JAIST Repository
https://dspace.jaist.ac.jp/

Title	raSAT: SMT for Polynomial Inequality
Author(s)	To, Van Khanh; Ogawa, Mizuhito
Citation	Research report (School of Information Science, Japan Advanced Institute of Science and Technology), IS-RR-2013-003: 1-23
Issue Date	2013-05-27
Type	Technical Report
Text version	publisher
URL	http://hdl.handle.net/10119/11349
Rights	
Description	リサーチレポート（北陸先端科学技術大学院大学情報科学研究科）

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

raSAT: SMT for Polynomial Inequality

To Van Khanh and Mizuhito Ogawa

School of Information Science
Japan Advanced Institute of Science and Technology, Japan
{khanhtv,mizuhito}@jaist.ac.jp

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

Keywords: interval arithmetic, affine arithmetic, SMT, polynomial constraints, testing.

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 encoded into polynomial constraints.

- **Automated detection of roundoff and overflow error**, which is our motivating application [1, 2]. For instance, consider DSP decoder like mpeg4. Usually, the decoder definition is given by a reference algorithm in C, which uses floating point number. In an embedded system, it is tempting to replace floating point into fixed point numbers. However, naive replacement would cause recognizable noise and locating such roundoff error source is not easy.
- **Automatic termination proving**, which is reduced to finding a suitable termination ordering [3]. There are lots of termination prover, e.g., T$_{\text{T}}$T$_2$ [1], Aprove [2].

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

- **Automatic loop invariant detection**. The use of Farkas's lemma is a popular approach in linear loop invariant generation [4]. Farkas's lemma uses products of matrices, and it requires solving polynomial constraints of degree 2. Non-linear loop invariant generation [5] and hybrid systems [6] require more complex polynomials.
- **Mechanical control design**. PID control is simple but widely used. Fujitsu used polynomial constraints solving to design PID control of HDD head movement [7].

Solving polynomial constraints on real numbers is decidable [8], though that on integers is undecidable (Hilbert's 10th problem). Quantifier elimination by cylindrical algebraic decomposition (QE-CAD) [9] is a well known technique, and implemented in Mathematica, Maple/SynRac, Reduce/Redlog, QEPCAD, and recently nlSAT [10]. It is DEXPTIME w.r.t. 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. There is an example to require over 20 hours on a supercomputer. Virtual substitution (VS) for polynomial constraints with degree smaller than or equal to 4 has better performance, but it is still EXPTIME.

SMT (SAT modulo theories) separates the case analysis and the core computation in a background theory, and many implementations are available, e.g., Z3, CVC3, yices. Presburger arithmetic (linear constraints) is one of the most popular background theory, and polynomial constraints (non-linear constraints) become recently evolving, which are mostly based on approximations. Their approaches can be classified as:

- **Interval Constraint Propagation (ICP)** Given bounded quantification, ranges of polynomials on real numbers are (over-) estimated by (classical) interval arithmetic. RSOLVER [11] and iSAT [12] are such examples.
- **Bit-Blasting** After setting bounds on values of variables, arithmetic operations on integers are encoded into a CNF over the fixed number of boolean variables. Bit-blasting is a main stream in QF-NIA (non-linear integer arithmetic) in SMT-COMP. MiniSmt [13] applies bit-blasting for rational numbers in QF_NRA (Non-linear Real Arithmetic), and by combining with symbolic manipulations, it can treat pre-fixed algebraic numbers. Drawback is, due to the bounds of bits, it cannot detect UNSAT.
- **Linearization** For QF-NIA, polynomial constraints are linearized by instantiating one of arguments in each multiplication with integers within given bounds. Barcelogic [14] is such an example. CORD uses another linearization, called CORDIC (COrdinate Rotation DIgital Computer) [15]. Both Barcelogic and CORD apply Yices for solving linear constraints.

Note that the second and third categories are affected a lot by the increase of degrees of polynomials. Our approach is based on the first category. We also focus on polynomial inequality constraints. This simplifies problems a lot still keeping applications above in the scope.

- In constructive analysis, 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. Thus, in implementation, we can easily fit exact computation on rational numbers (by representing as pairs of integers, such as num library of Ocaml), instead of floating point numbers.

This paper presents an iterative approximation refinement, called **raSAT** loop, which solves polynomial inequality constraints on real numbers. The approximation scheme consists of interval arithmetic (over-approximation, aiming to decide unsatisfiability) and testing (under-approximation, aiming to decide satisfiability). If both of them fail to decide, input intervals are refined by decomposition. **raSAT** loop is implemented as an SMT **raSAT** with miniSAT 2.2 as a backend SAT solver.

First, Section 2 starts with general framework of bounded quantification on over and under approximation theories. **raSAT** loop is presented to formalize refinement steps as case splitting. Soundness and (restricted) completeness of **raSAT** loop are also given in this section.

Next, Section 3 instantiates *Interval Arithmetic (IA)* and testing as over and under approximation theory, respectively. The former intends to detect UNSAT and the latter intends to detect SAT. Adding to Affine intervals [16], we apply another variant Chebyshev Affine interval [17]. A refinement step is an interval decomposition, and a termination heuristics $isHalt$ (which leads *unknown*) halts **raSAT** loop when all open boxes become smaller than a given bound.

There are immediate measures on inputs, *degrees of polynomials*, *the number of variables*, and *the number of atomic polynomial constraints*. Since **raSAT** loop depends on IA and testing, 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 will be a dominant factor.

In Section 4, we prepare strategies.

- UNSAT core detection for IA.
- Dynamic sorting by dependency to select an atomic polynomial inequality. Test data generation and interval decompositions are applied on variables appearing in it.
- Interval decomposition with certain bias.

Section 5 shows experimental results including QF_NRA benchmarks from SMT-LIB. They show that **raSAT** is comparable to Z3 4.3, and sometimes outperforms, especially with high degree of polynomials. For instance, polynomial inequality with a long monomial (e.g., 60), degree 6, and many variables (e.g., 14) in Zankl family are such examples.

Section 6 overviews related works, and Section 7 concludes the paper with observations and future works.

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

$$F = \exists x_1 \in I_1 \cdots x_n \in I_n . \psi(x_1, \cdots, x_n)$$

where $\psi(x_1, \cdots, x_n)$ is a CNF of literals.

When regarding as an SMT (SAT modulo theory) problem, boolean variables are assigned to each of $x_i \in I_i$ and each literal in $\psi(x_1, \cdots, x_n)$, and truth assignments is produced by a SAT solver, which are proved or disproved by a background theory T. As notational convention, m (the lower case) denotes an instance (m is aimed at variable assignments) of $x_i \in I_i$'s, and M (the upper case) denotes a (full) truth assignment on $x_i \in I_i$'s. Later, we will instantiate intervals and a polynomial constraint to I_i's and ψ, respectively (Section 3), and we will restrict $\psi(x_1, \cdots, x_n)$ to a conjunction.

As an SMT problem, we can switch a backend theory T. In *very lazy theory learning* [18], T is applied only for a full truth assignment M. We regard M as a conjunction of literals (in a background theory T).

- If an instance m of variables appearing in F satisfies F, we denote $m \models_T F$.
- For a truth assignment M, if an instance m satisfies M, we denote $m \in M$. 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 $M \models_T F$,*
- *T-satisfiable (T-SAT) if $m \models_T F$ for some $m \in M$, and*
- *T-unsatisfiable (T-UNSAT) if $M \models_T \neg F$.*

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

Note that F is T-valid if and only if $\forall x_1 \in I_1 \cdots x_n \in I_n . \psi(x_1, \cdots, x_n)$ is *true* under theory T. Fig. 1 illustrates Definition 1, when each I_i is an interval and ψ is a polynomial inequality.

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.

Later in Section 3, we will instantiate $O.T$ and $U.T$ with interval arithmetic and testing, respectively.

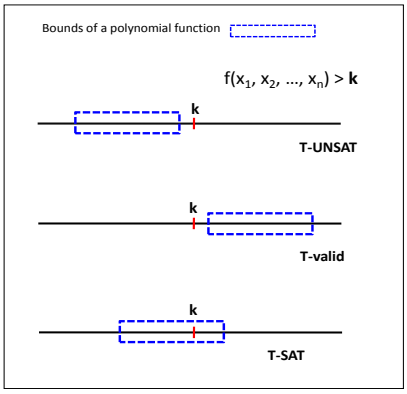

Fig. 1. Results of a target constraint F in a theory T

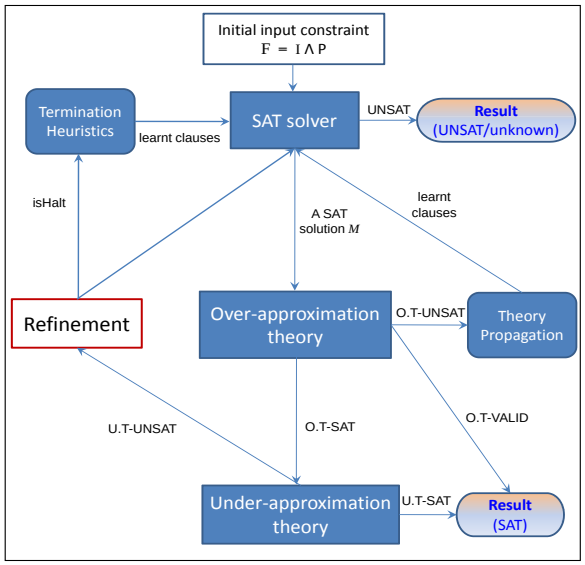

Fig. 2. Framework of raSAT loop

2.2 Refinement and Termination Heuristics: raSAT loop

By using over and under approximation theories, we apply **raSAT** loop (SAT by refinement of approximations) 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.

The refinement step is a case splitting on bounded quantification. That is, choose $x_i \in I_i$ and replace it with $x_i \in I_{i_1} \vee \cdots \vee x_k \in I_{i_k}$, where $I_{i_1}, \cdots, I_{i_k}$ satisfy that I_{i_j}'s are mutually disjoint and $I_i = I_{i_1} \cup \cdots \cup I_{i_k}$.

A *termination heuristic* is used to halt an **raSAT** loop when each I_{i_j} becomes *too small*, which is specified by the $isHalt$ rule. In case of I to be an interval, $isHalt$ is given by the size of each box becomes than a given threshold.

As an SMT, **raSAT** loop applies *very lazy theory learning* [18], that is, back-end theory is applied only for a full truth assignment M.

2.3 Soundness and Completeness of raSAT loop for Polynomial Inequality

We focus on *polynomial inequality constraints* with input ranges as open boxes which is described in Definition 3.

Definition 3. *A polynomial inequality constraint is a bounded quantification* $\exists x_1 \in I_1 \cdots x_n \in I_n.\psi(x_1, \cdots, x_n)$ *such that*

- *each* I_i *is an open interval* $x_i \in (a_i, b_i)$*, and*
- $\psi(x_1, \cdots, x_n)$ *is a conjunction of* $f_j > 0$ *where* f_j *is a polynomial over* $\{x_1, \cdots, x_n\}$*.*

$f_i > 0$ *is called an atomic polynomial inequality (API). We denote* $\mathbb{S}(F) = \{x \in \mathbb{R}^n \mid F \text{ holds}\}$*.*

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

Definition 4. *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 Euclidean space, a set U is compact if, and only if, U is a bounded closed set. Since a polynomial is a continuous function, $\mathbb{S}(\bigwedge_{i=1}^{m} f_i > 0)$ is an open set. Since $\mathbb{Q}$ is dense in $\mathbb{R}$, next two lemmas are immediate.

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 *are polynomials. Assume* $a_k < c < b_k$ *for* $1 \leq k \leq n$*. 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, at the refinement step, we can apply an interval decomposition $x_k \in (a_k, b_k)$ to $x_k \in (a_k, c) \vee x_k \in (c, b_k)$, instead of $x_k \in (a_k, b_k)$ to $x_k \in (a_k, c] \vee x_k \in (c, b_k)$ (i.e., missing c is allowed).

Note that although an initial polynomial inequality is SAT encoded to conjunctions only, by refinements, it becomes a CNF. For instance, recall Example 1. $x \in (-1, 3)$ and $y \in (2, 4)$ are refined to smaller intervals such that $\exists x \in (-1, 1) y \in (2, 4).(x^3 y - y^4 > 0) \wedge (y^3 - xy > 0) \vee \exists x \in (1, 3) y \in (2, 4).(x^3 y - y^4 > 0) \wedge (y^3 - xy > 0)$, which is SAT-encoded to a CNF $(x \in (-1, 1) \vee x \in (1, 3)) \wedge (y \in (2, 4)) \wedge (x^3 y - y^4 > 0) \wedge (y^3 - xy > 0)$.

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

Definition 5. *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 efficiently predict in advance. In our **raSAT** implementation, it is left as a heuristics.

Definition 6. *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 constraint such that each* I_i *is bounded. An over-approximation theory* $O.T$ *is* converging *(w.r.t.* F*) if, for each* $\delta > 0$ *and* $c = (c_1, \cdots, c_n)$ *satisfying* I*, 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 7. *Let* $F = \exists x_1 \in I_1 \cdots x_n \in I_n. \bigwedge_{j=1}^{m} f_j > 0$ *for* $I_i = (a_i, b_i)$*. An interval decomposition strategy is* fair*, if, for each* $c_j \in (a_j, b_j)$ *and* $\gamma > 0$*, an interval decomposition for* x_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 (w.r.t. F). If the threshold for isHalt is enough small and an interval decomposition strategy is fair, the followings hold.*

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

Proof. Soundness: From the definitions of O.T and U.T.
Completeness: If F is SAT (i.e., $\cap S(f_j) \neq \emptyset$), there is an open box in it with the size $\delta > 0$.

Assume that F is UNSAT (i.e., $\cap closure(S(f_j)) = \emptyset$) and each $closure(I_i)$ 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 $\cap closure(\mathbb{S}(f_j)) = \emptyset$, $\delta(f_j)(\bar{x}) > 0$ for each j. Since $\delta(f_j)$ is continuous and $closure(I_i)$ is compact, $\delta(f_j)(x)$ has the minimal value for $\bar{x} \in closure(I_1 \times \cdots \times I_n)$. Let $\delta_j = min\{\delta(f_i)(x) \mid x \in I\}$ and $\delta = \frac{min\{\delta_j\}}{2}$. Then $\delta > 0$.

In either case, since $O.T$ is complete, there exists $\gamma > 0$ for $\delta > 0$ satisfying Definition 6. We set γ to be the threshold of $isHalt$. Since an interval decomposition strategy is fair, decomposed boxes detect either SAT or UNSAT, respectively.

Limitations for detecting UNSAT occur on *kissing* and *convergence* cases. The left of Fig. 3 shows an example of kissing case for the constraint $x^2 + y^2 < 2^2 \wedge (x-4)^2 + (y-3)^2 < 3^2$. In this example, $closure(x^2+y^2 < 2^2) \cap closure((x-4)^2 + (y-3)^2 < 3^2) = (x = 1.6, y = 1.2)$, and there are no coverings to separate $x^2 + y^2 < 2^2$ and $(x-4)^2 + (y-3)^2 < 3^2$.

The right of Fig. 3 is an example of convergence for the constraint $y > x + \frac{1}{x} \wedge y < x \wedge x > 0$. The open box is $(0, \infty) \times (0, \infty)$ and there are no finite coverings to separate $y > x + \frac{1}{x}$ and $y < x \wedge x > 0$.

Note that Theorem requires only $O.T$ to be complete, since $O.T$-valid works as $U.T$-SAT. Later in Section 3, we apply an interval arithmetic as $O.T$ and testing as $U.T$. It is not difficult to see that an interval arithmetic is converging, and the aims of $U.T$ are,

- to obtain practical efficiency, and
- to guide interval decomposition strategy (like "First Test-UNSAT" in Section 4.2).

3 Over and Under-Approximations for Intervals

We present *Interval Arithmetic* and *Testing* as an over-approximation theory $\models_{O.T}$ and an under-approximation theory, $\models_{U.T}$, respectively. For interval arithmetic, we apply Affine intervals (AI) [16] and Chebyshev Affine interval (CAI) [17].

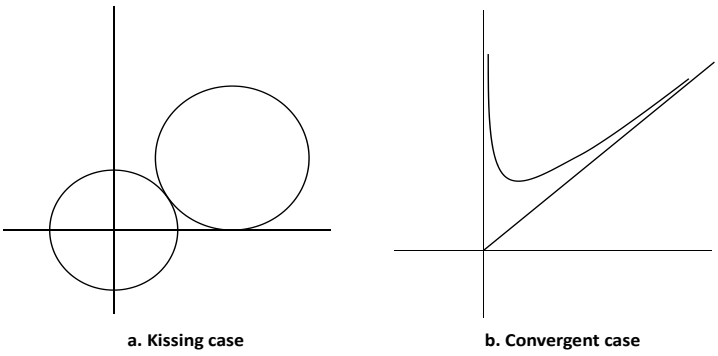

Fig. 3. Limitations for proving UNSAT

3.1 Interval Arithmetic

A typical example of IA is Classical Interval (CI) [19], 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 [20, 21] 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 may be less precise than CI. Forms of affine intervals vary by choices how to estimate multiplications. They are,

(i) $\epsilon\epsilon'$ is replaced with a fresh noise symbol (AF) [16, 21],
(ii) $\epsilon\epsilon'$ is pushed into the fixed error noise symbol $\epsilon_\pm$ (AF_1 and AF_2) [20],
(iii) $\epsilon\epsilon'$ is replaced with $(-1, 1)\epsilon$ (or $(-1, 1)\epsilon'$) (EAI) [1],
(iv) $\epsilon\epsilon$ is replaced by fixed noise symbols ϵ_+ or ϵ_- (AF_2) [20],
(v) keeping products of noise symbols up to degree 2 ($\epsilon_i\epsilon_j$),
(vi) Chebyshev approximation of x^2 [17] (Fig. 4), which introduces noise symbols for absolute values $|\epsilon|$, $\epsilon\epsilon = |\epsilon||\epsilon| = |\epsilon| + (-\frac{1}{4}, 0)$ and $\epsilon|\epsilon| = \epsilon + (-\frac{1}{4}, \frac{1}{4})$.

For designing CAI [17], we apply (ii) and (vi). Compared with (iv), (vi) keeps better precision for higher degrees greater than or equal to 2. Note that upper and lower bounds estimated by IA are over-approximation bounds of polynomials.

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,

- by AF_2, $f = -3 - \epsilon_1 - 2\epsilon_2 + 3\epsilon_+ + 3\epsilon_\pm$ and the bound of f is estimated as $(-9, 6)$,
- by CAI, $f = (-4, -\frac{11}{4}) + (-\frac{1}{4}, 0)\epsilon_1 - 2\epsilon_2 + \mathbf{3}|\epsilon_1| + (-2, 2)\epsilon_\pm$ and the bound of f is estimated as **(-8,4.5)**.

When IA has a noise symbol ϵ, we define *sensitivity* [1] of a variable as the absolute value of the coefficient of corresponding ϵ. In CAI of Example 2, the coefficient **3** of $|\epsilon_1|$ has the largest sensitivity, which indicates x is the most influence.

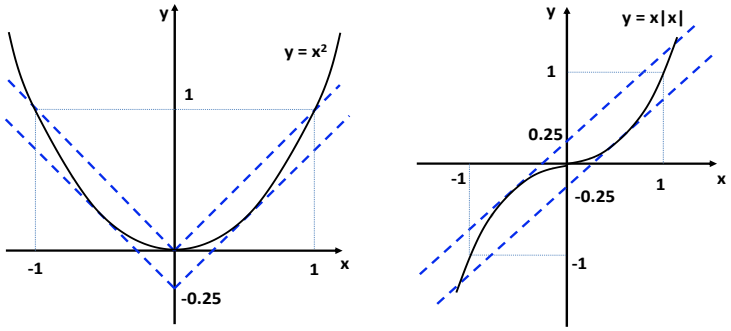

Fig. 4. Chebyshev approximation of x^2 and $x\ |x|$

3.2 Interval Arithmetic as Over Approximation

Interval arithmetic (IA) is applied for estimating bounds of polynomials under a given input range (a box), and we use it as an over-approximation theory. We instantiate IA to $O.T$ in Section 2, and obtain the definition below.

Definition 8. *Given a polynomial inequality constraint* $\exists x_1 \in (a_1, b_1) \cdots x_n \in (a_n, b_n).\ \bigwedge_{i=1}^{m} f_i(x_1, \cdots, x_n) > 0$.

Let $f_i^l(x_1, \cdots, x_n)$ *and* $f_i^u(x_1, \cdots, x_n)$ *be lower and upper bounds estimated by IA for* $x_i \in (a_i, b_i)$.

Let $I = x_1 \in (a_1, b_1) \cdots x_n \in (a_n, b_n)$ *and* $P = \bigwedge_{i=1}^{m} f_i(x_1, \cdots, x_n) > 0$, *we say*

- P *is* IA-VALID *under* I, *if IA evaluates* $\forall i \in [1, m].\ f_i^l(x_1, \cdots, x_n) > 0$,
- P *is* IA-UNSAT *under* I, $\exists i \in [1, m].\ f_i^u(x_1, \cdots, x_n) \leq 0$, *and*
- P *is* IA-SAT *under* I, *if* $(\exists j \in [1, m].\ f_j^l(x_1, \cdots, x_n) \leq 0) \wedge (\bigwedge_{i=1}^{m} f_i^u(x_1, \cdots, x_n) > 0)$.

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

3.3 Testing as Under Approximation

We instantiate testing to $U.T$ in Section 2.

Definition 9. *Given a polynomial inequality constraint* $\exists x_1 \in (a_1, b_1) \cdots x_n \in (a_n, b_n).\ \bigwedge_{i=1}^{m} f_i(x_1, \cdots, x_n) > 0$, *let* $I = x_1 \in (a_1, b_1) \cdots x_n \in (a_n, b_n)$ *and* $P = \bigwedge_{i=1}^{m} f_i(x_1, \cdots, x_n) > 0$. *Let a choice function* $\theta : (\mathbb{R} \times \mathbb{R})^n \to \mathbb{R}^n$ *with* $\theta(I) \in (a_1, b_1) \times \cdots \times (a_n, b_n)$. *For a finite set* Θ *of choice functions, we say*

– *P is* Test-SAT *under I if P holds for some $\theta \in \Theta$, and*
– *P is* Test-UNSAT *under I if P never hold for each $\theta \in \Theta$.*

The set Θ of choice functions in definition 9 is a set of test data. We apply *k-random ticks* [17], which consists of periodical k-test instances with a random offset, for generating test data of each variable. Note that Test-UNSAT does not imply UNSAT though Test-SAT implies SAT.

4 Strategies

4.1 Strategy for Over and Under Approximations for Intervals

UNSAT Core in a Polynomial Inequality An UNSAT core is defined as a minimal set $M_0 = \{l_1, \cdots, l_n\}$ that disproves P (w.r.t. $\models_{O.T}$) in the very lazy theory learning rule (Section 2). To obtain a precise minimal M_0 is not easy. As a strategy to obtain smaller M_0, we introduce an UNSAT core $\hat{f}$ of a polynomial f based on the IA-UNSAT judgment. Then, M_0 is selected as literals in M corresponding to variables in $\hat{f}$.

Definition 10. *$\hat{f}$ 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 constraint $f = x^2 - xy - xz > 0$ 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.

For instance, $\hat{f}_1 > 0$ is IA-UNSAT under $x \in (1, 2) \wedge y \in (4, 5)$, then $f > 0$ is IA-UNSAT under a set of input boxes $\{x \in (1, 2) \wedge y \in (4, 5) \wedge z \in (0, 1), x \in (1, 2) \wedge y \in (4, 5) \wedge z \in (1, 2), \cdots\}$, which is a set of all input boxes including $x \in (1, 2) \wedge y \in (4, 5)$. For removing unsatisfiable input boxes, $\neg(x \in (1, 2)) \vee \neg(y \in (4, 5))$ is added to interval constraints I as a learnt clause.

Incremental Test Data Generation Performing a large number of test data generations affects efficiency. For instance, if we consider a polynomial constraint with 30 variables and we generate 2 test data for each variable, we have 2^{30} test data as total, which is intractable. The ideas for incremental test data generation are (i) an API-wise test data generation with dynamic sorting of IA-SAT APIs, and (ii) thinning test data that does not satisfy an API. Note that, during test data generation, an interval decomposition is fixed, and test data are generated for IA-SAT APIs only. Let $\{f_j > 0\}$ be the set of IA-SAT APIs, and let $Var(f_j)$ be the set of variables appearing in an atomic polynomial inequality f_j.

For an API-wise test data generation, an ordering of testing of IA-SAT APIs affects the efficiency. Our ideas are,

– API with a smaller variable set,
– bottleneck API w.r.t. dependency ($Var(f_i) \subseteq Var(f_j)$), and
– API with a smaller additional test data generation

have high priority. To formalize them, let $DEP_{f_i} = \{f_j \mid f_j \in P \wedge Var(f_i) \subseteq Var(f_j)\}$ and $dep_{f_i} = |DEP_{f_i}|$. Then, during an API-wise test data generation, $\{f_j\}$ is dynamically sorted at the choice of next API to hold

(a) $Var(f_i) \subset Var(f_j)$ implies $i \leq j$,
(b) dep_{f_1} is the largest, and
(c) if, for some $j < m$, $Var(f_m) \subseteq \bigcup_{i=1}^{j} Var(f_i)$ and $\forall n.\ Var(f_n) \nsubseteq \bigcup_{i=1}^{j} Var(f_i)$, then $m \leq n$,

An API-wise test data generation requires storing previous test results of tested APIs. To reduce stored test results, test data refuting APIs are removed. When they become empty, it returns Test-UNSAT, and shifts to the refinement.

Example 4. Let $P = (2x - y^2 - 2 > 0) \wedge (x^2 - 1 > 0) \wedge (xy - yz - zx > 0) \wedge (u^2 - x^2y > 0) \wedge (2yv^2 - ux^2 - 1 > 0)$ with $x, y, z, u, v \in (0, 2)$ and let testing be 2-random ticks.

APIs of P are dynamically sorted as Fig. 5 for generating test data.

- First, $x^2 - 1 > 0$ is chosen, since $\{x\}$ is the smallest set of variables. Assume that generated test data are $\{x = 1.2, x = 0.5\}$, and $x^2 - 1 > 0$ holds for $\{x = 1.2\}$.
- Next, $2x - y^2 - 2 > 0$ is chosen, since $\{y\}$ is a smaller set of additional variables. Assume that generated test data are $\{y = 1.4, y = 0.5\}$. $\{x = 1.2, y = 0.5\}$ holds $2x - y^2 - 2 > 0$.
- $xy - yz - zx > 0$ is chosen (we can also chose $u^2 - x^2y > 0$). Assume that generated test data are $\{z = 0.8, z = 0.3\}$ and $\{x = 1.2, y = 0.5, z = 0.3\}$ holds it.
- For $u^2 - x^2y > 0$, assume that test data are $\{u = 1.05, u = 0.25\}$. $\{x = 1.2, y = 0.5, z = 0.3, u = 1.05\}$ holds it.
- Finally, for $2yv^2 - ux^2 - 1 > 0$, assume that generated test data are $\{v = 0.7, v = 1.3\}$. Neither satisfies it and Test-UNSAT is reported, and proceed to interval decomposition. If generated test data are $\{v = 1.13, v = 1.77\}$, $\{x = 1.2, y = 0.5, z = 0.3, u = 1.05, v = 1.77\}$ holds it and SAT is reported.

4.2 Strategies for Refinements

We need to consider the choice of intervals to decompose, and how to decompose an interval. Similar to explosion of test data generation, interval decomposition may cause exponential explosion of boxes. Certain strategies to choose variables to apply interval decompositions, and how to decompose intervals are crucial in practice.

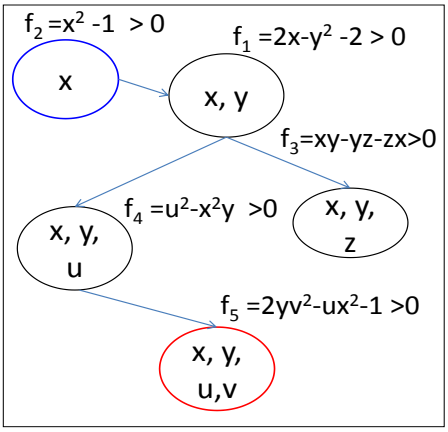

Fig. 5. Dynamically sorted APIs for test data generation

Selecting Intervals to Decompose The choice of intervals to decompose consists of 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 4.1. When an API $f_j > 0$ refutes all generated test data, it returns Test-UNSAT. Then, variables appearing in f_j are candidates for interval decompositions, 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 interval decompositions.

For (b), among variables in an API $f_j > 0$, we further filter variables that have sensitivity (Example 2) beyond a threshold, 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 Balanced decomposition decomposes an interval into two intervals exactly half.

Definition 11. *For an interval $x \in (a, b)$, a balanced decomposition is*

$$D_b(x \in (a, b)) = \{x \in (a, \frac{a+b}{2}), x \in (\frac{a+b}{2}, b)\}$$

Monotonic decomposition introduces bias δ to an interval decomposition, if a value of a corresponding variable monotonically affects on a value of a polynomial.

Definition 12. *Let $f(x_1, \cdots, x_k)$ be a polynomial, a variable x_i $(1 \leq i \leq k)$ is monotonically increasing in f if $\forall x_i' \geq x_i''$ implies $f(x_1, \cdots, x_i', \cdots, x_k) \geq f(x_1, \cdots, x_i'', \cdots, x_k)$, and is monotonically decreasing in f if $\forall x_i' \geq x_i''$ implies $f(x_1, \cdots, x_i', \cdots, x_k) \leq f(x_1, \cdots, x_i'', \cdots, x_k)$. Pos_f and Neg_f denote the sets of monotonically increasing and decreasing variables of a polynomial f, respectively.*

Definition 13. *Let $x \in (a, b)$ and $\delta < b - a$ for a bound δ. A monotonic decomposition on $x \in (a, b)$ is,*

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

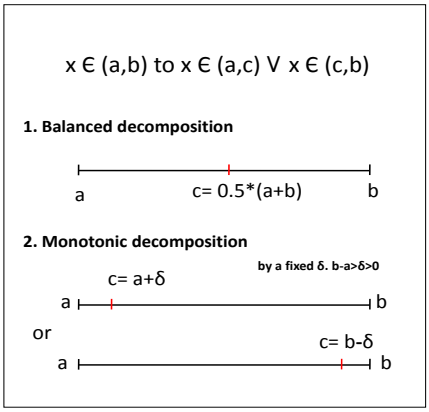

Fig. 6. Interval decomposition

We apply δ in Definition 13 (heuristic rule) as the bias δ, by regarding δ as a unit of searching. For a balanced decomposition, SAT solver will choose an arbitrary combination of input ranges. However, for a monotonic decomposition, we would like 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 makes upper and lower bounds of the polynomial f increase, and provide more chances to lead f satisfiable. MiniSAT 2.2 chooses literals by the "activity" measure, and we manually modify the activity of literals corresponding to a narrower sub-interval.

5 raSAT Implementation and 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 [10] shows the strength of **nlSAT** (equivalently, Z3 4.3). Note that our comparison is only on polynomial inequality.

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.

5.1 raSAT execution example

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. 7. 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. 7 (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.

5.2 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

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

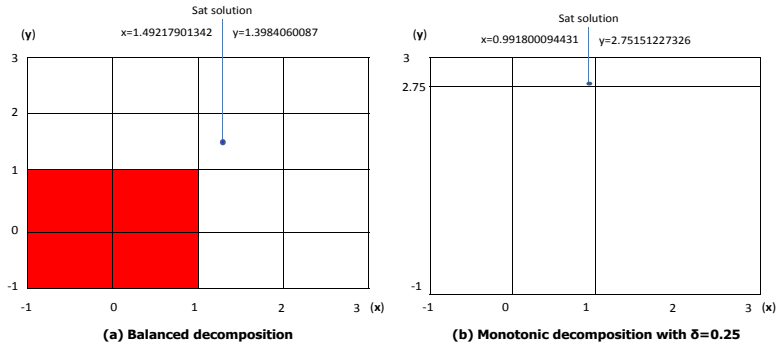

Fig. 7. Interval decompositions by **raSAT**

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 4.2). 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.

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,

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

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 $\psi = x_1^n + x_2^n < 1 \wedge (x_1 - r)^n + (x_2 - r)^n < 1$

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.

5.3 Benchmarks from SMT-LIB

In SMT-LIB [22], 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 [10], 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.

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

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 $\psi = \psi_1 \wedge \psi_2$

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

– **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 [10], 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**).

6 Related Work

Current decision procedures for solving polynomial constraints can be classified into one of five categories, or combinations among them.

1. **QE-CAD**. RAHD [23] is based on the core computation of QE-CAD proposed by Tarski. It applies different versions of QE-CAD implementations such as QEPCAD-B, Reduce/Redlog (and we expect Mathematica). Since

QE-CAD is DEXPTIME w.r.t. the number of variables, scalability is still challenging.

2. **Virtual substitution (VS)**. Virtual substitution is an EXPTIME algorithm applicable when the degree of each variable does not exceed 4. SMT-RAT toolbox [24][25] combines VS [26] and incremental DPLL with less lazy and eager theory propagation. Z3 (version 3.1) [27], the winner in the category QF_NRA of SMT competition in 2011, combines VS, ICP, and linearization.
3. **Interval constraint propagation (ICP)**. ICP applies interval arithmetic as an over-approximation for checking consistency in background theories. RSOLVER [11] and iSAT [12] are such examples. To remove unsatisfiable elements, while RSOLVER develops a pruning algorithm, iSAT apply a tight interaction of SAT solver and eager theory propagation. These approaches overlap with ours. Differences are, we apply Affine interval arithmetic (e.g., AF_1, AF_2, CAI), instead of classical interval in RSOLVER and iSAT. In addition to over-approximation (interval arithmetic), we also apply under-approximation (testing), which may find SAT instances and also it will guide next refinement steps and focus of an API.
4. **Bit-blasting**. MiniSmt [13] applies bounded bit encoding for rational numbers and extends symbolic representations for some fragments of real numbers. MiniSmt can show SAT quickly, but due to the bounded bit encoding, it cannot conclude UNSAT. UCLID [28] represents an input formula by a bit-vector formula on a given finite bit width. UCLID also applies both over- and under-approximation to refine each other. Reducing number of bits to represent an input formula is an under-approximation, and removing some clauses is an over-approximation. UCLID aims to finite-precision non-linear integer arithmetic, not for real numbers. Bit-blasting also suffers a lot when the degree of polynomials increases.
5. **Linearization**. Barcelogic [14] linearizes polynomial constraints on integers by exhaustive case analyses, which instantiate one of arguments in multiplication with all possible integers in a given-bound. CORD [15] uses another linearization, called CORDIC (COrdinate Rotation DIgital Computer). Both Barcelogic and CORD apply Yices for solving linear constraints. Linearization also suffers a lot when the degree of polynomials increases.

7 Conclusion

This paper presented an iterative approximation refinement, called **raSAT** loop, which is used for solving polynomial inequality constraints on real numbers. Experiments on simple benchmarks to estimate effects of input measures (the degree of polynomials, the number of variables, and the number of atomic polynomial constraints), and on benchmarks of the QF_NRA category in SMT-LIB showed that **raSAT** is comparable and sometimes outperforms Z3 4.3, which is believed one of the strongest SMT on the QF_NRA category.

7.1 Observation and Discussion

From experimental results in Section 5.2 and 5.3, we observe the followings.

- The degree of polynomials will not affect much.
- The number of variables are matters, but also for Z3 4.3. The experimental results do not show exponential growth, and we expect the strategy of selection of an API in which related intervals are decomposed seems effective in practice. By observing Zankl examples, we think the maximum number of variables of each API seems a dominant factor.
- Effects of the number of APIs are not clear at the moment. In simple benchmarks, **raSAT** is faster than Z3 4.3, however we admit that we have set small degree $n = 6$ for each API.

For instance, *matrix-2-all-5,8,11,12* in Zankl contain a long monomial (e.g., 60) with the max degree 6, and relatively many variables (e.g., 14), which cannot be solved by Z3 4.3, but **raSAT** does. As a general feeling, if an API contains more than $30 \sim 40$ variables, **raSAT** seems saturating. We expect that, adding to a strategy to select an API (Section 4.2), we need a strategy to select variables in the focus. We expect this can be designed with sensitivity (Example 2) and would work in practice. Note that sensitivity can be used only with noise symbols in Affine intervals. Thus, iSAT and RSOLVER cannot use this strategy, though they are based on IA, too.

7.2 Future Work

We are just in the beginning, and have lots of future work in both development of **raSAT** and its applications.

Extension of raSAT loop

- **Equality handling**: currently, **raSAT** loop can handle only inequalities. Before applying ideal based technique, such as *Gröbner basis*, we are planning to implement a non-constructive detection of equality by *intermediate value theorem*.
- **Solving polynomial constraints on integers**: In integer domain, the number of test data is finite if interval constraints are bounded. Then, Test-UNSAT implies UNSAT if all possible test data are generated. A tight interaction between testing and interval decomposition could be investigated. Mixed integers are also challenging.

raSAT Development

- **Avoiding local optimal**: we borrow an idea of *restart* in MiniSAT for escaping from hopeless local search (i.e., solution set is not dense or empty). *Heuristics* would be, after a deep interval decomposition of a box and Test-UNSAT are reported, backtrack occurs to choose a randomly selected box.

- **Separation of linear constraints**: Many benchmarks contain linear constraints. Current implementation does not have any tuning, but **raSAT** loop only. Practically, separating linear and non-linear constraints and solving them in a coordinated way between Presbuger arithmetic and **raSAT** would improve. During this separation, variables of intersecting linear constraints would be candidates for interval decompositions.
- **Incremental DPLL**: For interactions with the SAT solver, we currently apply the very lazy theory learning. Combination with *eager* theory propagation would improve, in which we can propagate a conflict from a partial truth assignment instead of waiting for a full truth assignment obtained by SAT solver.

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] Tarski, A.: A decision method for elementary algebra and geometry. Bulletin of the American Mathematical Society **59** (1951)

[9] 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

[10] 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

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

[12] 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

[13] 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
[14] 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
[15] 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
[16] Stolfi, J.: Self-Validated Numerical Methods and Applications. PhD thesis, PhD. Dissertation, Computer Science Department, Stanford University (1997)
[17] Khanh, T.V., Ogawa, M.: SMT for polynomial constraints on real numbers. Electr. Notes Theor. Comput. Sci. **289** (2012) 27–40
[18] 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
[19] Moore, R.E.: Interval Analysis. Prentice-Hall (1966)
[20] Messine, F.: Extensions of affine arithmetic: Application to unconstrained global optimization. Journal of Universal Computer Science **8**(2) (2002)
[21] Comba, J.L.D., Stolfi, J.: Affine arithmetic and its applications to computer graphics. In: Proceedings of VI SIBGRAPI. (1993) 9–18
[22] Barrett, C., Stump, A., Tinelli, C.: The satisfiability modulo theories library (SMT-LIB). (2010)
[23] 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
[24] 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
[25] 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
[26] Weispfenning, V.: Quantifier elimination for real algebra - the quadratic case and beyond. Applicable Algebra in Engineering, Communication and Computing **8** (1997) 85–101
[27] Microsoft: Z3
[28] 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