raSAT: SMT Solver for Polynomial Inequality

Vu Xuan Tung[1] and To Van Khanh[2] and Mizuhito Ogawa[1]

[1] Japan Advanced Institute of Science and Technology
{tung,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, called **raSAT** loop, which is an extension of the standard ICP (Interval Constraint Propagation) with testing. Two approximation schemes consist of interval arithmetic (over-approximation) and testing (under-approximation, to boost SAT detection. If both of them fail to decide, input intervals are refined by decomposition.
ICP is robust for large degrees, but the number of boxes (products of intervals) to explore exponentially explodes when the number of variables increases. We design strategies to choose a variable for decomposition and a box by targeting on SAT detection.
Several heuristic measures, called *SAT likelyhood*, *sensitivity*, and the number of unsolved atomic polynomial constraints, are compared on Zankl and Meta-Tarski benchmarks from QF_NRA category of SMT-LIB. They are also evaluated by comparing Z3 4.3 and isat3. We also show a simple modification to handle mixed intergers, and experiments on AProVE benchmark from QF_NIA category of SMT-LIB.

1 Introduction

Polynomial constraint solving is to find an instance that satisfies given polynomial inequality/equality. Many applications are reduced to solving polynomial constraints, such as

- **Locating roundoff and overflow errors**, which is our motivation [**?**, **?**].
- **Automatic termination proving**, which reduces termination detection to finding a suitable ordering [**?**], e.g., TTT2 [3], Aprove [4].
- **Loop invariant generation**. Farkas's lemma is a popular approach in linear loop invariant generation [**?**], and is reduced to degree 2 polynomials. Nonlinear loop invariant [**?**] requires more complex polynomials.
- **Hybrid system**. SMT for QF_NRA are often used as backend engines [**?**].
- **Mechanical contrnol design**. PID control is simple but widely used, and designing parameters is reduced to polynomial constraints [**?**].

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

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

A practical alternative is *ICP* (Interval Constraint Propagation), which are used in SMT solver community, e.g., isat3 [**?**], dReal [**?**]. ICP is based on over-approximation by interval arithmetics, and iteratively refines by interval decompositions. It is practically often more efficient than algebraic computation at the cost of loss of theoretical guarantees. It guarantees that, for polynomial inequality $\exists x_1 \in (a_1, b_1) \cdots x_n \in (a_n, b_n). \wedge_i f_i > 0$,

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

under the assumptions of *fair* decomposition and bounded intervals (a_i, b_i).

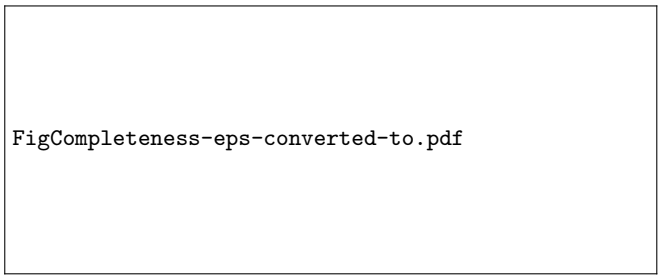


Fig. 1. SAT and UNSAT detection by ICP

The boundary part is reduced to polynomial equality checking, which would be solved algebraic methods, like Groebner basis. Alternatively, by loosening equality to δ-equality, δ-completeness is obtained [**?**, **?**].

This paper presents an SMT solver **raSAT** for polynomial inequality. It consists of a simple iterative approximation refinement, called **raSAT** loop, which is an extension of the standard ICP with testing to accelerate SAT detection. Two approximation schemes consist of interval arithmetic (over-approximation) and testing (under-approximation, to boost SAT detection. If both of them fail to decide, input intervals are refined by decomposition.

Compared to typical ICPs, **raSAT**

– intruoduces testing (as an under-approximation) to accelerate SAT detection,
– applies various interval arithmetic, e.g., Affine intervals [**?**, **?**, **?**], which enables to analyze the effects of input values, and
– SAT confirmation step by an error-bound guaranteed floating point package **iRRAM**, to avoid soundess bugs caused by roundoff errors, which is an alternative to [**?**].

These design is more on SAT detection oriented, since from our preliminary experiences, if the target problems have several hundred variables, solvable cases in practice are either SAT or UNSAT with small UNSAT core. Thus, acceleration of SAT detection and finding UNSAT core will be keys for scalability.

ICP is robust for larger degrees, but the number of boxes (products of intervals) to explore exponentially explodes when variables increase. Thus, design of strategies for selecting variables to decompose and boxes to explore is crucial for efficiency. Our strategy design is,

– a box with more possiblity to be SAT is selected to explore, which is estimated by several heuristic measures, called *SAT likelyhood*, and the number of unsolved atomic polynomial constraints, and
– a more influential variable is selected for multiple test cases and decomposition, which is estimated by *sensitivity*.

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

– Even if input ranges are open ended, first **rasat** loop starts with bounded ranges with certain values. If it is UNSAT, it will be enlarged to include infinity.
– Set the bound $isHalt$ such that interval will be be decomposed no smaller than it. If neither SAT nor UNSAT is detected, reset $isHalt$ with a smaller value and restart.

Note that *SAT likelyhood* and *sensitivity* are estimated during interval arithmetic. Especially, the latter can be applied only with Affine intervals.

Efficient UNSAT core and UNSAT confirmation with error bound guaranteed floating point arithmetic are left for future work.

They are compared on Zankl and Meta-Tarski benchmarks from QF_NRA category of SMT-LIBSMT-lib benchamrks[5]. They are also evaluated by comparing Z3 4.3[6] and HySAT[7]. We also show a simple modification to handle mixed intergers, and experiments on AProVE benchmark from QF_NIA category of SMT-LIB. Generally speking, the combination of *SAT likelyhood* and *sensitivity* works best.

	raSAT	Z3 4.3	Hysat
NRA/Zankl(166) (timeout 500sec)	56		
NRA/Meta=Tarski (7713) (timeout 60sec)	6304		
NIA/AProVE (8829) (timeout 60sec)	7582	8271	—

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

{**Mizuhito:** Concrete experimental results; the fastest is not in first priority, but fastest with single simple method.}

Related Work

Solving polynomial constraints on real numbers is decidable [**?**], though that on integers is undecidable (*Hilbert's 10th problem*). Quantifier elimination by cylindrical algebraic decomposition (QE-CAD) [**?**] is a well known technique, and implemented in Mathematica, Maple/SynRac, Reduce/Redlog, QEPCAD-B, and recently in some SMTs, e.g., nlSAT [**?**]. 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) [**?**] focusing on small degree polynomials (especially degree 2) has better performance, but it is still EXPTIME.

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 [**?**] and Z3 4.3 (which is referred as nlsat in [**?**]) 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 [**?**][**?**] 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 [**?**] 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 [**?**], 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 [**?**], 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 [**?**] 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 [**?**], HySAT [**?**], and dReal [**?**] 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 [**?**] 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 $m = (x_1, \cdots, x_n)$ satisfy all $x_i \in I_{i_j}$ that are assigned true.

We adopt *very lazy theory learning* [**?**], 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 **??**, we will instantiate an interval and an atomic polynomial inequality to I_i's and ψ_j, respectively. Then, Fig. **??** illustrates Definition **??**.

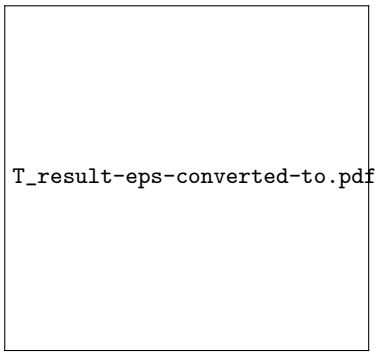


Fig. 2. Results of a target constraint F in a theory T

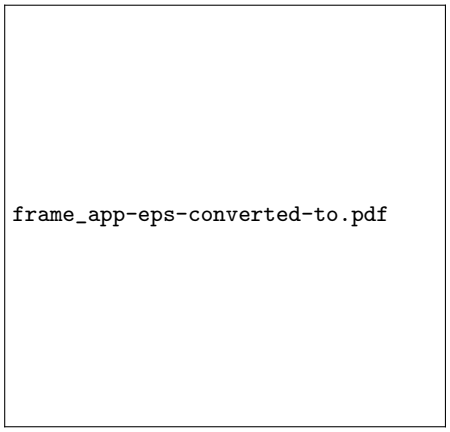


Fig. 3. raSAT loop

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.

A typical ICP applies $O.T$ only as an interval arithmetic. Later in Section **??**, we will instantiate interval arithmetic as $O.T$. Adding to $O.T$-valid, we introduce testing as $U.T$ to accelerate SAT detection.

2.2 Over-Approximation Theory Refinement

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 $\mathbb{S}(f_j) = \{x \in \mathbb{R}^n \mid f_j > 0 \text{ holds}\}$.

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

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

under the assumptions of *fair* decomposition and bounded intervals (a_i, b_i). We will prepare terminology and briefly review this fact.

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 $(\mathfrak{a}, \mathfrak{b})$.*

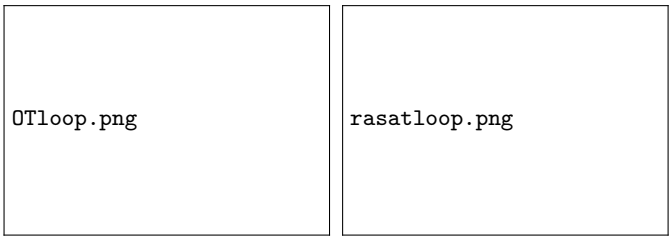
The set of all open boxes is a basis of Euclidean topology on $\mathbb{R}^n$. In $\mathbb{R}^n$, a set U is compact if, and only if, U is a bounded closed set. We denote a closure of a set $U(\subseteq \mathbb{R}^n)$ by $\overline{U}$. Since a polynomial is continuous, $\mathbb{S}(\bigwedge_{i=1}^{m} f_i > 0)$ is an open set. Note $\mathbb{Q}$ is dense in $\mathbb{R}$, thus we can replace an SAT instance in reals with one in rationals.

Initially, $\mathcal{I}$ is a conjunction. Later, by refinements, it becomes a CNF. In Example **??**, $x \in (-1, 3)$ 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 $(x \in (-1, 1) \vee x \in (1, 3)) \wedge (y \in (2, 4)) \wedge (x^3y - y^4 > 0) \wedge (y^3 - xy > 0)$.

A standard ICP applies an interval arithmetic as an over-approximation theory $O.T$. However, most of properties on an interval arithmeic are proved only with the following.

Definition 4. *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* $\bigwedge_{j=1}^{n} x_j \in (c_j - \gamma, c_j + \gamma) \models_{O.T} \bigwedge_{i=1}^{m} (f_i(c) - \delta < f_i(x) < f_i(c) + \delta)$.

$O.T$ refinemnet loop is shown in Fig. **??** (a). A standard ICP based algorithm of an SMT solver applies it with $O.T$ as a classical interval arithemtic. The variation of interval arithemtic will be presented in Section **??**.



(a) $O.T$ theory refinement loop **raSAT** loop

Fig. 4. Rfinement loops

Definition 5. *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 decomposition of* I_i *for each* i *eventually occurs in* $(c_j - \gamma, c_j + \gamma)$ *(as long as neither SAT nor UNSAT is detected).*

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, each* (a_i, b_i) *is bounded, and a refinement strategy is fair. Then,*

- *if* $\exists x_1 \in (a_1, b_1) \cdots x_n \in (a_n, b_n). \wedge_i f_i > 0$ *is SAT, O.T refinemnet loop eventually detects it, and*
- *if* $\exists x_1 \in [a_1, b_1] \cdots x_n \in [a_n, b_n]. \wedge_i f_i \geq 0$ *is UNSAT, ICP eventually detects it.*

Proof. The former is proved by the fact that, if F is SAT, there exists a non-empty neiborhood (open box) in $\cap\mathbb{S}(f_j)$, where $\mathbb{S}(f_j) = \{(x_1, \cdots, x_n) \mid f_j(x_1, \cdots, x_n) > 0\}$. If the box decomposition strategy is fair, the refinemnet will eventually find such an open box.

For the latter, assume that $\overline{F} = \exists x_1 \in [a_1, b_1] \cdots x_n \in [a_n, b_n]. \wedge_i f_i \geq 0$ is UNSAT, Thus, $\mathbb{S}(\overline{F}) = \emptyset$. 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 **??**. □

Limitations for detecting UNSAT occur on *kissing* and *convergent* cases. Fig. **??** 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. **??** right shows a convergent case $y > x + \frac{1}{x} \wedge y < x \wedge x > 0$, which is equivalent to $xy > x^2 + x \wedge y < x \wedge x > 0$. There are no finite coverings to separate them.

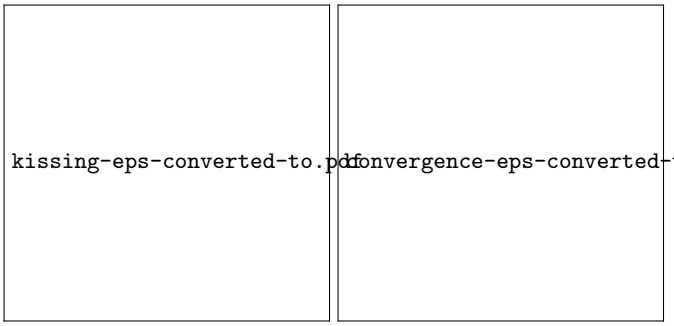


Fig. 5. Limitations for proving UNSAT

2.3 raSAT loop

From our preliminary experiences, when polynomial constraints have several hundred variables, it is very difficult to detect UNSAT unless it has a small UNSAT core.

Although an $O.T$ refinement loop is enough to implement an ICP based SMT solver, we extend it as **raSAT** (SAT by refinement of approximations) loop to boost SAT detection by adding an $U.T$ theory. **raSAT** loop works repeatedly as in Fig. **??** (b),

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.

Our design of an SMT solver **raSAT**, which implements **raSAT** loop, applies two heuristic features

- An incremental search by narrowing intervals and incrementaly deepening search (Section **??**).
- Heurstic measures *SAT-likelyhood* and *sensitivity*, for selection of a variable to decompose and a box to explore. (Section **??**).

raSAT also prepares various interval arithmetic as $O.T$ as in Section **??**, whereas currently only random tesing is prepared as $U.T$.

3 Over and Under-Approximations for Intervals

A typical theory for $O.T$ and $U.T$ are an interval arithmetic and testing, respectively. We say *IA-valid*, *IA-SAT*, and *IA-UNSAT*, when it is $O.T$-valid, $O.T$-SAT, and $O.T$-UNSAT, respectively. Similarly, we say *test-SAT* and *test-UNSAT*, when it is $U.T$-SAT and $U.T$-UNSAT, respectively. Note that IA-valid and test-SAT imply SAT, and IA-UNSAT implies UNSAT, whereas IA-SAT and test-UNSAT can conclude neither.

At the moment, we apply *k-random ticks* [**?**] as testing, which consists of periodical k-test instances with a random offset, for generating test data of each variable. However, a strategy based on *SAT-likelyhood* and *sensitivity* selects variables to generate multiple instances (currently 10 variables are choosen for periodical 2-test instances, and only single random instance is generated for the rest).

raSAT prepares various Affine intervals, adding to classical interval (CI) [**?**], 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 [**?**, **?**] 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) [**?**, **?**],
(ii) $\epsilon\epsilon'$ is reduced to the fixed error noise symbol $\epsilon_\pm$ (AF_1 and AF_2) [**?**],
(iii) $\epsilon\epsilon'$ is replaced with $(-1, 1)\epsilon$ (or $(-1, 1)\epsilon'$) (EAI) [**?**],
(iv) $\epsilon\epsilon$ is reduced to fixed noise symbols ϵ_+ or ϵ_- (AF_2) [**?**],
(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 $\epsilon|\epsilon| = \epsilon + (-\frac{1}{4}, \frac{1}{4})$ [**?**].

Remark 1. For Affine intervals, *sensitivity* [**?**] of a variable is a possible range of the absolute value of the coefficient of its corresponding ϵ.

Note that Affine interval works only for bounded intervals. For instance, $\infty + \infty\epsilon$ represents $(-\infty, \infty)$, which says nothing. Narrowing intervals as an incremental search (Section **??**) partilly depends on this fact. That is, if $\pm\infty$ is contained in an interval, first give finite upper/lower bounds and search within these bounds using an Affine interval. If UNSAT is concluded, then enlarge to the whole intervals using CI.

Example 1. 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 $(-8, 4.5)$.

4 Strategies in raSAT

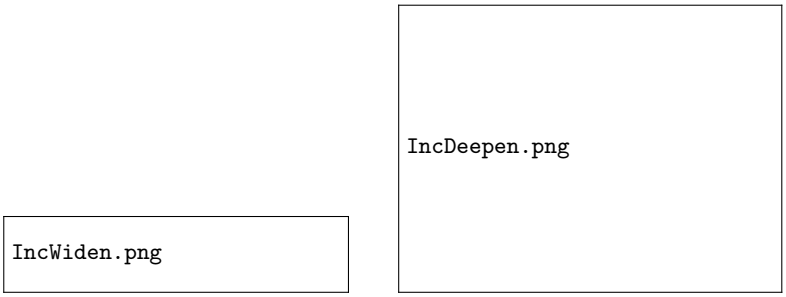
4.1 Incremental search

raSAT applies two incremental strategies, (1) *incremental windening*, and (2) *incremental deepening*.

Let $F = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} f_j > 0$ for $I_i = (a_i, b_i)$, where a_i, b_i would be $\pm\infty$.

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

Note that if $\delta_i < \infty$, **raSAT** applies an Affine interval; otherwise, it uses CI. Experiments in Section **??** are performed with $\delta_0 = 10$ and $\delta_1 = \infty$.



(a) Incremenal widening (b) Incremental Deepening

Fig. 6. Chebyshev approximation of x^2 and $x\ |x|$

Incremental deepening Starting with $F = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} f_j > 0$, $I_1 \times \cdots \times I_n$ is decomposed into many boxes. F becomes the disjunction of existential formulae corrsponding to these boxes. **raSAT** searches these boxes in depth-first manner, which may leads to local optimal search. To avoid it, **raSAT** applies a threshold γ, such that no more decomposition will be applied when a box becomes smaller than γ. If neither SAT nor UNSAT is detected, **raSAT** restarts with a smaller threshhold.

Let $\gamma_0 > \gamma_1 > \cdots > 0$, and **raSAT** incrementally deepens its search with these threshholds, i.e., starting with δ_0, and if it fails, restart with δ_1, and so on (Fig **??** (b)). Experiments in Section **??** are performed with $\gamma_0 = 0.1$ and $\gamma_{i+1} = \gamma_i/10$.

4.2 SAT directed heuristics measure

With several hundred variables, we observe possibilty of

- either SAT, or
- UNSAT with small UNSAT core.

For the latter, we need an efficient heuristics to find an UNSAT core, which is left as future work. For the former, the keys are how to choose variables to decompose, and how to choose a box to explore. The number of variables to decompose leads exponential growth of boxes, which shares the same problem with variables to generate multiple test instances (in $U.T$). **raSAT** choose such variables in two steps; first it selects *test-UNSAT APIs*, and then choose variables that appear in these APIs. We design SAT-directed heuristic measures based on the interval arithemtic ($O.T$).

Let $F = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} f_j > 0$ becomes $\vee \exists x_1 \in I'_1 \cdots x_n \in I'_n. \bigwedge_{j=1}^{m} f_j > 0$ after box decomposition. We denote the estimated range of f_j for $x_1 \in I'_1 \cdots x_n \in I'_n$ with IA ($O.T$) by $range(f_j, I'_1 \times \cdots \times I'_n)$. If an IA

is an Affine interval, f_j for $x_1 \in I'_1 \cdots x_n \in I'_n$ is estimated with the form $[c_1, d_1]\epsilon_1 + \cdots + [c_n, d_n]\epsilon_n$, and by instanciating ϵ_i with $[-1, 1]$, the resulting classical interval coincides with $range(f_j, I'_1 \times \cdots \times I'_n)$.

For $\exists x_1 \in I'_1 \cdots x_n \in I'_n. \bigwedge_{j=1}^{m} f_j > 0$, if some $f_j > 0$ is UNSAT, the box $I'_1 \times \cdots \times I'_n$ is UNSAT. If every $f_j > 0$ is SAT, F is SAT. Thus, if the box $I'_1 \times \cdots \times I'_n$ needs to be explore, it must contain a test-UNSAT APT (thus IA-SAT). We choose a test-UNSAT API $f_j > 0$ and let $I = |f_j|_{I'_1 \times \cdots \times I'_n}$. Thus $h > 0$. we define

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

Example 2. {**Mizuhito:** TO be filled}

SAT-likelyhood of an API intends to estimate APIs how likely to be SAT. For choosing variables, **raSAT** first choose a test-UNSAT API by SAT-likelyhood. There are two choices, either the largest or the least. *Sensitivity* of a variable intends to estimate variables how influencial to the value of an API. From selected APIs by SAT-likelyhood, **raSAT** selects variables with larger sensitivity. This selection of variables are used for (1) *multiple test instances generation*, and (2) *decomposition*.

For choosing a box to explore, **raSAT** chooses more likely to be SAT. There are two choice, (1) a box with the largest SAT-likelyhood, and (2) a box with the largest number of SAT (either IA-valid or test-SAT) APIs.

Experiments in Section **??** are performed with 10 variables (1 varable for each 10 APIs selected by SAT-likelyhood) for multiple test generation (2 instances), and 1 variable for decomposition.

5 Experiments

We implement **raSAT** loop as an SMT **raSAT**, based on MiniSat 2.2 as a backend SAT solver. Variaous combinations of strategies of **raSAT** (Section **??**) are compared. The best choice of **raSAT** is also compared with **Z3 4.3** and **iSAT3**, where the former is considered to be the state of the art ([**?**]), and the latter is an ICP based tool. Note that our comparison is only on polynomial inequality. The experiments are on a system with Intel Xeon E5-2680v2 2.80GHz and 4 GB of RAM.

Other than heuristics mentioned in Section **??**, there are lots of heuristic choices. For instance,

- how to generate test instances (in $U.T$),
- how to decompose an interval,

and so on. Experiments below are performed with randome generation (k-random tick) for the former and the blanced decomposition (dividing at the exact middle) for the latter. Further investigation is left for future.

5.1 Benchmarks from SMT-LIB

In SMT-LIB [8], 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 [**?**], 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.

- **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 Zankl, and Meti-Tarski families. Table **??** shows the number of solved problems and their total running time (in seconds).

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 non-linear 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 [**?**], 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] `http://www.smt-lib.org`

5.2 Polynomial constraints over integers

5.3 Effectiveness of designed strategies

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. Our strategies focus on selecting APIs(for testing) and selecting variable (for multiple test cases and for decomposition), selecting decomposed box of the chosen variable in TEST-UNSAT API. Thus, in order to justify the effectiveness of these strategies, we need to test them on problems with varieties of APIs number (selecting APIs) and varieties of variables number in each API (selecting variable). For this criteria, we use Zankl family for evaluating strategies.

The number of APIs for this family varies from 1 to 2231 and the maximum number of variables in each APIs varies from 1 to 422.

- Selecting APIs, selecting decomposed boxes:
  (1) Using SAT likely-hood
  (2) Randomly
- Selecting one variable each API for testing, one variable of TEST-UNSAT API for decomposition:
  (3) Using sensitivity
  (5) Randomly

We will compare the combinations $(1)-(5)$ and $(2)-(5)$ to see how SAT likely-hood affects the results. And comparison betweeen $(1)-(5)$ and $(1)-(3)$ illustrates the effectiveness of sensitivity. In these experiments, timeout is set to $500s$.

Table **??** shows the results of strategies. While $(1)-(5)$ solved 33 problems in $27.09s$, $(2)-(5)$ took $933.78s$ to solve 31 problems. In fact, among 31 solved problems of $(2)-(5)$, $(1)-(5)$ solved 30. The remaining problem is a SAT one, and $(2)-(5)$ solved it in $34.36s$. So excluding this problem, $(2)-(5)$ solved 30 problems in $933.78s - 34.36s = 899.42s$ and $(1)-(5)$ took $27.09s$ to solved all these 30 problems plus 3 other problems. Using SAT likely-hood to select the most difficult API first for testing is very reasonable. This is because the goal of testing is to find an assignment for variables that satisfies **all** APIs. Selecting the most difficult APIs first has the following benefits:

- The most difficult APIs are likely not satisfied by most of test cases (of variables in these APIs). This early avoids exploring additional unnecessary combination with test cases of variables in other easier APIs.
- If the most difficult APIs are satisfied by some test cases, these test cases might also satisfy the easier APIs.

raSAT uses SAT likely-hood to choose a decomposed box so that the chosen box will make the TEST-UNSAT API more likely to be SAT. This seems to work for SAT problems but not for UNSAT ones. In fact, this is not the case. Since for UNSAT problems, we need to check both of the decomposed boxes to prove the unsatisfiability (in any order). So for UNSAT constraints, it is not the problem

Strategies	SAT	UNSAT	time(s)
(1)-(5)	22	11	27.09
(2)-(5)	21	10	933.78
(1)-(3)	31	10	765.48

Table 1. Experiments of strategies on Zankl family

of how to choose a decomposed box, but the problem of how to decompose a box. That is how to choose a point within an interval for decomposition so that the UNSAT boxes are early separated. Currently, raSAT uses the middle point. For example, $[0, 10]$ will be decomposed into $[0, 5]$ and $[5, 10]$. The question of how to decompose a box is left for our future work.

Combination $(1) - (3)$ solved 41 problems in comparison with 33 problems of $(1) - (5)$. There are 5 large problems (more than 100 variables and more than 240 APIs in each problem) which were solved by $(1) - (3)$ (solving time was from $20s$ to $300s$) but not solved (timed out - more than $500s$) by $(1) - (5)$. Choosing the most important variable (using sensitivity) for multiple test cases and for decomposition worked here.

5.4 Comparison with other tools on QF_NRA and QF_NIA benchmarks

Comparison with Z3 and isat3 on QF_NRA. Table **??** presents the results of isat3, raSAT and Z3 on 2 families (Zankl and Meti-tarski) of QF_NRA. Only problems with inequalities are extracted for testing. The timeout for Zankl family is $500s$ and for Meti-tarski is $60s$ (the problems in Meti-tarski are quite small - less than 8 variables and less than 25 APIs for each problem). For isat3, ranges of all variables are uniformly set to $[-1000, 1000]$

Both isat3 and raSAT use interval arithmetics for deciding constraints. However, raSAT additionally use testing for accelerating SAT detection. As the result, raSAT solved larger number of SAT problems in comparison with isat3. In Table **??**, isat3 conludes UNSAT when variables are in the ranges $[-1000, 1000]$, while raSAT conclude UNSAT over $[-infinity, infinity]$.

In comparison with Z3, for Zankl family, Z3 4.3 showed better performance than raSAT in the problems with lots of small constraints: short monomial (less than 5), small number of variables (less than 10). For problems where most of constraints are long monomial (more than 5) and have large number of variables (more than 10), raSAT results are quite comparable with ones of Z3, often even outperfoming when the problem contains large number of vary long constraints (more than 40 monomials and/or more than 20 variables). For instance, *matrix-5-all-27, matrix-5-all-01, matrix-4-all-3, matrix-4-all-33, matrix-3-all-5, matrix-3-all-23, matrix-3-all-2, matrix-2-all-8* are such problems which can be solved by raSAT but not by Z3.

Solver	Zankl (151)			Meti-Tarski (5101)		
	SAT	UNSAT	time(s)	SAT	UNSAT	time(s)
isat3	14	15	209.20	2916	1225	885.36
raSAT	31	10	765.48	3322	1052	753.00
Z3 4.3	**54**	**23**	**1034.56**	**3528**	**1569**	**50235.39**

Table 2. Experimental results for Hong, Zankl, and Meti-Tarski families

Solver	SAT	UNSAT	time (s)
raSAT	6764	0	1230.54
Z3	**6784**	**36**	**139.78**

Table 3. Experiments on QF_NIA/AProVE

Meti-tarski family contains quite small problems (less than 8 variables and less than 25 APIs for each problem). Z3 solved all the SAT problems, while raSAT solved around 94% (33322/3528) of them. Approximately 70% of UNSAT problems are solved by raSAT. As mentioned, raSAT has to explore all the decomposed boxes for proving unsatisfiability. Thus the question "How to decompose boxes so that the UNSAT boxes are early detected?" needs to be solved in order to improve the ability of detecting UNSAT for raSAT.

Comparison with Z3 on QF_NIA. We also did experiments on QF_NIA/AProVE benchmarks to evaluate raSAT loop for Integer constraints. There are 6850 problems which do not contain equalities. The timeout for this experiment is $60s$. Table **??** shows the result of Z3 and raSAT for this family.

raSAT solved almost the same number of SAT problems as Z3 (6764 for raSAT with 6784 for Z3). Nearly 99% problems having been solved by raSAT is a quite encouraging number. Currently, raSAT cannot conclude any UNSAT problems. Again, this is due to exhaustive balanced decompositions.

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 [**?**, **?**].

Definition 6. $\bigwedge_j f_j \geq 0$ *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 1. *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 **??** 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. **??**.

Lemma 2. *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 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 **??** shows single equality handling. We are planning to extend the use of Intermediate Value Theorem to multiple equality with shared variables.

Infinite interval 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.

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.