-trier.de`

Chapter 2

Preliminaries

2.1 Abstract DPLL

2.2 SMT

2.3 Polynomial constraints

A polynomial is formally defined as:

polynomial ::= 1 | variable | rational_constant
| polynomial + polynomial
| polynomial - polynomial
| polynomial * polynomial

constraint ::= bound_constraint $\wedge$ polynomial_constraint

bound_constraint ::= $\bigwedge_i \bigvee_j bound\_atom_{ij}$

bound_atom ::= variable $\in$ [rational_constant, rational_constant]

polynomial_constraint ::= $\bigwedge_i \bigvee_j polynomial\_atom_{ij}$

polynomial_atom ::= polynomial <0
| polynomial >0
| polynomial ≥ 0
| polynomial ≤ 0
| polynomial $= 0$
| polynomial $\neq 0$

Chapter 3

Over-Approximation and Under-Approximation

This chapter presents Interval Arithmetic as an over-approximation theory, and Testing as an under-approximation theory. Let

3.1 Approximation theory

$F = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_j \psi_j(x_1, \cdots, x_n)$, where $\psi_j(x_1, \cdots, x_n)$ is an atomic formula. F is equivalnet to $\exists x_1 \ldots x_n.(\bigwedge_i x_i \in I_i) \wedge (\bigwedge_j \psi_j(x_1, \cdots, x_n))$, and we call $\bigwedge_i x_i \in I_i$ *interval constraints*, and we refer $\bigwedge_j \psi_j(x_1, \cdots, x_n)$ by $\psi(x_1, \cdots, x_n)$. Initially, interval constraints have a form of the conjunction $\bigwedge_i x_i \in I_i$, and later by refinement, $x_i \in I_i$ is decomposed into a clause $\bigvee_j x_i \in I_{i_j}$, which makes a CNF.

As an SMT (SAT modulo theory) problem, boolean variables are assigned to each $x_i \in I_{i_j}$, and truth assignments is produced by a SAT solver, which are proved or disproved by a background theory T whether it satisfies $\psi(x_1, \cdots, x_n)$.

As notational convention, m (the lower case) denotes a variable assignments on x_i's, and M (the upper case) denotes a truth assignment on $x_i \in I_{i_j}$'s. We write $m \in M$ when an instance $m = \{x_i \leftarrow c_i\}$ satisfies all $c_i \in I_{i_j}$ that are assigned true by M.

We assume *very lazy theory learning* [16], and a backend theory T is applied only for a full truth assignment M.

- If an instance m satisfies $\psi(x_1, \cdots, x_n)$, we denote $m \models_T \psi(x_1, \cdots, x_n)$.
- If each instance m with $m \in M$ satisfies $\psi(x_1, \cdots, x_n)$, we denote $M \models_T \psi(x_1, \cdots, x_n)$.

Definition 1 *Let $F = \exists x_1 \in I_1 \cdots x_n \in I_n.\psi(x_1, \cdots, x_n)$. For a truth assignment on M, F is*

- *T-valid if $M \models_T \psi(x_1, \cdots, x_n)$,*

- *T-satisfiable (T-SAT) if $m \models_T \psi(x_1, \cdots, x_n)$ for some $m \in M$, and*
- *T-unsatisfiable (T-UNSAT) if $M \models_T \neg\psi(x_1, \cdots, x_n)$.*

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

Definition 2 *Let $T, O.T, U.T$ be theories.*

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

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

A typical ICP applies $O.T$ only as an interval arithmetic. Later in Section **??**, we will instantiate interval arithmetic as $O.T$. Adding to $O.T$-valid, **raSAT** introduce testing as $U.T$ to accelerate SAT detection.

3.2 raSAT loop

3.3 Soundness - Completeness

refinement

3.4 Interval Arithmetic

Given a polynomial and intervals of the variables, interval arithmetic will compute an interval which contains all possible values of the polynomial.

3.4.1 Classical Interval

Coming soon...

3.4.2 Affine Interval

Coming soon...

3.5 Testing

Coming soon...

Chapter 4

SMT solver design

We design an SMT solver named raSAT [1] to solve polynomial constraint.

Although an $O.T$ refinement loop is enough to implement an ICP based SMT solver, we extend it as **raSAT** (SAT by refinement of approximations) loop to accelerate SAT detection by adding $U.T$, which works as in Fig. **??** (b).

1. When an over-approximation theory $O.T$ detects $O.T$-UNSAT (resp. $O.T$-valid), answer UNSAT (resp. SAT).
2. When an under-approximation theory $U.T$ detects $U.T$-SAT, answer SAT.
3. If neither holds, a refinement is applied.

Our design of an SMT solver **raSAT** applies two heuristic features.

- Incremental widening intervals, and incremental deeping search (Section 5.1).
- Heurstic measures *SAT-likelyhood* and *sensitivity*, for selection of a variable to decompose and a box to explore. (Section 5.2).

raSAT also prepares various interval arithmetic as $O.T$ as in Section **??**, whereas currently only random tesing (*k-random ticks*, which consists of periodical k-test instances with a random offset) is prepared as $U.T$.

A typical theory for $O.T$ and $U.T$ are an interval arithmetic and testing, respectively. We say *IA-valid*, *IA-SAT*, and *IA-UNSAT*, when it is $O.T$-valid, $O.T$-SAT, and $O.T$-UNSAT, respectively. Similarly, we say *test-SAT* and *test-UNSAT*, when it is $U.T$-SAT and $U.T$-UNSAT, respectively. Note that either IA-valid or test-SAT implies SAT, and IA-UNSAT implies UNSAT, whereas IA-SAT and test-UNSAT can conclude neither.

raSAT prepares various Affine intervals, adding to classical interval (CI) [**?**], which keep lower and upper bounds. The weakness of CI is loss of dependency among values. For instance, $x - x$ is evaluated to $(-2, 2)$ for $x \in (2, 4)$.

Affine Interval [**?** **?**] introduces *noise symbols* ϵ, which are interpreted as values in $(-1, 1)$. For instance, $x = 3 + \epsilon$ describes $x \in (2, 4)$, and $x - x = (3 + \epsilon) - (3 + \epsilon)$ is

[1] `http://www.jaist.ac.jp/~mizuhito/tools/rasat.html`

evaluated to 0. The drawback is that the multiplication without dependency might be less precise than CI. Affine intervals also cannot represent infinite intervals, e.g., $(0, \infty)$, since it becomes $\infty + \infty\ \epsilon$. Forms of affine intervals vary by choices how to approximate multiplications. They are,

(i) $\epsilon\epsilon'$ is replaced with a fresh noise symbol (AF) [**?** **?**],

(ii) $\epsilon\epsilon'$ is reduced to the fixed error noise symbol $\epsilon_\pm$ (AF_1 and AF_2) [**?**],

(iii) $\epsilon\epsilon'$ is replaced with $(-1, 1)\epsilon$ (or $(-1, 1)\epsilon'$) (EAI) [**?**],

(iv) $\epsilon\epsilon$ is reduced to fixed noise symbols ϵ_+ or ϵ_- (AF_2) [**?**],

(v) Chebyshev approximation of x^2 introduces a noise symbol $|\epsilon|$ as an absolute value of ϵ with $\epsilon\epsilon = |\epsilon||\epsilon| = |\epsilon| + (-\frac{1}{4}, 0)$ and $\epsilon|\epsilon| = \epsilon + (-\frac{1}{4}, \frac{1}{4})$ [**?**].

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

- *AF_2 estimates the range of f as $-3 - \epsilon_1 - 2\epsilon_2 + 3\epsilon_+ + 3\epsilon_\pm$, thus $(-9, 6)$,*
- *CAI estimates the range of f as $(-4, -\frac{11}{4}) + (-\frac{1}{4}, 0)\epsilon_1 - 2\epsilon_2 + \boldsymbol{3}|\epsilon_1| + (-2, 2)\epsilon_\pm$, thus $(-8, 4.5)$.*

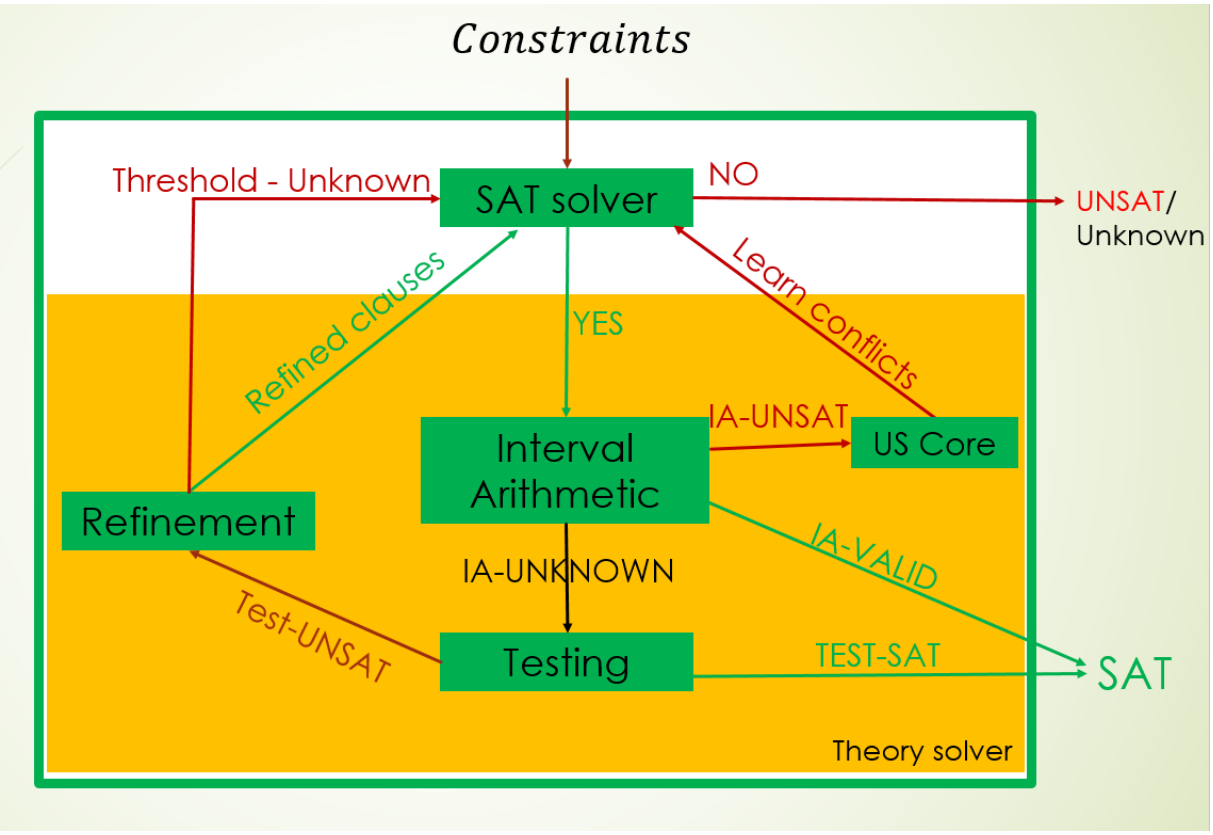


Figure 4.1: **raSAT** design

Chapter 5

Design strategies

We implemented a number of strategies for improving efficiency of raSAT: incremental search and refinement heuristics.

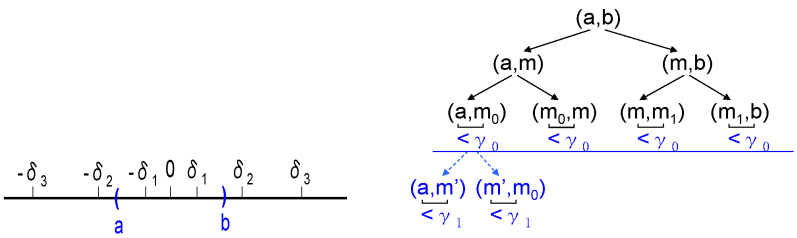
5.1 Incremental search

raSAT applies three incremental strategies, (1) *incremental windening*, (2) *incremental deepening* and (3) *incremental testing*. Let $F = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} f_j > 0$ for $I_i = (a_i, b_i)$.

5.1.1 Incremental windening

Given $0 < \delta_0 < \delta_1 < \cdots$, *incremental windening* starts with $F_0 = \exists x_1 \in I_1 \cap (-\delta_0, \delta_0) \cdots x_n \in I_n \cap (-\delta_0, \delta_0). \bigwedge_{j=1}^{m} f_j > 0$, and if it finishes with UNSAT, it runs with $F_1 = \exists x_1 \in I_1 \cap (-\delta_1, \delta_1) \cdots x_n \in I_n \cap (-\delta_1, \delta_1). \bigwedge_{j=1}^{m} f_j > 0$, and so on (Fig. 5.1 (a)).

Note that if $\delta_i < \infty$, **raSAT** applies an Affine interval; otherwise, it uses CI. Experiments in Section **??** are performed with $\delta_0 = 10$ and $\delta_1 = \infty$.



(a) Incremenal widening (b) Incremental Deepening

Figure 5.1: Incremental Widening and Deepening

5.1.2 Incremental deepening

Starting with $F = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} f_j > 0$, $I_1 \times \cdots \times I_n$ is decomposed into many boxes, and F becomes the disjunction of existential formulae corrsponding to these boxes. **raSAT** searches these boxes in depth-first manner. Figure 5.2 illustrates the search procedure of raSAT. First, the initially box I is examined (by IA and testing) and suppose raSAT detects neither SAT or UNSAT here. As a consequence, I is decomposed into 2 boxes I_{11} and I_{12} such that $\bar{x} \in I \leftrightarrow \bar{x} \in I_{11} \vee \bar{x} \in I_{12}$. raSAT will choose one of the two boxes to explore, for example I_{11}. If IA and testing cannot detect satisfiability of the given constraint in the box I_{11}, decomposition of this box takes place: $\bar{x} \in I_{11} \leftrightarrow \bar{x} \in I_{21} \vee \bar{x} \in I_{22}$. Either I_{21} or I_{22} is selected next. The process continues until SAT/UNSAT is detected which may leads to exhaustive local search. To avoid it, **raSAT** applies a threshold γ, such that no more decomposition will be applied when a box becomes smaller than γ. If neither SAT nor UNSAT is detected, **raSAT** restarts with a smaller threshold.

Let $\gamma_0 > \gamma_1 > \cdots > 0$, and **raSAT** incrementally deepens its search with these thresholds, i.e., starting with δ_0, and if it fails, restart with δ_1, and so on (Fig 5.1 (b)).

5.1.3 Incremental testing

One obstacle in testing is the exponentially large number of test instances. If 2 values are generated for each of n variables, 2^n test cases (combinations of generated values) will present.

Example 4 *Suppose* $\{x, y\}$ *is the set of variables which appears in the input constraint and let* $\{2, 9\}$ *and* $\{5, 8\}$ *are generated values for* x *and* y *respectively. In total* 4 *test cases arise:* $(x, y) = (2, 5), (2, 8), (9, 5), (9, 8)$.

In order to tackle the problem, the following strategies are proposed:

1. Restrict the number of test cases to 2^{10} by choosing most 10 influential variables which are decided by the following procedures for generating multiple (2) test values.
   - Select API by SAT-likelihood
   - Select 1 variable of each selected API.
2. Incrementally generate test values for variables to prune test cases that do not satisfy an API. This was proposed by Khanh and Ogawa in [12]:
   - Dynamically sort the IA-SAT APIs.
   - Generate the test values for variables of selected APIs.

   Example 5 *Let* $x^2 > 4$ *and* $x * y > 0$ *are two IA-VALID APIs to be tested and somehow they are sorted in that order, i.e.* $x^2 > 4$ *is selected before* $x * y > 0$. *Suppose* $\{1, 3\}$ *are generated as test values for* x *which are is enough to test the*

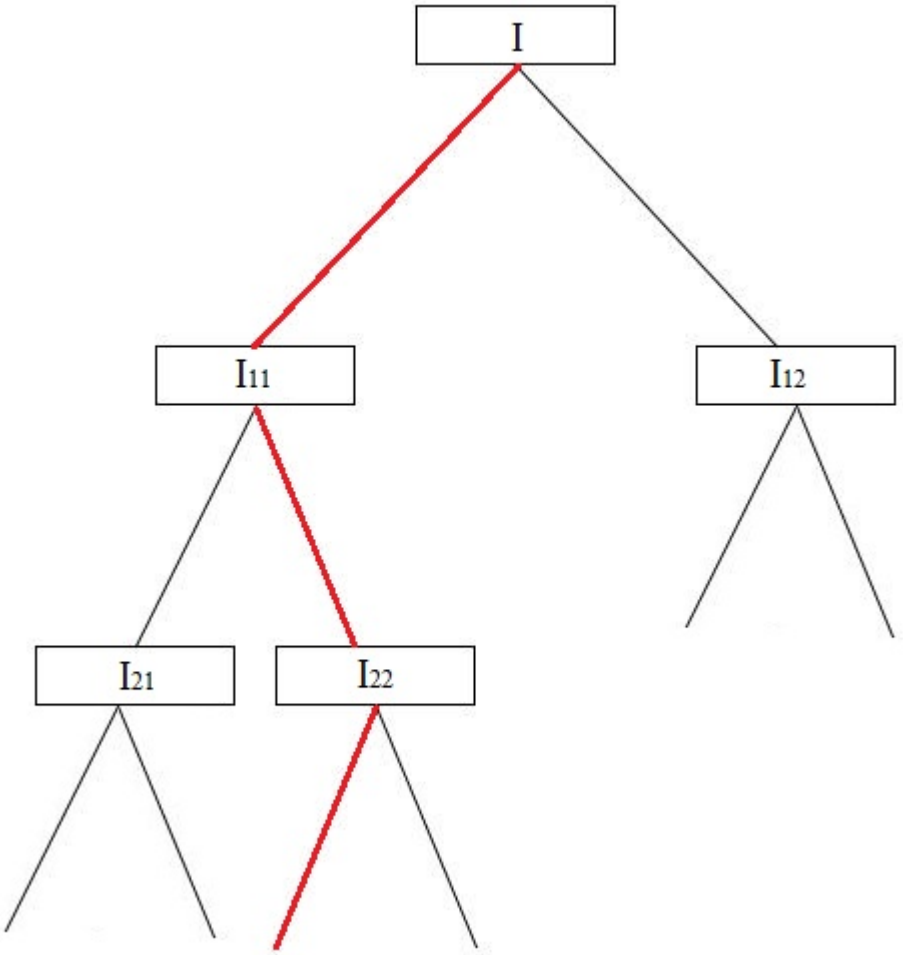


Figure 5.2: **raSAT** design

first selected API, i.e. $x^2 > 4$. *As a result of testing,* $x = 1$ *is excluded from the satisfiable test cases whilst* $x = 3$ *is not. Next, when* $x * y > 0$ *is considered,* y *needs to be generated* 2 *values, e.g.* $\{-3, 4\}$ *and two test cases* $(x, y) = (3, -3), (3, 4)$ *come out to be checked. In this example,* $(x, y) = (1, -3)$ *and* $(x, y) = (1, 4)$ *are early pruned by only testing* $x = 1$ *against* $x^2 > 4$.

5.2 SAT directed heuristics measure

With several hundred variables, we observe that an SMT solver works when either SAT, or UNSAT with small UNSAT core. For the latter, we need an efficient heuristics to find an UNSAT core, which is left as future work. For the former, the keys are how to choose variables to decompose, and how to choose a box to explore. **raSAT** chooses such

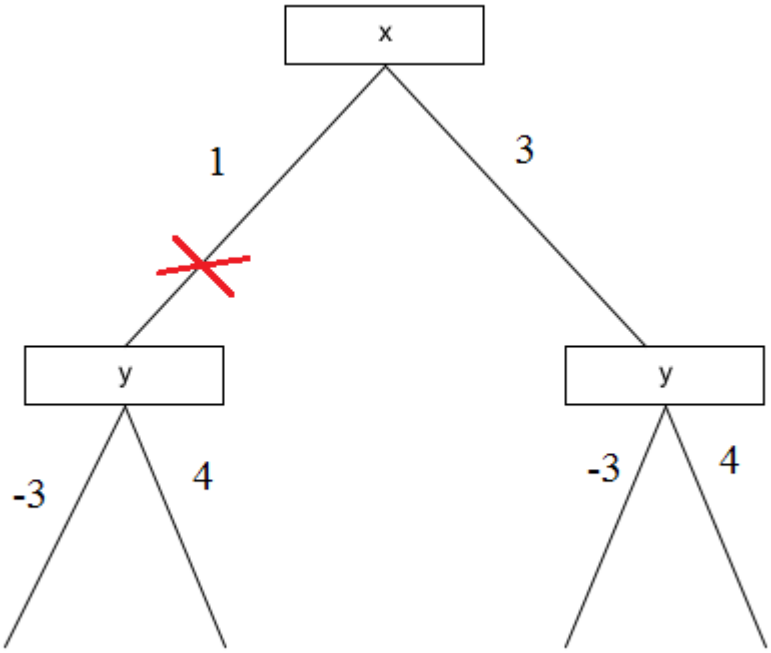


Figure 5.3: Incremental Testing Example

a variable in two steps; first it selects a *test-UNSAT API*, and then chooses a variable that appears in the API. We design SAT-directed heuristic measures based on the interval arithemtic ($O.T$).

Let $F = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} f_j > 0$ becomes $\vee$ $(\exists x_1 \in I'_1 \cdots x_n \in I'_n. \bigwedge_{j=1}^{m} f_j > 0)$ after box decomposition. For $\exists x_1 \in I'_1 \cdots x_n \in I'_n. \bigwedge_{j=1}^{m} f_j > 0$, if some $f_j > 0$ is UNSAT, the box $I'_1 \times \cdots \times I'_n$ is UNSAT. If every $f_j > 0$ is SAT, F is SAT. Thus, if the box $I'_1 \times \cdots \times I'_n$ needs to be explore, it must contain a test-UNSAT API (thus IA-SAT).

We denote the estimated range of f_j for $x_1 \in I'_1 \cdots x_n \in I'_n$ with IA ($O.T$) by $range(f_j, I'_1 \times \cdots \times I'_n)$. If an IA is an affine interval, it is in the form $[c_1, d_1]\epsilon_1 + \cdots + [c_n, d_n]\epsilon_n$, and by instanciating ϵ_i with $[-1, 1]$, the resulting classical interval coincides with $range(f_j, I'_1 \times \cdots \times I'_n)$. We define

- *Sensitivity* of a variable x_i in a test-UNSAT API $f_j > 0$ is $max(|c_i|, |d_i|)$.
- *SAT-likelyhood* of an API $f_j > 0$ is $|I \cap (0, \infty)|/|I|$, and
- *SAT-likelyhood* of a box $I'_1 \times \cdots \times I'_n$ is the least SAT-likelyhood of test-UNSAT APIs.

Example 6 *In Example 3,*

– *sensitivity is estimated* 1 *for* x *and* 2 *for* y *by* AF_2*, and* $3\frac{1}{4}$ *for* x *and* 2 *for* y*.*

– *SAT-likelyhood of* f *is estimated* $0.4 = \frac{6}{9-(-6)}$ *by* AF_2 *and* $0.36 = \frac{4.5}{4.5-(-8)}$ *by* CAI*.*

SAT-likelyhood intends to estimate APIs how likely to be SAT. For choosing variables, **raSAT** first choose a test-UNSAT API by SAT-likelyhood. There are two choices, either *the largest* or *the least*. *Sensitivity* of a variable intends to estimate how a variable is influencial to the value of an API. From a selected API by SAT-likelyhood, **raSAT** selects a variable with the largest sensitivity. This selection of variables are used for (1) *multiple test instances generation*, and (2) *decomposition*. For test generation, we will select multiple variables by repeating the selection.

For choosing a box to explore, **raSAT** chooses more likely to be SAT. There are two choice, (1) a box with the largest SAT-likelyhood, and (2) a box with the largest number of SAT (either IA-valid or test-SAT) APIs.

5.3 Other

UNSAT core, test case generation, interval decomposition

Chapter 6

Experiments

This chapter is going to present the experiments results which reflect how effective our designed strategies are. In addition, comparison between raSAT, Z3 and iSAT3 will be also shown. The experiments were done on a system with Intel Xeon E5-2680v2 2.80GHz and 4 GB of RAM. In the experiments, we exclude the problems which contain equalities because currently raSAT focuses on inequalities only.

6.1 Experiments on strategy combinations

We perform experiments only on Zankl, and Meti-Tarski families.

Our combinations of strategies mentioned in Section **??** are,

Selecting a test-UNSAT API	Selecting a box (to explore):	Selcting a variable:
(1) Least SAT-likelyhood.	(3) Largest number of SAT APIs.	(8) Largest sensitibity.
(2) Largest SAT-likelyhood.	(4) Least number of SAT APIs.	
	(5) Largest SAT-likelyhood.	
	(6) Least SAT-likelyhood.	
(10) Random.	(7) Random.	(9) Random.

Table 6.1 shows the experimental results of above mentioned combination. The timeout is set to 500s, and each time is the total of successful cases (either SAT or UNSAT).

Note that (10)-(7)-(9) means all random selection. Generally speaking, the combination of (5) and (8) show the best results, though the choice of (1),(2), and (10) shows different behavior on benchmarks. We tentatively prefer (1) or (10), but it needs to be investigated further.

Other than heuristics mentioned in Section **??**, there are lots of heuristic choices. For instance,

- how to generate test instances (in $U.T$),
- how to decompose an interval,

and so on.

Benchmark	(1)-(5)-(8)		(1)-(5)-(9)		(1)-(6)-(8)		(1)-(6)-(9)		(10)-(5)-(8)		(10)-(6)-(8)	
Matrix-1 (SAT)	20	132.72 (s)	21	21.48	19	526.76	18	562.19	21	462.57	19	155.77
Matrix-1 (UNSAT)	2	0.00	**3**	0.00	**3**	0.00	**3**	0	**3**	0.00	**3**	0.00
Matrix-2,3,4,5 (SAT)	**11**	632.37	1	4.83	0	0.00	1	22.50	9	943.08	1	30.48
Matrix-2,3,4,5 (UNSAT)	8	0.37	8	0.39	8	0.37	8	0.38	8	0.38	8	0.38

Benchmark	(2)-(5)-(8)		(2)-(5)-(9)		(2)-(6)-(8)		(2)-(6)-(9)		(2)-(7)-(8)		(10)-(7)-(9)	
Matrix-1 (SAT)	**22**	163.47 (s)	19	736.17	20	324.97	18	1068.40	21	799.79	21	933.39
Matrix-1 (UNSAT)	2	0	2	0.00	2	0.00	2	0.00	2	0.00	2	0.00
Matrix-2,3,4,5 (SAT)	5	202.37	1	350.84	1	138.86	0	0.00	0	0.00	0	0.00
Matrix-2,3,4,5 (UNSAT)	8	0.43	8	0.37	8	0.40	8	0.38	8	0.37	8	0.38

Benchmark	(1)-(3)-(8)		(1)-(4)-(8)		(2)-(3)-(8)		(2)-(4)-(8)		(10)-(3)-(8)		(10)-(4)-(8)	
Matrix-1 (SAT)	20	738.26 (s)	21	1537.9	18	479.60	21	867.99	20	588.78	19	196.21
Matrix-1 (UNSAT)	2	0.00	2	0.00	2	0.00	2	0.00	2	0.00	2	0.00
Matrix-2,3,4,5 (SAT)	0	0.00	2	289.17	1	467.12	1	328.03	1	195.18	2	354.94
Matrix-2,3,4,5 (UNSAT)	8	0.36	8	0.36	8	0.34	8	0.37	8	0.37	8	0.39

Benchmark	(1)-(5)-(8)		(1)-(5)-(9)		(10)-(5)-(8)		(10)-(7)-(9)	
Meti-Tarski (SAT, 3528)	3322	369.60 (s)	3303	425.37	**3325**	653.87	3322	642.04
Meti-Tarski (UNSAT, 1573)	1052	383.40	1064	1141.67	**1100**	842.73	1076	829.43

Table 6.1: Combnations of **raSAT** strategies on NRA/Zankl,Meti-Tarski benchmark

Experiments in Table 6.1 are performed with randome generation (k-random tick) for the former and the blanced decomposition (dividing at the exact middle) for the latter. Further investigation is left for future.

6.2 Comparison with other SMT solvers

We compare **raSAT** with other SMT solvers on NRA benchmarks, Zankl and Meti-Tarski. The timeouts for Zankl and Meti-tarski are $500s$ and $60s$, respectively. For **iSAT3**, ranges of all variables are uniformly set to be in the range $[-1000, 1000]$ (otherwise, it often causes segmentation fault). Thus, UNSAT detection of **iSAT3** means UNSAT in the range $[-1000, 1000]$, while that of **raSAT** and **Z3 4.3** means UNSAT over $[-\infty, \infty]$.

Among these SMT solvers, **Z3 4.3** shows the best performance. However, if we closely observe, there are certain tendency. **Z3 4.3** is very quick for small constraints, i.e., with short APIs (up to 5) and a small number of variables (up to 10). **raSAT** shows comparable performance on SAT detection with longer APIs (larger than 5) and a larger number of variables (more than 10), and sometimes outforms for SAT detection on vary long constraints (APIs longer than 40 and/or more than 20 variables). Such examples appear in Zankl/matrix-3-all-*, matrix-4-all-*, and matrix-5-all-* (total 74 problems), and **raSAT** solely solves

- *matrix-3-all-2* (47 variables, 87 APIs, and max length of an API is 27),
- *matrix-3-all-5* (81 variables, 142 APIs, and max length of an API is 20),
- *matrix-4-all-3* (139 variables, 244 APIs, and max length of an API is 73), and

Benchmark	**raSAT**				**Z3 4.3**				**iSAT3**			
	SAT		UNSAT		SAT		UNSAT		SAT		UNSAT	
Zankl/matrix-1 (53)	20	132.72 (s)	2	0.00	41	2.17	12	0.00	11	4.68	3	0.00
Zankl/matrix-2,3,4,5 (98)	11	632.37	8	0.37	13	1031.68	11	0.57	3	196.40	12	8.06
Meti-Tarski (3528/1573)	3322	369.60	1052	383.40	3528	51.22	1568	78.56	2916	811.53	1225	73.83

Table 6.2: Comparison among SMT solvers

– *matrix-5-all-01* (132 variables, 276 APIs, and max length of an API is 47).

Note that, for Zankl, when UNSAT is detected, it is detected very quickly. This is because SMT solvers detects UNSAT only when they find small UNSAT cores, without tracing all APIs. However, for SAT detection with ;arge problems, SMT solvers need to trace all problems. Thus, it takes much longer time.

6.3 Experiments on equality handling

6.4 Polynomial constraints over integers

raSAT loop is easily modified to NIA (nonlinear arithmetic over integers) from NRA, by setting $\gamma_0 = 1$ in incremental deepening in Section 5.1 and restrcting testdata generation on intergers. We also compare **raSAT** (combination (1)-(5)-(8)) with **Z3 4.3** on NIA/AProVE benchmark. **AProVE** contains 6850 inequalities among 8829. Some has several hundred variables, but each API has few variables (mostly just 2 variables).

The results are,

– **raSAT** detects 6764 SAT in 1230.54s, and 0 UNSAT.

– **Z3 4.3** detects 6784 SAT in 103.70s, and 36 UNSAT in 36.08s.

where the timeout is $60s$. **raSAT** does not successfully detect UNSAT, since UNSAT problems have quite large coefficients which lead exhaustive search on quite large area.

Chapter 7

Equality handling

7.1 SAT on Equality by Intermediate Value Theorem

For solving polynomial constraints with single 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 somewhere in between, as in Fig. **??**.

Lemma 1 *For* $F = \exists x_1 \in (a_1, b_1) \wedge \cdots \wedge x_n \in (a_n, b_n). \bigwedge_j^m f_j > 0 \ \wedge \ g = 0$, *F is SAT, if there is a box* $(l_1, h_1) \times \cdots \times (l_n, h_n)$ *with* $(l_i, h_i) \subseteq (a_i, b_i)$ *such that*

(i) $\bigwedge_j^m f_j > 0$ *is IA-VALID in the box, and*

(ii) there are two instances $\vec{t}, \vec{t'}$ *in the box with* $g(\vec{t}) > 0$ *and* $g(\vec{t'}) < 0$.

raSAT first tries to find an IA-VALID box for $\bigwedge_j^m f_j > 0$ by refinements. If such a box is found, it tries to find 2 instances for $g > 0$ and $g < 0$ by testing. Intermediate Value Theorem guarantees the existence of an SAT instance in between. Note that this method works for single equality and does not find an exact SAT instance. If multiple equalities do not share variables each other, we can apply Intermediate Value Theorem repeatedly to decide SAT. In Zankl benchmarks in SMT-lib, there are 15 gen-**.smt2 that contain equality (among 166 problems), and each of them satisty this condition.

In Table 7.1 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**.

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 7.1: Experimental results for 15 equality problems of Zankl family

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.

7.2 Equality handling using Grobner basis

Coming soon...

Chapter 8

UNSAT core

Chapter 9

Conclusion

This paper presented **raSAT** loop, which mutually refines over and under–approximation theories. For polynomial inequlaity, we adopted interval arithemtic and testing for over and under-approximation theories, respectively. **raSAT** loop is implemented as an SMT **raSAT**. The result of Experiments on QF_NRA in SMT-lib is encouraging, and **raSAT** shows comparable and sometimes outperforming to existing SMTs, e.g., Z3 4.3, HySAT, and dReal. For instance, ****** which has ** variables and degree ** was solved by **raSAT**, whereas none of above mentioned SMTS can.

9.1 Observation and Discussion

From experimental results in Section **??** and **??**, we observe the followings.

- The degree of polynomials will not affect much.
- The number of variables are matters, but also for Z3 4.3. The experimental results do not show exponential growth, and we expect the strategy of selection of an API in which related intervals are decomposed seems effective in practice. By observing Zankl examples, we think the maximum number of variables of each API seems a dominant factor.
- Effects of the number of APIs are not clear at the moment. In simple benchmarks, **raSAT** is faster than Z3 4.3, however we admit that we have set small degree $n = 6$ for each API.

For instance, *matrix-2-all-5,8,11,12* in Zankl contain a long monomial (e.g., 60) with the max degree 6, and relatively many variables (e.g., 14), which cannot be solved by Z3 4.3, but **raSAT** does. As a general feeling, if an API contains more than $30 \sim 40$ variables, **raSAT** seems saturating. We expect that, adding to a strategy to select an API (Section **??**), we need a strategy to select variables in the focus. We expect this can be designed with sensitivity (Example 3) and would work in practice. Note that sensitivity can be used only with noise symbols in Affine intervals. Thus, iSAT and RSOLVER cannot use this strategy, though they are based on IA, too.

9.2 Future Work

UNSAT core

Exact confirmation. Currently, **raSAT** uses iRRAM to verify SAT instances only. We are planning to implement confirmation phase to UNSAT cases.

Further strategy refinement. Test data generation, blanced box decomposition

Bibliography

[1] Hirokazu Anai. Algebraic methods for solving real polynomial constraints and their applications in biology. In *In Algebraic Biology Computer Algebra in Biology*, pages 139–147, 2005.

[2] George E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition: A synopsis. *SIGSAM Bull.*, 10(1):10–12, February 1976. ISSN 0163-5824. doi: 10.1145/1093390.1093393. URL `http://doi.acm.org/10.1145/1093390.1093393`.

[3] GeorgeE. Collins. Quantifier elimination by cylindrical algebraic decomposition twenty years of progress. In BobF. Caviness and JeremyR. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts and Monographs in Symbolic Computation, pages 8–23. Springer Vienna, 1998. ISBN 978-3-211-82794-9. doi: 10.1007/978-3-7091-9459-1_2. URL `http://dx.doi.org/10.1007/978-3-7091-9459-1_2`.

[4] MichaelA. Coln, Sriram Sankaranarayanan, and HennyB. Sipma. Linear invariant generation using non-linear constraint solving. In Jr. Hunt, WarrenA. and Fabio Somenzi, editors, *Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 420–432. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-40524-5. doi: 10.1007/978-3-540-45069-6_39. URL `http://dx.doi.org/10.1007/978-3-540-45069-6_39`.

[5] Florian Corzilius, Ulrich Loup, Sebastian Junges, and Erika brahm. Smt-rat: An smt-compliant nonlinear real arithmetic toolbox. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing SAT 2012*, volume 7317 of *Lecture Notes in Computer Science*, pages 442–448. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-31611-1. doi: 10.1007/978-3-642-31612-8_35. URL `http://dx.doi.org/10.1007/978-3-642-31612-8_35`.

[6] James H. Davenport and Joos Heintz. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, 5(12):29 – 35, 1988. ISSN 0747-7171. doi: http://dx.doi.org/10.1016/S0747-7171(88)80004-X. URL `http://www.sciencedirect.com/science/article/pii/S074771718880004X`.

[7] Martin Frnzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex

boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1: 209–236, 2007.

[8] M.K. Ganai and F. Ivancic. Efficient decision procedure for non-linear arithmetic constraints using cordic. In *Formal Methods in Computer-Aided Design, 2009. FMCAD 2009*, pages 61–68, Nov 2009. doi: 10.1109/FMCAD.2009.5351140.

[9] Sicun Gao, Soonho Kong, and EdmundM. Clarke. dreal: An smt solver for non-linear theories over the reals. In MariaPaola Bonacina, editor, *Automated Deduction CADE-24*, volume 7898 of *Lecture Notes in Computer Science*, pages 208–214. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-38573-5. doi: 10.1007/978-3-642-38574-2_14. URL http://dx.doi.org/10.1007/978-3-642-38574-2_14.

[10] Dejan Jovanovi and Leonardo de Moura. Solving non-linear arithmetic. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning*, volume 7364 of *Lecture Notes in Computer Science*, pages 339–354. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-31364-6. doi: 10.1007/978-3-642-31365-3_27. URL http://dx.doi.org/10.1007/978-3-642-31365-3_27.

[11] To Van Khanh and Mizuhito Ogawa. {SMT} for polynomial constraints on real numbers. *Electronic Notes in Theoretical Computer Science*, 289(0):27 – 40, 2012. ISSN 1571-0661. doi: http://dx.doi.org/10.1016/j.entcs.2012.11.004. URL http://www.sciencedirect.com/science/article/pii/S1571066112000746. Third Workshop on Tools for Automatic Program Analysis (TAPAS' 2012).

[12] To Van Khanh and Mizuhito Ogawa. rasat: Smt for polynomial inequality. Technical report, School of Information Science, Japan Advanced Institute of Science and Technology, 05 2013.

[13] Salvador Lucas and Rafael Navarro-Marset. Comparing csp and sat solvers for polynomial constraints in termination provers. *Electron. Notes Theor. Comput. Sci.*, 206:75–90, April 2008. ISSN 1571-0661. doi: 10.1016/j.entcs.2008.03.076. URL http://dx.doi.org/10.1016/j.entcs.2008.03.076.

[14] Do Thi Bich Ngoc and Mizuhito Ogawa. Overflow and roundoff error analysis via model checking. In *Proceedings of the 2009 Seventh IEEE International Conference on Software Engineering and Formal Methods*, SEFM '09, pages 105–114, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3870-9. doi: 10.1109/SEFM.2009.32. URL http://dx.doi.org/10.1109/SEFM.2009.32.

[15] Do Thi Bich Ngoc and Mizuhito Ogawa. Checking roundoff errors using counterexample-guided narrowing. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE '10, pages 301–304, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0116-9. doi: 10.1145/1858996.1859056. URL http://doi.acm.org/10.1145/1858996.1859056.

[16] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Abstract dpll and abstract dpll modulo theories. In *In LPAR04, LNAI 3452*, pages 36–50. Springer, 2005.

[17] Anh-Dung Phan, Nikolaj Bjørner, and David Monniaux. Anatomy of alternating quantifier satisfiability (work in progress). In *10th International Workshop on Satisfiability Modulo Theories (SMT)*, 2012.

[18] Stefan Ratschan. Efficient solving of quantified inequality constraints over the real numbers. *ACM Trans. Comput. Logic*, 7(4):723–748, October 2006. ISSN 1529-3785. doi: 10.1145/1183278.1183282. URL `http://doi.acm.org/10.1145/1183278.1183282`.

[19] Sriram Sankaranarayanan, Henny Sipma, and Zohar Manna. Constructing invariants for hybrid systems. In *in Hybrid Systems: Computation and Control, LNCS 2993*, pages 539–554. Springer-Verlag, 2004.

[20] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Non-linear loop invariant generation using gröbner bases. *SIGPLAN Not.*, 39(1):318–329, January 2004. ISSN 0362-1340. doi: 10.1145/982962.964028. URL `http://doi.acm.org/10.1145/982962.964028`.

[21] Alfred Tarski. A decision method for elementary algebra and geometry. In BobF. Caviness and JeremyR. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts and Monographs in Symbolic Computation, pages 24–84. Springer Vienna, 1998. ISBN 978-3-211-82794-9. doi: 10.1007/978-3-7091-9459-1_3. URL `http://dx.doi.org/10.1007/978-3-7091-9459-1_3`.

[22] Harald Zankl and Aart Middeldorp. Satisfiability of non-linear (ir)rational arithmetic. In *Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, LPAR'10, pages 481–500, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-17510-4, 978-3-642-17510-7. URL `http://dl.acm.org/citation.cfm?id=1939141.1939168`.