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 the tool **raSAT**, an SMT solver for polynomial constraints. It consists of a simple iterative approximation refinement, called **raSAT** loop, which is an extension of the standard ICP (Interval Constraint Propagation) with Testing to boost SAT detection and the use of the Intermediate Value theorem to show satisfiability of equations over the reals.
ICP is robust for large degrees, but slow convergence still remains as the main hurdle. We design strategies, namely *SAT likelihood* and *sensitivity*, for guiding the choice of a variable to decompose and a box to explore.

1 Introduction

Polynomial constraints are utilized to encode a wide range of problems in theorem proving and formal verification [17, 15, 5, 22, 21]. Solving such constraints over the real numbers was shown to be decidable by Alfred Tarski [23]. He provided a general decision procedure with non-elementary complexity, which has been consequently much improved. Quantifier elimination by cylindrical algebraic decomposition (CAD) [4] is the most notable improvement which is implemented in Mathematica, Maple/SynRac, Reduce/Redlog, QEPCAD-B, and recently in some SMT solvers [13, 6]. The CAD procedure solves more than the satisfiability with the complexity of doubly-exponential. By restricting on the satisfiability, *Variant quantifier elimination* [12] reduces to polynomial optimization problems, which are solved by Groebner basis in exponential time.

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

This paper presents an SMT solver **raSAT** for polynomial constraints over reals. It consists of a simple iterative approximation refinement, called **raSAT** *loop*, which adds Testing to boost SAT detection to a standard ICP. Two approximation schemes consist of Interval Arithmetic (over-approximation) and Testing (under-approximation), to boost SAT detection. If both the estimation by a Interval Arithmetic and Testing fail, input intervals are refined by decompositions. The features of **raSAT** are,

- **raSAT** loop, which adds Testing to boost SAT detection to a standard ICP,
- various Interval Arithmetics support, e.g., Affine intervals [16, 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 inequalities, 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. 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 equality 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**.

raSAT also applies incremental search not to fall into local optimals.

- **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.

2 raSAT loop

Our target problem is solving polynomial constraints which are defined as follows.

Definition 1. *A polynomial 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 $p_j(x_1, ..., x_n) \diamond 0$ with $p_j(x_1, ..., x_n)$ is a polynomial and $\diamond \in \{<, \leq, >, \geq, =\}$. 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 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 $\mathbb{S}(\psi(x_1, \cdots, x_n)) = \{(r_1, \cdots, r_n) \in \mathbb{R}^n \mid \psi(r_1. \cdots, r_n) \text{ holds}\}$. Given a box $B =$

This section is going to present **raSAT** (refinement of approximations for SAT) loop [14] which is an extension of Interval Constraint Propagation (ICP)[2]. The main difference is that **raSAT** loop implements Testing after the estimation by Interval Arithmetic (IA) to boost SAT detection. In addition, the Intermediate Value Theorem is employed to show satisfiability of equations.

2.1 raSAT loop

Let $\psi = \psi_1 \wedge \psi_2$ be a conjunctive polynomial constraint where $\psi_1 = \bigwedge_{j=1}^{l} g_j(x_1, \cdots, x_n) \diamond 0, \diamond \in \{<, \leq, >, \geq\}$ and $\psi_2 = \bigwedge_{j=l+1}^{m} g_j(x_1, \cdots, x_n) = 0$. Algorithm 1 displays **raSAT** loop where

- Testing and IA of **raSAT** are discussed in [14]. In the current implementation, we support solving constraints over the integers by generating integers as test cases.
- The function $IVT(\psi_2, B)$ intuitively returns true if the Intermediate Value Theorem can conclude SAT of ψ_2 inside the box B, and returns false otherwise.

The idea of $IVT(\psi_2, B)$ is from the following two lemmas which corresponds to cases of single and multiple equations.

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.

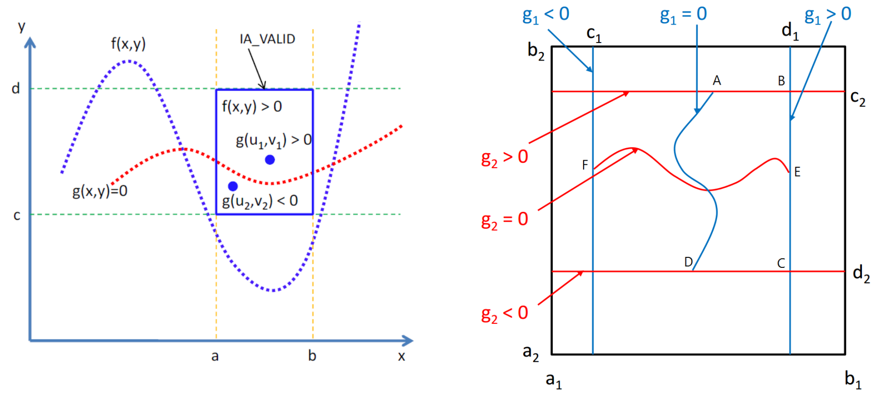
Lemma 1. *For $\psi = \psi_1 \wedge \psi_2$ and a box $B = [l_1, h_1] \times \cdots \times [l_n, h_n]$ where $\psi_1 = \bigwedge_{j=1}^{m-1} g_j(x_1, \cdots, x_n) \diamond 0$, $\diamond \in \{<, \leq, >, \geq\}$ and $\psi_2 = g_m(x_1, \cdots, x_n) = 0$, ψ is SAT if the following two conditions hold.*

(i) The inequalities ψ_1 is satisfied in the box by IA.
(ii) There are two instances $\boldsymbol{t}, \boldsymbol{t'} \in B$ such that $g(\boldsymbol{t}) > 0$ and $g(\boldsymbol{t'}) < 0$.

Example 1. Let $\varphi = f(x, y) > 0 \wedge g(x, y) = 0$ be a polynomial constraint. Suppose we find a box $B = (a, b) \times (c, d)$ such that $f(x, y) > 0$ is true for any points in B (concluded by IA). (Fig. 1a). 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.

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

while $\varPi$ 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 $\varPi$
  $B \leftarrow$ 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**
    $\varPi \leftarrow \varPi \wedge \neg(\bigwedge_{i=1}^{n} \bigwedge_{k=1}^{i_k} x_i \in I_{ik})$
  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 if B satisfies ψ_1 by IA and $IVT(\psi_2, B)$ **then**
    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
    $\varPi \leftarrow \varPi \wedge (x_i \in I_{ik} \leftrightarrow (x_i \in I_1 \vee x_i \in I_2)) \wedge \neg(x_i \in I_1 \wedge x_i \in I_2)$
  end if
end while
return UNSAT



(a) Single equality

(b) Multiple equalities

Fig. 1: Example on solving equality using the Intermediate Value Theorem

Multiple Equations The idea of using the Intermediate Value Theorem is generalized for solving multiple equations. Consider m equalities ($m \geq 1$): $\bigwedge_{j=1}^{m} g_j = 0$ and an box $B = [l_1, h_1] \times \cdots [l_n, h_n]$. For $V = \bigcup_{j=1}^{m} var(g_j)$ and $V' = \{x_{i_1}, \cdots x_{i_k}\} \subseteq V$, we denote $\{(r_1, \cdots, r_n) \in B \mid r_i = l_i \text{ for } i = i_1, ..., i_k\}$ and $\{(r_1, \cdots, r_n) \in B \mid r_i = h_i \text{ for } i = i_1, ..., i_k\}$ 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. $\psi = \psi_1 \wedge \psi_2$ *and a box* $B = [l_1, h_1] \times \cdots \times [l_n, h_n]$ *where* $\psi_1 = \bigwedge_{j=1}^{l} g_j(x_1, \cdots, x_n) \diamond 0, \diamond \in \{<, \leq, >, \geq\}$ *and* $\psi_2 = \bigwedge_{j=l+1}^{m} g_j(x_1, \cdots, x_n) = 0$*,* ψ *is SAT if the following two conditions hold.*

1. $\bigwedge_{j=1}^{l+1} \psi_j(x_1, \cdots, x_n)$ *is satisfied on* B *by IA, and*
2. *there is a* check basis $(V_{l+1}, \cdots, V_m)$ *of* $(g_{l+1}, \cdots, g_m)$ *on* B*.*

The idea is, from the Intermediate Value Theorem, with $l + 1 \leq j \leq m$, each g_j has a $n - |V_j|$ dimension 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 $n - \sum_{j=l+1}^{m} |V_j|$. Thus, this method has a limitation that the number of variables must be greater than or equal to the number of equations.

Example 2. 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. 1b). 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.

2.2 SAT directed Strategies of raSAT

As mentioned, slow convergence is an obstacle for ICP-based solvers. The strategies for selecting a variable whose interval will be decomposed and selecting a box to explore are the keys in quick detection of satisfiability.

In **raSAT**, selecting a variable is done in the following two steps. (1) First select the most likely influential *API*, and (2) then choose the 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.

Definition 3. *Let $g \diamond 0$ be an API and B is a box of intervals where $\diamond \in \{<, \leq, >, \geq, =\}$. Then the* SAT-likelihood *of $g > 0$ in B is defined as*

$$\frac{|I \cap \mathbb{S}(x \diamond 0)|}{|I|}$$

for $I = range(g, B)$.

If IA is an Affine Interval whose details can be found in [14], we assume that the estimated range of g is of the form $[c_1, d_1]\epsilon_1 + \cdots + [c_n, d_n]\epsilon_n$. By instantiating $[-1, 1]$ to ϵ_i, we obtain $range(g, B)$. Furthermore, from this form, we can define sensitivity of variables.

Definition 4. *Let $[c_1, d_1]\epsilon_1 + \cdots + [c_n, d_n]\epsilon_n$ be an affine form of a polynomial g, the* sensitivity *of a variable x_i in g is defined as $max(|c_i|, |d_i|)$.*

While SAT-likelihood intends to estimate how likely it is for an API to be SAT; *sensitivity* of a variable intends to estimate how a variable is influential to the value of an polynomials. At the time of decomposition, **raSAT** selects a variable by:

- first selecting the API with the least value of SAT-likelihood, and then
- selecting the variable with the largest value of sensitivity in the polynomial of the selected API.

After a box is decomposed into two smaller ones, **raSAT** selects the most likely influential one to explore. The criteria for influence is likewise estimated by *SAT-likelihood*.

Definition 5. *For a polynomial constraint $\psi = \bigwedge_{j=1}^{m} \psi_j(x_1, \cdots, x_n)$, the* SAT-likelihood *of a box B by the least SAT-likelihood among those of $\psi_j(x_1, \cdots, x_n)$.*

Since we desire to detect SAT instances expeditiously (UNSAT needs to check both decomposed boxes), the box with higher SAT-likelihood will be explored first.

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 (for simplicity in explanation, suppose the API is of the form $g > 0$, the other cases proceed similarly). 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 $g = -x_{15} * x_8 + x_{15} * x_2 - x_{10} * x_{16}$ and consider an API $g > 0$. 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 estimated by $AF2$ [14] for g. 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 as a result we have $g = 100$ which satisfies $g > 0$.

Other Strategies Because sensitivity is only used when the intervals of variables are bounded (required by affine intervals), **raSAT** incrementally widens the intervals and deepens the search. Note that both ICP and **raSAT** are suffered from roundoff errors of the floating arithmetic. To guarantee the soundness, IA adopts outward rounding [11] 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**.

Incremental widening Given a polynomial constraint ψ and a sequence of real numbers $\delta_0, \delta_1, \cdots, \delta_k$ where $0 < \delta_0 < \cdots < \delta_n$, $\delta_k = \infty$, *incremental widening* starts with a box $B_0 = [-\delta_0, \delta_0]^n$, and if ψ is UNSAT, then enlarge the intervals as $B_1 = [-\delta_1, \delta_1]^n$. This continues until either SAT, timeout, or UNSAT with $B_k = [-\infty, \infty]^n$.

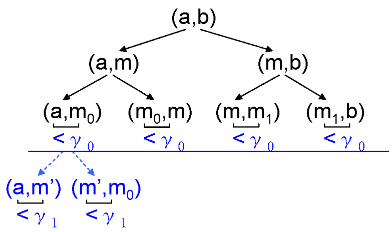


Fig. 2: Incremental deepening

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 repeatation (Fig. 2).

3 Results

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 Interval

Arithmetics. Various combinations of strategies of **raSAT** (in Section 2.2) 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. 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 4. The experiments are with Intel Xeon E5-2680v2 2.80GHz and 4 GB of RAM.

3.1 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 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 variables (mostly up to 5 variables), and up to 20 APIs.
- **Keymaera** contains 68 inequalities among 680.
- **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 generate 2 test cases for each of these, and single random test data is generated for each of the rest of variables.
- For interval decomposition, **raSAT** splits at exactly the middle.

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

3.2 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		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

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):	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 1) 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.

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.

3.3 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

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: Combinations 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

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** 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 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), and

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

– *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.

3.4 Polynomial Constraints Over Integers

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 **??** 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 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.

4 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 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 equality case. Note that before solving on equality, we assume a box that makes $\bigwedge_{j=1}^{m} \psi_j(x_1, \cdots, x_n)$ IA-valid.

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

Current **raSAT** implementation on equalities 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 inequalities IA-valid. Preliminary experiments on equalities from QF_NRA/Zankl and QF_NRA/Meti-tarski are summarized in Table 6. We hope that the sensitivity will give their effective strategies.

5 Related Work

Non-linear constraints are still under ivestigation, and many techniques appear in 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 [24] 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 It 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 special 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 interval arithmetic.

dReal It has different judgements 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-comp benchmarks. With the weaking of SAT to δ-SAT, it obtains the completeness of δ-SAT and δ-UNSAT [9]. Note that **dReal** uses only CI as an interval arithmetic, and lazy theory propagation as **raSAT**.

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 4 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 Interval Arithmetic .We are planning to refine strategies such that previous IA and Testing 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] Benhamou, F., Granvilliers, L.: Continuous and Interval Constraints. In: F. Rossi, P.v.B., Walsh, T. (eds.) Handbook of Constraint Programming, pp. 571–604. Elsevier (2006)

[3] 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)

[4] 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)

[5] 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)

[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] 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)