raSAT: SMT Solver for Polynomial Inequality

Vu Xuan Tung[1], To Van Khanh[2], and Mizuhito Ogawa[1]

[1] Japan Advanced Institute of Science and Technology
{tungvx,mizuhito}@jaist.ac.jp
[2] University of Engineering and Technology, Vietnam National University, Hanoi
khanhtv@vnu.edu.vn

Abstract. 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 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 with respect to the number of variables. 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.

1 Introduction

Polynomial Constraint solving over real (integer) numbers is to find an assignment from real (integer) numbers to variables that satisfies given polynomial inequality/equality. Many applications are reduced to solving polynomial constraints, such as

- **Locating roundoff and overflow errors**, which is our motivation [1, 2].
- **Automatic termination proving**, which reduces termination detection to finding a suitable ordering [3], e.g., T_TT₂ [3], AProVE [4].
- **Loop invariant generation**. Farkas's lemma is a popular approach in linear loop invariant generation [4], and is reduced to degree 2 polynomials. Nonlinear loop invariant [5] requires more complex polynomials.
- **Hybrid system**. SMT for QF_NRA are often used as backend engines [6].

[3] http://cl-informatik.uibk.ac.at/software/ttt2/
[4] http://aprove.informatik.rwth-aachen.de

– **Mechanical contrnol design**. PID control is simple but widely used, and designing parameters is reduced to polynomial constraints [7].

Solving polynomial constraints on real numbers is decidable [8], though that on integers is undecidable (*Hilbert's 10th problem*). Quantifier elimination by cylindrical algebraic decomposition (QE-CAD) [9] is a well known technique, and implemented in Mathematica, Maple/SynRac, Reduce/Redlog, QEPCAD-B, and recently in some SMT solvers [10]. It can solve general formulae at the cost of DEXPTIME, which hardly work up to 8 varaibles and degree 10. Satisfiability targets on an existential problem, and *Variant quantifier elimination* [**?**] reduces polynomial constraint solving to polynomial optimization problems, which are solved by Groebner basis in EXPTIME.

A practical alternative is *ICP* (Interval Constraint Propagation), which are used in SMT solver community, e.g., **iSAT3** [11], **dReal** [12], and RSOLVER [13]. 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. 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.
– 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 bounded intervals.

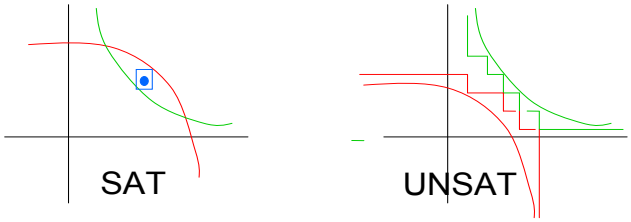


Fig. 1. SAT and UNSAT detection by ICP

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 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**[5], 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 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[6]. They are also evaluated by comparing **Z3 4.3**[7] 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*.

Related Work

Non-linear constraints are still under development, and SMT solvers adapt several approachs other than ICP.

[5] `http://irram.uni-trier.de`
[6] `http://www.smtlib.org/`
[7] `http://z3.codeplex.com`

QE-CAD. RAHD [**?**] and Z3 4.3 (which is referred as nlsat in [10]) include QE-CAD. QE-CAD is precise and detects beyond SAT instances (e.g., SAT regions), scalability is still challenging, since it is DEXPTIME.

Virtual substitution (VS). SMT-RAT toolbox [14][15] 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.

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

Linearization. Linearization of polynomials is often used over integers, such as Barcelogic [**?**], which substitutes all possible integers in a given-bound to an argument of a multiplication. Then, multiplications are reduced to an exhaustive search on linear constraints. 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.

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

2 Over and Under Approximation Theories and Their Refinement

2.1 Approximation Theory

Let $F = \exists x_1 \in I_1 \cdots x_n \in I_n . \bigwedge_j \psi_j(x_1, \cdots, x_n)$, 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 3, 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 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 3. *An* open box *of dimension n is a set $(a_1, b_1) \times \cdots \times (a_n, b_n)$ where $a_i, b_i \in \mathbb{R}, a_i \leq b_i$. 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.

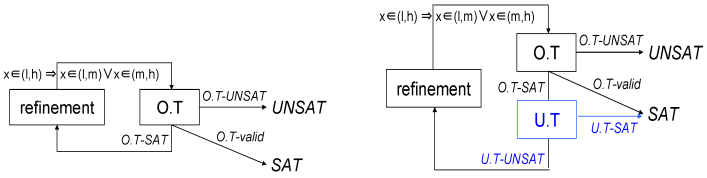
Example 1. $\exists x \in (-1,3)\ y \in (2,4).(x^3y - y^4 > 0) \wedge (y^3 - xy > 0)$ is an example of a polynomial inequality with 2 variables and 2 APIs.

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 4. *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. 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 3.



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

Fig. 2. Rfinement loops

Definition 5. *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. 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 4, 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. 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. 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.

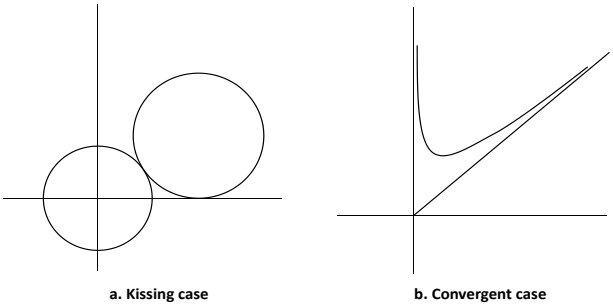


Fig. 3. Limitations for proving UNSAT

2.3 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. 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 4.1).
- Heurstic measures *SAT-likelyhood* and *sensitivity*, for selection of a variable to decompose and a box to explore. (Section 4.2).

raSAT also prepares various interval arithmetic as $O.T$ as in Section 3, whereas currently only random tesing (*k-random ticks*, which consists of periodical k-test instances with a random offset) is prepared as $U.T$.

3 Interval arithmetic

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 2. 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 $(-8, 4.5)$.

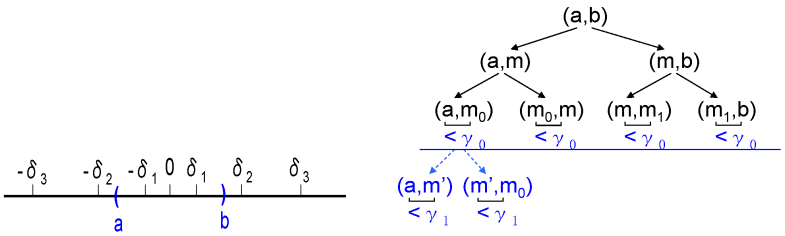
4 Strategies in raSAT

4.1 Incremental search

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

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. 4 (a)).

Note that if $\delta_i < \infty$, **raSAT** applies an Affine interval; otherwise, it uses CI. Experiments in Section 6 are performed with $\delta_0 = 10$ and $\delta_1 = \infty$.



(a) Incremenal widening (b) Incremental Deepening

Fig. 4. Chebyshev approximation of x^2 and $x\ |x|$

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 threshholds, i.e., starting with δ_0, and if it fails, restart with δ_1, and so on (Fig 4 (b)).

4.2 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 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 3. In Example 2,

- 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.

Test case generation using variables sensitivity. The value of variables sensitivity can also be used to approximate how likely the value of a polynomial increases when the value of that variable increases. Consider the constraint $f = -x_{15} * x_8 + x_{15} * x_2 - x_{10} * x_{16} > 0$. With $x_2 \in [9.9, 10], x_8 \in [0, 0.1], x_{10} \in [0, 0.1], x_{15} \in [0, 10]$, and $x_{16} \in [0, 10]$. The result of AF2 for f is: $0.25\epsilon_2 - 0.25\epsilon_8 - 0.25\epsilon_{10} + 49.5\epsilon_{15} - 0.25\epsilon_{16} + 0.75\epsilon_{+-} + 49.25$.

The coefficient of ϵ_2 is 0.25 which is positive, then we expect that if x_2 increases, the value of f also increase. As the result, the test case of x_2 is expected as high as possible in order to satisfy $f > 0$. We will thus take the upper bound value of x_2, i.e. 10. Similarly, we take the test cases for other variables: $x_8 = 0, x_{10} = 0, x_{15} = 10, x_{16} = 0$. With these test cases, we will have $f = 100 > 0$.

5 SAT on Equality by Intermediate Value Theorem

Single Equation

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.

Lemma 1. *For* $\varphi = \bigwedge_j^m f_j > 0 \ \wedge \ g = 0$, *F is SAT, if there is a box represented by* $\Pi = \bigwedge_{v_i \in V} v_i \in (l_i, h_i)$ *such that*

(i) $\bigwedge_j^m f_j > 0$ *is* $\Pi^p_{\mathbb{R}}$*-VALID, and*
(ii) there are two instances $\boldsymbol{t}, \boldsymbol{t'}$ *in the box with* $g(\boldsymbol{t}) > 0$ *and* $g(\boldsymbol{t'}) < 0$.

Proof. It is clear from the Intermediate Value Theorem that there exist an point $\boldsymbol{t_0}$ between $\boldsymbol{t}$ and $\boldsymbol{t'}$ such that $g(\boldsymbol{t_0}) = 0$. In addition, because $\bigwedge_j^m f_j > 0$ is $\Pi^p_{\mathbb{R}}$-VALID, $\boldsymbol{t_0}$ also satisfies $\bigwedge_j^m f_j > 0$. As a result, φ is satisfiable with $\boldsymbol{t_0}$ as the SAT instance.

Example 4. Consider the constraint $\varphi = f(x, y) > 0 \wedge g(x, y) = 0$. Suppose we can find a box represented by $\Pi = x \in \langle a, b\rangle \wedge y \in \langle c, d\rangle$ such that $f(x, y) > 0$ is $\Pi^p_{\mathbb{R}}$-VALID (Figure 5). In addition, inside that box, we can find two points (u_1, v_1) and (u_2, v_2) such that $g(u_1, v_1) > 0$ and $g(u_2, v_2) < 0$. By Lemma 1, the constraint is satisfiable.

raSAT first tries to find a box of variables' intervals (by refinements) such that $\bigwedge_j^m f_j > 0$ is VALID inside that box. Then 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 does not find an exact SAT instance.

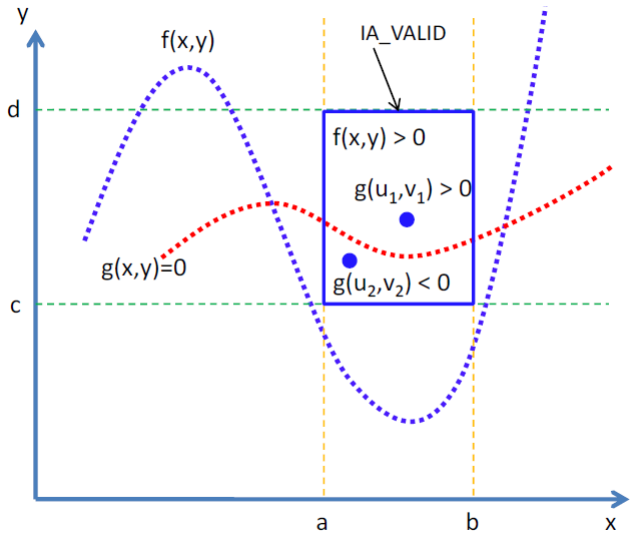


Fig. 5. Example on solving single equation using the Intermediate Value Theorem

Multiple Equations

The idea of using the Intermediate Value Theorem can also be used for solving multiple equations. Consider n equations ($n \geq 1$): $\bigwedge_{i=1}^{n} g_i = 0$ and an interval constraint $\bigwedge_{v_i \in V} v_i \in \langle l_i, h_i \rangle$. If we can find a set $\{V_1, \cdots, V_n\}$ that satisfies the following properties, then we can conclude that $\bigwedge_{i=1}^{n} g_i = 0$ is satisfiable in $\bigwedge_{v_i \in V} v_i \in \langle l_i, h_i \rangle$.

- For all $i = 1, \cdots, n$; we have $V_i \subset var(g_i)$.
- For all $i \neq j$, we have $V_i \neq V_j$.
- For all $i = 1, \cdots, n$; let $k_i = |V_i|$ and $V_i = \{v_{ij} \mid 1 \leq j \leq k_i\}$. Then, there exist two values $(v_{i1}, \cdots, v_{ik_I}) = (x_{i1}, \cdots, x_{ik_i})$ and $(v_{i1}, \cdots, v_{ik_I}) = (x'_{i1}, \cdots, x'_{ik_i})$ such that

$$g_i(x_{i1}, \cdots, x_{ik_i}, \cdots, v_{ik}, \cdots) > 0$$

and

$$g_i(x'_{i1}, \cdots, x'_{ik_i}, \cdots, v_{ik}, \cdots) < 0$$

for all values of v_{ik} in $\langle l_{ik}, h_{ik} \rangle$ where $v_{ik} \in var(g_i) \setminus V_i$. We denote $ivt(g_i, V_i, \Pi)$ to represent that the polynomial g_i enjoy this property with respect to V_i and Π.

By the first two properties, this method restricts that the number of variables must be greater than or equal to the number of equations.

Example 5. Consider two equations $g_1(x, y) = 0$ and $g_2(x, y) = 0$ (Figure 6) which satisfy the above restriction on the number of variables, and the variable intervals is $\Pi = x \in \langle c_1, d_1 \rangle \wedge y \in \langle d_2, c_2 \rangle$. Let $V_1 = \{x\}$ and $V_2 = \{y\}$, we have:

$$g_1(c_1, y) < 0 \text{ and } g_1(d_1, y) < 0 \text{ for all } y \in \langle d_2, c_2 \rangle; \text{ and}$$

$$g_2(x, d_2) > 0 \text{ and } g_2(x, c_2) < 0 \text{ for all } x \in \langle c_1, d_1 \rangle$$

Thus we can conclude that $g_1(x, y) = 0 \wedge g_2(x, y) = 0$ has a solution inside the box represented by Π.

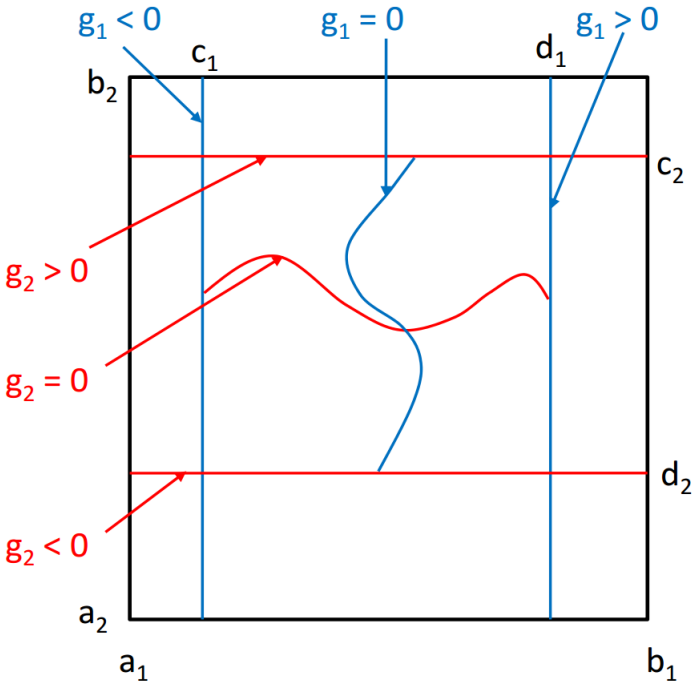


Fig. 6. Example on solving single equation using the Intermediate Value Theorem

Our current implementation of handling multiple equations is very naive which is described in Algorithm 1 because for each polynomial, **raSAT** checks every possible subsets of its variables. As the result, given the constraint $\bigwedge_{i=1}^{n} g_i = 0$, in the worst case **raSAT** will check $2^{|var(g_1)|} * \cdots * 2^{|var(g_n)|}$ cases. As a future work, we may use variables' sensitivity to give priority on subsets of variables.

Algorithm 1 Solving multiple equations $\bigwedge_{i=1}^{n} g_i = 0$ with interval constraint $\Pi = \bigwedge_{v_i \in V} v_i \in \langle l_i, h_i \rangle$

```
1: function EQUATIONSPROVER(⋀_{i=j}^{n} g_i = 0, Π, V_0)
2:     if j > n then                                   ▷ All equations are checked
3:         return SAT
4:     end if
5:     for V_j ∈ P(var(g_j)) do                ▷ P(var(g_j)) is the powerset of var(g_j)
6:         if V_j ∩ V = ∅ and ivt(V', g_j, Π) then
7:             V_0 ← V_0 ∪ V'
8:             if EQUATIONSPROVER(⋀_{i=j+1}^{n} g_i = 0, Π, V_0) = SAT then
9:                 return SAT
10:            end if
11:        end if
12:    end for
13:    return UNSAT
14: end function
15: EQUATIONSPROVER(⋀_{i=1}^{n} g_i = 0, Π, ∅)
```

6 Experiments

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 4) 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 ([10]), 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.

6.1 Benchmarks from SMT-LIB

In SMT-LIB [8], benchmark programs on non-linear real number arithmetic (QF_NRA) are categorized into Meti-Tarski, Keymaera, Kissing, Hong, and

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

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 [10], 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.

- **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.

The setting of the experiments are

- For test data generation, raSAT chooses 10 variables (1 variable from each of 10 APIs with largest SAT-likelyhood) and apply random 2-ticks, and single random test data is generated for each of the rest of variables.
- For interval decomposition, raSAT applies the balanced decomposition.
- For incremental widening, $\delta_0 = 10, \delta_1 = \infty$
- For incremental deepening, $\gamma_i = 10^{-(i+1)}$ for $i \geq 0$.

6.2 Experiments on ICP solvers

We perform experiments on ICP-based SMT solvers against inequalities of Zankl, Meti-Tarski and Keymaera families. The solvers consis of:

- previous version of raSAT [17],
- current version of raSAT with above mentioned strategies,
- iSAT3 [11], and
- dREAL [12].

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 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).

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)	18	101.07	15	1064.76	14	562.19	**21**	462.57	18	788.46
Matrix-1 (UNSAT)	2	0.01	2	0.01	2	0.01	2	0.01	2	0.01	2	0.01
Matrix-2,3,4,5 (SAT)	**10**	632.37	3	140.27	1	3.46	0	0.00	5	943.08	0	0.00
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)	20	163.47 (s)	21	736.17	19	953.97	18	1068.40	19	799.79	19	230.39
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	514.37	1	350.84	0	0.00	0	0.00	0	0.00	1	13.43
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)	18	1438.47 (s)	20	1537.9	19	1100.60	17	916.32	17	87.78	20	710.21
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	1	33.17	1	201.32	2	328.03	0	0.00	1	20.94
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 1. Combnations of **raSAT** strategies on NRA/Zankl,Meti-Tarski benchmark

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 4, there are lots of heuristic choices. For instance,

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

and so on.

Experiments in Table 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.

6.3 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 $[-\infty, \infty]$.

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 2. Comparison among SMT solvers

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
- *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.

6.4 Experiments on SAT solving of equations

Zankl(15)				Meti-tarski(2612)			
SAT(11)		UNSAT(4)		SAT(1497)		UNSAT(1115)	
11	0.07(s)	4	0.17(s)	1343	150.11(s)	804	188.61(s)

6.5 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 4.1 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.

7 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. **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}

References

[1] Ngoc, D.T.B., Ogawa, M.: Overflow and roundoff error analysis via model checking. In: Proceedings of the 2009 Seventh IEEE International Conference on Software Engineering and Formal Methods. SEFM '09, Washington, DC, USA, IEEE Computer Society (2009) 105–114

[2] Ngoc, D.T.B., Ogawa, M.: Checking roundoff errors using counterexample-guided narrowing. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. ASE '10, New York, NY, USA, ACM (2010) 301–304

[3] Lucas, S., Navarro-Marset, R.: Comparing csp and sat solvers for polynomial constraints in termination provers. Electron. Notes Theor. Comput. Sci. **206** (2008) 75–90

[4] Coln, M., Sankaranarayanan, S., Sipma, H.: Linear invariant generation using non-linear constraint solving. In Hunt, WarrenA., J., Somenzi, F., eds.: Computer Aided Verification. Volume 2725 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2003) 420–432

[5] Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Non-linear loop invariant generation using gröbner bases. SIGPLAN Not. **39** (2004) 318–329

[6] Sankaranarayanan, S., Sipma, H., Manna, Z.: Constructing invariants for hybrid systems. In: in Hybrid Systems: Computation and Control, LNCS 2993, Springer-Verlag (2004) 539–554

[7] Anai, H.: Algebraic methods for solving real polynomial constraints and their applications in biology. In: In Algebraic Biology Computer Algebra in Biology. (2005) 139–147

[8] Tarski, A.: A decision method for elementary algebra and geometry. In Caviness, B., Johnson, J., eds.: Quantifier Elimination and Cylindrical Algebraic Decomposition. Texts and Monographs in Symbolic Computation. Springer Vienna (1998) 24–84

[9] Collins, G.: Quantifier elimination by cylindrical algebraic decomposition twenty years of progress. In Caviness, B., Johnson, J., eds.: Quantifier Elimination and Cylindrical Algebraic Decomposition. Texts and Monographs in Symbolic Computation. Springer Vienna (1998) 8–23

[10] Jovanovi, D., de Moura, L.: Solving non-linear arithmetic. In Gramlich, B., Miller, D., Sattler, U., eds.: Automated Reasoning. Volume 7364 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 339–354

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

[12] Gao, S., Kong, S., Clarke, E.: dreal: An smt solver for nonlinear theories over the reals. In Bonacina, M., ed.: Automated Deduction CADE-24. Volume 7898 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 208–214

[13] Ratschan, S.: Efficient solving of quantified inequality constraints over the real numbers. ACM Trans. Comput. Logic **7** (2006) 723–748

[14] Corzilius, F., Loup, U., Junges, S., brahm, E.: Smt-rat: An smt-compliant nonlinear real arithmetic toolbox. In Cimatti, A., Sebastiani, R., eds.: Theory and Applications of Satisfiability Testing SAT 2012. Volume 7317 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 442–448

[15] Corzilius, F., brahm, E.: Virtual substitution for smt-solving. In Owe, O., Steffen, M., Telle, J., eds.: Fundamentals of Computation Theory. Volume 6914 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2011) 360–371

[16] Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Abstract dpll and abstract dpll modulo theories. In: In LPAR04, LNAI 3452, Springer (2005) 36–50

[17] Khanh, T.V., Ogawa, M.: {SMT} for polynomial constraints on real numbers. Electronic Notes in Theoretical Computer Science **289** (2012) 27 – 40 Third Workshop on Tools for Automatic Program Analysis (TAPAS' 2012).