raSAT: SMT for Polynomial Inequality

To Van Khanh[1] and Mizuhito Ogawa[2]

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

Abstract. This paper presents an iterative approximation refinement, called **raSAT** loop, which solves a polynomial inequality on real numbers. The approximation scheme consists of interval arithmetic (over-approximation, aiming to decide UNSAT) and testing (under-approximation, aiming to decide SAT). If both of them fail to decide, input intervals are refined by decomposition.
The **raSAT** loop is implemented as an SMT **raSAT** with miniSAT 2.2 as a backend SAT solver. Experiments are performed on QF_NRA benchmarks from SMT-LIB and simple benchmarks intending to estimate effects of input measures (i.e., degrees, number of variables, and number of polynomials). They show that **raSAT** is comparable to Z3 4.3, and sometimes outperforms, especially with high degree of polynomials.

1 Introduction

Polynomial constraint solving is to find an instance that satisfies given polynomial inequality/equality. For instance, $\exists xy.-y^2+(x^2-1)y-1>0 \wedge -x^2-y^2+4>0$ is such an example. This is an easy formula, but proving its satisfiability and showing a satisfiable instance (e.g., $x=1.8$, $y=0.9$) are not so easy.

Many applications are reduced to solving polynomial constraints, such as

- **Locating roundoff and overflow errors**, which is our motivation [1, 2]. DSP decorders in practice are defined by reference algorithms in C using floating point arithmetic. In embedded systems, often it is replaced with fixed point arithmetic, which may cause visible noises.
- **Automatic termination proving**, e.g., TTT2 [3], Aprove [4]. is reduced to finding a suitable termination ordering [3].
- **Loop invariant generation**. Farkas's lemma is a popular approach in linear loop invariant generation [4], and is reduced to degree 2 polynomials. Nonlinear loop invariant [5] requires more complex polynomials.
- **Hybrid system**. SMT for QF_NRA are often used as backend engines [6].
- **Mechanical contrnol design**. PID control is simple but widely used, and designing parameters is reduced to polynomial constraints [7].

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

We focus on polynomial inequality, which simplifies problems a lot still keeping applications. This view is shared with δ-completeness [8, 9].

- Inequality $a < b$ on real numbers is computable whereas equality $a = b$ is not. ($a = b$ can be decided for algebraic numbers by ideal computation.)
- Inequality can be decided by enough fine approximation.
- Since rational numbers are dense in real numbers, inequality on real numbers can be reduced to that on rational numbers.

This paper presents an iterative approximation refinement, called **raSAT** loop, which solves polynomial inequality on real numbers. The scheme consists of interval arithmetic (over-approximation, aiming to decide UNSAT) and testing (under-approximation, aiming to decide SAT). If neither decides, input intervals are refined by decomposition. Elementary extensions for handling *greater-than-equal* and equality are also discussed.

raSAT loop is implemented as an SMT **raSAT** with a backend SAT solver miniSAT 2.2 and backend theories in Ocaml. **raSAT** loop is not only a simple framework, but also allows us to design mutually refining strategies, e.g., the result of interval arithmetic refines both test data generation and next refinements, and the result of testing refines next refinements.

There are immediate complexity measures on inputs, *the degree of polynomials*, *the number of variables*, and *the length of polynomial inequalities*. Since **raSAT** loop depends on interval arithmetic and testing in floating point arithmetic, higher degrees of polynomials do not affect much on efficiency. However, refinement steps (interval decompositions) easily introduce exponential blowup of the number of boxes, and the number of variables is a dominant factor.

Experiments (Section 7 are performed mainly comparison with Z3 4.3[5], HySAT[6], and dReal[7] on SMT-lib benchamrks[8] in QF_NRA. The result is encouraging. Although our proto-type **raSAT** implementation includes only **raSAT** loop, **raSAT** shows comparable (and sometimes outperfoming) results.

Related Work

Solving polynomial constraints on real numbers is decidable [10], though that on integers is undecidable (*Hilbert's 10th problem*). Quantifier elimination by cylindrical algebraic decomposition (QE-CAD) [11] is a well known technique, and implemented in Mathematica, Maple/SynRac, Reduce/Redlog, QEPCAD-B, and recently in some SMTs, e.g., nlSAT [12]. It is DEXPTIME wrt the number of variables. In practice, it works fine up to 5-6 variables with lower degrees, but solving 8 variables and degree 10 may be the current limit. Virtual substitution (VS) [13] focusing on small degree polynomials (especially degree 2) has better performance, but it is still EXPTIME.

[5] `http://z3.codeplex.com`
[6] `http://hysat.informatik.uni-oldenburg.de/`
[7] `http://dreal.cs.cmu.edu/`
[8] `http://www.smtlib.org/`

SMT (SAT modulo theories) separates the case analysis and feasibility in a background theory, and many implementations are available. Presburger arithmetic (linear constraints) is one of the most popular background theory, and polynomial constraints (non-linear constraints) become evolving. Their approaches can be classified as follows. The community of SMT starts to interact with that of symbolic computation, such as the first and the second techniques.

QE-CAD. RAHD [14] and Z3 4.3 (which is referred as nlsat in [12]) include QE-CAD. QE-CAD is precise and detects beyond SAT instances (e.g., SAT regions), scalability is still challenging, since it is DEXPTIME.

Virtual substitution (VS). SMT-RAT toolbox [15][16] combines VS, incremental DPLL, and eager theory propagation. Z3 (version 3.1), the winner of QF_NRA in SMT competition 2011, combines VS, ICP, and linearization.

Bit-blasting. Bid-blasting in bounded bit width is often used in SMTs for QF_NIA. UCLID [17] reduces the number of bits (i.e., narrowing bounds for SAT instances) as an under-approximation, and removes clauses as an over-approximation. They refine each other, which shares a similar sprit with **raSAT** loop. MiniSmt [18], the winner of QF_NRA in SMT competition 2010, applies it for rational numbers with symbolic representations for prefixed algebraic numbers. MiniSmt can show SAT quickly with small degree polynomials, but due to the bounded bit encoding, it cannot conclude UNSAT. Bit-blasting also suffers a lot when the degree of polynomials increases.

Linearization. Linearization of polynomials is often used over integers, such as Barcelogic [19], which substitutes all possible integers in a given-bound to an argument of a multiplication. Then, multiplications are reduced to an exhaustive search on linear constraints. CORD [20] uses another linearization, called CORDIC (COrdinate Rotation DIgital Computer) for real numbers. Both Barcelogic and CORD apply Yices for solving linear constraints. Linearization also suffers a lot when the degree of polynomials increases.

Interval constraint propagation (ICP). ICP applies a *branch-and-bound* approach with interval arithmetic as an over-approximation, which can conclude UNSAT. If a box is refined enough, it may satisfy a constraint everywhere in the box, which conclued SAT. RSOLVER [21], HySAT [22], and dReal [23] are such examples. As an acceleration of excluding unsatisfiable boxes, RSOLVER uses a pruning algorithm, whereas HySAT applies eager theory propagation with a tight interaction with a SAT solver. **raSAT** is also in this category. Additional features of **raSAT** are, (1) the use of Affine intervals [24] adding to a classical interval used in RSOLVER and HySAT, and (2) testing as an under-approximation to acceralate SAT instance detection and mutually refining strategies.

2 Over and Under Approximation Theories and Their Refinement

2.1 Approximation Theory

We start with a general framework, and assume that a target constraint is an existential bounded quantification $F = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_j \psi_j(x_1, \cdots, x_n)$, where $\psi_j(x_1, \cdots, x_n)$ is an atomic formula.

F is equivalnet to $\exists x_1 \ldots x_n.(\bigwedge_i x_i \in I_i) \wedge (\bigwedge_j \psi_j(x_1, \cdots, x_n))$, and we refer $\bigwedge_i x_i \in I_i$ by $\mathcal{I}$ and $\bigwedge_j \psi_j(x_1, \cdots, x_n)$ by $\mathcal{P}$, respectively. Initially, $\mathcal{I}$ has a form of the conjunction $\bigwedge_i x_i \in I_i$, and later by refinement (Section refsubsec:rasatloop), $x_i \in I_i$ is decomposed into a clause $\bigvee_j x_i \in I_{i_j}$ and $\mathcal{I}$ becomes a CNF.

As an SMT (SAT modulo theory) problem, boolean variables are assigned to each $x_i \in I_{i_j}$ in $\mathcal{I}$, and truth assignments is produced by a SAT solver, which are proved or disproved by a background theory T whether it satisfies $\mathcal{P}$.

As notational convention, m (the lower case) denotes an instance (m is aimed at variable assignments) of $x_i \in I_{i_j}$'s, and M (the upper case) denotes a (full) truth assignment on $x_i \in I_{i_j}$'s. We write $m \in M$ when an instance $m = (x_1, \cdots, x_m)$ satisfy all $x_i \in I_{i_j}$ that are assigned true.

We adopt *very lazy theory learning* [25], and a backend theory T is applied only for a full truth assignment M. We regard M as a conjunction $\bigwedge_i x_i \in I_{i_j}$.

- If an instance m of variables appearing in $\mathcal{P}$ satisfies F, we denote $m \models_T F$.
- If m satisfies F for each instance $m \in M$, we denote $M \models_T F$.

Definition 1. *Let $F = \exists x_1 \in I_1 \cdots x_n \in I_n.\psi(x_1, \cdots, x_n)$. For a truth assignment on M, F is*

- *T-valid if $M \models_T \psi(x_1, \cdots, x_n)$,*
- *T-satisfiable (T-SAT) if $m \models_T \psi(x_1, \cdots, x_n)$ for some $m \in M$, and*
- *T-unsatisfiable (T-UNSAT) if $M \models_T \neg\psi(x_1, \cdots, x_n)$.*

If T is clear from the context, we simply say valid, satisfiable, and unsatisfiable.

Later in Section 4, we will instantiate an interval and an atomic polynomial inequality to I_i's and ψ_j, respectively. Then, Fig. 1 illustrates Definition 1.

Definition 2. *Let $T, O.T, U.T$ be theories.*

- *$O.T$ is an* over-approximation theory *(of T) if $O.T$-UNSAT implies T-UNSAT, and*
- *$U.T$ is an* under-approximation theory *(of T) if $U.T$-SAT implies T-SAT.*

We further assume that $O.T$-valid implies T-valid.

Note that $O.T$-valid can be regarded as $U.T$, since $O.T$-valid implies T-valid, thus T-SAT. Later in Section 4, we will instantiate interval arithmetic as $O.T$. Adding to $O.T$-valid, we introduce testing as $U.T$ to accelerate SAT detection.

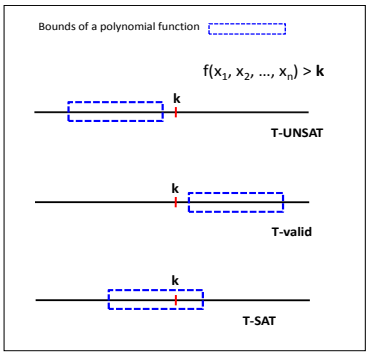

Fig. 1. Results of a target constraint F in a theory T

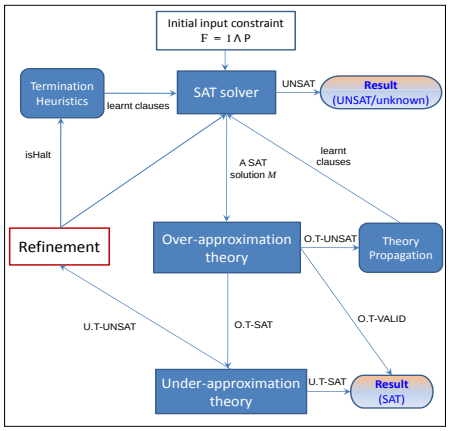

Fig. 2. raSAT loop

2.2 DPLL(T) rules for raSAT loop

By using over and under approximation theories, we define **raSAT** *(SAT by refinement of approximations) loop* in Fig. 2 to decide SAT modulo theory T. It works repeatedly as

1. When an over-approximation theory $O.T$ detects $O.T$-UNSAT (resp. $O.T$-valid), answer UNSAT (resp. SAT).
2. When an under-approximation theory $U.T$ detects $U.T$-SAT, answer SAT.
3. If neither holds, a refinement is applied.

More formally, they are described by abstract DPLL(T) rules. Note that our DPLL(T) rules are restricted than in [25]. $M \parallel \mathcal{I} \mid \mathcal{P}$ means $M \parallel \mathcal{I}, \mathcal{P}$ in [25], where $\mid$ means a guard and P is untoughed. A literal l means some $x \in I$ in $\mathcal{I}$.

– **SAT rule** is applied when either $M \models_{O.T} \mathcal{P}$, or $m \models_{U.T} \mathcal{P}$ for some $m \in M$.

$$M \parallel \mathcal{I} \mid \mathcal{P} \Longrightarrow_{VL} \mathit{SAT} \quad \text{if } M \models_{O.T} \mathcal{P} \text{ or } \exists m \in M. m \models_{U.T} \mathcal{P}.$$

– **Fail rule** is applied when SAT solver returns UNSAT.

$$\emptyset \parallel \mathcal{I} \mid \mathcal{P} \Longrightarrow_{VL} \mathit{fail} \quad \mathit{if}\ \mathcal{I} \text{ is UNSAT}$$

– **Very lazy theory learning rule** is applied when $M \models_{O.T} \neg P$.

$$M \parallel \mathcal{I} \mid \mathcal{P} \Longrightarrow_{VL} \emptyset \parallel \mathcal{I}, (\neg l_1 \vee \cdots \vee \neg l_n) \mid \mathcal{P} \quad \mathit{if} \quad \begin{cases} \{l_1, \ldots, l_n\} \subseteq M \text{ and} \\ l_1 \wedge \cdots \wedge l_n \models_{O.T} \neg P. \end{cases}$$

Whenever either SAT rule or fail rule are applied, DPLL(T) terminates and informs SAT or UNSAT, respectively. Note that in *very lazy theory learning rule*, if $l_1 \wedge \cdots \wedge l_n$ is chosen to be minimal, which is an *UNSAT core*.

The refinement step is a case splitting on bounded quantification. That is, choose $x \in I$ in $\mathcal{I}$ and replace it with $x \in I_1 \vee \cdots \vee x \in I_k$, where $I = I_1 \cup \cdots \cup I_k$ and I_i's are mutually disjoint. We refer $x \in I$ and $x \in I_i$ by l and l_i, respectively.

– **Refinement rule** is applied when $\models_{O.T}$ and $\models_{U.T}$ neither proves nor disproves M, and $\mathcal{I}$ is refined by case splitting.

$$M \parallel \mathcal{I} \mid \mathcal{P} \Longrightarrow_{VL} \emptyset \parallel \mathcal{I}, l \Leftrightarrow l_1 \vee \cdots \vee l_n, \neg l_1 \vee \neg l_2, \ldots, \neg l_{n-1} \vee \neg l_n \mid \mathcal{P}$$

A *termination heuristic* interrupts too fine exploration of a specific M.

– **Heuristic rule** interrupts exploration to refine M when $isHalt(M)$ holds.

$$M \parallel \mathcal{I} \mid \mathcal{P} \Longrightarrow_{VL} \emptyset \parallel \mathcal{I}, \neg M \mid \mathcal{P} \quad \textit{if } isHalt(M)$$

3 Soundness and Completeness of raSAT loop for Polynomial Inequality

From now on, We focus on a *polynomial inequality* such that I_i and $\psi_j(x_1, \cdots, x_n)$ are an open interval (a_i, b_i) and an atomic polynomial inequaltiy (API) $f_j > 0$, respectively. We denote $\mathbb{S}(f_j) = \{x \in \mathbb{R}^n \mid f_j > 0 \text{ holds}\}$.

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

Definition 3. *An* open box *of dimension n is a set $(a_1, b_1) \times \cdots \times (a_n, b_n)$ where $a_i, b_i \in \mathbb{R}, a_i \leq b_i$. For $\mathfrak{a} = (a_1, \cdots, a_n)$ and $\mathfrak{b} = (b_1, \cdots, b_n)$, we denote $(a_1, b_1) \times \cdots \times (a_n, b_n)$ by $(\mathfrak{a}, \mathfrak{b})$.*

The set of all open boxes is a basis of Euclidean topology on $\mathbb{R}^n$. In $\mathbb{R}^n$, a set U is compact if, and only if, U is a bounded closed set. We denote a closure of a set $U(\subseteq \mathbb{R}^n)$ by $\overline{U}$. Since a polynomial is continuous, $\mathbb{S}(\bigwedge_{i=1}^{m} f_i > 0)$ is an open set. Since $\mathbb{Q}$ is dense in $\mathbb{R}$, next lemmas hold.

Lemma 1. *For a polynomial inequality $F = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} f_j > 0$, If there exists an SAT instance of F in $\mathbb{R}^n$, there exists also in $\mathbb{Q}^n$.*

Lemma 2. *Suppose that $a_j < b_j$ for $1 \leq j \leq n$ and f_i's are polynomials. Assume $a_k < c < b_k$ for some k. Then, $\exists x_1 \in (a_1, b_1) \cdots x_n \in (a_n, b_n). \bigwedge_{i=1}^{m} f_i > 0$ is SAT (resp. UNSAT) if, and only if, $\exists x_1 \in (a_1, b_1) \cdots x_k \in (a_k, c) \cdots x_n \in (a_n, b_n). \bigwedge_{i=1}^{m} f_i > 0 \vee \exists x_1 \in (a_1, b_1) \cdots x_k \in (c, b_k) \cdots x_n \in (a_n, b_n)). \bigwedge_{i=1}^{m} f_i > 0$ is SAT (resp. UNSAT).*

Lemma 1 says that proving SAT of F in $\mathbb{R}$ is reduced to that in $\mathbb{Q}$. Lemma 2 says that, in the refinement step, we can apply refinement $x_k \in (a_k, b_k)$ to $x_k \in (a_k, c) \vee x_k \in (c, b_k)$, instead of $x_k \in (a_k, c] \vee x_k \in (c, b_k)$ (i.e., c is ignored).

Initially, $\mathcal{I}$ is a conjunction. Later, by refinements, it becomes a CNF. In Example 1, $x \in (-1, 3)$ and $y \in (2, 4)$ are refined to smaller intervals such

that $\exists x \in (-1,1) y \in (2,4).(x^3y - y^4 > 0) \wedge (y^3 - xy > 0) \vee \exists x \in (1,3) y \in (2,4).(x^3y - y^4 > 0) \wedge (y^3 - xy > 0)$, which results a CNF $(x \in (-1,1) \vee x \in (1,3)) \wedge (y \in (2,4)) \wedge (x^3y - y^4 > 0) \wedge (y^3 - xy > 0)$.

For a polynomial inequality, termination condition $isHalt$ is that lengths of intervals become less than a given threshold.

Definition 4. *For $x_1 \in (l_1, h_1), \cdots, x_n \in (l_n, h_n)$ and $\delta > 0$, $isHalt(M) = (h_1 - l_1 < \delta) \wedge \cdots \wedge (h_n - l_n < \delta)$.*

If we set the threshold γ in $isHalt$ enough small, soundness and (restricted) completeness of **raSAT** loop are shown. Note that such threshold is difficult to predict. In our **raSAT** implementation, it is left as a heuristics.

Definition 5. *Let $F = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} f_j > 0$ be a polynomial inequality such that each I_i is bounded. An over-approximation theory $O.T$ is* converging *if, for each $\delta > 0$ and $c = (c_1, \cdots, c_n) \in I_1 \times \cdots \times I_n$, there exists $\gamma > 0$ such that $\bigwedge_{j=1}^{n} x_j \in (c_j - \gamma, c_j + \gamma) \models_{O.T} \bigwedge_{i=1}^{m} (f_i(c) - \delta < f_i(x) < f_i(c) + \delta)$.*

Definition 6. *Let $F = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} f_j > 0$ for $I_i = (a_i, b_i)$. A refinement strategy is* fair*, if, for each $c_j \in (a_j, b_j)$ and $\gamma > 0$, a refinement of $x_i \in I_i$ for each i eventually occurs in $(c_j - \gamma, c_j + \gamma)$ (as long as an open box is not detected either SAT or UNSAT).*

Theorem 1. *Let $F = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} f_j > 0$ for $I_i = (a_i, b_i)$. Assume that an over-approximation theory $O.T$ is converging, the threshold for isHalt is enough small, and a refinement strategy is fair. Then, the followings hold.*

***Soundness**: If* **raSAT** *loop finds SAT (resp. UNSAT), F is really SAT (resp. UNSAT).*

***Completeness**:*

- *If F is SAT,* **raSAT** *loop eventually finds an SAT instance.*
- *If $\cap \mathbb{S}(f_j) = \emptyset$ and each $\overline{I_i}$ is compact,* **raSAT** *loop eventually detects UNSAT.*

A proof is in Appendix A. Limitations for detecting UNSAT occur on *kissing* and *convergent* cases. Fig. 3 left shows a kissing case $x^2 + y^2 < 2^2 \wedge (x-4)^2 + (y-3)^2 < 3^2$ such that $\overline{\mathbb{S}(-x^2 - y^2 + 2^2)} \cap \overline{\mathbb{S}(-(x-4)^2 - (y-3)^2 + 3^2)} = \{(x,y) \mid (1.6, 1.2)\}$. Thus, there are no coverings to separate them. Fig. 3 right shows a convergent case $y > x + \frac{1}{x} \wedge y < x \wedge x > 0$, which is equivalent to $xy > x^2 + x \wedge y < x \wedge x > 0$. There are no finite coverings to separate them.

Note that Theorem requires only $O.T$ to be converging, since $O.T$-valid works as $U.T$-SAT. Later in Section 4, we apply an interval arithmetic as $O.T$ (which is converging) and testing as $U.T$. The aims of $U.T$ are,

- to accelerate SAT detection, and
- to guide a refinement strategy (e.g., "First Test-UNSAT" in Section 5.2).

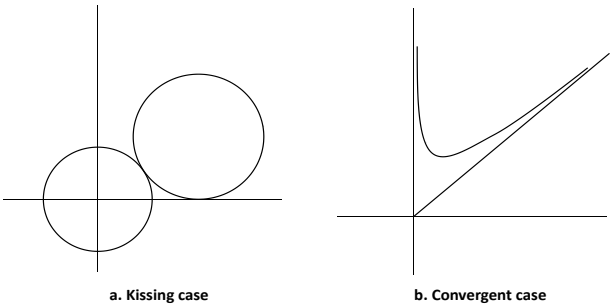

Fig. 3. Limitations for proving UNSAT

4 Over and Under-Approximations for Intervals

4.1 Interval Arithmetic as Over-Approximation

A typical example of IA is Classical Interval (CI) [26], which keeps a lower bound and an upper bound. The weakness of CI is loss of dependency among values. For instance, if $x \in (2, 4)$ then, $x - x$ is evaluated to $(-2, 2)$.

Affine Interval [27, 28] 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 estimate multiplications. They are,

(i) $\epsilon\epsilon'$ is replaced with a fresh noise symbol (AF) [29, 28],
(ii) $\epsilon\epsilon'$ is reduced to the fixed error noise symbol $\epsilon_\pm$ (AF_1 and AF_2) [27],
(iii) $\epsilon\epsilon'$ is replaced with $(-1, 1)\epsilon$ (or $(-1, 1)\epsilon'$) (EAI) [1],
(iv) $\epsilon\epsilon$ is reduced to fixed noise symbols ϵ_+ or ϵ_- (AF_2) [27],
(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 $\epsilon|\epsilon| = \epsilon + (-\frac{1}{4}, \frac{1}{4})$ (Fig. 4).

CAI [30] consists of (ii) and (v), which keeps better precision than iv) for multiplicatins of the same variables, e.g., Taylor expansion.

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

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

For Affine intervals, *sensitivity* [1] of a variable is the absolute value of the coefficient of its corresponding ϵ. In Example 2, CAI estimates the coefficient of $|\epsilon_1|$ as $\mathbf{3}$, which has the largest sensitivity and indicates x the most influencial.

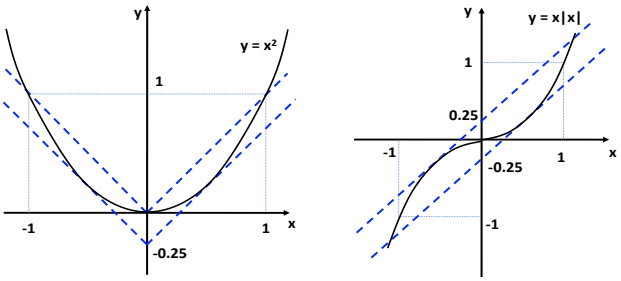

Fig. 4. Chebyshev approximation of x^2 and $x\ |x|$

Definition 7. *Let* $M = \bigwedge_{i=1}^{m} x_i \in (a_i, b_i)$ *and* $\mathcal{P} = \bigwedge_{i=1}^{m} f_i(x_1, \cdots, x_n) > 0$. *Let* δ_i^l *and* δ_i^u *be lower and upper bounds of* $f_i(x_1, \cdots, x_n)$ *estimated by IA for* $x_i \in (a_i, b_i)$. *Then, we say*

- $\mathcal{P}$ *is* IA-VALID *under* M, *if IA evaluates* $\forall i \in [1, m].\ \delta_i^l > 0$,
- $\mathcal{P}$ *is* IA-UNSAT *under* M, $\exists i \in [1, m].\ \delta_i^u \leq 0$, *and*
- $\mathcal{P}$ *is* IA-SAT *under* M, *if* $(\exists j \in [1, m].\ \delta_j^l \leq 0) \wedge (\bigwedge_{i=1}^{m} \delta_i^u > 0)$.

IA-VALID and IA-UNSAT safely reason satisfiability (SAT) and unsatisfiability (UNSAT), respectively. However, IA-SAT cannot conclude SAT.

4.2 Testing as Under Approximation

Definition 8. *Let* $M = \bigwedge_{i=1}^{m} x_i \in (a_i, b_i)$ *and* $\mathcal{P} = \bigwedge_{i=1}^{m} f_i(x_1, \cdots, x_n) > 0$. *Let a choice function* $\theta : (\mathbb{R} \times \mathbb{R})^n \to \mathbb{R}^n$ *such that* $\theta(M) \in (a_1, b_1) \times \cdots \times (a_n, b_n)$. *Testing is a finite set* Θ *of choice functions. Then, we say*

- $\mathcal{P}$ *is* Test-SAT *under* M *if* $\theta(M)$ *holds* $\mathcal{P}$ *for some* $\theta \in \Theta$, *and*
- $\mathcal{P}$ *is* Test-UNSAT *under* M *if* $\theta(M)$ *never holds* $\mathcal{P}$ *for each* $\theta \in \Theta$.

Test-UNSAT does not imply UNSAT though Test-SAT implies SAT. We apply *k-random ticks* [30] as testing, which consists of periodical k-test instances with a random offset, for generating test data of each variable. Note that we keep previously generated test data to avoid overlaps of test data.

5 Strategies in raSAT

raSAT loop is implemented as **raSAT**, which uses (modified) miniSAT 2.2 as a backend SAT solver. Backend theories (in Section 4) are implementd in Ocaml. [9]

[9] **raSAT** download site at `http://www.jaist.ac.jp/m̃izuhito/tools/rasat.html`

5.1 Strategy for Over and Under Approximations for Intervals

UNSAT Core in Polynomial Inequality. An UNSAT core is a minimal set $M_0 = \{l_1, \cdots, l_n\}$ that disproves P (wrt $\models_{O.T}$) in **very lazy theory learning rule** (Section 2.2). However, finding a precise M_0 is not easy, and we apply sound approximation of an UNSAT core $\hat{f}$ of a polynomial f to obtain smaller M_0.

Definition 9. *$\hat{f}$ is an UNSAT core of a polynomial f if IA-UNSAT of $\hat{f} > 0$ implies IA-UNSAT of $f > 0$.*

Example 3. Given a polynomial inequality $f = x^2 - xy - xz > 0$ with $x, y, z \in (0, \infty)$, $\hat{f}_1 = x^2 - xy$ and $\hat{f}_2 = x^2 - xz$ are two UNSAT cores of f.

Incremental Test Data Generation The number of test data affects efficiency a lot. For instance, if we consider a polynomial inequality with 30 variables and we generate 2 test data for each variable, we have 2^{30} test data as a total, which is intractable. The ideas for incremental test data generation are

1. IA-VALID APIs are excluded,
2. IA-SAT APIs are dynamically sorted,
3. test data are incrementally generated during the enumeration of APIs, and
4. discharge test data that do not satisfy previous APIs.

Our ideas of dynamic sorting among APIs are,

(i) an API with a smaller variable set,
(ii) an API that have more APIs whose variable sets are supersets, and
(iii) an API with a smaller additional test data generation,

have higher priority. Example 4 in Appendix B illustrates them.

Let $\{f_j > 0\}$ be the set of IA-SAT APIs and let $Var(f_j)$ be the set of variables in f_j. Let $DEP_{f_i} = \{f_j \mid f_j \in \mathcal{P}, Var(f_i) \subseteq Var(f_j)\}$ and $dep_{f_i} = |DEP_{f_i}|$. An ordering $\preceq$ on $\{f_j\}$ is given by lexicographic applications of $\preceq_a$, $\preceq_b$, and $\preceq_c$.

(a) $dep_{f_i} \geq dep_{f_j}$ implies $f_i \preceq_a f_j$.
(b) If $f_j \prec f_m$ with $Var(f_m) \subseteq \bigcup_{f_i \succeq f_j} Var(f_i)$ and $Var(f_n) \nsubseteq \bigcup_{f_i \succeq f_j} Var(f_i)$ is found, set $f_m \prec_b f_n$.
(c) $f_i \prec f_j, f_k$ and $|Var(f_j) \setminus Var(f_i)| \leq |Var(f_k) \setminus Var(f_i)|$ imply $f_j \preceq_c f_k$.

Note that $\prec_b$ may violate the anti-symmetry. **raSAT** finds $\preceq$ by a procedure, and if $f_i \prec_b f_j$ has set in advance, it will not set $f_j \prec_b f_i$. We intend that (a) for (i), (ii) (since $Var(f_i) \subseteq Var(f_j)$ implies $dep_{f_i} \geq dep_{f_j}$), and (b), (c) for (iii).

An ordering $\preceq$ may be partial, and **raSAT** randomly fulfills it to be total. Then, it renumbers the indecies such that $f_1 \prec f_2 \prec \cdots$. Test data are incrementally generated during this enumeration, i.e., generate test data for $Var(f_1)$, $Var(f_2)$, and so far. During the enumeration, test data that failed previous APIs are removed. When they become empty, it reports Test-UNSAT.

5.2 Interval Refinements

Selecting Intervals to Refine Refinement shares the same explosion problem with test data generation. The choice of intervals to refine has two steps.

(a) Choose an API such that its variables are candidates for refinements.
(b) Among variables, choose influential ones.

(a) follows incremental test data generation in Section 5.1. When an API $f_j > 0$ refutes all generated test data, it returns Test-UNSAT. Then, variables appearing in f_j are candidates for refinement, since f_j is a direct cause of Test-UNSAT. In Example 4, Test-UNSAT of $2yv^2 - ux^2 - 1 > 0$ is reported with $\{v = 0.7, v = 1.3\}$, and $x, y, u, v \in (0, 2)$ become candidates for refinement.

For (b), among variables in the candidate API $f_j > 0$, we further filter variables that have larger sensitivity (Example 2), since they are expected to be more influential. Sensitivity is detected by previous IA-SAT detection phase. Among presented strategies, only this step is not implemented in current **raSAT**.

Interval Decomposition The first choice of interval decomposition is a *Balanced decomposition*, which exactly decomposes an interval into half and half. Instead, we adopt *monotonic decomposition*, which introduces bias δ when a value of a corresponding variable monotonically affects to that of a polynomial.

Definition 10. *Let $f(x_1, \cdots, x_k)$ be a polynomial, a variable x_i is monotonic (resp. anti-monotonic) in f if $x_i' \geq x_i''$ implies $f(\cdots, x_i', \cdots) \geq f(\cdots, x_i'', \cdots)$ (resp. $f(\cdots, x_i', \cdots) \leq f(\cdots, x_i'', \cdots)$). Pos_f (resp. Neg_f) denotes the set of monotonic (resp. anti-monotonic) variables in f.*

Then, a monotonic decomposition on $x \in (a, b)$ with $\delta < b - a$ is,

$$\begin{cases} \{x \in (a, b-\delta), x \in (b-\delta, b)\} & \text{if } x \in Pos_f \\ \{x \in (a, a+\delta), x \in (a+\delta, b)\} & \text{if } x \in Neg_f \\ \{x \in (a, \frac{a+b}{2}), x \in (\frac{a+b}{2}, b)\} & \text{otherwise} \end{cases}$$

The bias δ in Definition 10 is shared with δ in Definition 4. As a choice of SAT instances, we hope to force SAT solver to choose a narrower sub interval (i.e., $(b - \delta, b)$ for Pos_f, and $(a, a + \delta)$ for Neg_f), which would make bounds of f approaching, and provide more chances to lead f satisfiable. MiniSAT 2.2 decides it by the "activity" measure, and we modify that part in MiniSAT 2.2.

6 Equality handling

6.1 Greater-than-or-Equal Handling

raSAT loop is designed to solve polynomial inequality. There are several ways to extend to handle equality, in which our idea shares similarity with dReal [23, 9].

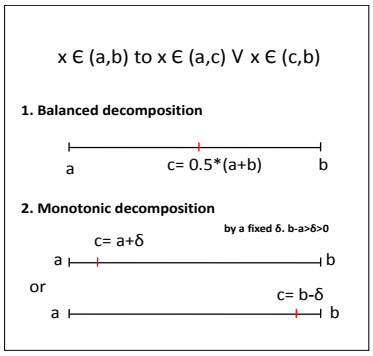

Fig. 5. Interval decompositions

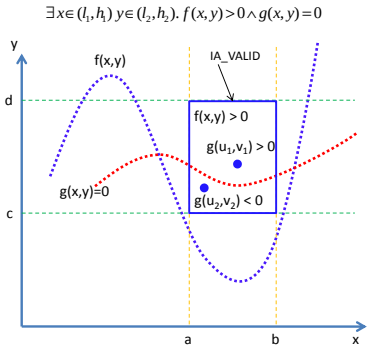

Fig. 6. Intermediate Value Theorem

Definition 11. $\bigwedge_j f_j \geq 0$ *is* strict-SAT *(resp.* strict-UNSAT*) if* $\bigwedge_j f_j > \delta_j$ *is SAT (resp.* $\bigwedge_j f_j > -\delta_j$ *is UNSAT) for some* $\delta_j > 0$.

Lemma 3. *If* $\bigwedge_j f_j \geq 0$ *is* strict-SAT *(resp.* strict-UNSAT*), it is SAT (resp. UNSAT).*

Note that netiher strict-SAT nor strict-UNSAT (i.e., kissing situation), Lemma 3 cannot conclude anything, and **raSAT** says *unknown*.

6.2 SAT on Equality by Intermediate Value Theorem

For solving polynomial constraints with single equality ($g = 0$), we apply *Intermediate Value Theorem*. That is, if existing 2 test cases such that $g > 0$ and $g < 0$, then $g = 0$ is SAT somewhere in between, as in Fig. 6.

Lemma 4. *For* $F = \exists x_1 \in (a_1, b_1) \wedge \cdots \wedge x_n \in (a_n, b_n). \bigwedge_j^m f_j > 0 \ \wedge \ g = 0$, *F is SAT, if there is a box* $(l_1, h_1) \times \cdots \times (l_n, h_n)$ *with* $(l_i, h_i) \subseteq (a_i, b_i)$ *such that*

(i) $\bigwedge_j^m f_j > 0$ *is IA-VALID in the box, and*

(ii) there are two instances $\boldsymbol{t}, \boldsymbol{t'}$ *in the box with* $g(\boldsymbol{t}) > 0$ *and* $g(\boldsymbol{t'}) < 0$.

raSAT first tries to find an IA-VALID box for $\bigwedge_j^m f_j > 0$ by refinements. If such a box is found, it tries to find 2 instances for $g > 0$ and $g < 0$ by testing. Intermediate Value Theorem guarantees the existence of an SAT instance in between. Note that this method works for single equality and does not find an exact SAT instance. If multiple equalities do not share variables each other, we can apply Intermediate Value Theorem repeatedly to decide SAT. In Zankl benchmarks in SMT-lib, there are 15 gen-**.smt2 that contain equality (among 166 problems), and each of them satisty this condition.

7 Experiments

We implement **raSAT** loop as an SMT **raSAT**, based on MiniSat 2.2 as a backend SAT solver. We will compare **raSAT** with **Z3 4.3** (the latest version), since Table.1 in [12] shows the strength of **nlSAT** (equivalently, Z3 4.3). Note that our comparison is only on polynomial inequality. Example 5 in Appendix C illustrates how **raSAT** works.

We apply *2-random ticks* for testing, *Test-UNSAT* of testing and *monotonic decomposition* for refinements. In these experiments, we do not include UNSAT core and sensitivity strategy, which are not implemented yet. All tests are run on a system with Intel Core Duo L7500 1.6 GHz and 2 GB of RAM.

7.1 Preliminary Evaluation

There are three immediate measures on the size of polynomial constraints. They are the highest degree of polynomials, the number of variables, and the number of APIs. We prepare simple benchmarks focusing on these measures.

For the first and the second measures, we apply

$$\psi = \sum_{i=1}^{k} x_i^n < 1 \wedge \sum_{i=1}^{k} (x_i - r)^n < 1 \tag{1}$$

For experiments on these problems, To make SAT and UNSAT problems, we adjust values of r around the threshold $\sqrt[n]{\frac{1}{k}}$ (for fixed k and n), which separates SAT and UNSAT. In our experiments we choose values of r with $|r - \sqrt[n]{\frac{1}{k}}| < 0.01$.

For termination heuristics $isHalt$, δ is set to 0.005, and the initial interval constraints are $\bigwedge_i x_i \in (-1, 1)$.

The degree of polynomials

Table 1 shows results of **raSAT** and **Z3 4.3** for the problems $\psi = x_1^n + x_2^n < 1 \wedge (x_1 - r)^n + (x_2 - r)^n < 1$ with $k = 2$, $n = 8, 10, 12, ..., 22$. The first column indicates whether SAT or UNSAT, followed by the columns of k, n and r. Running time (in seconds) of **Z3 4.3** and **raSAT** are in the last two columns. For the degree 22, the results of **Z3 4.3** are $''? > 3600''$, which means that **Z3 4.3** cannot finish in 3600 seconds. **raSAT** outperforms **Z3 4.3**, and this is not surprising since IA and testing are not affected much by the increase of degrees.

The number of variables

We set simple benchmarks by instantiating $k = 3, 4, 5, 6$ and $n = 4, 6, 8$ to the formula 1. **raSAT** loop decomposes boxes, and the number of boxes can easily grow exponentially. To hold down it in practice, we introduce a strategy to select an API in which each variable is applied an interval decomposition (Section 5.2). Here, the timeout is set to 600 seconds for each problem, and the results is shown in Table 2, which show that the selection strategy seems working well.

SAT/UNSAT	k	n	r	Time(s)	
				Z3 4.3	raSAT
SAT	2	8	1.83	1.330	0.265
UNSAT	2	8	1.84	0.580	0.328
SAT	2	10	1.86	4.530	0.140
UNSAT	2	10	1.87	125.000	0.796
SAT	2	12	1.88	0.360	0.140
UNSAT	2	12	1.89	40.280	1.390
SAT	2	14	1.90	0.480	0.296
UNSAT	2	14	1.91	78.730	0.531
SAT	2	16	1.91	2.250	0.109
UNSAT	2	16	1.92	174.000	0.484
SAT	2	18	1.92	289.110	0.562
UNSAT	2	18	1.93	391.670	0.765
SAT	2	20	1.93	1259.560	1.468
UNSAT	2	20	1.94	1650.860	0.921
SAT	2	22	1.93	$? > 3600$	0.437
UNSAT	2	22	1.94	$? > 3600$	3.203

Table 1. Experimental results for $\psi = x_1^n + x_2^n < 1 \wedge (x_1 - r)^n + (x_2 - r)^n < 1$

SAT/UNSAT	k	n	r	Time(s)	
				Z3 4.3	raSAT
SAT	3	4	1.51	0.030	0.027
UNSAT	3	4	1.52	10.560	*timeout*
SAT	4	4	1.41	*timeout*	0.390
UNSAT	4	4	1.42	*timeout*	*timeout*
SAT	5	4	1.33	*timeout*	51.578
UNSAT	5	4	1.34	*timeout*	*timeout*
SAT	6	4	1.27	*timeout*	111.031
UNSAT	6	4	1.28	*timeout*	*timeout*
SAT	3	6	1.66	*timeout*	0.890
UNSAT	3	6	1.67	*timeout*	62.765
SAT	4	6	1.58	*timeout*	1.156
UNSAT	4	6	1.59	*timeout*	*timeout*
SAT	5	6	1.52	*timeout*	73.937
UNSAT	5	6	1.53	*timeout*	*timeout*
SAT	6	6	1.48	*timeout*	239.968
UNSAT	6	6	1.49	*timeout*	*timeout*
SAT	3	8	1.74	*timeout*	3.125
UNSAT	3	8	1.75	*timeout*	37.156
SAT	4	8	1.68	*timeout*	69.843
UNSAT	4	8	1.69	*timeout*	*timeout*

Table 2. Experimental results for $\psi = \sum_{i=1}^{k} x_i^n < 1 \wedge \sum_{i=1}^{k} (x_i - r)^n < 1$

The number of APIs

Simple benchmarks to measure the effect of the number of APIs are prepared as the formula $\psi = \psi_1 \wedge \psi_2$ where,

- $\psi_1 = x_0^n + x_1^n < 1 \wedge x_1^n + x_2^n < 1 \wedge \cdots \wedge x_k^n + x_0^n < 1$
- $\psi_2 = (x_0 - r)^n + (x_1 - r)^n < 1 \wedge (x_1 - r)^n + (x_2 - r)^n < 1 \wedge \cdots \wedge (x_k - r)^n + (x_0 - r)^n < 1$

We fixed $n = 6$ and k is from 3 to 15. The timeout is set by 600 seconds and $|r - \sqrt[n]{\frac{1}{2}}| < 0.01$. The results are shown in Table 3. Generally, **raSAT** shows better results than Z3 4.3.

SAT/UNSAT	k	n	r	Time(s)	
				Z3 4.3	raSAT
SAT	3	6	1.78	*timeout*	0.171
UNSAT	3	6	1.79	0.280	0.796
SAT	5	6	1.78	*timeout*	0.375
UNSAT	5	6	1.79	0.280	0.640
SAT	7	6	1.78	*timeout*	0.765
UNSAT	7	6	1.79	0.250	0.734
SAT	9	6	1.78	*timeout*	2.671
UNSAT	9	6	1.79	0.300	1.921
SAT	11	6	1.78	*timeout*	3.328
UNSAT	11	6	1.79	0.220	1.343
SAT	13	6	1.78	*timeout*	4.460
UNSAT	13	6	1.79	0.300	1.875
SAT	15	6	1.78	*timeout*	6.640
UNSAT	15	6	1.79	0.300	2.265

Table 3. Experimental results for $\psi = \psi_1 \wedge \psi_2$

7.2 Benchmarks from SMT-LIB

In SMT-LIB [10], benchmark programs on non-linear real number arithmetic (QF_NRA) are categorized into Meti-Tarski, Keymaera, Kissing, Hong, and Zankl families. Until SMT-COMP 2011, benchmarks are only Zankl family. In SMT-COMP 2012, other families have been added, and currently growing. General comparison among various existing tools on these benchmarks is summarized in Table.1 in [12], which shows Z3 4.3 is one of the strongest.

From them, we take problems of polynomial inequality only The brief statistics and explanation are as follows.

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

Solver	Hong (20)			Zankl (151)			Meti-Tarski (832)		
	SAT	UNSAT	time(s)	SAT	UNSAT	time(s)	SAT	UNSAT	time(s)
Z3 4.3	0	8	5.620	**50**	**24**	**1144.320**	**502**	**330**	**33.350**
raSAT	0	**20**	381.531	42	9	2417.931	501	156	21.989

Table 4. Experimental results for Hong, Zankl, and Meti-Tarski families

- **Meti-Tarski** contains 5364 inequalities among 8377, taken from elementary physics. Typically, they are small problems which have lower degrees and few variables, i.e., 3 or 4 variables in each problem. Frequently, linear constraints are mixed in these problems.
- **Keymaera** contains 161 inequalities among 4442.
- **Kissing** has 45 problems, all of which contains equality (mostly single equality).
- **Hong** has 20 inequalities among 20, tuned for QE-CAD and quite artificial.
- **Zankl** has 151 inequalities among 166, taken from termination provers. Problems may contain many (> 100) variables, in which some APIs have > 15 variables

We perform experiments only on Hong, Zankl, and Meti-Tarski families. Table 4 shows the number of solved problems and their total running time (in seconds).

Among 20 problems of Hong family, their degrees distribute from 1 to 20. **raSAT** solved all of them (all are UNSAT). **Z3 4.3** solved 8 problems, whose degrees are up to 8. Note that iSAT can also solve all of problems in Hong family.

For Zankl family, **Z3 4.3** shows better performance than **raSAT**. **Z3 4.3** runs very fast for problems that contain linear constraints combined with nonlinear constraints of lower degrees (e.g., degree 4). We observe that **raSAT** outperforms **Z3 4.3** when problems have a long monomial (e.g., 60), higher degrees (e.g., 6), and APIs with more variables (e.g., > 14). For instance, only **raSAT** can solve *matrix-2-all-5,8,11,12*, and is quicker to show SAT (by testing) in *matrix-2-all-9,10*.

Among large number of problems in Meti-Tarski, we extract 832 problems for the experiment. **Z3 4.3** solved all problems, and **raSAT** solved 657 (SAT/UNSAT) problems among 832. Actually, **raSAT** solved almost all SAT problems, but UNSAT problems are less. One reason is that kissing cases occur frequently in UNSAT problems of Meti-Tarski, which **raSAT** cannot handle. Note that, in Table.1 in [12], only QE-CAD based tools work fine (**Z3 3.1** does not apply QE-CAD, whereas **Z3 4.3** = **nlSAT** includes QE-CAD). Although **raSAT** has certain limitations on UNSAT problems, it shows enough comparable results in Meti-Tarski benchmarks and seems faster than most of QE-CAD based tools (except for **Z3 4.3**).

8 Conclusion

This paper presented **raSAT** loop, which mutually refines over and under–approximation theories. For polynomial inequlaity, we adopted interval arithemtic and testing for over and under-approximation theories, respectively. **raSAT** loop is implemented as an SMT **raSAT**. The result of Experiments on QF_NRA in SMT-lib is encouraging, and **raSAT** shows comparable and sometimes outperforming to existing SMTs, e.g., Z3 4.3, HySAT, and dReal. For instance, ****** which has ** variables and degree ** was solved by **raSAT**, whereas none of above mentioned SMTS can. **raSAT** still remains in naive proto-type status, and there are lots of future work.

Mixed integers. Mixed integers are additional requirements on variables on which SAT instances are integers. This is quite straightforward in **raSAT**, since a box is easy to find grid points.

Exact confirmation. Currently, **raSAT** uses floating point arithmetic. Thus, results can be unsound. We are planning to add a confirmation phase to confirm whether an SAT instance is exact by roundoff error bound guaranteed floating arithmetic libraries, such as ****.

Sensitivity implementation. For incremental test data generation and refinement, we adopt dynamic sorting on APIs. We further design a strategy to select target variables by sensitivity. However, it is not implemented yet, and must be.

Multiple equality handling. Section 6.2 shows single equality handling. We are planning to extend the use of Intermediate Value Theorem to multiple equality with shared variables.

Infinite interva; handling Affine intervals do not work for infinite intervals. Currently, **raSAT** applies classical intervals for infinite intervals. Hoever, infinite intervals may become finite after refinements. We hope to apply affine intervals when they become finite.

Randamization Current strategies may be lead to explore local optimals. As *restart* in SAT solvers, we hope to inlcude restart and randamization techiniques to aviod sticking to local optimals.

References

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

[2] Ngoc, D.T.B., Ogawa, M.: Checking roundoff errors using counterexample-guided narrowing. In: Proceedings of the IEEE/ACM international conference on Automated software engineering. ASE '10, ACM (2010) 301–304

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

[4] Colón, M., Sankaranarayanan, S., Sipma, H.: Linear invariant generation using non-linear constraint solving. In: CAV. Volume 2725 of Lecture Notes in Computer Science., Springer (2003) 420–432
[5] Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Non-linear loop invariant generation using gröbner bases. In: Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages. POPL '04, New York, NY, USA, ACM (2004) 318–329
[6] Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Constructing invariants for hybrid systems. Form. Methods Syst. Des. **32**(1) (2008) 25–55
[7] Anai, H.: Algebraic methods for solving real polynomial constraints and their applications in biology. In: Algebraic Biology Computer Algebra in Biology. (2005) 139–147
[8] Gao, S., Avigad, J., Clarke, E.M.: δ-complete decision procedures for satisfiablity over the reals. In: International Joint Conference on Automated Reasoning. IJCAR 2012, Springer-Verlag (2012) 286–300
[9] Gao, S., Avigad, J., Clarke, E.M.: Delta-decidability over the reals. In: 27th IEEE Symposium on Logic in Computer Science. LICS 2012, IEEE (2012) 305–314
[10] Tarski, A.: A decision method for elementary algebra and geometry. Bulletin of the American Mathematical Society **59** (1951)
[11] Collins, G.E.: Quantifier elimination by cylindrical algebraic decomposition – twenty years of progress. In Caviness, B.F., Johnson, J.R., eds.: Quantifier Elimination and Cylindrical Algebraic Decomposition, Springer-Verlag (1998) 8–23
[12] Jovanović, D., de Moura, L.: Solving non-linear arithmetic. In: Proceedings of the 6th international joint conference on Automated Reasoning. IJCAR'12, Springer-Verlag (2012) 339–354
[13] Weispfenning, V.: Quantifier elimination for real algebra - the quadratic case and beyond. Applicable Algebra in Engineering, Communication and Computing **8** (1997) 85–101
[14] Passmore, G.O., Jackson, P.B.: Combined decision techniques for the existential theory of the reals. In: Proceedings of the 16th Symposium, 8th International Conference. Held as Part of CICM '09 on Intelligent Computer Mathematics. Calculemus '09/MKM '09, Springer-Verlag (2009) 122–137
[15] Corzilius, F., Loup, U., Junges, S., Ábrahám, E.: SMT-RAT: an SMT-compliant nonlinear real arithmetic toolbox. In: Proceedings of the 15th international conference on Theory and Applications of Satisfiability Testing. SAT'12, Springer-Verlag (2012) 442–448
[16] Corzilius, F., Ábrahám, E.: Virtual substitution for SMT-solving. In: Proceedings of the 18th international conference on Fundamentals of computation theory. FCT'11, Springer-Verlag (2011) 360–371
[17] Bryant, R.E., Kroening, D., Ouaknine, J., Seshia, S.A., Strichman, O., Brady, B.: Deciding bit-vector arithmetic with abstraction. In: Proceedings of the 13th international conference on Tools and algorithms for the construction and analysis of systems. TACAS'07, Springer-Verlag (2007) 358–372
[18] Zankl, H., Middeldorp, A.: Satisfiability of non-linear (ir)rational arithmetic. In: Proceedings of the 16th international conference on Logic for programming, artificial intelligence, and reasoning. LPAR'10, Springer-Verlag (2010) 481–500
[19] Borralleras, C., Lucas, S., Navarro-Marset, R., Rodríguez-Carbonell, E., Rubio, A.: Solving non-linear polynomial arithmetic via sat modulo linear arithmetic. In: Proceedings of the 22nd International Conference on Automated Deduction. CADE-22, Springer-Verlag (2009) 294–305

[20] Ganai, M., Ivancic, F.: Efficient decision procedure for non-linear arithmetic constraints using cordic. In: Formal Methods in Computer-Aided Design, 2009. FMCAD 2009. (2009) 61 –68
[21] Ratschan, S.: Efficient solving of quantified inequality constraints over the real numbers. ACM Trans. Comput. Logic **7**(4) (October 2006) 723–748
[22] Franzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. Journal on Satisfiability, Boolean Modeling and Computation **1** (2007) 209–236
[23] Gao, S., Kong, S., Clarke, E.M.: dReal: An SMT solver for nonlinear theories over the reals. In: Conference on Automated Deduction. CADE-24, Springer-Verlag (2013) 208–214
[24] Stolfi, J., de Figueiredo, L.: An introduction to affine arithmetic. Tendencias em Matematica Aplicada e Computacional **3**(4) (2003) 297–312
[25] Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Abstract DPLL and abstract DPLL modulo theories. In Baader, F., Voronkov, A., eds.: Logic for Programming, Artificial Intelligence, and Reasoning. Volume 3452 of Lecture Notes in Computer Science. Springer-Verlag (2005) 36–50
[26] Moore, R.E.: Interval Analysis. Prentice-Hall (1966)
[27] Messine, F.: Extensions of affine arithmetic: Application to unconstrained global optimization. Journal of Universal Computer Science **8**(2) (2002)
[28] Comba, J.L.D., Stolfi, J.: Affine arithmetic and its applications to computer graphics. In: Proceedings of VI SIBGRAPI. (1993) 9–18
[29] Stolfi, J.: Self-Validated Numerical Methods and Applications. PhD thesis, PhD. Dissertation, Computer Science Department, Stanford University (1997)
[30] Khanh, T.V., Ogawa, M.: SMT for polynomial constraints on real numbers. Electr. Notes Theor. Comput. Sci. **289** (2012) 27–40

A Proof of Theorem 1

Proof. Soundness: From the definitions of O.T and U.T.
Completeness: If F is SAT (i.e., $\cap\mathbb{S}(f_j) \neq \emptyset$), there is an open box in it with the size $\delta > 0$. Assume that F is UNSAT (i.e., $\cap\overline{\mathbb{S}(f_j)} = \emptyset$) and each $\overline{I_i}$ is compact. Let $\delta(f_j)(\bar{x}) = max\{|f_j(\bar{x}) - f_1(\bar{x})|, \cdots, |f_j(\bar{x}) - f_m(\bar{x})| \mid \bar{x} \in I_1 \times \cdots \times I_n\}$. From $\cap\overline{\mathbb{S}(f_j)} = \emptyset$, $\delta(f_j)(\bar{x}) > 0$ for each j. Since $\delta(f_j)$ is continuous and $\overline{I_i}$ is compact, $\delta(f_j)(x)$ has the minimum value $\delta_j > 0$. Let $\delta = \frac{min\{\delta_j\}}{2}$. Then $\delta > 0$.

In either case, since $O.T$ is converging, there exists $\gamma > 0$ for $\delta > 0$ satisfying Definition 5. We set γ to be the threshold of $isHalt$. Since a refinement is fair, refined boxes detect either SAT or UNSAT, respectively. □

B Incremental Test Data Generation Example

Example 4. Let $\mathcal{P} = (2x - y^2 - 2 > 0) \wedge (x^2 - 1 > 0) \wedge (xy - yz - zx > 0) \wedge (u^2 - x^2y > 0) \wedge (2yv^2 - ux^2 - 1 > 0)$ with $x, y, z, u, v \in (0, 2)$ and let testing be 2-random ticks. Fig. 7 shows the generated ordering on $\mathcal{P}$. Then, incremental test data generation proceeds as follows.

1. For $x^2 - 1 > 0$, assume that generated test data are $\{x = 1.2, x = 0.5\}$. Then, $x^2 - 1 > 0$ holds for $\{x = 1.2\}$. ($x = 0.5$ is discharged.)
2. Next $2x - y^2 - 2 > 0$. Test data is generated for y, and assume that they are $\{y = 1.4, y = 0.5\}$. Then, $\{x = 1.2, y = 0.5\}$ holds $2x - y^2 - 2 > 0$.
3. Similary, for $xy - yz - zx > 0$ (an altenative choice is $u^2 - x^2y > 0$), $\{z = 0.8, z = 0.3\}$ are generated. Then, $\{x = 1.2, y = 0.5, z = 0.3\}$ holds it.
4. For $u^2 - x^2y > 0$, $\{u = 1.05, u = 0.25\}$ are generated. Then, $\{x = 1.2, y = 0.5, z = 0.3, u = 1.05\}$ holds it.
5. Finally, for $2yv^2 - ux^2 - 1 > 0$, if $\{v = 0.7, v = 1.3\}$ are generated, neither satisfies it and Test-UNSAT is reported. Instead, if $\{v = 1.13, v = 1.77\}$ are generated, $\{x = 1.2, y = 0.5, z = 0.3, u = 1.05, v = 1.77\}$ is an SAT instance.

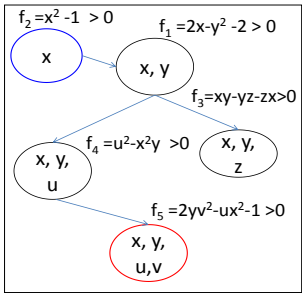

Fig. 7. Sorting APIs

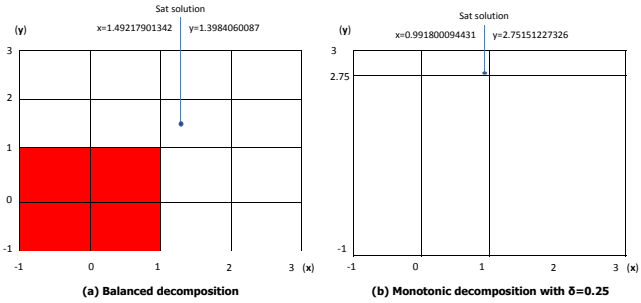

Fig. 8. Interval decompositions by **raSAT**

C saSAT Execution Example

Example 5. We explain how **raSAT** works by $F = \exists x \in (-1, 3) y \in (-1, 3).x^3 - x^2 + y - 1.99 > 0$. **raSAT** initially evaluates with an open box $x \in (-1, 3) \wedge y \in (-1, 3)$. IA results IA-SAT and testing results Test-UNSAT. Then, a refinement step is applied to decompose the intervals $x \in (-1, 3)$ and $y \in (-1, 3)$.

– **Balanced decomposition**: Balanced decomposition decomposed into a CNF $(x \in (-1, 1) \vee x \in (1, 3)) \wedge (y \in (-1, 1) \vee y \in (1, 3))$. Assume that the SAT solver chooses the interval combination $x \in (-1, 1) \wedge y \in (-1, 1)$. Then, IA results IA-SAT and testing results Test-UNSAT again. Balanced decomposition is applied again, and the initial box is decomposed into boxes shown in the left of Fig. 8. IA concludes IA-UNSAT on red boxes, and they will be removed from the boxes of further searching. When the SAT solver chooses $x \in (1, 2) \wedge y \in (1, 2)$, IA results IA-SAT, and testing finally finds a satisfiable test instance $x = 1.49217901342$ and $y = 1.3984060087$ (Test-SAT).

– **Monotonic decomposition**: Monotonic decomposition is described in the right of Fig. 8 (where $\delta = 0.25$). When a monotonic decomposition is applied, the initial interval constraint is decomposed into a CNF $(x \in (-1, 1) \vee x \in (1, 3)) \wedge (y \in (-1, 2.75) \vee y \in (2.75, 3))$, since $y \in Pos_f$. The SAT solver chooses $x \in (-1, 1) \wedge y \in (2.75, 3)$ for IA and testing, which result IA-SAT and Test-UNSAT.
The second monotonic decomposition is applied and $(x \in (-1, 0) \vee x \in (0, 1) \vee x \in (1, 3)) \wedge (y \in (-1, 2.75) \vee y \in (2.75, 3))$ is obtained. Note that $(2.75, 3)$ has already reached to $isHalt$ with $\delta = 0.25$, and is not decomposed further. The SAT solver chooses $x \in (0, 1) \wedge y \in (2.75, 3)$ and testing finds a satisfiable test instance $x = 0.991800094431$ and $y = 2.75151227326$ (Test-SAT). With monotonic decomposition, **raSAT** finds a satisfiable instance with fewer decompositions.