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 solver **raSAT** for polynomial inequality. It consists of a simple iterative approximation refinement, **raSAT** loop, which is an extension of the standard ICP (Interval Constraint Propagation) with testing, aiming to accelerate SAT detection. If it fails to decide, input intervals are refined by decomposition.
ICP is robust for large degrees, but the number of boxes (products of intervals) may exponentially explode with respect to the number of variables. For boosting SAT detection, we design strategies on the choice of a variable to decompose and a box to explore. Heuristic measures, e.g., *SAT likelihood*, *sensitivity*, and *the number of unsolved atomic polynomial constraints*, are compared on Zankl, Meti-tarski, and Keymaera benchmarks from QF_NRA of SMT-LIB. The results are compared with those of **Z3 4.3**, **dReal-2.15.01** and **iSAT3**.
At last, **raSAT** loop is extended for polynomial equality based on the Intermediate Value Theorem. This extension is again evaluated on Zankl, Meti-tarski, and Keymaera families. We also show a simple modification to handle mixed integers, and experimental results on AProVE benchmark from QF_NIA of SMT-LIB.

1 Introduction

Polynomial constraint solving over reals (resp. integers) is to find an instance from reals (resp. integers) that satisfies given polynomial inequality/equality. Solving polynomial constraints on reals is decidable [21], though that on integers is undecidable (*Hilbert's 10th problem*). Quantifier Elimination by Cylindrical Algebraic Decomposition (QE-CAD) [4] is a well known technique, and implemented in Mathematica, Maple/SynRac, Reduce/Redlog, QEPCAD-B, and recently in some SMT solvers [13, 6]. QE-CAD solves more than the satisfiability, and is DEXPTIME. By restricting on the satisfiability, *Variant quantifier elimination* [12] reduces to polynomial optimization problems, which are solved by Groebner basis in EXPTIME.

A practical alternative is Interval Constraint Propagation(*ICP*)[2], which is implemented in **iSAT3** [7], **dReal** [10], and **RSolver** [20]. ICP is an over-approximation by an interval arithmetic, and iteratively refines by interval decomposition. Although ICP is not complete for UNSAT detection with unbounded intervals, it is practically often more efficient than algebraic methods.

This paper presents an SMT solver **raSAT** for polynomial constraints over reals. It consists of a simple iterative approximation refinement, **raSAT** *loop*, which adds testing to boost SAT detection to the standard ICP, aiming to accelerate SAT detection. If both the estimation by an interval arithmetic and testing fail, input intervals are refined by decomposition. The features of **raSAT** are,

– **raSAT** loop, which adds testing to boost SAT detection to a standard ICP,
– various interval arithmetic support, e.g., Affine intervals [15, 17, 14],
– sound use of floating point arithmetic, i.e., outward rounding in interval arithmetic [11], and confirmation of an SAT instance by an error-bound guaranteed floating point package **iRRAM**[3].

ICP and **raSAT** loop are robust for large degrees, but the number of boxes (products of intervals) grows exponentially. First, we target on polynomial inequations, and design SAT detection-directed strategies on the choice of a variable to decompose, a box to explore, and a variable to generate multiple test cases. They are based on heuristic measures, *SAT likelihood*, *sensitivity*, and the number of unsolved atomic polynomial constraints. Another strategy of **raSAT** is an incremental avoiding local optimal.

– **Incremental widening**. Starting **raSAT** loop with a smaller initial 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, set a smaller bound and restart.

The combinations are examined on Zankl, Meti-tarski and Keymaera benchmarks from QF_NRA of SMT-LIB, to find clear differences from random choices. We also show two extensions, (1) handling polynomial equations by using the Intermediate Value Theorem, and (2) polynomial constraints over integers (e.g., AProVE benchmark in QF_NIA). These results are also compared with **Z3 4.3**, **dReal-2.15.01** and **iSAT3**.

2 ICP overview and raSAT loop

Our target problems is a nonlinear constraint, especially over real numbers. That over integer numbers will be briefly shown in Section 4.4. We mainly discuss on polynomial inequations, and later in Section 5, we show an extension to cover polynomial equations based on the 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 inequation (API) of the form $p_j(x_1, ..., x_n) > 0$. We denote the set of variables appearing in p_j by $var(p_j)$.

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

Note that φ is equivalent to $\exists x_1 \dots x_n.(\bigwedge_{i=1}^{n} x_i \in I_i) \wedge (\bigwedge_{j=1}^{m} \psi_j(x_1, \cdots, x_n))$. We refer $\bigwedge_i x_i \in I_i$ and $\bigwedge_{j=1}^{m} \psi_j(x_1, \cdots, x_n)$ by I_φ and $\psi(x_1, \cdots, x_n)$, respectively. 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 review Interval Constraint Propagation (ICP)[2], and then introduce **raSAT** (refinement of approximations for SAT) loop [14]. The main difference is that **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 outward rounding [11] on the lower/upper bounds of intervals. In **raSAT**, when testing says SAT, it is confirmed with the error bound guaranteed package **iRRAM**.

2.1 ICP overview

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

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

1: $S \leftarrow \{B_0\}$ ▷ 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 ψ by using IA **then**
9: **return** SAT
10: **else** ▷ IA cannot conclude the constraint $\Longrightarrow$ *Refinement* Step
11: $\{B_1, B_2\} \leftarrow split(B')$ ▷ 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

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 satisfiability of equations. In contrast, although the number of boxes increase exponentially, ICP always detects 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. 1. The left shows a kissing case $x^2+y^2 < 2^2 \wedge (x-4)^2+(y-3)^2 < 3^2$ such that $\overline{\mathbb{S}(-x^2-y^2+2^2>0)} \cap \overline{\mathbb{S}(-(x-4)^2-(y-3)^2+3^2>0)} = \{(x,y) \mid (1.6, 1.2)\}$. 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.

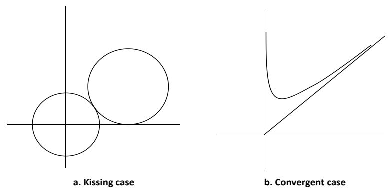


Fig. 1: Limitations for detection of UNSAT

2.2 raSAT loop

ICP is extended to **raSAT** loop, which adds testing to boost SAT detection [14].

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
    choose $(x_i \in I_{ik}) \in \pi$ such that $\forall k_1 \in \{1, \cdots, i_k\} I_{ik} \subseteq I_{ik_1}$
    $\{I_1, I_2\} \leftarrow split(I_{ik})$ ▷ 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

Let $\psi = \bigwedge_{j=1}^{m} g_j(x_1, \cdots, x_n) > 0$ be a target constraint. Algorithm 2 displays **raSAT** loop. Its implementation **raSAT** adapts various IAs, and applies two main heuristics (the use of Affine intervals enable us the latter).

- Incremental widening and incremental deepening (Section 3.1).
- Heuristic measures *SAT-likelihood* and *sensitivity*, for selection of a variable at line 11 (of Algorithm 2) and a box at line 13 (Section 3.2).

2.3 Interval Arithmetic

Various Affine intervals [5] are prepared in **raSAT**, adding to Classical Interval (CI) [16]. Although precision is incomparable, Affine interval partially preserves 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) [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 $\epsilon|\epsilon| = \epsilon + (-\frac{1}{4}, \frac{1}{4})$ [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 $(-8, 4.5)$.

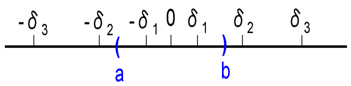
3 SAT directed Strategies of raSAT

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 I_φ becomes the disjunction of existential formulae corresponding 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 by **Z3** where UNSAT either is quickly detected or leads to timeout. For SAT detection, the keys will be a strategic control not to fall into local optimal and a strategy to choose most likely decomposition/boxes.

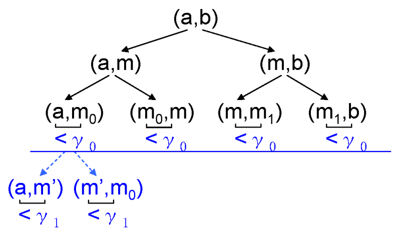
3.1 Incremental search

Two incremental search strategies are prepared in **raSAT**, (1) *incremental widening*, 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 widening Given $0 < \delta_0 < \delta_1 < \cdots$, *incremental widening* starts with $I_\varphi^0 = \exists x_1 \in I_1 \cap (-\delta_0, \delta_0) \cdots x_n \in I_n \cap (-\delta_0, \delta_0). \bigwedge_{j=1}^{m} g_j > 0$, and if it stays UNSAT, then enlarge the intervals as $I_\varphi^1 = \exists x_1 \in I_1 \cap (-\delta_1, \delta_1) \cdots x_n \in I_n \cap (-\delta_1, \delta_1). \bigwedge_{j=1}^{m} g_j > 0$. This continues until either SAT, timeout, or a given bound of repetition (Fig. 2 (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.



(a) Incremental widening



(b) Incremental Deepening

Fig. 2: 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 $\gamma = \gamma_0$. If neither SAT nor UNSAT is detected, **raSAT** restarts with the threshold γ_1. This continues until either SAT, timeout, or a given bound of repetition (Fig. 2 (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 *API*, and (2) then choose a most likely influential variable in the selected API. For *most likely influential* measures, we apply the *SAT-likelihood* on APIs and the *sensitivity* on variables, respectively. Note that the latter measure is defined only by Affine intervals.

In line 3 of Algorithm 2, an IA will estimate the ranges of polynomials in a box B. We denote the estimated range of g_j by $range(g_j, B)$. If an IA is an Affine interval, we assume that the estimated range of g_j 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 $range(g_j, B)$. We define as follows.

- The *SAT-likelihood* of an API $g_j > 0$ is $|I \cap (0,\infty)|/|I|$ for $I = range(g_j, B)$.
- The *sensitivity* of a variable x_i in an API $g_j > 0$ is $max(|c_i|, |d_i|)$.

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.

SAT-likelihood intends to estimate an API 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 B by the least SAT-likelihood of APIs. Since the SAT-likelihood of each box is computed when it is created by the decomposition, **raSAT** simply compares newly decomposed boxes with the previous 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 (concluded by either IA or testing) 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 $x_{16} \in [0, 10]$, $0.25\epsilon_2 - 0.25\epsilon_8 - 0.25\epsilon_{10} + 49.5\epsilon_{15} - 0.25\epsilon_{16} + 0.75\epsilon_{+-} + 49.25$ is the estimation of g by AF_2. The coefficient of ϵ_2 is 0.25, which is positive. Thus, the value of g is likely to increase if x_2 increases. 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 others, $x_8 = 0, x_{10} = 0, x_{15} = 10, x_{16} = 0$, which lead $g = 100 > 0$.

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 outward rounding in the IAs. Various combinations of strategies of **raSAT** (in Section 3) and random strategies are compared on *Zankl* and *Meti-Tarski* in NRA and *AProVE* in NIA from SMT-LIB. The best combination of choices are

1. an test-UNSAT API (API that cannot be satisfied by any test cases in testing) 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[4] benchmark on non-linear real number arithmetic (QF_NRA) has Meti-Tarski, Keymaera, Kissing, Hong, and Zankl families, of which brief statistics are 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 inequatity problems 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 inequality problems among 7713, taken from elementary physics. They are mostly small problems, up to 8 variables (mostly up to 5 variables), and up to 20 APIs.

Keymaera contains 68 inequality problems among 680.

Kissing has 45 problems, all of which contains equations (mostly a single equation).

Hong has 20 inequality problems among 20, tuned for QE-CAD.

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 generate 2 test cases for each, and one random test case is for each of the rest of variables.
– For interval decomposition, **raSAT** splits exactly at the middle.

4.2 Experiments on Strategy Combinations

Selection of Incremental Strategies We run options for incremental widening and deepening on Zankl family in order to select the best combination. Though the results are preliminary, from now on, we choose

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		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: Different options in incremental widening and deepening

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

Table 2 shows the results of the strategy combinations (below) on QD_NRA/Zankl and Meti-Tarski. The timeout is set to 500s, and time shows the total of successful cases (either SAT or UNSAT).

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

where we randomly select points in the box B (Algorithm 2) as test cases in testing. 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.

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: Strategy combinations of **raSAT** on QF_NRA/Zankl and Meti-Tarski

Experiments with test case generation using variable sensitivity
We also examine the effectiveness of the sensitivity in test case generation, which

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

is referred by (11). Table 3 presents the results on QF_NRA/Zankl and Meti-tarski, which show the improvement by the strategy (11) in SAT detection.

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 the variable 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 bounded interval $[-1000, 1000]$ (otherwise, it often causes segmentation fault). Thus, UN-SAT 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** concludes 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 outperforms on SAT detection of 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),
- *matrix-5-all-01* (132 variables, 276 APIs, and max length of an API is 47).

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 polynomial inequality

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

The **raSAT** loop is easily modified to QF_NIA (nonlinear arithmetic over integer numbers) 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 integer numbers, 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 inequality and 1979 equality problems. 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.

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

Table 5: Comparison on NIA/AProVE

5 Extension for Equality Handling

For a polynomial 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 equations 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 equations, we assume a box that makes $\bigwedge_{j=1}^{m} \psi_j(x_1, \cdots, x_n)$ IA-valid.

5.1 Single Equation

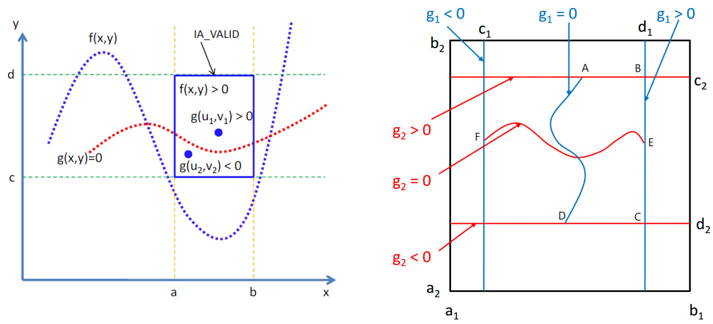
For solving polynomial constraints with a single equation ($g = 0$), we can apply in a simple way. That is, finding 2 test cases with $g > 0$ and $g < 0$ implies $g = 0$ somewhere in between.

Lemma 1. *For* $\varphi = \exists x_1 \in I_1 \cdots x_n \in I_n(\bigwedge_{j=1}^{m} g_j > 0 \wedge g = 0)$*. Suppose decomposition creates a box* $B = (l_1, h_1) \times \cdots \times (l_n, h_n)$ *such that* $(l_i, h_i) \subseteq I_i$ *and* $\bigwedge_{j=1}^{m} g_j > 0$ *is IA-VALID in the box. For* $(l_g, h_g) = range(g, B)$*,*

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

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

Example 4. Let $\varphi = f(x,y) > 0 \land g(x,y) = 0$. Suppose we find a box $B = (a,b) \times (c,d)$ such that $f(x,y) > 0$ is VALID in B. (Fig. 3a). 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.



(a) Single equation

(b) Multiple equations

Fig. 3: Example on solving equations using the Intermediate Value Theorem

raSAT first tries to find a box (by the decomposition) such that $\bigwedge_{j=1}^{m} g_j > 0$ 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.

5.2 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 $B = (l_1,h_1) \times \cdots (l_n,h_n)$. For $V = \bigcup_{j=1}^{m} var(g_j)$ and $V' = \{x_{j_1}, \cdots x_{j_k}\} \subseteq V$, we denote $\{(r_1,\cdots,r_n) \in B \mid r_i = l_i \text{ for } i = j_1,...,j_k\}$ and $\{(r_1,\cdots,r_n) \in B \mid r_i = h_i \text{ for } i = j_1,...,j_k\}$ by $B\downarrow_{V'}$ and $B\uparrow_{V'}$, respectively.

Definition 2. *A sequence $(V_1,\cdots,V_m)$ of subsets of V is a* check basis *of $(g_1,\cdots,g_m)$ on a box B, 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* $B \uparrow_{V_j}$ *and* $g_j < 0$ *on* $B \downarrow_{V_j}$*, or* $g_j < 0$ *on* $B \uparrow_{V_j}$ *and* $g_j > 0$ *on* $B \downarrow_{V_j}$*.*

Lemma 2. *For a polynomial constraint containing multiple equations*

$$\varphi = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} \psi_j(x_1, \cdots, x_n) \wedge \bigwedge_{j=1}^{m'} g_j(x_1, \cdots, x_n) = 0$$

and a box $B \subseteq I_1 \times \cdots \times I_n$*, assume that*

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

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

The idea is, from the Intermediate Value Theorem, each g_j has a $n - |V_j|$ dimensional surface of null points of g_j between $B \uparrow_{V_j}$ and $B \downarrow_{V_j}$. 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. 3b). 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** for polynomial constraints with equations is implemented in a naive way. 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 problems with equations from QF_NRA/Zankl and QF_NRA/Meti-tarski are summarized in Table 6. We hope that the sensitivity will give effective strategies.

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 Related Work

There are many techniques appearing in various SMT solvers.

QE-CAD RAHD [19] and Z3 4.3 (nlsat in [13]) include QE-CAD.

Virtual substitution (VS) SMT-RAT toolbox [6] combines VS, incremental DPLL, and eager theory propagation. Z3 3.1 combines VS, ICP, and linearization.

Bit-blasting UCLID [3] and MiniSmt [22] give a bound on the number of bits to encode integers and rationals, respectively.

Linearization CORD [8] linearizes multiplications of reals by CORDIC encoding. Linearization suffers from the increase of the polynomial degrees.

ICP-based SMT solvers are iSAT3 and dReal, adding to **raSAT**.

iSAT3 has tighter integration between DPLL procedure [18] and ICP. Fresh variables are introduced to decompose a polynomial to atomic representations, and each of them is assigned to an atomic proposition. A data structure is prepared to store intervals such that they correspond to the decision level one in DPLL. Its unit propagation is strengthened by combining with eager theory propagation. In a clause, if all except one literals are falsified, the remaining literal causes unit propagation and it becomes a candidate when next decomposition occurs. Note that **iSAT3** uses only CI as an IA.

dReal has different judgments on SAT, called δ-SAT, which allows the deviation of the width δ. Thus, δ-SAT does not imply really SAT. This is the reason why dReal quite often concludes SAT (actually δ-SAT) for UNSAT problems in SMT-LIB benchmarks. With the weakening of SAT to δ-SAT, it obtains the completeness of δ-SAT and δ-UNSAT [9]. Note that **dReal** uses only CI as an IA, and lazy theory propagation as **raSAT**.

7 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 of SMT-LIB, we found two heuristic measures, *SAT-likelihood* and *sensitivity*, which lead an effective strategy for SAT detection. Currently, **raSAT** is in proto-type status, and lots of future work remain.

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, but current implementation has no effective strategies on finding a box on which all inequations are IA-valid and a decomposition V_j's. We hope to find one based on the sensitivity. We also would like to additionally apply Groebner basis, to overcome the limitation that the number of variables is greater-than-equal to that of equations.

Further strategy refinement. Currently, raSAT uses information only from IA. We hope to refine strategies such that previous IA and testing results mutually guide. For instance, a box decomposition strategy can be a target.

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. ICFP, ACM (2012)
[2] Benhamou, F., Granvilliers, L.: Continuous and Interval Constraints. Handbook of Constraint Programming (2006), pp.571–604
[3] Bryant, R.E., Kroening, D., Ouaknine, J., Seshia, S.A., Strichman, O., Brady, B.: Deciding bit-vector arithmetic with abstraction. TACAS 2007. LNCS 4424, pp. 358–372 (2007)
[4] Collins, G.: Quantifier elimination by cylindrical algebraic decomposition - twenty years of progress. Quantifier Elimination and Cylindrical Algebraic Decomposition, pp. 8–23 (1998)
[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. SAT 2012, vol. 7317, pp. 442–448
[7] Frnzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. JSAT 1, 209–236 (2007)
[8] Ganai, M., Ivancic, F.: Efficient decision procedure for non-linear arithmetic constraints using cordic. FMCAD 2009. pp. 61–68 (2009).
[9] Gao, S., Avigad, J., Clarke, E.M.: Delta-complete decision procedures for satisfiability over the reals. IJCAR'12, pp. 286–300 (2012)
[10] Gao, S., Kong, S., Clarke, E.: dReal: An SMT solver for nonlinear theories over the reals. CADE-24, LNCS 7898, pp. 208–214 (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)
[13] Jovanovi, D., de Moura, L.: Solving non-linear arithmetic. Automated Reasoning, LNCS 7364, pp. 339–354 (2012)
[14] Khanh, T.V., Ogawa, M.: SMT for polynomial constraints on real numbers. TAPAS 2012, ENTCS 289(0), pp.27 – 40 (2012)
[15] Messine, F.: Extensions of affine arithmetic: Application to unconstrained global optimization, Journal of Universal Computer Science 8(11), pp.992–1015 (2002)
[16] Moore, R.: Interval analysis. Prentice-Hall (1966)
[17] Ngoc, D.T.B., Ogawa, M.: Overflow and roundoff error analysis via model checking. SEFM 2009. pp. 105–114 (2009)
[18] Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Abstract dpll and abstract dpll modulo theories. LPAR04, LNAI 3452. pp. 36–50 (2005)
[19] Passmore, G.O., Jackson, P.B.: Combined decision techniques for the existential theory of the reals. CALCULEMUS. LNCS 5625, pp. 122–137 (2009)
[20] Ratschan, S.: Efficient solving of quantified inequality constraints over the real numbers. TOCL 7(4), 723–748 (2006)
[21] Tarski, A.: A decision method for elementary algebra and geometry. Quantifier Elimination and Cylindrical Algebraic Decomposition, pp. 24–84 (1998)
[22] Zankl, H., Middeldorp, A.: Satisfiability of non-linear (ir)rational arithmetic. LPAR, LNCS 6355, pp. 481–500 (2010)