raSAT: SMT Solver for Polynomial Constraints over Reals

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 [18].
- **Automatic termination proving**, which reduces termination detection to finding a suitable ordering [15], e.g., T$_{\text{T}}$T$_2$[3], AProVE[4].

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

- **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 [23] requires more complex polynomials.
- **Hybrid system**. SMT solvers for polynomial constraints over real numbers (QF_NRA) are often used as backend engines [22].

Solving polynomial constraints on real numbers is decidable [24], 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 [13]. 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* [12] 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** [10], and **RSolver** [21]. 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 [16, 18, 14], 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 [11] 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

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

– 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.

– **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 [20] and Z3 4.3 (which is referred as nlsat in [13]) 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 [25] 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 [19] 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. **dReal**'s methodology [9] also cooperates DPLL with ICP in the lazy manner as in **raSAT**.

2 ICP overview and raSAT loop

Our target problems is a nonlinear constraint, especially over reals. That over integer will be briefly shown in Section 4.4. We mainly discuss on polynomial inequaltiies, and later in Section 5, we show an extension to cover polynomial equality based on Intermediate Value Theorem.

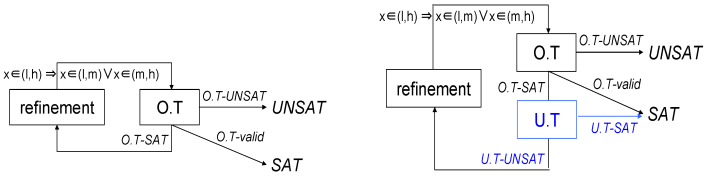
Definition 1. *A polynomial inequality constraint is*

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

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 polynomial. We denote the set of varialbes appearing in* p_j *by* $var(p_j)$.

Note that φ is equivalent to $\exists x_1 \ldots x_n.(\bigwedge_{i=1}^{n} x_i \in I_i) \wedge (\bigwedge_{j=1}^{m} \psi_j(x_1, \cdots, x_n))$. We call $\bigwedge_i x_i \in I_i$ an *interval constraint*, and we refer $\bigwedge_{j=1}^{m} \psi_j(x_1, \cdots, x_n)$ by $\psi(x_1, \cdots, x_n)$. We denote the set of solutions of the constraint $\psi(x_1, \cdots, x_n)$ as $\mathbb{S}(\psi(x_1, \cdots, x_n)) = \{(r_1, \cdots, r_n) \in \mathbb{R}^n \mid \psi(r_1. \cdots, r_n) \text{ holds}\}$.

We first review Interval Constraint Propagation (ICP), and then introduce **raSAT** (refinemnet of approximations for SAT) loop [14]. The main difference is **raSAT** loop has testing after the estimation by interval arithmetic (IA) to boost SAT detection. Note that both ICP and **raSAT** are suffered from roundoff errors of the floating arithmetic. To guarantee the soundness, IA adopts the round down/up for the estimation of lower/upper bounds of intervals. In **raSAT**, when testing says SAT, it is confirmed with the error bound guaranteed package **iRRAM**.



(a) ICP loop **raSAT** loop

Fig. 1. Refinement loops

2.1 ICP overview

Algorithm 1 describe the basic ICP for solving polynomial inequalities. Let $\psi = \bigwedge_{j=k}^{m} g_j(x_1, \cdots, x_n) > 0$ be a target constraint.

Algorithm 1 ICP starting from the initial box $B_0 = I_1 \times \cdots \times I_n$

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

where two functions $prune(B, \psi)$ and $split(B)$ satisfy

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

Since ICP concludes SAT (line 8) only when it finds a box in which the constraint becomes valid by IA. It is also suffered from roundoff errors, and the basic ICP cannot conclude the satifiability of equations. In contrast, although the number of boxes increase exponentially, ICP always detect SAT of an inequality constraint $\exists x_1 \in (a_1, b_1) \cdots x_n \in (a_n, b_n). \wedge_j g_j > 0$ if the *split* is fairly applied and each I_j is bounded. However, ICP may miss to detect UNSAT. Limitations for detecting UNSAT come from the *kissing* and *convergent* cases in Fig. 2. The 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, it cannot be separated by the covering by (enough small) boxes. The right shows a convergent case $y > x + \frac{1}{x} \wedge y < x \wedge x > 0$, i.e., $xy > x^2 + x \wedge y < x \wedge x > 0$. The latter does not appear if all intervals I_j are finitely bounded.

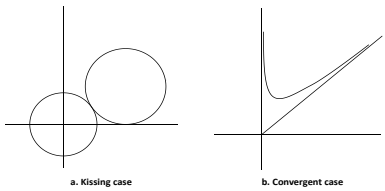


Fig. 2. Limitations for detection of UNSAT

2.2 raSAT loop

We extend ICP with testing to accelerate SAT detection, which is called **raSAT** loop [14]. Let $\psi = \bigwedge_{j=k}^{m} g_j(x_1, \cdots, x_n) > 0$ be a target constraint. Algorithm 2 displays the **raSAT** loop, which detects as below.

Algorithm 2 raSAT loop starting from the initial box $\Pi = \bigwedge_{i=1}^{n} x_i \in I_i^0$

while Π is satisfiable **do** ▷ Some more boxes exist
  $\pi = \{x_i \in I_{ik} \mid i \in \{1, \cdots, n\}, k \in \{1, \cdots, i_k\}\} \leftarrow$ a solution of Π
  $B \leftarrow$ the box represented by $\bigwedge_{i=1}^{n} \bigwedge_{k=1}^{i_k} x_i \in I_{ik}$
  if B does not satisfy ψ by using IA **then**
    $\Pi \leftarrow \Pi \wedge \neg(\bigwedge_{i=1}^{n} \bigwedge_{k=1}^{i_k} x_i \in I_{ik})$
  else if B satisfies ψ by using IA **then**
    return SAT
  else if B satisfies ψ by using *Testing* **then** ▷ Different from ICP
    return SAT
  else ▷ Neither IA nor *Testing* conclude the constraint $\Longrightarrow$ *Refinement* Step
    $(x_i \in I_{ik}) \in \pi$ such that $\forall k_1 \in \{1, \cdots, i_k\} I_{ik} \in I_{ik_1}$
    $\{I_1, I_2\} \leftarrow split(I_{ik})$ ▷ split I_{ik} into two smaller intervals I_1 and I_2
    $\Pi \leftarrow \Pi \wedge (x_i \in I_{ik} \leftrightarrow (x_i \in I_1 \vee x_i \in I_2)) \wedge \neg(x_i \in I_1 \wedge x_i \in I_2)$
  end if
end while
return UNSAT

Our design of an SMT solver **raSAT** adapts various interval arithmetic, and applies two main heuristics. The use of Affine intervals enable us to apply the latter.

- 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, Algorithm 2). (Section 3.2).

2.3 Interval Arithmetic

raSAT prepares various Affine Intervals [5], adding to Classical Interval (CI) [17]. Although precision of the estimations is incomparable, Affine intervals preserve a part of the dependency among values, which are lost in CI. For instance, $x - x$ is evaluated to $(-2, 2)$ for $x \in (2, 4)$ by CI, but 0 by an Affine interval.

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) [5],
(ii) $\epsilon\epsilon'$ is reduced to the fixed error noise symbol $\epsilon_\pm$ (AF_1 and AF_2) [16],
(iii) $\epsilon\epsilon'$ is replaced with $(-1, 1)\epsilon$ (or $(-1, 1)\epsilon'$) (EAI) [18],
(iv) $\epsilon\epsilon$ is reduced to fixed noise symbols ϵ_+ or ϵ_- (AF_2) [16],
(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 $\epsilon|\epsilon| = \epsilon + (-\frac{1}{4}, \frac{1}{4})$ [14].

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 SAT directed Strategies of raSAT

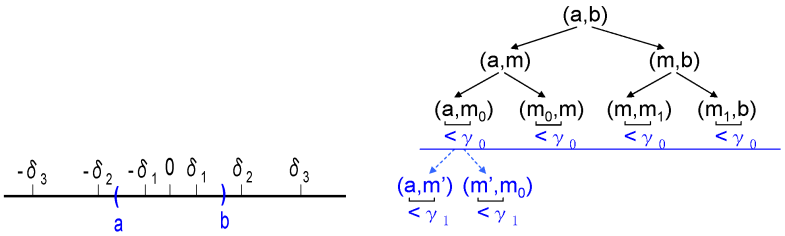
ICP is affected less with the degree of polynomials, but affected most with the number of variables. Starting with $\varphi = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} g_j > 0$, $I_1 \times \cdots \times I_n$ is decomposed into exponentially many boxes, and F becomes the disjunction of existential formulae corrsponding to these boxes. The detection of UNSAT requires exhaustive search on all boxes, and finding a small UNSAT core is the key. This is often observed **Z3**; UNSAT is either very quickly detected, or leads timeout. For SAT detection, the keys will be a starategic control not to fall into local optimals and a strategy to choose most likely decomposition/boxes.

3.1 Incremental search

Two incremental search strategies are prepared for **raSAT** applies, (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 stays UNSAT, then enlarge the intervals as $F_1 = \exists x_1 \in I_1 \cap (-\delta_1, \delta_1) \cdots x_n \in I_n \cap (-\delta_1, \delta_1). \bigwedge_{j=1}^{m} g_j > 0$. This continues until eihter SAT, timeout, or a given bound of repeatation (Fig. 3 (a)).

Note that if $\delta_i = \infty$, we cannot use an Affine interval For instance, $(-\infty, \infty) = \infty\epsilon$ does not make sense. In **raSAT**, AF_2 is used if $\delta_i < \infty$, and CI is used otherwise. Experiments in Section 4 are performed with $\delta_0 = 10$ and $\delta_1 = \infty$.



(a) Incremenal widening (b) Incremental Deepening

Fig. 3. Two incremental search strategies

Incremental deepening To combine depth-first-search and breadth-first search among decomposed boxes, **raSAT** applied *incremental deepening*. Let $\gamma_0 > \gamma_1 > \cdots > 0$. It applies a threshold γ, such that no more decomposition occurs when a box becomes smaller than γ. γ is initially $\gamma = 0$. If neither SAT nor UNSAT is detected, **raSAT** restarts with the threshhold γ_1. This contrinues until either SAT, timeout, or a given bound of repeatation (Fig. 3 (b)).

3.2 SAT directed heuristics measure

In **raSAT**, a strategy to select a variable to decompose is in the following two steps. (1) First select a most likely influential *test-UNSAT API*, and (2) then choose a most likely influential variable in the API. For *most likely influential*

measures, we apply the *SAT-likelyhood* on APIs and the *sensitivity* on variables, respectively. Note that the latter measure is defined only for Affine intervals.

Let $\vee\ (\exists x_1 \in I'_1 \cdots x_n \in I'_n.\ \bigwedge_{j=1}^{m} g_j > 0)$ be the results of decompositions starting from the initial constraint $\varphi = \exists x_1 \in I_1 \cdots x_n \in I_n.\ \bigwedge_{j=1}^{m} g_j > 0$. 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, we conclude SAT. In case that still a test-UNSAT (thus IA-SAT) API remains, the next decomposition starts.

For a box $x_1 \in I'_1 \cdots x_n \in I'_n$, the estimated range of g_j by IA is denoted by $range(g_j, I'_1 \times \cdots \times I'_n)$. If IA is an Affine Interval, we assume that the estimated range of $g_j > 0$ has the form $(c_1, d_1)\epsilon_1 + \cdots + (c_n, d_n)\epsilon_n$. By instantiating $(-1, 1)$ to ϵ_i, we obtain $range(g_j, I'_1 \times \cdots \times I'_n)$. We define

- the *SAT-likelihood* of an API $g_j > 0$ by $|I \cap (0, \infty)|/|I|$ for $I = range(g_j, I'_1 \times \cdots \times I'_n)$, and
- the *sensitivity* of a variable x_i in an API $g_j > 0$ by $max(|c_i|, |d_i|)$.

Example 2. In Example 1,

- SAT-likelihood of f is $0.4 = \frac{6}{9-(-6)}$ by AF_2 and $0.36 = \frac{4.5}{4.5-(-8)}$ by CAI.
- the sensitivity of x is 1 by AF_2 and $3\frac{1}{4}$ by CAI, and that of y is 2 by both AF_2 and CAI.

The SAT-likelihood intends to estimate APIs how likely to be SAT. There are two choices on the SAT-likelihood, either *the largest* or *the least*. The *sensitivity* of a variable intends to estimate how a variable is influential to the value of an API, and the largest sensitivity is considered to be the most influential. This selection of variables are used both for (1) *decomposition*, and (2) *test case generation*. For multiple test generation, we select multiple variables that have larger sensitivity.

At the decomposition, **raSAT** also examines the choice of the box. We define the *SAT-likelihood* of a box $I'_1 \times \cdots \times I'_n$ by the least SAT-likelihood of test-UNSAT APIs. Since the SAT-likelihood each box is computed when it is created by the decoposition, **raSAT** simply cocmpares newly decomposed boxes with the prievious ones. There are two choices of boxes, (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. These combinations of strategy choices are compared by experiments in Section 4.

Test case generation strategy The sensitivity of variables is also used for test case generation. That is, **raSAT** generates two test cases for the specified number of variables, and one for the rest. Such variables are selected from those with larger sensitivity. When two test cases are generated, **raSAT** also observes the sign of the coefficients of noise symbols. If positive, it takes the upper bound of possible values as the first test case; otherwise, the lower bound. The second test case is generated randomly.

Example 3. Let $g = -x_{15} * x_8 + x_{15} * x_2 - x_{10} * x_{16}0$ and consider a constraint $g > 0$. 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 Affine Interval (AF_2) estimates it $0.25\epsilon_2 - 0.25\epsilon_8 - 0.25\epsilon_{10} + 49.5\epsilon_{15} - 0.25\epsilon_{16} + 0.75\epsilon_{+-} + 49.25$. The coefficient of ϵ_2 is 0.25, which is positive. Thus, if x_2 increases, the value of g is likely to increase. Then, we take the upper bound of possible values of x_2 as a test case, i.e. 10. Similarly, we take the test cases for other variables: $x_8 = 0, x_{10} = 0, x_{15} = 10, x_{16} = 0$, and we have $g = 100 > 0$ with them.

4 Experiments

We implement **raSAT** loop as an SMT solver **raSAT**, based on MiniSat 2.2 as a backbone SAT solver and the library in [1] for round-down/up in each 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 choice by the least SAT-likelihood,
2. a variable choice by the largest sensitivity, and
3. a box choice by the largest SAT-likelihood.

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 ([13]), and the remaining ones are a popular ICP based tools. Note that our comparison in this section is only on polynomial inequality. Preliminary results on equality will be reported in Section 5. The experiments are with Intel Xeon E5-2680v2 2.80GHz and 4 GB of RAM.

4.1 Benchmarks from SMT-LIB

SMT-LIB[6] benchmark on non-linear real number arithmetic (QF_NRA) has Meti-Tarski, Keymaera, Kissing, Hong, and Zankl families, of which brief statistics is summarized below. 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 [13], which shows Z3 4.3 is one of the strongest.

- **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 variables, 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.

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

– **Keymaera** contains 161 inequalities among 4442.
– **Kissing** has 45 problems, all of which contains equality (mostly single equation).
– **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 the 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** splits at exactly the middle.

4.2 Experiments on Strategy Combinations

Selection of Incremental Strategies We run some options for incremental widening and incremental deepening on Zankl family in order to select the best combination. From now on, we set as below.

– 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		$\delta-$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

Table 2 shows the experimental results of above mentioned combination. The timeout is set to 500s, and time shows 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.

Here, (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 with test case generation using variables sensitivity

This section examines the effectiveness of the sensitivity in test case generation, which is referred by (11). Table 3 presents the result on Zankl and Meti-tarski benchmarks, which show that this strategy made improvements in SAT detection.

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

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

4.3 Comparison with other SMT solvers

We compare **raSAT** with other SMT solvers in Table 4. The timeout is $500s$. For **iSAT3**, the 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 SAT statements by **dReal-2.15.01** means δ-SAT, which allows δ deviation. Thus, it does not mean really SAT, and a number of UNSAT problems in Zankl, **dReal** are concluded SAT.

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 on SAT detection of vary long constraints (APIs longer than 40 and/or

Benchmark	**raSAT**				**Z3 4.3**				**iSAT3**				**dReal**			
	SAT		UNSAT		SAT		UNSAT		SAT		UNSAT		δ−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

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 5. Comparison on NIA/AProVE

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 find small UNSAT cores, without tracing all APIs. Otherwise, it leads timeout. However, for SAT detection with large problems, SMT solvers need to trace all APIs. Thus, it takes much longer time.

4.4 Polynomial Constraints Over Integers

raSAT loop is easily modified to QF_NIA (nonlinear arithmetic over integers) from QF_NRA. We obtain SAT detection over integers by setting $\gamma_0 = 1$ in the incremental deepening in Section 3.1 and restricting test data generation on intergers, where UNSAT detection is the same as for QF_NIA benchmarks. We compare **raSAT** with **Z3 4.3** on benchmarks of QF_NIA/AProVE, which consists of 6850 inequalities and 1979 equalities. Some has several hundred variables, but each API has few variables (mostly just 2 variables). The preliminary results (with the time out $500s$) are presented in Table 5. **raSAT** does not detect UNSAT well since UNSAT problems have quite large coefficients, which lead exhaustive search on quite large area.

5 Extension for Equality Handling

For a polynoial constraint with equality $\varphi : \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} \psi_j(x_1, \cdots, x_n) \wedge \bigwedge_{j=1}^{m'} g_j(x_1, \cdots, x_n) = 0$, one typical way to solve equality

is an algebraic method, e.g., Groebner basis. In this section, we try a simple method based on *Intermediate Value Theorem*. It is illustrated by a single equation case. Note that before solving on equality, we assume a(n enough small) box that makes $\bigwedge_{j=1}^{m} \psi_j(x_1, \cdots, x_n)$ IA-valid.

Single Equation For solving polynomial constraints with single equality ($g = 0$), we can apply in a simple way. That is, finding 2 test cases with $g > 0$ and $g < 0$ implies $g = 0$ 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.*

If neither (i) nor (ii) holds, **raSAT** continues the decomposition.

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

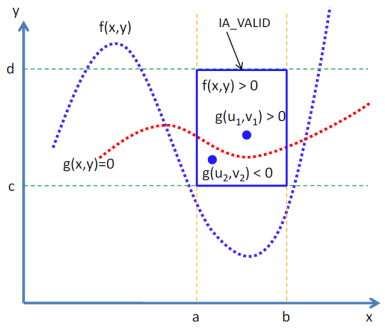


Fig. 4. Example on solving single equation by Intermediate Value Theorem

raSAT first tries to find a box (by the decomposition) such that $\bigwedge_{j}^{m} f_j > 0$ is IA-VALID in the box. Then it tries to find 2 instances with $g > 0$ and $g < 0$ by testing. Intermediate Value Theorem guarantees the existence of an SAT instance in between. Note that this method does not find a SAT instance itself.

Multiple Equations The idea of Intermediate Value Theorem is extended for solving multiple equations. Consider m equations ($m \geq 1$): $\bigwedge_{j=1}^{m} g_j = 0$ and an box $I = (l_1, h_1) \times \cdots (l_n, h_n)$. For $V = \bigcup_{j=1}^{m} var(g_j)$ and $V' = \{x_{j_1}, \cdots x_{j_k}\} \subseteq V$, we denote $\{(v_1, \cdots, v_n) \in I \mid v_{j_i} = l_{j_i} \text{ for each } i\}$ and $\{(v_1, \cdots, v_n) \in I \mid v_{j_i} = h_{j_i} \text{ for each } i\}$ by $I \downarrow_{V'}$ and $I \uparrow_{V'}$, respectively.

Definition 2. *A seqeunce $(V_1, \cdots, V_m)$ of subsets of V is a* check basis *of $(g_1, \cdots, g_m)$, if, for each j, j' with $1 \leq j, j' \leq m$,*

1. *$V_j (\neq \emptyset) \subseteq var(g_j)$,*
2. *$V_j \cap v_{j'} = \emptyset$ if $j \neq j'$, and*
3. *either $g_j > 0$ on $I \uparrow_{V_j}$ and $g_j < 0$ on $I \downarrow_{V_j}$, or $g_j < 0$ on $I \uparrow_{V_j}$ and $g_j > 0$ on $I \downarrow_{V_j}$.*

Lemma 2. *For $\varphi : \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} \psi_j(x_1, \cdots, x_n) \wedge \bigwedge_{j=1}^{m'} g_j(x_1, \cdots, x_n) = 0$ and $I = (l_1, h_1) \times \cdots (l_n, h_n) \subseteq I_1 \times \cdots \times I_n$, assume that*

1. *$\bigwedge_{j=1}^{m} \psi_j(x_1, \cdots, x_n)$ is IA-valid on I, and*
2. *there is a* check basis *$(V_1, \cdots, V_m)$ of $(g_1, \cdots, g_m)$.*

Then, $\bigwedge_{j=1}^{m} g_j = 0$ has a SAT instance in I (and thus φ is SAT).

The idea is, from the Intermediate Value Theorem, each g_j has a $n - |V_j|$ dimesion surface of null points of g_j between $I \uparrow_{V_j}$ and $I \downarrow_{V_j}$. Since V_j's are mutually disjoint (and g_j' are continuous), we have the intersection of all such surfaces of null points with the dimension $n - \sum_{j=1}^{m} |V_j|$. Thus, this method has a limitation 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 $g_2(x, y) = 0$, and assume that $(\{x\}, \{y\})$ is a check basis of (g_1, g_2) on $[c_1, d_1] \times [c_2, d_2]$ (Fig. 5). Then, the blue line (null points of g_1) and the red line (null points of g_2) must have an intersection. We can explain this by Jordan curve theorem. Since $ABCD$ is a closed curve such that E is inner and F is outer, a continuous (red) line EF must have an intersection by Jordan curve theorem.

Current **raSAT** implementation on equations has only naive strategies. For instance, for each $g_j = 0$, **raSAT** checks all possible subsets of its variables as candidates for V_j. Thus, in the worst case **raSAT** checks $2^{|var(g_1)|} * \cdots * 2^{|var(g_m)|}$ cases. It also does not prepare a strategy to find a box that makes all inequations IA-valid. Preliminary experiments on handling equations from QF_NRA/Zankl and QF_NRA/Meti-tarski are summmarized in Table 6. We hope that the sensitivity will give their effective strategies.

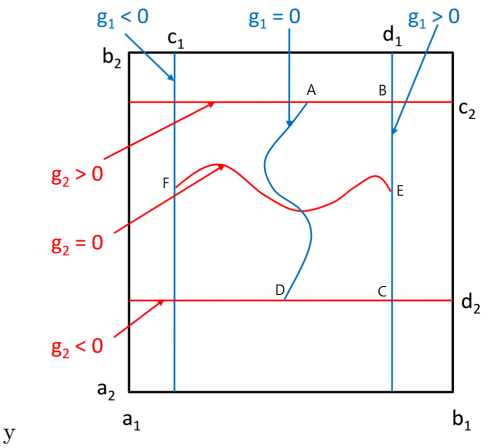


Fig. 5. Example on solving single equation using the Intermediate Value Theorem

Benchmark	raSAT				Z3 4.3				iSAT3				dReal			
	SAT		UNSAT		SAT		UNSAT		SAT		UNSAT		δ−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 6. Comparison among SMT solvers with equations

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., Avigad, J., Clarke, E.M.: Delta-complete decision procedures for satisfiability over the reals. In: Proceedings of the 6th International Joint Conference on Automated Reasoning. pp. 286–300. IJCAR'12, Springer-Verlag, Berlin, Heidelberg (2012)

[10] 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)

[11] Hickey, T., Ju, Q., Van Emden, M.H.: Interval arithmetic: From principles to implementation. J. ACM 48(5), 1038–1068 (Sep 2001)

[12] 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)

[13] 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)

[14] 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)

[15] 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)

[16] Messine, F.: Extensions of affine arithmetic: Application to unconstrained global optimization

[17] Moore, R.: Interval analysis. Prentice-Hall series in automatic computation, Prentice-Hall (1966)

[18] 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)

[19] 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)

[20] 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)

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

[22] 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)

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

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

[25] 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)