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, 2003

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 Satoshi Sato
Professor Takuya Katayama
Associate Professor Katsuhiko Gondow

February, 2003 (Submitted)

Abstract

This thesis presents an SMT solver **raSAT** for polynomial constraints. Solving polynomial constraints is raised from many applications of Software Verification such as round-off/overflow error analysis, automatic termination proving or loop invariant generation. In 1948, Tarski proposed a decision method for solving polynomial constraints over real numbers. **raSAT** consists of a simple iterative approximation refinement, called **raSAT** loop, which is an extension of the standard ICP (Interval Constraint Propagation) with testing. Two approximation schemes consist of interval arithmetic (over-approximation) and testing (under-approximation), to accelerate SAT detection. If both fails, input intervals are refined by decomposition.

ICP is robust for large degrees, but the number of boxes (products of intervals) to explore exponentially explodes when the number of variables increases. We design strategies for boosting SAT detection on the choice of a variable to decompose and a box to explore.

to choose a variable for decomposition and a box by targeting on SAT detection.

Several heuristic measures, called *SAT likelyhood*, *sensitivity*, and the number of unsolved atomic polynomial constraints, are compared on Zankl and Meti-Tarski benchmarks from QF_NRA category of SMT-LIB. They are also evaluated by comparing **Z3 4.3** and **iSAT3**. We also show a simple modification to handle mixed intergers, and experiments on AProVE benchmark from QF_NIA category of SMT-LIB.

Chapter 1

Introduction

1.1 Polynomial constraint solving

Polynomial constraint solving is a task of computing an assignment of variables to real/integer numbers 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 [8, 9]
- **Automatic termination proving**, which reduces termination detection to finding a suitable ordering [7], e.g., TTT2 [1], Aprove [2].
- **Loop invariant generation**. Farkas's lemma is a popular approach in linear loop invariant generation [6], and is reduced to degree 2 polynomials. Non-linear loop invariant [11] requires more complex polynomials.
- **Hybrid system**. SMT for QF_NRA are often used as backend engines [10].
- **Mechanical contrnol design**. PID control is simple but widely used, and designing parameters is reduced to polynomial constraints [**?**].

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

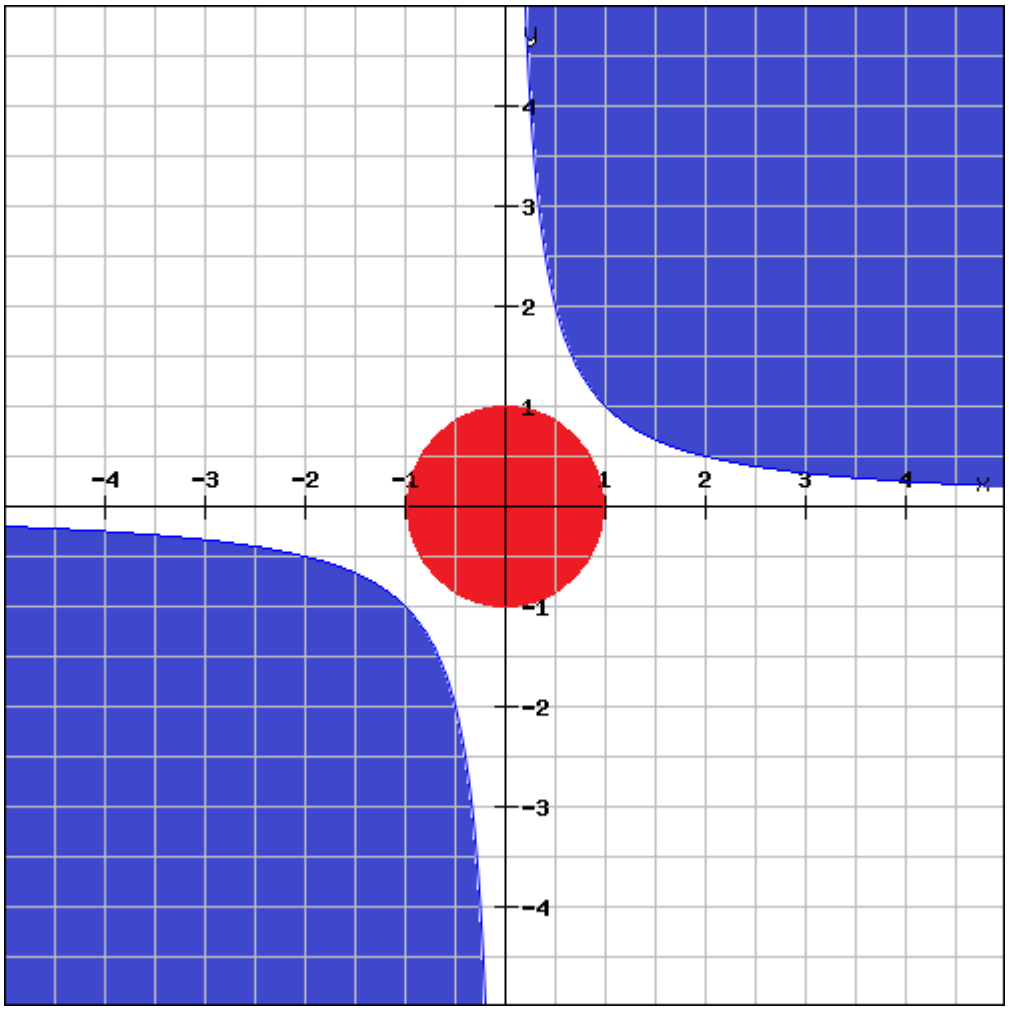


Figure 1.1: Example of UNSAT constraint

1.2 Existing approaches

Although solving polynomial constraints on real numbers is decidable [1], 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)** [2] 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 [3]). Although QE-CAD is precise and detects beyond SAT instances (e.g., SAT regions), scalability is still challenging, since it is DEXPTIME.

2. **Virtual substitution (VS)**. SMT-RAT toolbox [4][5] combines VS, incremental DPLL, and eager theory propagation. Z3 (version 3.1), the winner of QF_NRA in SMT competition 2011, combines VS, ICP, and linearization.

3. **Bit-blasting**. Bit-blasting in bounded bit width is often used in SMTs for QF_NIA.

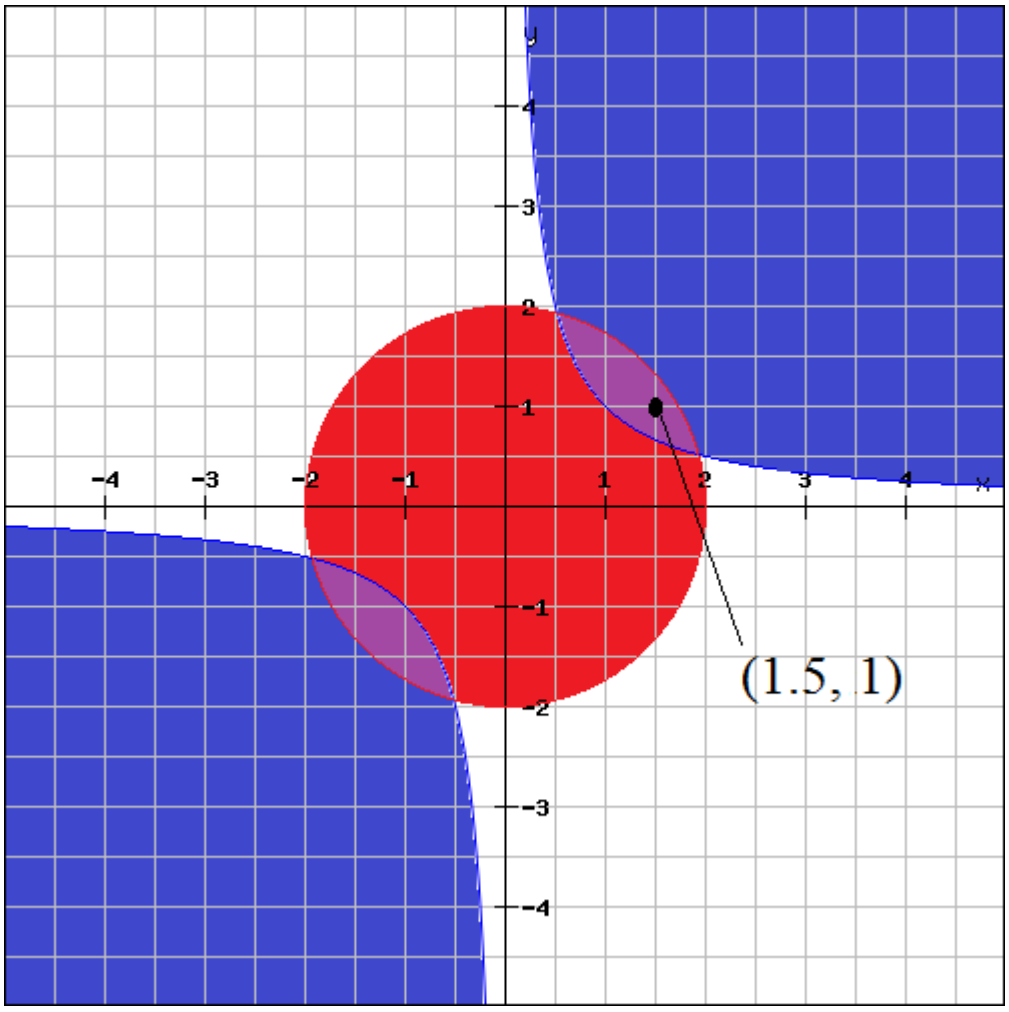


Figure 1.2: Example of SAT constraint

UCLID [?] reduces the number of bits (i.e., narrowing bounds for SAT instances) as an under-approximation, and removes clauses as an over-approximation. They refine each other, which shares a similar sprit with **raSAT** loop. MiniSmt [?], the winner of QF_NRA in SMT competition 2010, applies it for rational numbers with symbolic representations for prefixed algebraic numbers. MiniSmt can show SAT quickly with small degree polynomials, but due to the bounded bit encoding, it cannot conclude UNSAT. Bit-blasting also suffers a lot when the degree of polynomials increases.

4. **Linearization**. CORD [?] uses another linearization, called CORDIC (COrdinate Rotation DIgital Computer) for real numbers. Both Barcelogic and CORD apply Yices for solving linear constraints. Linearization also suffers a lot when the degree of polynomials increases.

5. **Interval Constraint Propagation** which are used in SMT solver community, e.g., **iSAT3** [?], **dReal** [?], and RSOLVER [?]. ICP is based on over-approximation

by interval arithmetics, and iteratively refines by interval decompositions. It is practically often more efficient than algebraic computation with weaker theoretical completeness.

1.3 Proposed Approach and Contributions

Our aim is an efficient decision method for solving polynomial constraint. We combine ICP with Grobner basis method in order to solve the following cases.

1.3.1 Inequalities

Constraints with inequalities can be categorized into four cases:

1. **SAT with without touching**

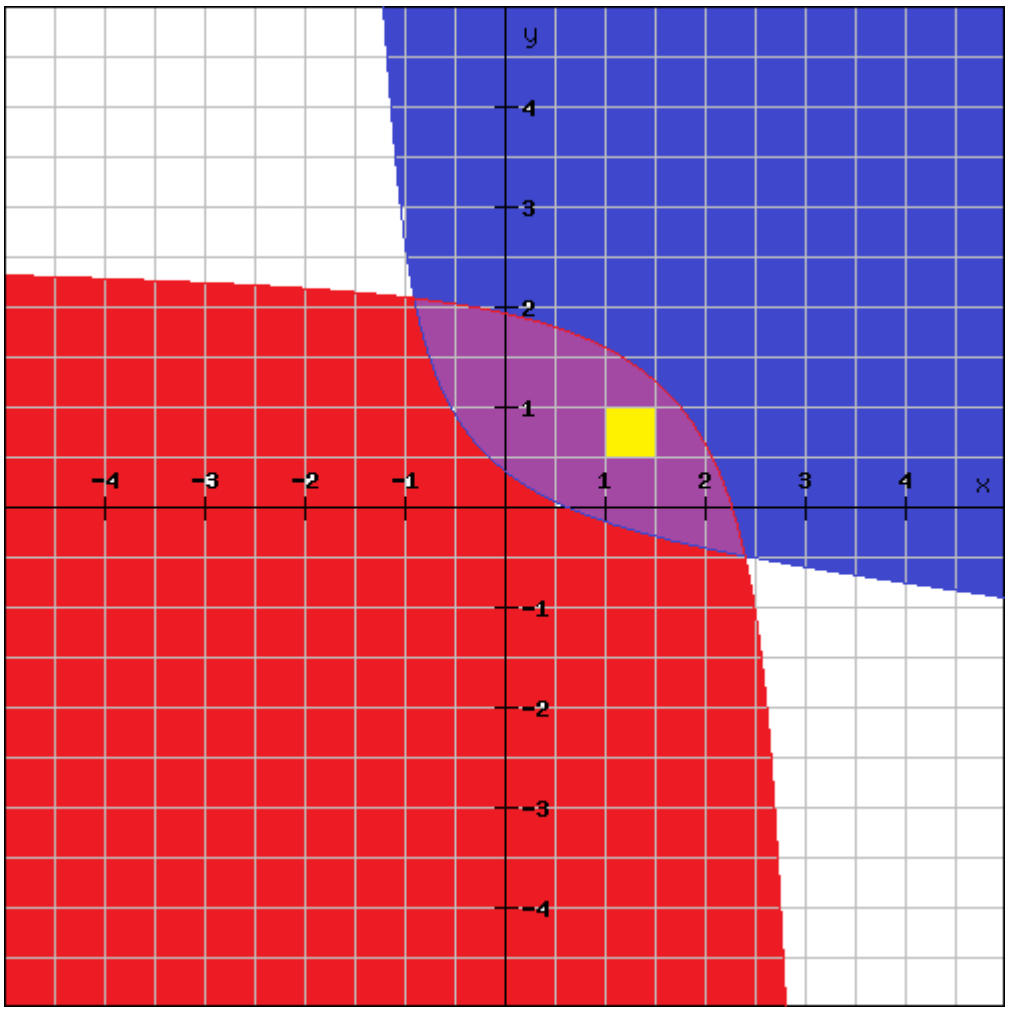


Figure 1.3: SAT without touching dectected by ICP

2. **SAT/UNSAT with touching/convergence**

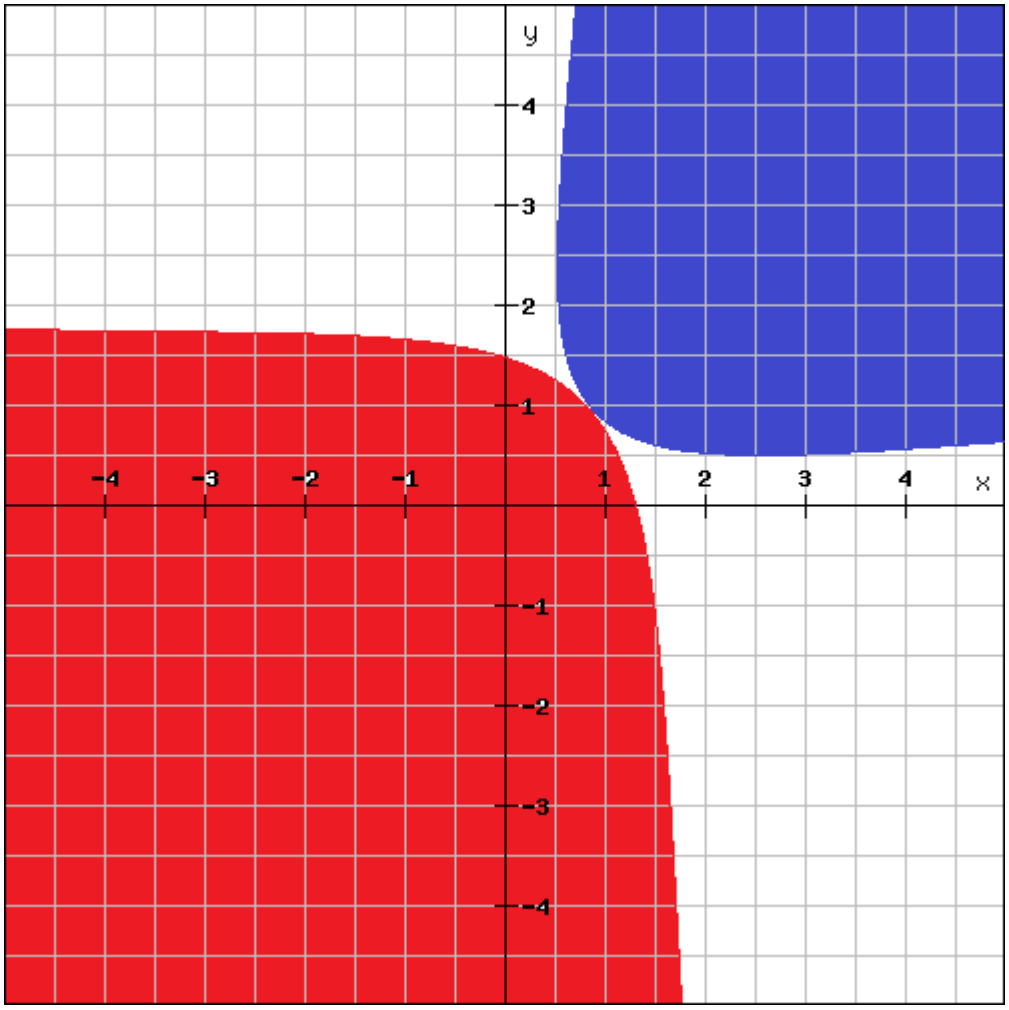


Figure 1.4: SAT(UNSAT) detected by Grobner basis method

3. **UNSAT without touching/convergence**

$\exists x_1 \in (a_1, b_1) \cdots x_n \in (a_n, b_n).\ \wedge_i f_i > 0$,

- If $\exists x_1 \in (a_1, b_1) \cdots x_n \in (a_n, b_n).\ \wedge_i f_i > 0$ is SAT, ICP eventually detects it.
- If $\exists x_1 \in [a_1, b_1] \cdots x_n \in [a_n, b_n].\ \wedge_i f_i \geq 0$ is UNSAT, ICP eventually detects it

under the assumptions of *fair* decomposition and bounded intervals.

The boundary part is reduced to polynomial equality checking, which would be solved algebraic methods, like Groebner basis. Alternatively, by loosening equality to δ-equality, δ-completeness is obtained [**?**, **?**].

This paper presents an SMT solver **raSAT** for polynomial inequality. It consists of a simple iterative approximation refinement, called **raSAT** *loop*, which is an extension of

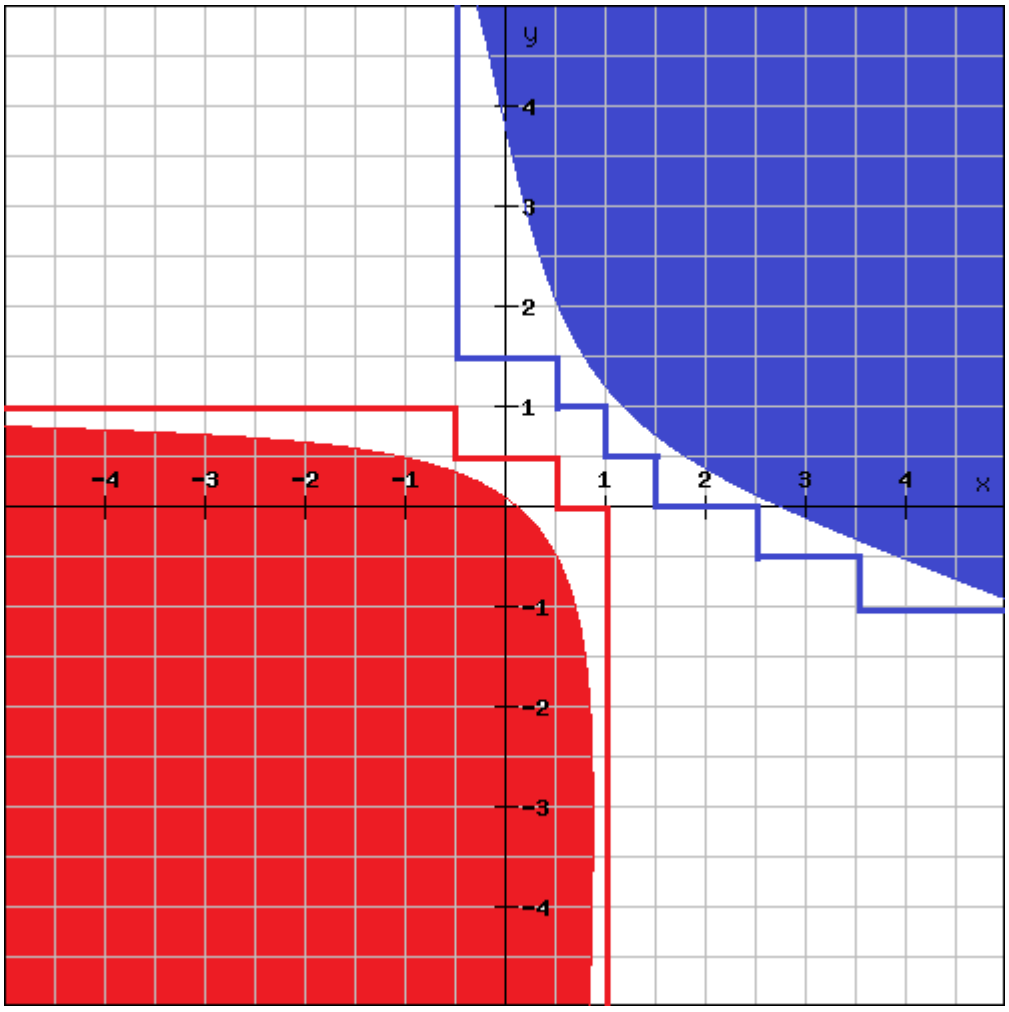


Figure 1.5: UNSAT detected by ICP

the standard ICP with testing to accelerate SAT detection. Two approximation schemes consist of interval arithmetic (over-approximation) and testing (under-approximation), to accelerate SAT detection. If both fails, input intervals are refined by decomposition. Compared to typical ICPs, **raSAT**

- introduces testing (as an under-approximation) to accelerate SAT detection,
- applies various interval arithmetic, e.g., Affine intervals [**?**, **?**, **?**], which enables to analyze the effects of input values, and
- SAT confirmation step by an error-bound guaranteed floating point package **iRRAM**[3], to avoid soundess bugs caused by roundoff errors.

This design is more on SAT detection oriented, since from our preliminary experiences, if the target problems have several hundred variables, solvable cases in practice are either

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

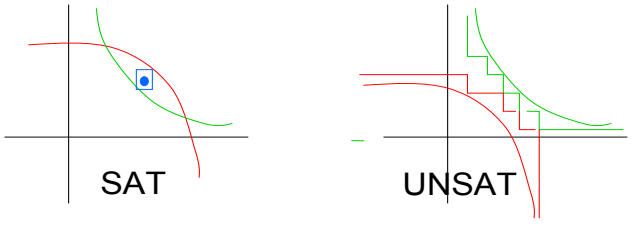


Figure 1.6: SAT and UNSAT detection by ICP

SAT or UNSAT with small UNSAT core. Thus, acceleration of SAT detection and finding UNSAT core will be keys for scalability.

ICP is robust for larger degrees, but the number of boxes (products of intervals) to explore exponentially explodes when variables increase. Thus, design of strategies for selecting variables to decompose and boxes to explore is crucial for efficiency. Our strategy design is,

- 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, and
- a more influential variable is selected for multiple test cases and decomposition, which is estimated by *sensitivity*.

Note that *SAT likelyhood* and *sensitivity* are estimated during interval arithmetic. Especially, the latter can be applied only with Affine intervals. **raSAT** also applies incremental search, which is often faster in practice.

- **Incremental widening**. Starting **raSAT** loop with a smaller interval, and if it is UNSAT, enlarge the input intervals and restart.
- **Incremental deepening**. Starting with the bound that each interval will be decomposed no smaller than it. If neither SAT nor UNSAT is detected, take a smaller bound and restart.

Efficient UNSAT core and UNSAT confirmation are left for future work.

They are compared on Zankl and Meti-Tarski benchmarks from QF_NRA category of SMT-LIB[4]. They are also evaluated by comparing **Z3 4.3**[5] and **iSAT3**. Another advantage of **raSAT** is the ease to handle mixed intergers, and experiments on AProVE benchmark from QF_NIA category of SMT-LIB compares **raSAT** with **Z3 4.3**. Although **Z3 4.3** performs the best, **raSAT** shows comparable SAT detection on very large problems (e.g., with several hundred variables) with the combination of *SAT likelyhood* and *sensitivity*.

raSAT applies SAT confirmation to avoid soundness errors caused by roundoff/overflow errors. Another static analysis based approach is found in [**?**].

[4] `http://www.smtlib.org/`
[5] `http://z3.codeplex.com`

Chapter 2

Over-Approximation and Under-Approximation for Polynomials

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

2.1 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* [**?**], 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.

2.2 Interval Arithmetic

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

2.2.1 Classical Interval

2.2.2 Affine Interval

2.3 Testing

Chapter 3

Soundness - Completeness

3.1 Soundness

3.2 Completeness

Chapter 4

SMT solver design

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

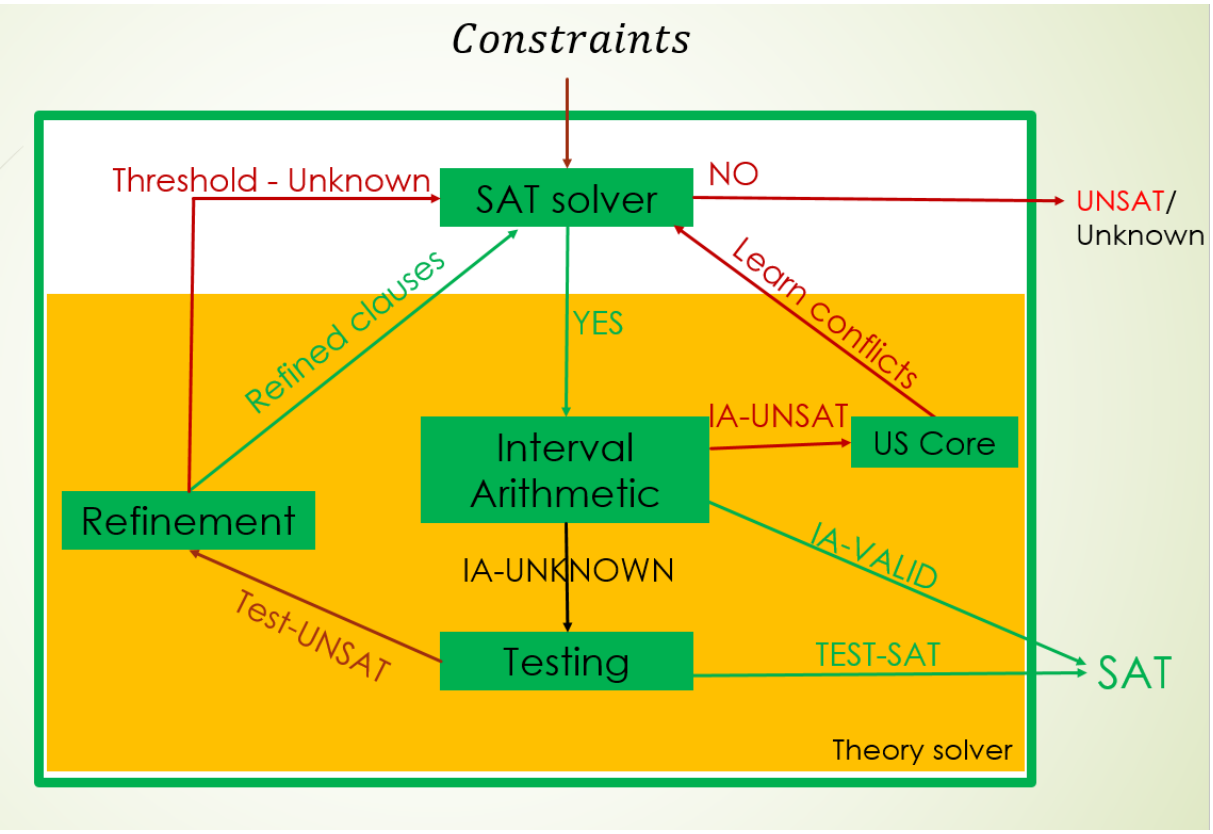


Figure 4.1: raSAT design

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

Chapter 5

Design strategies

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

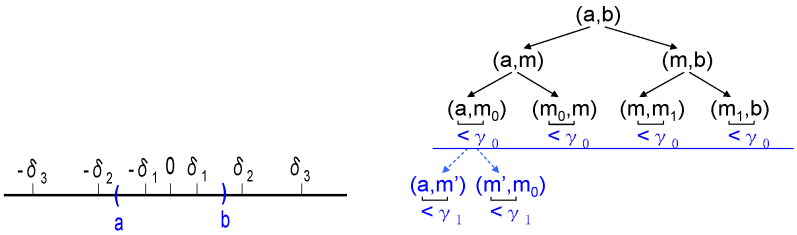
5.1 Incremental search

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

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, which may leads to local optimal 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 threshhold.

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

Large number of test cases affect the efficiency of testing phase. Suppose n variables appear in the polynomial of constraint and 2 test cases are generated for each variable, totally 2^n combinations of test cases are generated.

Example 3 *Variables set of the constraint in Example 1 is* $\{x, y\}$ *(the number of variables* $n = 2$*). If we generate* 2 *test cases for each variable:* $\{5, 6\}$ *for* x *and* $\{1, 7\}$ *for* y*,* $2^n = 4$ *combinations of those test cases are presented* $\{(5, 1), (5, 7), (6, 1), (6, 7)\}$.

raSAT solves this problem by:

1. restricting the number of test cases to 2^{10}:
   - Ten variables are selected and raSAT generates 2 test cases for each.
   - One test case are prepared for each remaining variable.
2. on-demand generation of test cases: Each inequality/equation is sequentially selected for testing. Only test cases for variables of the chosen inequality/equation are generated. This can early avoid unsatisfiable combinations.

   Example 4 *Let the constraint is* $x > 1 \wedge y * z > 3$*. In testing phase, e.g.,* $x > 1$ *will be selected first and test cases for* x *is* $\{0, 2\}$*. Here the test case* $x = 0$ *does not satisfy* $x > 1$*, it will be removed without combining with test cases of* y *and* z*. On the other hand, if test cases of* y *and* z *are also generated at the beginning, e.g,* $\{2, 7\}$ *for* y *and* $\{5, 9\}$ *for* z*; the 4 unnecessary combinations:* $\{(0, 2, 5), (0, 2, 9), (0, 7, 5), (0, 7, 9)\}$ *need to be all checked.*

SAT-likelihood and sensitivity are criteria used for selecting inequality/equation and variable respectively.

Definition 3 *Let* $F = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} f_j > 0$ *be a constraint. 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 instantiating* 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.*

5.1.4 Over-Approximation Theory Refinement

From now on, We focus on a *polynomial inequality* such that I_i and $\psi_j(x_1, \cdots, x_n)$ are an open interval (a_i, b_i) and an atomic polynomial inequaltiy (API) $f_j > 0$, respectively. We denote $\mathbb{S}(f_j) = \{x \in \mathbb{R}^n \mid f_j > 0 \text{ holds}\}$.

For ICP, it is folklore that, for polynomial inequality $\exists x_1 \in (a_1, b_1) \cdots x_n \in (a_n, b_n).\ \wedge_i f_i > 0$,

- if $\exists x_1 \in (a_1, b_1) \cdots x_n \in (a_n, b_n).\ \wedge_i f_i > 0$ is SAT, ICP eventually detects it, and
- if $\exists x_1 \in [a_1, b_1] \cdots x_n \in [a_n, b_n].\ \wedge_i f_i \geq 0$ is UNSAT, ICP eventually detects it,

under the assumptions of *fair* decomposition and bounded intervals (a_i, b_i). We will prepare terminology and briefly review this fact.

Definition 4 *An* open box *of dimension* n *is a set* $(a_1, b_1) \times \cdots \times (a_n, b_n)$ *where* $a_i, b_i \in \mathbb{R}, a_i \leq b_i$*. For* $\mathfrak{a} = (a_1, \cdots, a_n)$ *and* $\mathfrak{b} = (b_1, \cdots, b_n)$*, we denote* $(a_1, b_1) \times \cdots \times (a_n, b_n)$ *by* $(\mathfrak{a}, \mathfrak{b})$*.*

The set of all open boxes is a basis of Euclidean topology on $\mathbb{R}^n$. In $\mathbb{R}^n$, a set U is compact if, and only if, U is a bounded closed set. We denote a closure of a set U by $\overline{U}$. Since a polynomial is continuous, $\mathbb{S}(\bigwedge_{i=1}^{m} f_i > 0)$ is an open set. Note that $\mathbb{Q}$ is dense in $\mathbb{R}$, and an SAT instance in reals can be replaced with one in rationals.

Initially, interval constraints consists of conjunction only. Later, by refinements, it becomes a CNF.

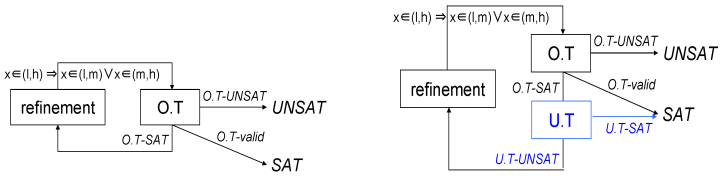
$\exists x \in (-1, 3)\ y \in (2, 4).(x^3y - y^4 > 0) \wedge (y^3 - xy > 0)$ is an example of a polynomial inequality with 2 variables and 2 APIs.

For instance, $x \in (-1, 3)$ and $y \in (2, 4)$ are refined to smaller intervals such that $\exists x \in (-1, 1) y \in (2, 4).(x^3y - y^4 > 0) \wedge (y^3 - xy > 0) \vee \exists x \in (1, 3) y \in (2, 4).(x^3y - y^4 > 0) \wedge (y^3 - xy > 0)$, which results a CNF $(x \in (-1, 1) \vee x \in (1, 3)) \wedge (y \in (2, 4)) \wedge (x^3y - y^4 > 0) \wedge (y^3 - xy > 0)$.

Note that an interval arithmetic used in ICP is a converging theory.

Definition 5 *Let* $F = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} f_j > 0$ *be a polynomial inequality such that each* I_i *is bounded. An over-approximation theory* $O.T$ *is* converging *if, for each* $\delta > 0$ *and* $c = (c_1, \cdots, c_n) \in I_1 \times \cdots \times I_n$*, there exists* $\gamma > 0$ *such that* $\bigwedge_{j=1}^{n} x_j \in (c_j - \gamma, c_j + \gamma) \models_{O.T} \bigwedge_{i=1}^{m} (f_i(c) - \delta < f_i(x) < f_i(c) + \delta)$.

$O.T$ refinemnet loop is shown in Fig. 5.2 (a). A standard ICP based algorithm of an SMT solver applies it with $O.T$ as a classical interval arithemtic. The variation of interval arithemtic will be presented in Section **??**.



(a) $O.T$ refinement loop **raSAT** loop

Figure 5.2: Rfinement loops

Definition 6 *Let* $F = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} f_j > 0$ *for* $I_i = (a_i, b_i)$*. A refinement strategy is* fair*, if, for each* $c_j \in (a_j, b_j)$ *and* $\gamma > 0$*, a decomposition of* I_i *for each* i *eventually occurs in* $(c_j - \gamma, c_j + \gamma)$ *(as long as neither SAT nor UNSAT is detected).*

Theorem 1 *Let* $F = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} f_j > 0$ *for* $I_i = (a_i, b_i)$*. Assume that an over-approximation theory* $O.T$ *is converging, each* (a_i, b_i) *is bounded, and a refinement strategy is fair. Then,*

- *if* $\exists x_1 \in (a_1, b_1) \cdots x_n \in (a_n, b_n). \wedge_i f_i > 0$ *is SAT,* $O.T$ *refinemnet loop eventually detects it, and*
- *if* $\exists x_1 \in [a_1, b_1] \cdots x_n \in [a_n, b_n]. \wedge_i f_i \geq 0$ *is UNSAT,* $O.T$ *refinement loop eventually detects it.*

Proof 1 *The former is proved by the fact that, if* F *is SAT, there exists a non-empty neiborhood (open box) in* $\cap\, \mathbb{S}(f_j)$*. If the box decomposition strategy is fair, the refinemnet will eventually find such an open box.*

For the latter, assume that $\overline{F} = \exists x_1 \in [a_1, b_1] \cdots x_n \in [a_n, b_n]. \wedge_i f_i \geq 0$ *is UNSAT. Thus,* $\cap\, \overline{\mathbb{S}(f_i)} = \emptyset$*. Let* $\delta_{j,k} = min\{|f_j(\bar{x}) - f_k(\bar{x})| \mid \bar{x} \in I_1 \times \cdots \times I_n\}$*. Since* f_i*'s are continuous and* $\overline{I_i}$*'s are compact,* $\delta_{j,k}$ *is well defined, and* $\delta_{j.k} > 0$ *for some* j, k*. Let* $\delta = \frac{min\{\delta_{j,k}\}}{2}$*. Since* $O.T$ *is converging, there exists* $\gamma > 0$ *for* $\delta > 0$ *satisfying Definition 5, and fair decomposition eventually finds open boxes such that* $\mathbb{S}(f_j)$ *and* $\mathbb{S}(f_k)$ *are separated.*

Limitations for detecting UNSAT occur on *kissing* and *convergent* cases. Fig. 5.3 left shows a kissing case $x^2 + y^2 < 2^2 \wedge (x-4)^2 + (y-3)^2 < 3^2$ such that $\overline{\mathbb{S}(-x^2 - y^2 + 2^2)} \cap \overline{\mathbb{S}(-(x-4)^2 - (y-3)^2 + 3^2)} = \{(x, y) \mid (1.6, 1.2)\}$. Thus, there are no coverings to separate them. Fig. 5.3 right shows a convergent case $y > x + \frac{1}{x} \wedge y < x \wedge x > 0$, which is equivalent to $xy > x^2 + x \wedge y < x \wedge x > 0$. There are no finite coverings to separate them.

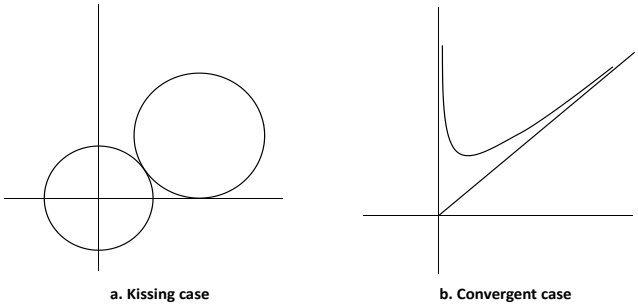


Figure 5.3: Limitations for proving UNSAT

5.1.5 raSAT loop

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. 5.2 (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 5.2.1).

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 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 5 *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 $(-4, -\frac{11}{4}) + (-\frac{1}{4}, 0)\epsilon_1 - 2\epsilon_2 + \mathbf{3}|\epsilon_1| + (-2, 2)\epsilon_\pm$, thus $(-8, 4.5)$.*

5.2 Refinements strategies

5.2.1 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 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 $(O.T)$.

Example 6 *In Example 5,*

- *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.

Chapter 6

Equality handling

6.0.2 Greater-than-or-Equal Handling

raSAT loop is designed to solve polynomial inequality. There are several ways to extend to handle equality, in which our idea shares similarity with dReal [**?**, **?**].

Definition 7 $\bigwedge_j f_j \geq 0$ *is* strict-SAT *(resp.* strict-UNSAT*) if* $\bigwedge_j f_j > \delta_j$ *is SAT (resp.* $\bigwedge_j f_j > -\delta_j$ *is UNSAT) for some* $\delta_j > 0$.

Lemma 1 *If* $\bigwedge_j f_j \geq 0$ *is* strict-SAT *(resp.* strict-UNSAT*), it is SAT (resp. UNSAT).*

Note that netiher strict-SAT nor strict-UNSAT (i.e., kissing situation), Lemma 1 cannot conclude anything, and **raSAT** says *unknown*.

6.0.3 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 2 *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.

Chapter 7

Experiments

We implemented **raSAT** loop as an SMT **raSAT**, based on MiniSat 2.2 as a backend SAT solver. In this section, we are 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.

7.0.4 Effectiveness of designed strategies

There are three immediate measures on the size of polynomial constraints. They are the highest degree of polynomials, the number of variables, and the number of APIs. Our strategies focus on selecting APIs(for testing) and selecting variable (for multiple test cases and for decomposition), selecting decomposed box of the chosen variable in TEST-UNSAT API. Thus, in order to justify the effectiveness of these strategies, we need to test them on problems with varieties of APIs number (selecting APIs) and varieties of variables number in each API (selecting variable). For this criteria, we use Zankl family for evaluating strategies. The number of APIs for this family varies from 1 to 2231 and the maximum number of variables in each APIs varies from 1 to 422.

As mentioned, raSAT uses SAT likely-hood evaluation to judge the difficulty of an API. In testing, raSAT chooses the likely most difficult API to test first. In box decomposition, again SAT likely-hood is used to evaluate decomposed boxes again the TEST-UNSAT API. For easy reference, we will name the strategies as followings

- Selecting APIs, selecting decomposed boxes:
  - **(1)** Using SAT likely-hood
  - **(2)** Randomly
- Selecting one variable each API for testing, one variable of TEST-UNSAT API for decomposition:
  - **(3)** Using sensitivity

Strategies	Group	SAT	UNSAT	time(s)
(1)-(5)	matrix-1-all-*	22	11	27.09
(2)-(5)	matrix-1-all-*	21	10	933.78
(1)-(3)	matrix-1-all-*	31	10	765.48

Table 7.1: Experiments of strategies on Zankl family

(5) Randomly

We will compare the combinations $(1)-(5)$ and $(2)-(5)$ to see how SAT likely-hood affects the results. And comparison betweeen $(1)-(5)$ and $(1)-(3)$ illustrates the effectiveness of sensitivity. In these experiments, timeout is set to $500s$.

Table 7.1 shows the results of strategies. While $(1)-(5)$ solved 33 problems in $27.09s$, $(2)-(5)$ took $933.78s$ to solve 31 problems. In fact, among 31 solved problems of $(2)-(5)$, $(1)-(5)$ solved 30. The remaining problem is a SAT one, and $(2)-(5)$ solved it in $34.36s$. So excluding this problem, $(2)-(5)$ solved 30 problems in $933.78s - 34.36s = 899.42s$ and $(1)-(5)$ took $27.09s$ to solved all these 30 problems plus 3 other problems. Using SAT likely-hood to select the most difficult API first for testing is very reasonable. This is because the goal of testing is to find an assignment for variables that satisfies **all** APIs. Selecting the most difficult APIs first has the following benefits:

- The most difficult APIs are likely not satisfied by most of test cases (of variables in these APIs). This early avoids exploring additional unnecessary combination with test cases of variables in other easier APIs.

- If the most difficult APIs are satisfied by some test cases, these test cases might also satisfy the easier APIs.

raSAT uses SAT likely-hood to choose a decomposed box so that the chosen box will make the TEST-UNSAT API more likely to be SAT. This seems to work for SAT problems but not for UNSAT ones. In fact, this is not the case. Since for UNSAT problems, we need to check both of the decomposed boxes to prove the unsatisfiability (in any order). So for UNSAT constraints, it is not the problem of how to choose a decomposed box, but the problem of how to decompose a box. That is how to choose a point within an interval for decomposition so that the UNSAT boxes are early separated. Currently, raSAT uses the middle point. For example, $[0, 10]$ will be decomposed into $[0, 5]$ and $[5, 10]$. The question of how to decompose a box is left for our future work.

Combination $(1)-(3)$ solved 41 problems in comparison with 33 problems of $(1)-(5)$. There are 5 large problems (more than 100 variables and more than 240 APIs in each problem) which were solved by $(1)-(3)$ (solving time was from $20s$ to $300s$) but not solved (timed out - more than $500s$) by $(1)-(5)$. Choosing the most important variable (using sensitivity) for multiple test cases and for decomposition worked here.

Solver	Zankl (151)			Meti-Tarski (5101)		
	SAT	UNSAT	time(s)	SAT	UNSAT	time(s)
isat3	14	15	209.20	2916	1225	885.36
raSAT	31	10	765.48	3322	1052	753.00
Z3 4.3	**54**	**23**	**1034.56**	**3528**	**1569**	**50235.39**

Table 7.2: Experimental results for Hong, Zankl, and Meti-Tarski families

7.0.5 Comparison with other tools on QF_NRA and QF_NIA benchmarks

Comparison with Z3 and isat3 on QF_NRA.

Table 7.2 presents the results of isat3, raSAT and Z3 on 2 families (Zankl and Meti-tarski) of QF_NRA. Only problems with inequalities are extracted for testing. The timeout for Zankl family is $500s$ and for Meti-tarski is $60s$ (the problems in Meti-tarski are quite small - less than 8 variables and less than 25 APIs for each problem). For isat3, ranges of all variables are uniformly set to $[-1000, 1000]$

Both isat3 and raSAT use interval arithmetics for deciding constraints. However, raSAT additionally use testing for accelerating SAT detection. As the result, raSAT solved larger number of SAT problems in comparison with isat3. In Table 7.2, isat3 conludes UNSAT when variables are in the ranges $[-1000, 1000]$, while raSAT conclude UNSAT over $[-infinity, infinity]$.

In comparison with Z3, for Zankl family, Z3 4.3 showed better performance than raSAT in the problems with lots of small constraints: short monomial (less than 5), small number of variables (less than 10). For problems where most of constraints are long monomial (more than 5) and have large number of variables (more than 10), raSAT results are quite comparable with ones of Z3, often even outperfoming when the problem contains large number of vary long constraints (more than 40 monomials and/or more than 20 variables). For instance, *matrix-5-all-27, matrix-5-all-01, matrix-4-all-3, matrix-4-all-33, matrix-3-all-5, matrix-3-all-23, matrix-3-all-2, matrix-2-all-8* are such problems which can be solved by raSAT but not by Z3.

Meti-tarski family contains quite small problems (less than 8 variables and less than 25 APIs for each problem). Z3 solved all the SAT problems, while raSAT solved around 94% (33322/3528) of them. Approximately 70% of UNSAT problems are solved by raSAT. As mentioned, raSAT has to explore all the decomposed boxes for proving unsatisfiability. Thus the question "How to decompose boxes so that the UNSAT boxes are early detected?" needs to be solved in order to improve the ability of detecting UNSAT for raSAT.

Solver	SAT	UNSAT	time (s)
raSAT	6764	0	1230.54
Z3	**6784**	**36**	**139.78**

Table 7.3: Experiments on QF_NIA/AProVE

Comparison with Z3 on QF_NIA.

We also did experiments on QF_NIA/AProVE benchmarks to evaluate raSAT loop for Integer constraints. There are 6850 problems which do not contain equalities. The timeout for this experiment is 60s. Table 7.3 shows the result of Z3 and raSAT for this family.

raSAT solved almost the same number of SAT problems as Z3 (6764 for raSAT with 6784 for Z3). Nearly 99% problems having been solved by raSAT is a quite encouraging number. Currently, raSAT cannot conclude any UNSAT problems. Again, this is due to exhaustive balanced decompositions.

We implement **raSAT** loop as an SMT solver **raSAT**, based on MiniSat 2.2 as a backend SAT solver. Various combinations of strategies of **raSAT** (in Section **??**) and random stategies are compared on *Zankl*, *Meti-Tarski* in NRA category and *AProVE* in NIA category from SMT-LIB. The best combination of choices are

1. a test-UNSAT API by the least SAT-likelyhood,
2. a variable by the largest sensitivity, and
3. a box by the largest SAT-likelyhood,

and sometimes a random choice of a test-UNSAT API (instead of the least SAT-likelyhood) shows an equaly good result. They are also compared with **Z3 4.3** and **iSAT3**, where the former is considered to be the state of the art ([3]), and the latter is a popular ICP based tool. Note that our comparison is only on polynomial inequality. The experiments are on a system with Intel Xeon E5-2680v2 2.80GHz and 4 GB of RAM.

7.0.6 Benchmarks from SMT-LIB

In SMT-LIB [1], benchmark programs on non-linear real number arithmetic (QF_NRA) are categorized into Meti-Tarski, Keymaera, Kissing, Hong, and Zankl families. Until SMT-COMP 2011, benchmarks are only Zankl family. In SMT-COMP 2012, other families have been added, and currently growing. General comparison among various existing tools on these benchmarks is summarized in Table.1 in [3], which shows Z3 4.3 is one of the strongest.

From them, we extract problems of polynomial inequality only The brief statistics and explanation are as follows.

[1] `http://www.smt-lib.org`

- **Zankl** has 151 inequalities among 166, taken from termination provers. A Problem may contain several hundred variables, an API may contain more than one hundred variable, and the number of APIs may be over thousands, though the maximum degree is up to 6.
- **Meti-Tarski** contains 5101 inequalities among 7713, taken from elementary physics. They are mostly small problems, up to 8 varaibles (mostly up to 5 variables), and up to 20 APIs.
- **Keymaera** contains 161 inequalities among 4442.
- **Kissing** has 45 problems, all of which contains equality (mostly single equality).
- **Hong** has 20 inequalities among 20, tuned for QE-CAD and quite artificial.

{**Mizuhito:** Experiments in Section **??** are performed with 10 variables (1 varable for each 10 APIs selected by SAT-likelyhood) for multiple test generation (random 2 ticks), and variable for decomposition.

Experiments in Section **??** are performed with $\gamma_0 = 0.1$ and $\gamma_{i+1} = \gamma_i/10$.
$\delta_0 = 10$, $\delta_1 = \infty$. }

7.0.7 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 7.4 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,

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

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

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

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

and so on.

Experiments in Table 7.4 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.

7.0.8 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

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

Table 7.5: Comparison among SMT solvers

- *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
- *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.

7.0.9 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 8

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.

8.0.10 Observation and Discussion

From experimental results in Section **??** and 7.0.6, 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 5) 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.

8.0.11 Future Work

raSAT still remains in naive proto-type status, and there are lots of future work.

UNSAT core. {**Mizuhito:** to be filled}

Exact confirmation. Currently, **raSAT** uses floating point arithmetic. Thus, results can be unsound. We are planning to add a confirmation phase to confirm whether an SAT instance is exact by roundoff error bound guaranteed floating arithmetic libraries, such as ****.

Equality handling. We are planning to extend the use of Intermediate Value Theorem to multiple equality with shared variables.

Further strategy refiment. {**Mizuhito:** Test data generation, blanced box decomposition}

Bibliography

[1]

[2]

[3]

[4]

[5]

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Non-linear loop invariant generation using gröbner bases. *SIGPLAN Not.*, 39(1):318–329, January 2004.