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 (Satisfiability Modulo Theory) 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 (IA) and Testing, 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.
Several heuristic measures, called *SAT likelihood*, *sensitivity*, and the number of unsolved atomic polynomial constraints, are compared on Zankl, Meti-tarski and Keymaera benchmarks from QF_NRA category of SMT-LIB. They are also evaluated by comparing **Z3 4.3**, **dReal-2.15.01** and **iSAT3**. raSAT loop is extended with the use of the Intermediate Value Theorem to solve equality. This extension is evaluated on equalities of Zankl, Meti-tarski and Keymaera families. We also show a simple modification to handle mixed integers, 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 [17].
- **Automatic termination proving**, which reduces termination detection to finding a suitable ordering [14], e.g., $\mathsf{T_TT_2}$[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. Non-linear loop invariant [22] requires more complex polynomials.

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

– **Hybrid system**. SMT solvers for polynomial constraints over real numbers (QF_NRA) are often used as backend engines [21].

Solving polynomial constraints on real numbers is decidable [23], though that on integers is undecidable (*Hilbert's 10th problem*). Quantifier elimination by cylindrical algebraic decomposition (QE-CAD) [3] is a well known technique, and implemented in Mathematica, Maple/SynRac, Reduce/Redlog, QEPCAD-B, and recently in some SMT solvers [12]. It can solve general formula at the cost of DEXPTIME, which hardly work up to 8 variables and degree 10. Satisfiability targets on an existential problem, and *Variant quantifier elimination* [11] reduces polynomial constraint solving to polynomial optimization problems, which are solved by Groebner basis in EXPTIME.

A practical alternative is Interval Constraint Propagation (*ICP*), which are used in SMT solver community, e.g., **iSAT3** [7], **dReal** [9], and **RSolver** [20]. ICP is based on over-approximation by Interval Arithmetic, and iteratively refines by interval decompositions. It is practically often more efficient than algebraic computation with weaker theoretical completeness.

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 ICP solvers, **raSAT**

– introduces testing (as an under-approximation) to accelerate SAT detection,
– applies various Interval Arithmetic, e.g., Affine intervals [15, 17, 13], which enables to analyze the effects of input values, and
– implements SAT confirmation step by an error-bound guaranteed floating point package **iRRAM**[5], to avoid soundess bugs caused by roundoff errors.

As **iSAT3**, **raSAT** applies outward rounding [10] in Interval Arithmetic to avoid soundless bugs due to round-off error of floating arithmetic operations. As a consequence, answers of raSAT (SAT or UNSAT) (SAT instances found in testing is verified by **iRRAM**) are guaranteed to be sound.

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 likelihood*, 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*.

raSAT also applies incremental search, which is often faster in practice.

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

– **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.

Related Work

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

QE-CAD. RAHD [19] and Z3 4.3 (which is referred as nlsat in [12]) 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 [6] combines VS, incremental DPLL, and eager theory propagation. Z3 (version 3.1) combines VS, ICP, and linearization.

Bit-blasting. UCLID [2] and MiniSmt [24] reduces the number of bits (i.e., narrowing bounds for SAT instances) , for integer and rational numbers respectively.

Linearization. CORD [8] linearize multiplications using a technique called CORDIC for real numbers. Linearization suffers when the degree of polynomials increases.

Because **raSAT** in the same category of using ICP with iSAT3 and dReal, next part is going to take a look at details of methodologies used in these solvers.

iSAT3

Although **iSAT3** also uses Interval Arithmetic (IA), its algorithm integrates IA with DPLL procedure [18] tighter than that of **raSAT**. During DPLL procedure, in addition to an assignment of literals, **iSAT3** also prepares a data structure to store interval boxes where each box corresponds to one decision level of DPLL procedure's assignment. In **UnitPropagation** rule, intead of using standard rule, **iSAT3** searches for clauses that have all but one atoms being inconsistence with the current interval box. When some atom are selected for the literals assignment, this tool tries to use the selected atoms to contract the corresponding box to make it smaller. In order to do this, **iSAT3** convert each inequality/equation in the given constraints into the conjunction of the atoms of the following form by introducing additional variables:

$$\begin{array}{ll}
\text{atom} & ::= \text{bound} \mid \text{equation} \\
\text{bound} & ::= \text{variable relation rational\_constant} \\
\text{relation} & ::= <|\leq|=|\geq|> \\
\text{equation} & ::= \text{variable} = \text{variable bop variable} \\
\text{bop} & ::= + \mid - \mid \times
\end{array}$$

In other words, the resulting atoms are of the form, e.g., either $x > 10$ or $x = y + z$. For example, the constraint

$$x^2 + y^2 < 1$$

is converted into:

$$\begin{cases} x_1 = x^2 \\ x_2 = y^2 \\ x_3 = x_1 + x_2 \\ x_3 < 1 \end{cases}$$

From the atoms of these form, the contraction can be easily done for interval boxes:

- For the bound atom of the form, e.g., $x > 10$, if the bound is $x \in [0, 100]$, then the contracted box contain $x \in [10, 100]$.
- For the equations of three variables $x = y$ bop z, from bounds of any two variables, we can infer the bound for the remaining one. For example, from

  $$\begin{cases} x = y.z \\ x \in [1, 10] \\ y \in [3, 7] \end{cases}$$

  we can infer that

  $$z \in [\frac{1}{7}, \frac{10}{3}]$$

When the **UnitPropagation** and contraction can not be done, **iSAT3** split one interval (decomposition) in the current box and select one decomposed interval to explore which corresponds to **decide** step. If the contraction yields an empty box, a conflict is detected and the complement of the bound selection in the last split needs to be asserted. This is done via **learn** the causes of the conflict and **backjump** to the previous bound selection of the last bound selection. In order to reason about causes of a conflict, **iSAT3** maintains an implication graph to represents which atoms lead to the asserting of one atom.

dReal

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

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

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

- **Removing** $\neq$: Each formula of the form $f \neq 0$ is transformed into $f > 0 \vee f < 0$.

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

Note that the satisfiability of φ^δ does not imply that of φ.

2 raSAT loop

2.1 Target Problem

Definition 1. *A (bounded) polynomial inequality constraint is*

$$\varphi : \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} \psi_j(x_1, \cdots, x_n)$$

where $\psi_j(x_1, \cdots, x_n)$ is an atomic polynomial inequality (API) of the form $p_j(x_1, ..., x_n) > 0$ with $p_j(x_1, ..., x_n)$ is a composition of functions: $+$ (addition), $-$ (subtraction) and $\times$ (multiplication).

Note that φ is equivalent to $\exists x_1 \ldots x_n.(\bigwedge\limits_{i=1}^{n} x_i \in I_i) \wedge (\bigwedge\limits_{j=1}^{m} \psi_j(x_1, \cdots, x_n))$. We call $\bigwedge\limits_{i} x_i \in I_i$ an *interval constraint*, and we refer $\bigwedge\limits_{j=1}^{m} \psi_j(x_1, \cdots, x_n)$ by $\psi(x_1, \cdots, x_n)$. We denote the set of solutions of the constraint $\psi(x_1, \cdots, x_n)$ as $\mathbb{S}(\psi(x_1, \cdots, x_n)) = \{(r_1, \cdots, r_n) \in \mathbb{R}^n \mid \psi(r_1. \cdots, r_n) \text{ holds}\}$. Because our methodology is an extension of ICP, next section is going to review ICP before the details of **raSAT** loop.

2.2 ICP overview

Algorithm 1 describe the standard ICP methodology for solving polynomial inequalities. Two functions *prune*(B, ψ) and *branch*(B) satisfies the following properties:

- If $B' = prune(B, \psi)$, then $B' \subseteq B$ and $B' \cap \mathbb{S}(\psi) = B \cap \mathbb{S}(\psi)$
- If $\{B_1, B_2\} = branch(B)$, then $B = B_1 \cup B_2$ and $B_1 \cap B_2 = \emptyset$

Because ICP concludes SAT (line 8) only when there exists a box that satisfy the constraints by IA, ICP cannot show the satisfiability of equalities.

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

Algorithm 1 ICP for solving the constraint $\psi = \bigwedge_{j=k}^{m} g_j(x_1, \cdots, x_n) > 0$ given the initial box $B_0 = I_1 \times \cdots \times I_n$

```
1: $S \leftarrow \{B_0\}$                                   ▷ Set of boxes
2: while $S \neq \emptyset$ do
3:     $B \leftarrow S.choose()$                            ▷ Get one box from the set
4:     $B' \leftarrow prune(B, \psi)$
5:     if $B' = \emptyset$ then                             ▷ The box does not satisfy the constraint
6:         $S \leftarrow S \setminus \{B\}$
7:         continue
8:     else if $B'$ satisfies $\psi$ by using $IA$ then
9:         return SAT
10:    else                ▷ $IA$ cannot conclude the constraint $\implies$ Refinement Step
11:        $\{B_1, B_2\} \leftarrow branch(B')$             ▷ split $B'$ into two smaller boxes $B_1$ and $B_2$
12:        $S \leftarrow (S \setminus \{B\}) \cup \{B_1, B_2\}$
13:    end if
14: end while
15: return UNSAT
```

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

under the assumptions of fair decomposition and bounded intervals (a_i, b_i) for all $i \in \{1, \cdots, n\}$. We will prepare terminology and briefly review this fact.

Definition 2. *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, $\mathbb{S}(\bigwedge_{j=1}^{m} g_j > 0)$ 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.

Note that an Interval Arithmetic used in ICP is a converging theory.

Definition 3. *Let $\varphi = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} g_j > 0$ be a polynomial inequality such that each I_i is bounded. An theory 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 T can concludes that for all $x_i \in (c_i - \gamma, c_i + \gamma)$, $i = 1, \cdots, n$, we have $\bigwedge_{j=1}^{m} (g_j(c) - \delta < g_j(x) < g_j(c) + \delta)$.*

Definition 4. *Let $\varphi = \exists x_1 \in I_1 \cdots x_n \in I_n.\ \bigwedge_{j=1}^{m} g_j > 0$ for $I_i = (a_i, b_i)$. A refinement strategy is* fair*, if, for each $c_i \in (a_i, b_i)$ and $\gamma > 0$, a decomposition of I_i for each i eventually occurs in $(c_i - \gamma, c_i + \gamma)$ (as long as neither SAT nor UNSAT is detected).*

Theorem 1. *Let $\varphi = \exists x_1 \in I_1 \cdots x_n \in I_n.\ \bigwedge_{j=1}^{m} g_j > 0$ for $I_i = (a_i, b_i)$. Assume that each (a_i, b_i) is bounded, and a refinement strategy is fair. Then,*

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

Proof. The former is proved by the fact that, if φ is SAT, there exists a non-empty neighborhood (open box) in $\cap\ \mathbb{S}(g_j > 0)$. If the box decomposition strategy is fair, the refinement will eventually find such an open box.

For the latter, assume that $\overline{\varphi} = \exists x_1 \in [a_1, b_1] \cdots x_n \in [a_n, b_n].\ \wedge_j\, g_j \geq 0$ is UNSAT. Thus, $\cap\ \overline{\mathbb{S}(g_j > 0)} = \emptyset$. Let $\delta_{j,k} = min\{|g_j(\bar{x}) - g_k(\bar{x})| \mid \bar{x} \in I_1 \times \cdots \times I_n\}$. Since g_j'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 IA is converging, there exists $\gamma > 0$ for $\delta > 0$ satisfying Definition 3, and fair decomposition eventually finds open boxes such that $\mathbb{S}(g_j \geq 0)$ and $\mathbb{S}(g_k \geq 0)$ are separated. $\square$

Limitations for detecting UNSAT occur on *kissing* and *convergent* cases. Fig. 1 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. 1 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.

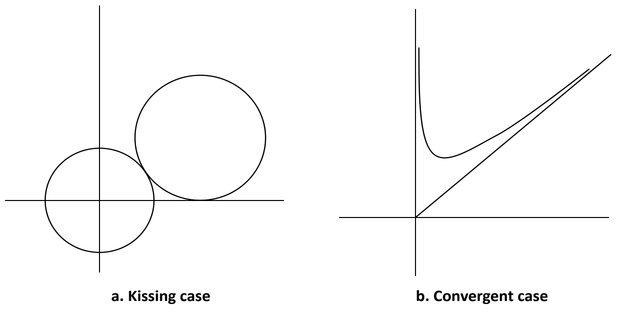


Fig. 1. Limitations for proving UNSAT

2.3 raSAT loop

Although an ICP is enough to implement an SMT solver, we extend it as **raSAT** (SAT by refinement of approximations) loop to accelerate SAT detection by adding testing as an under-approximation, which works as in Algorithm 2.

Algorithm 2 **raSAT** loop for solving the constraint $\psi = \bigwedge_{j=k}^{m} g_j(x_1, \cdots, x_n) > 0$ given the initial box represented by $\Pi = \bigwedge_{i=1}^{n} x_i \in I_i^0$

```
 1: while $\Pi$ is satisfiable do                                    ▷ Some more boxes exist
 2:     $\pi = \{x_i \in I_{ik} \mid i \in \{1, \cdots, n\}, k \in \{1, \cdots, i_k\}\} \leftarrow$ a solution of $\Pi$
 3:     $B \leftarrow$ the box represented by $\bigwedge_{i=1}^{n} \bigwedge_{k=1}^{i_k} x_i \in I_{ik}$
 4:     if $B$ does not satisfy $\psi$ by using $IA$ then
 5:         $\Pi \leftarrow \Pi \wedge \neg(\bigwedge_{i=1}^{n} \bigwedge_{k=1}^{i_k} x_i \in I_{ik})$
 6:     else if $B$ satisfies $\psi$ by using $IA$ then
 7:         return SAT
 8:     else if $B$ satisfies $\psi$ by using Testing then               ▷ Different from ICP
 9:         return SAT
10:     else  ▷ Neither IA nor Testing conclude the constraint $\Longrightarrow$ Refinement Step
11:         $(x_i \in I_{ik}) \in \pi$ such that $\forall k_1 \in \{1, \cdots, i_k\} I_{ik} \in I_{ik_1}$
12:         $\{I_1, I_2\} \leftarrow branch(I_{ik})$        ▷ split $I_{ik}$ into two smaller intervals $I_1$ and $I_2$
13:         $\Pi \leftarrow \Pi \wedge (x_i \in I_{ik} \leftrightarrow (x_i \in I_1 \vee x_i \in I_2)) \wedge \neg(x_i \in I_1 \wedge x_i \in I_2)$
14:     end if
15: end while
16: return UNSAT
```

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

– Incremental widening intervals, and incremental deeping search (Section 3.1).
– Heurstic measures *SAT-likelihood* and *sensitivity*, for selection of a variable to decompose (line 11 of Algorithm 2) and a box to explore line 13 of Algorithm 2. (Section 3.2).

Interval Arithmetic

raSAT prepares various Affine Intervals, adding to Classical Interval (CI) [16], 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 [5] 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) [5],
(ii) $\epsilon\epsilon'$ is reduced to the fixed error noise symbol $\epsilon_\pm$ (AF_1 and AF_2) [15],
(iii) $\epsilon\epsilon'$ is replaced with $(-1,1)\epsilon$ (or $(-1,1)\epsilon'$) (EAI) [17],
(iv) $\epsilon\epsilon$ is reduced to fixed noise symbols ϵ_+ or ϵ_- (AF_2) [15],
(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 $\epsilon|\epsilon| = \epsilon + (-\frac{1}{4}, \frac{1}{4})$ [13].

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

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

3 Strategies in raSAT

3.1 Incremental search

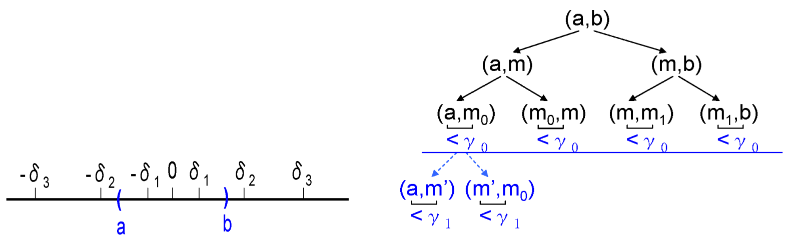
raSAT applies two incremental strategies, (1) *incremental windening*, and (2) *incremental deepening*. Let $\varphi = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} g_j > 0$ for $I_i = (a_i, b_i)$.

Incremental windening Given $0 < \delta_0 < \delta_1 < \cdots$, *incremental windening* starts with $F_0 = \exists x_1 \in I_1 \cap (-\delta_0, \delta_0) \cdots x_n \in I_n \cap (-\delta_0, \delta_0). \bigwedge_{j=1}^{m} g_j > 0$, and if it finishes with UNSAT, it runs with $F_1 = \exists x_1 \in I_1 \cap (-\delta_1, \delta_1) \cdots x_n \in I_n \cap (-\delta_1, \delta_1). \bigwedge_{j=1}^{m} g_j > 0$, and so on (Fig. 2 (a)).

If $\delta_i < \infty$, **raSAT** combines the result of an Affine Interval (currently AF2) with that of CI by taking the conjunction of two results; otherwise, it uses CI only. Experiments in Section 4 are performed with $\delta_0 = 10$ and $\delta_1 = \infty$.

Incremental deepening Starting with $\varphi = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} g_j > 0$, $I_1 \times \cdots \times I_n$ 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 exhaustive local search. To avoid it, **raSAT** applies a threshold γ, such that no more decomposition will be applied when a box becomes smaller than γ. If neither SAT nor UNSAT is detected, **raSAT** restarts with a smaller 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 2 (b)).



(a) Incremenal widening (b) Incremental Deepening

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

3.2 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 Arithmetic.

Let $\varphi = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} g_j > 0$ becomes $\vee\ (\exists x_1 \in I'_1 \cdots x_n \in I'_n. \bigwedge_{j=1}^{m} g_j > 0)$ after box decomposition. For $\exists x_1 \in I'_1 \cdots x_n \in I'_n. \bigwedge_{j=1}^{m} g_j > 0$, if some $g_j > 0$ is UNSAT, the box $I'_1 \times \cdots \times I'_n$ is UNSAT. If every $g_j > 0$ 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 g_j for $x_1 \in I'_1 \cdots x_n \in I'_n$ with IA by $range(g_j, I'_1 \times \cdots \times I'_n)$. If IA is an Affine Interval, it is in the form $(c_1, d_1)\epsilon_1 + \cdots + (c_n, d_n)\epsilon_n$, and we can obtain $range(g_j, I'_1 \times \cdots \times I'_n)$ by instantiating ϵ_i with $(-1, 1)$ for $i \in \{1, \cdots, n\}$. We define

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

Example 2. In Example 1,

- sensitivity is estimated as 1 for x and 2 for y by AF_2, and $3\frac{1}{4}$ for x and 2 for y.
- SAT-likelihood 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-likelihood intends to estimate APIs how likely to be SAT. For choosing variables, **raSAT** first choose a test-UNSAT API by SAT-likelihood. There are two choices, either *the largest* or *the least*. *Sensitivity* of a variable intends to estimate how a variable is influential to the value of an API. From a selected API by SAT-likelihood, **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 one which is more likely to be SAT. There are two choice, (1) a box with the largest SAT-likelihood, 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.

Example 3. Consider the constraint $g > 0$ with $g = -x_{15} * x_8 + x_{15} * x_2 - x_{10} * x_{16}$. For $x_2 \in [9.9, 10], x_8 \in [0, 0.1], x_{10} \in [0, 0.1], x_{15} \in [0, 10]$, and $x_{16} \in [0, 10]$, the result of Affine Interval, e.g. AF2, for g 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 g is likely to increase. As the result, the test case of x_2 is expected as high as possible in order to satisfy $g > 0$. We will thus take the upper bound value of x_2, i.e. 10, as a test case. 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 $g = 100 > 0$.

4 Experiments

We implement **raSAT** loop as an SMT solver **raSAT**, based on MiniSat 2.2 as a backend SAT solver and the library in [1] for round-down/up in Interval Arithmetics. Various combinations of strategies of **raSAT** (in Section 3) and random strategies are compared on *Zankl* and *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-likelihood,
2. a variable by the largest sensitivity, and
3. a box by the largest SAT-likelihood,

and sometimes a random choice of a test-UNSAT API (instead of the least SAT-likelihood) shows an equally good result. They are also compared with **Z3 4.3**, **iSAT3** and **dReal-2.15.01** where the former is considered to be the state of the art ([12]), and the remaining ones are a popular ICP based tools. 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.

4.1 Benchmarks from SMT-LIB

In SMT-LIB[6], 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 [12], 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-likelihood) 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.

4.2 Experiments on Strategy Combinations

Selection of Incremental Strategies We run some options for incremental widening and incremental deepening on inequalities of Zankl family in order to select the best combination. From Table 1 we prefer the following setting:

- For incremental widening, $\delta_0 = 10, \delta_1 = \infty$.
- For incremental deepening, $\gamma_i = 10^{-(i+1)}$ for $i = 0, 1, \cdots$

Benchmark	$\delta_0 = \infty, \gamma_i = 0.1$				$\delta_0 = \infty, \gamma_i = 10^{-(i+1)}$				$\delta_0 = 10, \delta_1 = \infty, \gamma_i = 10^{-(i+1)}$				$\delta_0 = 1, \delta_1 = 10, \delta_3 = \infty, \gamma_i = 10^{-(i+1)}$			
	SAT		UNSAT		SAT		UNSAT		SAT		UNSAT		δ−SAT		UNSAT	
Zankl	4	5.75 (s)	10	3.47 (s)	5	6.16 (s)	10	3.47 (s)	**20**	244.34 (s)	10	3.47 (s)	2	205.64 (s)	10	3.47 (s)

Table 1. Options for Incremental Strategies

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

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 (s)	15	1064.76 (s)	14	562.19 (s)	**21**	462.57 (s)	18	788.46(s)
Matrix-1 (UNSAT)	2	0.01 (s)	2	0.01 (s)	2	0.01 (s)	2	0.01 (s)	2	0.01 (s)	2	0.01 (s)
Matrix-2,3,4,5 (SAT)	**10**	632.37 (s)	3	140.27 (s)	1	3.46 (s)	0	0.00 (s)	5	943.08 (s)	0	0.00 (s)
Matrix-2,3,4,5 (UNSAT)	8	0.37 (s)	8	0.39 (s)	8	0.37 (s)	8	0.38 (s)	8	0.38 (s)	8	0.38 (s)

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 (s)	19	953.97 (s)	18	1068.40 (s)	19	799.79 (s)	19	230.39 (s)
Matrix-1 (UNSAT)	2	0.00(s)	2	0.00 (s)	2	0.00 (s)	2	0.00 (s)	2	0.00 (s)	2	0.00 (s)
Matrix-2,3,4,5 (SAT)	5	514.37 (s)	1	350.84 (s)	0	0.00 (s)	0	0.00 (s)	0	0.00 (s)	1	13.43 (s)
Matrix-2,3,4,5 (UNSAT)	8	0.43 (s)	8	0.37 (s)	8	0.40 (s)	8	0.38 (s)	8	0.37 (s)	8	0.38 (s)

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 (s)	19	1100.60 (s)	17	916.32 (s)	17	87.78 (s)	20	710.21 (s)
Matrix-1 (UNSAT)	2	0.00 (s)	2	0.00(s)	2	0.00 (s)	2	0.00 (s)	2	0.00 (s)	2	0.00 (s)
Matrix-2,3,4,5 (SAT)	0	0.00 (s)	1	33.17 (s)	1	201.32 (s)	2	328.03 (s)	0	0.00 (s)	1	20.94 (s)
Matrix-2,3,4,5 (UNSAT)	8	0.36 (s)	8	0.36 (s)	8	0.34 (s)	8	0.37 (s)	8	0.37 (s)	8	0.39 (s)

Benchmark	(1)-(5)-(8)		(1)-(5)-(9)		(10)-(5)-(8)		(10)-(7)-(9)	
Meti-Tarski (SAT)	3452	713.16 (s)	**3456**	644.21 (s)	3454	747.25 (s)	3451	895.14 (s)
Meti-Tarski (UNSAT)	1052	822.09 (s)	1044	957.71 (s)	**1061**	321.00 (s)	1060	233.46 (s)

Table 2. Combnations of **raSAT** strategies on NRA/Zankl,Meti-Tarski benchmark

We perform experiments only on inequalities of Zankl, and Meti-Tarski families. Table 2 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). Our combinations of strategies are,

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

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.

Experiments in Table 2 are performed with random generation (k-random tick) and the balanced decomposition (dividing at the exact middle).

Experiments with test case generation using variables sensitivity

From above section, we can see that the combination (1)-(5)-(8) shows the best performance on benchmarks. This section is going to examine the effectiveness of variables sensitivity in generation of test cases which is named as (11). Table 3 presents the result of the experiments on QF_NRA/Zankl and QF_NRA/Meti-tarski benchmarks, which show that this strategy made some improvements.

Benchmark	(1)-(5)-(8)		(1)-(5)-(8)-(11)	
Matrix-1 (SAT)	20	132.72 (s)	**25**	414.99(s)
Matrix-1 (UNSAT)	2	0.01(s)	2	0.01(s)
Matrix-2,3,4,5 (SAT)	10	632.37 (s)	**11**	1264.77(s)
Matrix-2,3,4,5 (UNSAT)	8	0.37(s)	8	0.38(s)
Meti-Tarski (SAT)	3452	713.16 (s)	**3473**	419.25 (s)
Meti-Tarski (UNSAT)	1052	822.09 (s)	1052	821.85 (s)

Table 3. Effectiveness of variables sensitivity on test cases generation

Benchmark	**raSAT**				**Z3 4.3**				**iSAT3**				**dReal**			
	SAT		UNSAT		SAT		UNSAT		SAT		UNSAT		$\delta-$SAT		UNSAT	
Zankl/matrix-1 (53)	25	414.99 (s)	2	0.01 (s)	**41**	2.17 (s)	12	0.00 (s)	11	4.68 (s)	3	0.00 (s)	46	3573.43 (s)	0	0.00 (s)
Zankl/matrix-2,3,4,5 (98)	11	1264.77 (s)	8	0.38 (s)	**13**	1031.68 (s)	11	0.57 (s)	3	196.40 (s)	12	8.06 (s)	19	2708.89 (s)	0	0.00 (s)
Meti-Tarski (5101)	3473	419.25 (s)	1052	821.85 (s)	**3528**	51.22 (s)	1568	78.56 (s)	2916	811.53 (s)	1225	73.83 (s)	3523	441.35 (s)	1197	55.39 (s)
Keymaera (68)	0	0.00 (s)	16	0.06 (s)	0	0.00 (s)	68	0.36 (s)	0	0.00 (s)	16	0.07 (s)	8	0.18 (s)	0	0.00 (s)

Table 4. Comparison among SMT solvers over inequalities

4.3 Comparison with other SMT solvers

We compare **raSAT** with other SMT solvers on NRA benchmarks: Zankl, Meti-Tarski and Keymaera. The timeout is $500s$. 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**, **dReal-2.15.01** and **Z3 4.3** means UNSAT over $[-\infty, \infty]$. Another note is that if **dReal-2.15.01** concludes SAT, the constraint is δ-SAT, which cannot imply the satisfiability of the constraint.

Table 4 shows the results of experiments. 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 when they find small UNSAT cores, without tracing all APIs. However, for SAT detection with large problems, SMT solvers need to trace all problems. Thus, it takes much longer time.

5 Extensions

5.1 Extensions for Equality Handling

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* $\varphi = \exists x_1 \in I_1 \cdots x_n \in I_n(\bigwedge_j^m f_j > 0 \ \wedge \ g = 0)$*. Suppose decomposition creates a box* $I = (l_1, h_1) \times \cdots \times (l_n, h_n)$ *where* $(l_i, h_i) \subseteq I_i$ *for all* $i \in \{1, \cdots, n\}$*, such that* $\bigwedge_j^m f_j > 0$ *is IA-VALID in the box. Let* $(l_g, h_g) = range(g, I)$*.*

(i) If $l_g > 0$ *or* $h_g < 0$*, then* F *is UNSAT in the box.*
(ii) If there are two instances $\boldsymbol{t}, \boldsymbol{t'}$ *in the box with* $g(\boldsymbol{t}) > 0$ *and* $g(\boldsymbol{t'}) < 0$*, then* F *is SAT.*

Proof. (i) If $l_g > 0$ or $h_g < 0$, then $g = 0$ cannot be satisfied in box I. As a result, F is UNSAT in I.
(ii) If there are two instances $\boldsymbol{t}, \boldsymbol{t'}$ in the box with $g(\boldsymbol{t}) > 0$ and $g(\boldsymbol{t'}) < 0$, it is clear from the Intermediate Value Theorem that there exist one 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 IA-VALID in I, $\boldsymbol{t_0}$ also satisfies $\bigwedge_j^m f_j > 0$. As a result, F is satisfiable with $\boldsymbol{t_0}$ as the SAT instance.

In the case that both (i) and (ii) do not occur, then **raSAT** continue by decomposition.

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 3). 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.

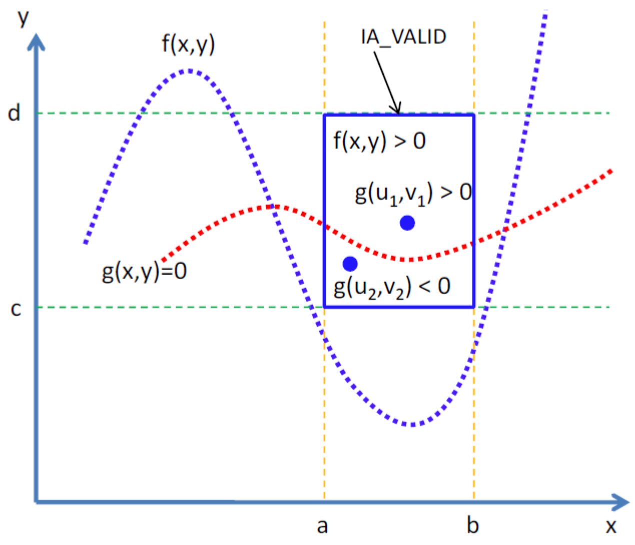


Fig. 3. Example on solving single equation using the Intermediate Value Theorem

Multiple Equations The idea of using the Intermediate Value Theorem can also be used for solving multiple equations. Consider m equations ($m \geq 1$): $\bigwedge_{j=1}^{m} g_j = 0$ and an box $I = (l_1, h_1) \times \cdots (l_n, h_n)$. If we can find a set $\{V_1, \cdots, V_m\}$ that satisfies the following properties, then we can conclude that $\bigwedge_{j=1}^{m} g_j = 0$ is satisfiable in I.

- For all $j \in \{1, \cdots, m\}$, we have $V_j \subseteq var(g_j)$.
- For all $j_1 \neq j_2 \in \{1, \cdots, m\}$, we have $V_{j_1} \cap V_{j_2} = \emptyset$.
- For all $j \in \{1, \cdots, m\}$, let $k_j = |V_j|$ and $V_j = \{v_{jk} \mid 1 \leq k \leq k_j\}$, then, there exist two combinations $(x_{j1}, \cdots, x_{jk_j}) = (t_{j1}, \cdots, t_{jk_j})$ and $(x_{j1}, \cdots, x_{jk_j}) = (t'_{j1}, \cdots, t'_{jk_j})$ where $t_{jk} \neq t'_{jk} \in (l_{jk}, h_{jk})$, $1 \leq k \leq k_j$ such that
$$g_j(t_{j1}, \cdots, t_{jk_j}, \cdots, x_{jk}, \cdots) > 0$$
and
$$g_j(t'_{j1}, \cdots, t'_{jk_j}, \cdots, x_{jk}, \cdots) < 0$$
for all values of x_{jk} in (l_{jk}, h_{jk}) where $x_{jk} \in var(g_j) \setminus V_j$. We denote $ivt(g_j, V_j, I)$ to represent that the polynomial g_j enjoy this property with respect to V_j and I.

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 4) which satisfy the above restriction on the number of variables, and the variable intervals are $x \in (c_1, d_1)$ and $y \in (d_2, c_2)$. Let $V_1 = \{x\}$ and $V_2 = \{y\}$, we have:

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

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

This is because $ABCD$ creates a Jordan Curve and the continuous graph of g_2 connects point E in the interior to point F in the exterior of that curve, so the graph of g_2 must intersects the curve somewhere. Furthermore, the graph of g_2 can intersect with neither AB nor DC. As a result, it must intersect with the graph of g_1. Thus we can conclude that $g_1(x, y) = 0 \wedge g_2(x, y) = 0$ has a solution inside the box $(c_1, d_1) \times (d_2, c_2)$.

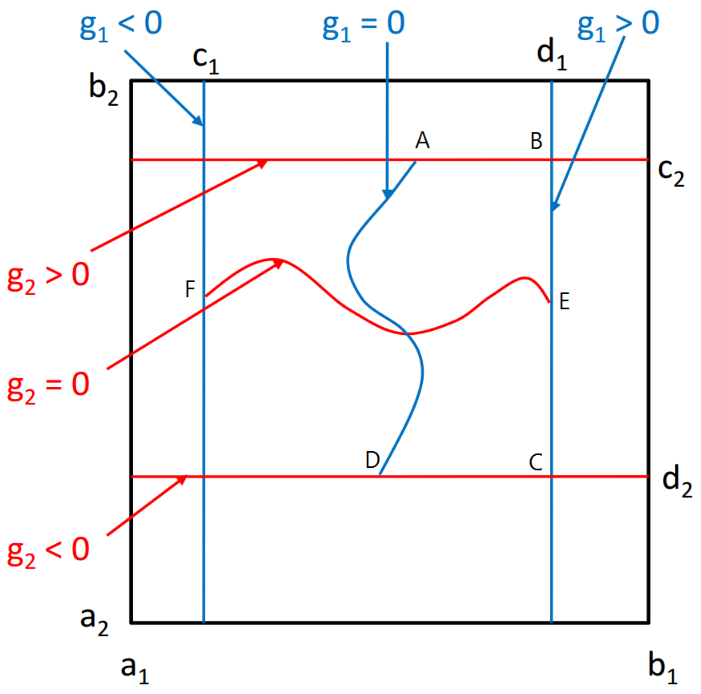


Fig. 4. Example on solving single equation using the Intermediate Value Theorem

Our current implementation of handling multiple equations is very naive which is described in Algorithm 3 because for each equality $g_j = 0$, **raSAT** checks every possible subsets of its variables as candidates for V_j. As a result, given the constraint $\bigwedge_{j=1}^{m} g_j = 0$, in the worst case **raSAT** will check $2^{|var(g_1)|} * \cdots * 2^{|var(g_m)|}$

Benchmark	**raSAT**				**Z3 4.3**				**iSAT3**				**dReal**			
	SAT		UNSAT		SAT		UNSAT		SAT		UNSAT		$\delta-$SAT		UNSAT	
Zankl (15)	**11**	0.07 (s)	**4**	0.17 (s)	**11**	0.17 (s)	**4**	0.02 (s)	0	0.00 (s)	**4**	0.05 (s)	11	0.06 (s)	**4**	0.02(s)
Meti-Tarski (3528/1573)	875	174.90 (s)	781	401.15 (s)	**1497**	21.00 (s)	**1115**	74.19 (s)	1	0.28 (s)	1075	22.6 (s)	1497	72.85 (s)	943	21.40 (s)
Keymaera (612)	0	0.00 (s)	312	66.63 (s)	0	0.00 (s)	**610**	2.92 (s)	0	0.00 (s)	226	1.63 (s)	13	4.03 (s)	318	1.96 (s)

Table 5. Comparison among SMT solvers over equalities

cases. As a future work, we may use variables' sensitivity to give priority on subsets of variables.

Algorithm 3 Solving multiple equations $\bigwedge_{j=1}^{m} g_j = 0$ with interval constraint $\Pi = \bigwedge_{i=1}^{n} x_i \in (l_i, h_i)$

```
 1: function EQUATIONSPROVER($\bigwedge_{j=k}^{m} g_j = 0, \Pi, V_0$)
 2:     if $k > m$ then                                   ▷ All equations are checked
 3:         return SAT
 4:     end if
 5:     for $V_k \in P(var(g_k))$ do              ▷ $P(var(g_k))$ is the powerset of $var(g_k)$
 6:         if $V_k \cap V = \emptyset$ and $ivt(g_k, V_k, \Pi)$ then
 7:             $V_0 \leftarrow V_0 \cup V_k$
 8:             if EQUATIONSPROVER($\bigwedge_{j=k+1}^{m} g_j = 0, \Pi, V_0$) = SAT then
 9:                 return SAT
10:             end if
11:         end if
12:     end for
13:     return UNSAT
14: end function
15: EQUATIONSPROVER($\bigwedge_{j=1}^{m} g_j = 0, \Pi, \emptyset$)
```

We also do experiments on constraints with equalities from QF_NRA/Zankl and QF_NRA/Meti-tarski. Table 5

5.2 Extension for Polynomial Constraints Over Integers

raSAT loop is easily modified to QF_NIA (nonlinear arithmetic over integers) from QF_NRA, by setting $\gamma_0 = 1$ in incremental deepening in Section 3.1 and restricting testdata generation on intergers. We also compare **raSAT** with **Z3 4.3** on benchmarks of QF_NIA/AProVE. **AProVE** consists of 6850 inequalities and 1979 equalities. Some has several hundred variables, but each API has few variables (mostly just 2 variables). Note that the use of the Intermediate Value

Benchmark	**raSAT**				**Z3 4.3**			
	SAT		UNSAT		SAT		UNSAT	
inequalities (6850)	**6784**	65.60 (s)	0	0.00 (s)	**6784**	97.77 (s)	**36**	32.46 (s)
equalities (1979)	891	33721.37 (s)	16	27.34 (s)	**900**	1951.01(s)	**250**	3104.74(s)

Table 6. Comparison on NIA/AProVE

Theorem cannot be applied for QF_NIA constraints since the polynomials are not continuous. However, Interval Arithmetics can conclude UNSAT of QF_NIA benchmarks because UNSAT over real numbers simply implies UNSAT over integer numbers. For SAT benchmarks (both inequalities and equalities), **raSAT** concludes satisfiability only by testing.

The results are presented in Table 6 where the timeout is $500s$. **raSAT** does not detect unsatisfiability well since UNSAT problems have quite large coefficients which lead exhaustive search on quite large area.

6 Conclusion

This paper presented **raSAT** loop, which extends ICP with testing to accelerate SAT detection and implemented as an SMT solver **raSAT**. With experiments on benchmarks from QF NRA category of SMT-lib, we found two heuristic measures SAT-likelihood and sensitivity, which lead effective strategy combination for SAT detection. **raSAT** still remains in naive proto-type status, and there are lots of future work.

UNSAT core. Currently, **raSAT** focuses on SAT detection. For UNSAT detection, the target is to find a small UNSAT core in a large problem.

Equality handling. Section 5 shows equality handling where UNSAT constraints can be completely solved by ICP (with the assumption of bounded intervals). The Intermediate Value Theorem can be used to show satisfiability with restrictions on variables of polynomials. Moreover, the use of this theorem is not complete in showing satisfiability. As a future work, we will apply Groebner basis in addition to the current use of the Intermediate Value Theorem.

Further strategy refiment. Currently, raSAT uses only information from O.T (Interval Arithmetic).We are planning to refine strategies such that previous O.T and U.T results mutually guide to each other. For instance, a box decomposition strategy can be more focused.

References

[1] Alliot, J.M., Gotteland, J.B., Vanaret, C., Durand, N., Gianazza, D.: Implementing an interval computation library for OCaml on x86/amd64 architectures. In: ICFP. ACM (2012)

[2] Bryant, R.E., Kroening, D., Ouaknine, J., Seshia, S.A., Strichman, O., Brady, B.: Deciding bit-vector arithmetic with abstraction. In: IN PROC. TACAS 2007. Lecture Notes in Computer Science, vol. 4424, pp. 358–372. Springer (2007)
[3] Collins, G.: Quantifier elimination by cylindrical algebraic decomposition twenty years of progress. In: Quantifier Elimination and Cylindrical Algebraic Decomposition, pp. 8–23. Springer Vienna (1998)
[4] Coln, M., Sankaranarayanan, S., Sipma, H.: Linear invariant generation using non-linear constraint solving. In: Computer Aided Verification, Lecture Notes in Computer Science, vol. 2725, pp. 420–432. Springer Berlin Heidelberg (2003)
[5] Comba, J.L.D., Stolfi, J.: Affine arithmetic and its applications to computer graphics (1993)
[6] Corzilius, F., Loup, U., Junges, S., brahm, E.: Smt-rat: An smt-compliant nonlinear real arithmetic toolbox. In: Theory and Applications of Satisfiability Testing SAT 2012, vol. 7317, pp. 442–448. Springer Berlin Heidelberg (2012)
[7] 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, 209–236 (2007)
[8] Ganai, M., Ivancic, F.: Efficient decision procedure for non-linear arithmetic constraints using cordic. In: Formal Methods in Computer-Aided Design, 2009. FMCAD 2009. pp. 61–68 (Nov 2009)
[9] Gao, S., Kong, S., Clarke, E.: dreal: An smt solver for nonlinear theories over the reals. In: Bonacina, M. (ed.) Automated Deduction CADE-24, Lecture Notes in Computer Science, vol. 7898, pp. 208–214. Springer Berlin Heidelberg (2013)
[10] Hickey, T., Ju, Q., Van Emden, M.H.: Interval arithmetic: From principles to implementation. J. ACM 48(5), 1038–1068 (Sep 2001)
[11] Hong, H., Din, M.S.E.: Variant quantifier elimination. Journal of Symbolic Computation 47(7), 883 – 901 (2012), international Symposium on Symbolic and Algebraic Computation (ISSAC 2009)
[12] Jovanovi, D., de Moura, L.: Solving non-linear arithmetic. In: Gramlich, B., Miller, D., Sattler, U. (eds.) Automated Reasoning, Lecture Notes in Computer Science, vol. 7364, pp. 339–354. Springer Berlin Heidelberg (2012)
[13] Khanh, T.V., Ogawa, M.: {SMT} for polynomial constraints on real numbers. Electronic Notes in Theoretical Computer Science 289(0), 27 – 40 (2012), third Workshop on Tools for Automatic Program Analysis (TAPAS' 2012)
[14] Lucas, S., Navarro-Marset, R.: Comparing csp and sat solvers for polynomial constraints in termination provers. Electron. Notes Theor. Comput. Sci. 206, 75–90 (Apr 2008)
[15] Messine, F.: Extensions of affine arithmetic: Application to unconstrained global optimization
[16] Moore, R.: Interval analysis. Prentice-Hall series in automatic computation, Prentice-Hall (1966)
[17] 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. pp. 105–114. SEFM '09, IEEE Computer Society, Washington, DC, USA (2009)
[18] Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Abstract dpll and abstract dpll modulo theories. In: In LPAR04, LNAI 3452. Lecture Notes in Computer Science, vol. 3452, pp. 36–50. Springer (2005)
[19] Passmore, G.O., Jackson, P.B.: Combined decision techniques for the existential theory of the reals. In: CALCULEMUS. Lecture Notes in Computer Science, vol. 5625, pp. 122–137. Springer-Verlag (2009)

[20] Ratschan, S.: Efficient solving of quantified inequality constraints over the real numbers. ACM Trans. Comput. Logic 7(4), 723–748 (Oct 2006)

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

[22] Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Non-linear loop invariant generation using gröbner bases. SIGPLAN Not. 39(1), 318–329 (Jan 2004)

[23] Tarski, A.: A decision method for elementary algebra and geometry. In: Quantifier Elimination and Cylindrical Algebraic Decomposition, pp. 24–84. Springer Vienna (1998)

[24] Zankl, H., Middeldorp, A.: Satisfiability of non-linear (ir)rational arithmetic. In: Clarke, E., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning. Lecture Notes in Computer Science, vol. 6355, pp. 481–500. Springer Berlin Heidelberg (2010)