# SMT for Polynomial Constraints on Real Numbers

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

*School of Information Science*
*Japan Advanced Institute of Science and Technology*
*Ishikawa, Japan*

**Abstract**

This paper preliminarily reports an SMT for solving polynomial inequalities over real numbers. Our approach is a combination of interval arithmetic (over-approximation, aiming to decide unsatisfiability) and testing (under-approximation, aiming to decide Satisfiability) to sandwich precise results. Addition to existing interval arithmetic's, such as classical intervals and affine intervals, we newly design Chebyshev Approximation Intervals, focusing on multiplications of the same variables, like Taylor expansions. When it decides neither Satisfiability nor unsatisfiability, this framework enables us a refinement loop by splitting input ranges into smaller ones (although this refinement loop implementation is left to future work). Preliminary experiments on small benchmarks from SMT-LIB are also shown.

*Keywords:* interval arithmetic, affine arithmetic, SAT modulo theories - SMT, polynomial constraints, testing.

## 1 Introduction

Solving polynomial constraints plays an important role in program verification, e.g., roundoff/overflow error detection [14], termination proving [8], hybrid systems, loop invariant generation, and parameter design of control.

Tarski proved that polynomial constraints over real numbers (algebraic numbers) is decidable [17], and later Collins proposed Quantifier Elimination by Cylindrical Algebraic Decomposition [19], which is nowadays implemented in Mathematica, Maple/SynRac, Reduce/Redlog, and QEPCAD. However, it is DEXPTIME wrt the number of variables, and works fine in practice up to 5

[1] Email: khanhtv@jaist.ac.jp
[2] Email: mizuhito@jaist.ac.jp

variables and lower degrees. 8 variables with degree 10 already require 20-30 hours by supercomputer.

An alternative approach is approximation, which is typically implemented as SMTs, which participate in QF_NRA category of SMT-Comp, e.g., iSAT [7], MiniSmt [18], Barcelogic [1], and CVC3 [3]. Among them, iSAT aims to detect unsatisfiability, and others aim to detect Satisfiability of polynomial constraints.

This paper preliminarily reports an SMT for solving polynomial inequalities over real numbers. Our approach is a combination of interval arithmetic (over-approximation, aiming to decide unsatisfiability) and testing (under-approximation, aiming to decide Satisfiability) to sandwich precise results. Addition to existing interval arithmetic's, such as classical intervals and affine intervals, we newly design Chebyshev Approximation Intervals (called CAI1 and CAI2), focusing on multiplications of the same variables, like Taylor expansions. Chebyshev approximation in interval arithmetic is not new, but we newly introduce noise symbols for absolute values.

We apply very lazy theory learning [13] for interaction with MiniSat 2.2. Initially, CNF given to SAT solver describes possible combination of input ranges. If interval arithmetic reports unsatisfiable, some combinations are removed for next SAT searching by memorizing them as learnt clauses to SAT solver. If interval arithmetic finds neither Satisfiable nor unsatisfiable, each polynomial is examined by random testing.

If testing cannot find a satisfiable instance, it is memorized as unsatisfiable by test (test-UNSAT) as heuristics, and removed from the following search.

When SMT decides neither Satisfiability nor unsatisfiability, this framework enables us a refinement loop by splitting input ranges into smaller ones (although this refinement loop implementation is left to future work).

The structure of paper is organized as follows. Section 2 describes the polynomial constraints and theory learning strategy in terms of abstract DPLL [13]. Section 3 explains variations of interval arithmetic and newly propose Chebyshev Approximation Intervals, CAI1, CAI2. Section 4 describes testing strategy. The framework of our SMT solver is described with examples in section 5. Preliminary experiments on small examples from SMT-LIB benchmark [2] are reported in section 6. Section 7 discusses some related work, and Section 8 concludes the paper with future work.

## 2 Polynomial constraints and Abstract DPLL

Among polynomial constraints over real numbers, our current target problem is *satisfiable* problem of *polynomial inequality constraints*, as in Definition 2.1. Handling polynomial equality's is left to future work. We assume input ranges are given by intervals (as in the most of SMT-LIB benchmark).

**Definition 2.1** A polynomial inequality constraint is in the form of

$$\bigwedge(\exists x_1 \in [l_1, h_1] \cdots x_, \in [l_n, h_n].\ \bigwedge_j^m f_j(x_1, \cdots, x_n) > 0)$$

where $l_i, h_i \in \mathbb{R}$ and $f_j(x_1, \cdots, x_n)$ is a polynomial over variables $x_1, \cdots, x_n$.

`Satisfiability Modulo Theories (SMT)` is a procedure to detect satisfiable instances under theory. A typical arithmetic theory is Presburger arithmetic (linear arithmetic) over integers and real numbers. It decomposes a problem into SAT solving as case analysis and theory as arithmetic conjunctive constraint solving. Interaction between SAT solving and theory has `Lazy` and `Eager` strategies, which are described below as *Abstract DPLL modulo theories* [13].

- *Very lazy theory learning* interacts with theory when an SAT instance is found, and learns a clause $l_1 \vee ... \vee \neg l_n \vee \neg l$ when theory refutes $l_1 \wedge ... \wedge l_n \wedge l$.

$$MlM_1 \parallel F \Longrightarrow \emptyset \parallel F \wedge (\neg l_1 \vee ... \vee \neg l_n \vee \neg l) \qquad \text{if} \begin{cases} MlM_1 \models F \\ \{l_1, ..., l_n\} \subseteq M \\ l_1 \wedge ... \wedge l_n \models_T \neg l \end{cases}$$

  where $F$ is a CNF, $l_i$ is a literal, and $\models_T$ is reasoning by theory $T$.
- *Eager theory propagation* interacts with theory during DPLL procedure of SAT, and DPLL procedure continues when theory admits the current decisions.

$$Ml \parallel F \Longrightarrow Ml \parallel F \qquad \text{if} \begin{cases} M \models_T l \\ \text{l is undefined in M} \\ l \text{ or } \neg l \text{ occurs in F} \end{cases}$$

We adopt *very lazy theory learning* on `MiniSat2.2`, since *eager theory propagation* requires tighter interaction between SAT solver and theory, which needs internal modification of MiniSat, and it naturally memorizes UNSAT combination of input ranges for a polynomial as a learnt clause.

## 3 Interval Arithmetic

Interval arithmetic (IA) estimates bounds of polynomials under given input ranges, and we use it as an over-approximation theory. For a closed existential polynomial constraint

$$C = \exists x_1 \in [l_1, h_1] \cdots x_k \in [l_k, h_k]\ .\ \bigwedge_i f_i(x_1, \cdots, x_k) > 0,$$

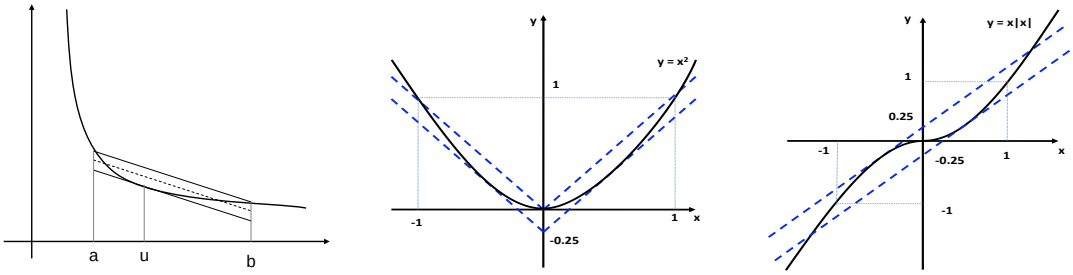


Fig. 1. Chebyshev approximation with noise symbols for absolute values

we denote $\vdash_{IA} C$ if the estimation of each $f_i(x_1, \cdots, x_k)$ contains positive results. Then, $\vdash C$ implies $\vdash_{IA} C$ holds, but not vice versa. We can safely infer $\vdash C$ from $\vdash_{IA} C$ only when $\forall x_1 \in [l_1, h_1] \cdots x_k \in [l_k, h_k]$ . $\bigwedge_i f_i(x_1, \cdots, x_k) > 0$ is estimated by IA. We say

- $C$ is *SAT* by IA if $\forall x_1 \in [l_1, h_1] \cdots x_k \in [l_k, h_k]$ . $\bigwedge_i f_i(x_1, \cdots, x_k) > 0$ is estimated by IA, and
- $C$ is *UNSAT* by IA if $\neg\exists x_1 \in [l_1, h_1] \cdots x_k \in [l_k, h_k]$ . $\bigwedge_i f_i(x_1, \cdots, x_k) > 0$ is estimated by IA.

$\vdash_{IA} C$ only is interpreted as $C$ to be *unknown*.

A popular example of IA is Classical Interval (CI) [9], which keeps a lower bound and an upperbound. 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 (AF) introduces *noise symbols* $\epsilon$, which is interpreted as some value in $[-1, 1]$ [4,5,6], for partial symbol manipulation. For instance, $x \in [2, 4]$ is represented as $x = 3 + \epsilon$, and $x - x = (3 + \epsilon) - (3 + \epsilon)$ is safely evaluated to 0. The drawback is that the multiplication without dependency may be less precise than CI. For instance, let $x \in [2, 4]$ and $y \in [3, 7]$. Then $x = 3 + \epsilon$ and $y = 5 + 2\epsilon'$, and $xy = 15 + 5\epsilon + 6\epsilon' + 2\epsilon\epsilon'$. Choices are,

- $\epsilon\epsilon'$ is replaced with a fresh noise symbol [4,5],
- $\epsilon\epsilon'$ is replaced with $[-1, 1]\epsilon$ (or $[-1, 1]\epsilon'$), called Extended Affine Interval (EAI) [14], and
- $\epsilon\epsilon'$ is pushed into the fixed error noise symbol$\epsilon_{\pm}$, denoted AF1 [10].

Either of treatments estimates that $xy$ is in $[2, 28]$, whereas CI results $[6, 28]$. We will use the last choice as default except for AF.

We newly design Chebyshev Approximation Interval (CAI1, CAI2) and implement Classical Interval (CI), Affine Intervals (AF, AF1, AF2) [10], and

ChebyshevApproximation Intervals (CAI1, CAI2). Their forms are, e.g.,

$$\text{AF1} \quad \hat{x} = a_0 + \sum_{i=1}^{n} a_i \epsilon_i + a_{n+1}\epsilon_{\pm}$$
$$\text{AF2} \quad \ddot{x} = a_0 + \sum_{i=1}^{n} a_i \epsilon_i + a_{n+1}\epsilon_{+} + a_{n+2}\epsilon_{-} + a_{n+3}\epsilon_{\pm}$$
$$\text{CAI1} \quad \mathring{x} = \bar{a}_0 + \sum_{i=1}^{n} \bar{a}_i \epsilon_i + \sum_{i=1}^{n} \bar{a}_{i+n}\epsilon_{i+n} + \bar{a}_{2n+1}\epsilon_{\pm}$$

where $\epsilon_+ \in [0,1]$, $\epsilon_- \in [-1,0]$, $\epsilon_\pm \in [-1,1]$ is the error noise symbol and $\epsilon_{i+n}$ represents the absolute value $|\epsilon_i|$ of $\epsilon_i$. Ideas behind are,

(i) introduction of noise symbols [4,5,10],

(ii) keeping products $\epsilon_i\epsilon_j$ of noise symbols up to degree 2 [10] (beyond degree 2, products are pushed into the error noise symbol $\epsilon_\pm$), and

(iii) Chebyshev approximation of $x^2$ with noise symbols for absolute values.

(iii) comes from the observation that, for $x \in [-1,1]$,

$$|x| - \frac{1}{4} \le x^2 = |x|^2 \le |x| \quad \text{and } x - \frac{1}{4} \le x|x| \le x + \frac{1}{4}$$

which are explained in Figure 1. This observation leads symbolic manipulation on products of the same noise symbol $\epsilon$ as

$$\epsilon \times \epsilon = |\epsilon| \times |\epsilon| = |\epsilon| + [-\frac{1}{4}, 0] \quad \text{and} \quad \epsilon \times |\epsilon| = \epsilon + [-\frac{1}{4}, \frac{1}{4}].$$

**Remark 3.1** Introduction of Chebyshev approximation is not new. For instance, [16] proposed it based on the mean-value theorem, as in the left of Figure 1. [12] applied not only for products of the same noise symbols but also those of different noise symbols. However, their estimation on $x^2$ is only in the positive interval using the fact $x - \frac{1}{4} \le x^2 \le x$ for $x \in [0,1]$. We newly introduce noise symbols for absolute values. The advantage is, coefficients are half compared to them, which reduce the effect of the offset $[-\frac{1}{4}, 0]$. Currently, we only focus on products of the same noise symbols, which is useful for computation like in Taylor expansion.

Roughly speaking, AF and AF1 apply (i) only, AF2 applies (i) and (ii) [11], CAI1 applies (i) and (iii), and CAI2 applies all. The definitions of CAI1 arithmetic are found in Appendix.

**Example 3.2** Given $f = (x^2 - 2y^2 + 7)^2 + (3x + y - 5)^2$ with $x \in [-1,1]$ and $y \in [-2,0]$, the bound of $f$ computed by AF1, AF2, CAI1 and CAI2 are as follows:

- *AF1:* $[-98, 220]$
- *AF2:* $[-53, 191]$
- *CAI1:* $[-4.6875, 163.25]$
- *CAI2:* $[3.3125, 147.25]$

**Example 3.3** Given $sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!}$ with $x \in [0, 0.523598]$, the bounds of $sin(x)$ are as follows:

- AF1 : $10^{-6}[-6290.49099241,\ 523927.832027]$
- AF2 : $10^{-6}[-6188.00580507,\ 514955.797111]$
- CAI1: $10^{-6}[-1591.61467700,\ 503782.471931]$
- CAI2: $10^{-6}[-1591.61467700,\ 503782.471931]$

In the example 3.2, CAI2 gives the best bound comparing with CAI1, AF2 and AF1 because it can keep information about $\epsilon_i\epsilon_j$. The example 3.3 is *Taylor expansion* of $sin(x)$. Bounds of $sin(x)$ are estimated for x ranged from 0 to $\frac{\pi}{6}$. In this example, CAI1 and CAI2 give the same bound better than AF1 and AF2.

## 4 Testing

Testing is a popular methodology to find satisfiable instances. For real numbers, only finitely many instances can be tested, and we use it as an under-approximation theory. For a closed existential polynomial constraint

$$C = \exists x_1 \in [l_1, h_1] \cdots x_k \in [l_k, h_k]\ .\ \bigwedge_i f_i(x_1, \cdots, x_k) > 0$$

and finite set $\Theta$ of substitutions, we denote $\vdash_{test(\Theta)} C$ if $\bigwedge_i f_i(\theta(x_1), \cdots, \theta(x_k)) > 0$ holds for some $\theta \in \Theta$ with $\theta(x_1) \in [l_1, h_1] \cdots \theta(x_k) \in [l_k, h_k]$. Then, $\vdash_{test_\Theta} C$ implies $\vdash C$ holds, but not vice versa. We say

- $C$ is *SAT if* $\vdash_{test_\Theta} C$, *and*
- $C$ *is* test-UNSAT if $\neg \bigwedge_i f_i(\theta(x_1), \cdots, \theta(x_k)) > 0$ for each $\theta \in \Theta$ such that $\theta(x_1) \in [l_1, h_1] \cdots \theta(x_k) \in [l_k, h_k]$.

Test-UNSAT does not imply UNSAT, but we will use it as heuristics to focus on more likely target ranges of inputs during SMT-procedure (Section 5). This is useful since reducing the number of test data is a serious matter For instance, consider if we have 10 variables, then just 2 test data for each becomes $2^{10}$ instances as a total.

There are two immediate strategies to generate test data.

**Definition 4.1** For an interval $[l, h]$ and $k \geq 1$,

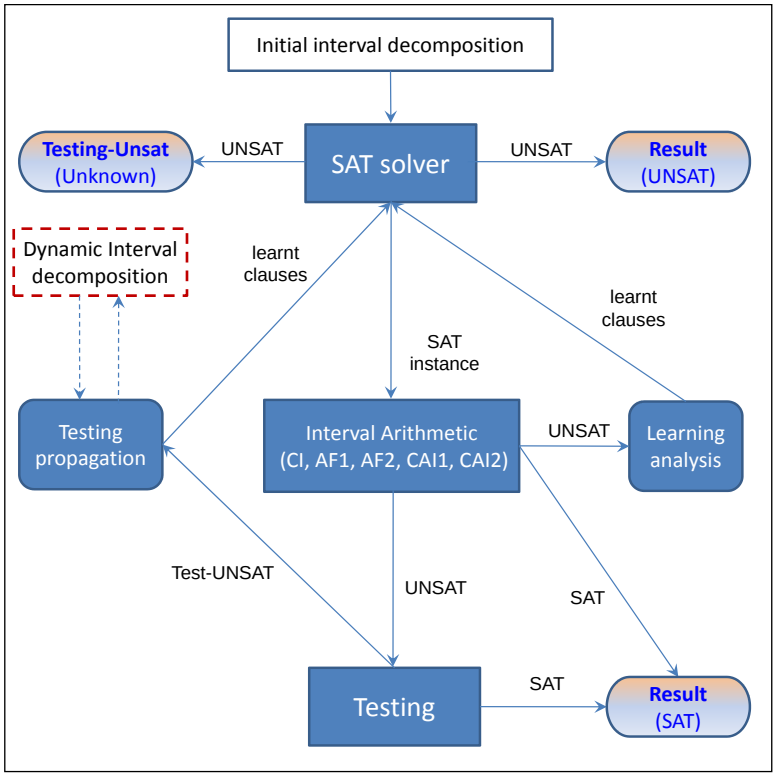


Fig. 2. Framework of SMT solver

- the $k$-**random ticks** are $\{c_1, \cdots, c_k\}$, and
- the $k$-**periodic ticks** are $\{c, c+\Delta, \cdots, c+(k-1)\Delta\}$,

where $\Delta = \frac{h-l}{k}$, and $c \in [l, l+\Delta]$, $c_i \in [l+(i-1)\Delta, l+i\Delta]$ are randomly generated (with $i \in \{1, ..., k\}$).

## 5 SMT on polynomial inequality constraints

The main idea of our SMT solver is applications of two theories, IA (CI, AF1, AF2, CAI1, CAI2) for over-approximation and testing for under-approximation to sandwich the precise results. Although currently not implemented yet, we plan an automatic decomposition of input ranges to refine the detected results as in [15]. Fig 2 describes its design framework.

- `Initial interval decomposition`: An interval of a variable is split into *small intervals*, which are represented as disjunction. For instance, $x \in [a, b]$ is represented by $x \in [a, a_1] \vee x \in [a_1, a_2] \vee \cdots \vee x \in [a_n, b]$ for $a < a_1 < a_2 < \cdots < a_n < b$. After encoding $x \in [a_i, a_{i+1}]$ and a polynomial $f_i(x_1, \cdots, x_k) > 0$ (initially, not appearing in CNF) by atomic propositions, we obtain CNF, which is sent to SAT solver.
- `SAT solver`: We use MiniSat2.2 as a backend SAT solver. SAT solver

finds a satisfiable combination of input ranges of all variables. A satisfiable instance is sent to IA to check either SAT, UNSAT, or unknown. If SAT, we have done. If UNSAT, the negation of the combination is added to CNF as a learnt clause, and SAT solver returns next satisfiable combination. If unknown, the combination is sent to testing.

- `Interval Arithmetic`: We implement CI, AF1, AF2, CAI1 and CAI2 as IA. IA decides each polynomial $f_i(x_1, \cdots, x_k) > 0$ appearing in a constraint either SAT, UNSAT, or *unknown*, under given input ranges. If some of them are UNSAT, we return UNSAT to SAT solver as learnt clauses. If each of them is SAT, we have done. If some of them remains *unknown*, all unknown polynomials are sent to testing (still memorizing polynomials detected to be SAT).
- `Testing`: In current implementation, 2-random ticks are generated for each variable to test a polynomial $f_i(x_1, \cdots, x_k) > 0$. If a polynomial is SAT for a test instance, we have done. If it cannot find a successful test instance, it returns *test-UNSAT*.
- `Testing propagation`: When testing of polynomials returns *test-UNSAT*, the negation of the combination of input ranges is added to CNF as a learnt clause. This is heuristics to narrow the search and intends to find SAT for next evaluation. If none of SAT instances are found, we conclude *unknown* for this input range decomposition.

  The SMT solver will perform `Dynamic interval decomposition` to split into smaller input ranges, and refine the search. In current implementation, `Dynamic interval decomposition` is left to future work.

**Example 5.1** Fig 3 describes how the SMT solver works on a polynomial constraint $\exists x \in [-1, 4]\ y \in [-1, 4]\ .\ 4x + 3y - xy > 12$. Its input format is

```
x = [-1,4]
y = [-1,4]
(assert (f = 4x + 3y -xy > 12))
```

First, by *Initial interval decomposition*, the input ranges $[-1, 4]$ and $[-1, 4]$ of variables $x$ and $y$ are split to 5 small input ranges. By IA, the red areas ($x \in [-1, 2]$ and $y \in [-1, 3]$) are detected to be UNSAT. The remaining areas remain white, which means *unknown*. Then, testing is applied, for instance on $x \in [3, 4]$ and $y \in [1, 2]$, and fortunately finds a satisfiable instance with $x = 3.33821$ and $y = 1.31143$.

With *Dynamic interval decomposition*, for instance the area $x \in [2, 3]$ and $y \in [-1, 0]$ is split into quarters. By IA, two left quarters are detected to be UNSAT. Similarly, in the area $x \in [3, 4]$ and $y \in [3, 4]$, the right below quarter is detected to be SAT (blue) by IA.

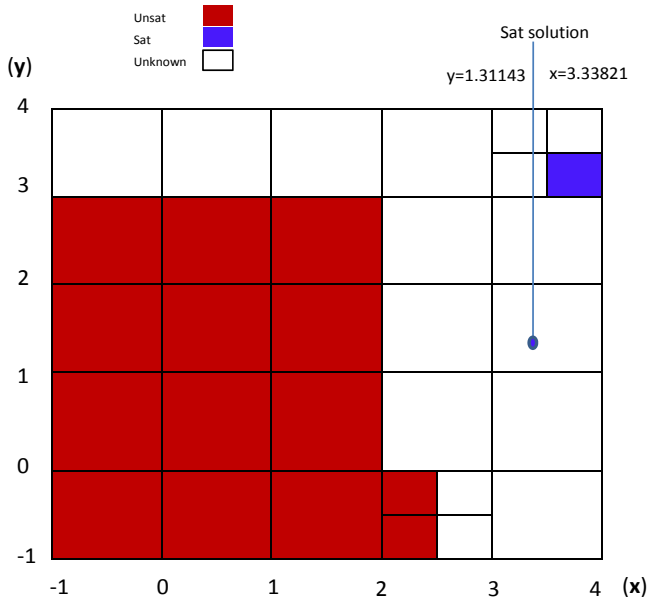


Fig. 3. Solver working on P1

## 6 Preliminary experiment

In this section, preliminary results with the problem P1

$$\exists x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10} \in [0, 3]\ x_{11} \in [-3, 2]\ x_{12} \in [-1, 3].$$
$$x_1x_3 - x_1x_7 \geq 0\ \wedge\ x_1x_2 - x_1x_6 > 0\ \wedge x_1x_3 - x_3 \geq 0\ \wedge x_1x_2 - x_4 \geq 0\ \wedge$$
$$x_7 - x_3 \geq 0\ \wedge\ x_6 - x_2 > 0\ \wedge\ x_8 + x_6x_9 - x_{10} > 0\ \wedge\ x_3x_9 - x_7x_9 \geq 0\ \wedge$$
$$x_2x_9 - x_6x_9 > 0\ \wedge\ x_{11}^3 - 2x_{11}^2(1 + x_{12}^2) - 2x_{12}(x_{11} + x_{12}) + x_{12} - 6.5 > 0$$

and 18 problems in the division QF_NRA of SMT-LIB [2] benchmark. We choose problems with up to 20 variables.

P1 was checked by CI, AF1, AF2, CAI1 and CAI2. Both CAI1 and CAI2 detect *UNSAT* while other three IAs return *unknown*. The *Initial interval decomposition* divides given ranges of variables into ranges with the width 1.

Division QF_NRA of SMT-LIB benchmarks consists of a family *zankl*, which comes from termination analysis of term rewriting. All variables in problems are originally set a lower bound with $\geq 0$. In this experiment, we set an upper bound for these variable and evaluate these problems with a range $[0, 2.5]$. The range $[0, 2.5]$ is split to 5 ranges with the width 0.5.

We apply IA depending on the number of variables in a problem, e.g., CAI2 for $< 10$, CAI1 for $\leq 15$ *(except problem matrix-1-all-21)* and AF1, AF2

| Problem name | No. variables | No. constraints | Interval arithmetic | Result | Time (s) |
|---|---|---|---|---|---|
| P1 | 13 | 10 | $CI$ | unknown | 0.031 |
| P1 | 13 | 10 | $AF1$ | unknown | 0.109 |
| P1 | 13 | 10 | $AF2$ | unknown | 0.046 |
| P1 | 13 | 10 | $CAI1$ | UNSAT | 0.796 |
| P1 | 13 | 10 | $CAI2$ | UNSAT | 124.89 |
| matrix-1-all-01 | 19 | 22 | $AF2$ | unknown | 0.093 |
| matrix-1-all-2 | 14 | 9 | $CAI1$ | sat | 8.328 |
| matrix-1-all-3 | 19 | 21 | $AF1$ | sat | 175.968 |
| matrix-1-all-4 | 16 | 20 | $AF2$ | sat | 20.328 |
| matrix-1-all-11 | 19 | 7 | $AF1$ | sat | 17.687 |
| matrix-1-all-14 | 14 | 16 | $CAI1$ | sat | 66.484 |
| matrix-1-all-15 | 10 | 14 | $CAI1$ | unknown | 26.656 |
| matrix-1-all-18 | 6 | 10 | $CAI2$ | sat | 14.156 |
| matrix-1-all-20 | 16 | 16 | $AF2$ | sat | 1.062 |
| matrix-1-all-21 | 13 | 17 | $AF1$ | sat | 2753.72 |
| matrix-1-all-24 | 11 | 12 | $CAI1$ | unknown | 50.828 |
| matrix-1-all-33 | 13 | 6 | $CAI1$ | sat | 68.765 |
| matrix-1-all-34 | 20 | 14 | $AF2$ | sat | 3349.89 |
| matrix-1-all-36 | 18 | 19 | $AF2$ | sat | 54.015 |
| matrix-1-all-37 | 19 | 46 | $AF2$ | unknown | 3730.66 |
| matrix-1-all-39 | 19 | 23 | $AF2$ | unknown | 85.781 |
| matrix-1-all-43 | 16 | 9 | $AF2$ | unknown | 0.343 |
| matrix-2-all-6 | 17 | 10 | $AF2$ | unknown | 15.75 |

Table 1
Experimental results with P1 and QF_NRA

for $> 15$, due to efficiency reason of preliminary implementation. Efficiency of CAI1 can be compared to AF1 and AF2, but CAI2 is much slower.

Table 1 includes 6 columns of the problem name, number of variables, number of constraints, type of interval arithmetic, result, and time in second.

## 7 Related work

There are several solvers which aim polynomial constraints e.g., iSAT [7], MiniSmt [18], Barcelogic [1] and CVC3 [3].

iSAT [7] applies CI for propagating theory information. Instead of lazy theory learning, iSAT applies a tight interaction of SAT solver and eager theory propagation. Thus, conflict detecting and propagated implication from theory are directly applied SAT solver to provide new assignments. iSAT splits ranges into smaller ranges until finding a satisfiable solution or terminating after reaching to the bound of splitting. Due to the lack of under-approximation such as testing, iSAT can show unsatisfiability, but cannot conclude true Satisfiability.

MiniSmt [18] applies bit encoding of non-linear arithmetic over integers under given bounds, and reduced to SAT solving. First, solving constraints over rational numbers is defined and then fixed number of real numbers introduced symbolically. MiniSmt can show Satisfiability quickly, but due to given bounds of the search, it cannot conclude unsatisfiability.

Barcelogic [1] focus on integers. Non-linear constraints are linearized and linear constraints are solved by Presburger arithmetic. CVC3 [3] is also a popular SMT, which can handle non-linear arithmetic. However, we could not find references that provide its technical details.

## 8 Conclusion and Future work

This paper preliminarily reported an SMT for solving polynomial inequalities over real numbers. Our approach is a combination of interval arithmetic (over-approximation, aiming to decide unsatisfiability) and testing (under-approximation, aiming to decide Satisfiability) to sandwich precise results. Addition to existing interval arithmetic's, such as classical intervals and affine intervals, we newly designed Chebyshev Approximation Intervals.

Interval arithmetic can indicate *unsatisfiable areas* and remove these areas from search space. Testing only focus on areas that solver decides neither satisfiable nor unsatisfiable. When testing cannot find a satisfiable instance in an area, *heuristics* is applied to make the solver not to search that area again.

Preliminary experiments on small examples from SMT-LIB contain precise detection of satisfiable examples and unsatisfiable examples. To our restricted knowledge, our SMT implementation firstly enables us to detect both.

Our status is preliminary, and there are lots to do for future work.

- `Test data generation strategy`: When the number of variables becomes large, the number of test data to generate is a serious matter. Fortunately, interval arithmetic with noise symbols keeps sensitivity on variables. For instance, if an input range of $x_i$ is described by a noise symbol $\epsilon_i$, the coefficient of $\epsilon_i$ in the result reflects strength of its influence. We can generate more test data for such sensitive variables. This was proposed in [15] under the program analysis context, and we hope to apply to our SMT.

- Dynamic interval decomposition and refinement loop: In Figure 2, dynamic interval composition is connected with dotted lines, which means not yet implemented. Depending on interval arithmetic and testing results, we can focus on areas more likely to be unsatisfiable or contain satisfiable instances. For instance, even if $f_i(\theta(x_1), \cdots, \theta(x_n))$ fails to be positive, we can expect that nearer to 0 would be nearer to satisfiable instances. If the result of interval arithmetic has smaller overlap with positive values, it is more likely to be unsatisfiable. This kind of refinement loop was proposed in [15] under the program analysis context, and we hope to apply to our SMT.
- Polynomial equality Currently, we can handle polynomial inequality only. However, for instance

$$\exists x_1 \in [l_1, h_1] \cdots x_, \in [l_n, h_n].\ \bigwedge_j^m f_j(x_1, \cdots, x_n) > 0\ \wedge\ g(x_1, \cdots, x_n) = 0$$

  can be decomposed into two phases. First, find some area (by interval arithmetic) such that

$$\forall x_1 \in [l_1, h_1] \cdots x_, \in [l_n, h_n].\ \bigwedge_j^m f_j(x_1, \cdots, x_n) > 0.$$

  and find two instances (by testing) such that $g(a_1, \cdots, a_n) > 0$ and $g(b_1, \cdots, b_n) < 0$. By Intermediate value theorem, we can conclude $\exists x_1 \in [l_1, h_1] \cdots x_, \in [l_n, h_n].\ g(x_1, \cdots, x_n) = 0$.
- `Scalability and practical experiments`: Scalability is very important in practice, and we expect the partial use of eager theory propagation will improve efficiency.

## References


[1] Borralleras, C., Salvador Lucas, Rafael Navarro-marset, Enric Rodriguez-carbonell and Albert Rubio, *Solving Non-linear Polynomial Arithmetic via SAT Modulo Linear Arithmetic*, "Proceedings of the $22^{nd}$ International Conference on Automated Deduction CADE-22," Lecture Notes in Computer Science, Vol. **5663** (2009), Springer-Verlag, 294–305.

[2] Barrett, C., Aaron Stump, and Cesare Tinelli, The Satisfiability Modulo Theories Library (SMT-LIB), URL: `http://www.smt-lib.org`, 2010.

[3] Barrett, C., and Cesare Tinelli, *CVC3*, "Proceedings of the $19^{th}$ International Conference on Computer Aided Verification (CAV '07)," Lecture Notes in Computer Science, Vol. **4590** (2007), Springer-Verlag, 298–302.

[4] Comba, J. L. D., and Jorge Stolfi, *Affine arithmetic and its applications to computer graphics*, Proceedings of VI SIBGRAPI (1993), 9–18.

[5] De Figueiredo, L., and Jorge Stolfi, "Self-Validated Numerical Methods and Applications," Brazilian Mathematics Colloquium monographs, IMPA/CNPq, Brazil, 1997.

[6] De Figueiredo, L., and Jorge Stolfi, *Affine Arithmetic: Concepts and Applications*, Numerical Algorithms, Vol. **37** (2004), 147–158.

[7] Franzle, M., Christian Herde, Tino Teige, Stefan Ratschan and Tobias Schubert, *Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure*, Journal on Satisfiability, Boolean Modeling and Computation, Vol. **1** (2007), 209–236.

[8] Lucas, S. and Rafael Navarro Marset, *Comparing CSP and SAT Solvers for Polynomial Constraints in Termination Provers*, Electron. Notes Theor. Comput. Sci., Vol. **206** (2008), 75–90.

[9] Moore, R. E., "Interval Analysis", Prentice Hall, 1966.

[10] Messine, F., *Extensions of affine arithmetic: Application to unconstrained global optimization*, Journal of Universal Computer Science, Vol. **8** (2002), 992-1015.

[11] Messine, F. and Ahmed Touhami,*A General Reliable Quadratic Form: An Extension of Affine Arithmetic*, Reliable Computing, Vol. **12** No. 3 (2006), 171-192.

[12] Miyajima, S., Takatomi Miyata and Masahide Kashiwagi, *A new dividing method in affine arithmetic*, IEICE Transactions, Vol. **E86-A** No. 9 (2003), 2192–2196.

[13] Nieuwenhuis, R., Albert Oliveras and Cesare Tinelli, *Abstract DPLL and abstract DPLL modulo theories*, "Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference LPAR04," Lecture Notes in Computer Science, Vol. **3452** (2005), Springer, 36–50.

[14] Ngoc, D., Mizuhito Ogawa, *Overflow and Roundoff Error Analysis via Model Checking*, "Proceedings of the 2009 Seventh IEEE International Conference on Software Engineering," SEFM 2009, IEEE Computer Society, 105–114.

[15] Ngoc, D., Mizuhito Ogawa, *Checking Roundoff Errors using Counterexample-Guided Narrowing*, "Checking Roundoff Errors using Counterexample-Guided Narrowing," ASE 2010, 301-304.

[16] Stolfi, J., "Self-Validated Numerical Methods and Applications," Ph. D. Dissertation, Computer Science Department, Stanford University, 1997.

[17] Tarski, A., *A decision method for elementary algebra and geometry*, University of California Press, 1951.

[18] Zankl, H., and Aart Middeldorp, *Satisfiability of non-linear (Ir)rational arithmetic*, "Proceedings of the 16th international conference on Logic for programming, artificial intelligence, and reasoning," LPAR'10 (2010), Springer-Verlag, 481–500.

[19] Caviness, B.F., Jeremy R. Johnson, "Quantifier Elimination and Cylindrical Algebraic Decomposition," Springer=Verlag, 1998.

## Appendix

**Definition of CAF1** Given $\mathring{x}$, $\mathring{y}$ are represented by CAI1 form:

$$\mathring{x} = \bar{a}_0 + \sum_{i=1}^{n} \bar{a}_i \epsilon_i + \sum_{i=1}^{n} \bar{a}_{i+n} \epsilon_{i+n} + \bar{a}_{2n+1} \epsilon_{\pm}$$

$$\mathring{y} = \bar{b}_0 + \sum_{i=1}^{n} \bar{b}_i \epsilon_i + \sum_{i=1}^{n} \bar{b}_{i+n} \epsilon_{i+n} + \bar{b}_{2n+1} \epsilon_{\pm}$$

and $\bar{c} = [-1, 1]$. Standard operations $\{\mathring{+}, \mathring{-}, \mathring{\times}\}$ of `CAI1 arithmetic` are defined as follows *(for simplicity we denote $\bar{a}\bar{b}$ for $\bar{a} \bar{\times} \bar{b}$)*:

- $\mathring{x} \mathring{+} \mathring{y} = (\bar{a}_0 \bar{+} \bar{b}_0) + \sum_{i=1}^{2n} (\bar{a}_i \bar{+} \bar{b}_i)\epsilon_i + (\bar{c}\bar{a}_{2n+1} \bar{+} \bar{c}\bar{b}_{2n+1})\epsilon_{\pm}$

- $\mathring{x} \mathbin{\mathring{-}} \mathring{y} = (\bar{a}_0 \mathbin{\bar{-}} \bar{b}_0) + \sum_{i=1}^{2n} (\bar{a}_i \mathbin{\bar{-}} \bar{b}_i)\epsilon_i + (\bar{c}\bar{a}_{2n+1} \mathbin{\bar{+}} \bar{c}\bar{b}_{2n+1})\epsilon_\pm$
- $\mathring{x} \mathbin{\mathring{\times}} \mathring{y} = K_0 + \sum_{i=1}^{n} (\bar{a}_0\bar{b}_i \mathbin{\bar{+}} \bar{a}_i\bar{b}_0 \mathbin{\bar{+}} \bar{a}_i\bar{b}_{i+n} \mathbin{\bar{+}} \bar{a}_{i+n}\bar{b}_i)\epsilon_i \mathbin{\bar{+}} \sum_{i=1}^{n} (\bar{a}_0\bar{b}_{i+n} \mathbin{\bar{+}} \bar{a}_{i+n}\bar{b}_0 \mathbin{\bar{+}} \bar{a}_i\bar{b}_i \mathbin{\bar{+}} \bar{a}_{i+n}\bar{b}_{i+n})\epsilon_{i+n} + K\epsilon_\pm$

  where:
  - $K_0 = \bar{a}_0\bar{b}_0 \mathbin{\bar{+}} \sum_{i=1}^{n} (\bar{a}_i\bar{b}_i[-\frac{1}{4}, 0] \mathbin{\bar{+}} \bar{a}_i\bar{b}_{i+n}[-\frac{1}{4}, \frac{1}{4}] \mathbin{\bar{+}} \bar{b}_i\bar{a}_{i+n}[-\frac{1}{4}, \frac{1}{4}] \mathbin{\bar{+}} \bar{a}_{i+n}\bar{b}_{i+n}[-\frac{1}{4}, 0])$
  - $K = (\bar{c}\bar{a}_0\bar{b}_{2n+1} \mathbin{\bar{+}} \bar{c}\bar{b}_0\bar{a}_{2n+1}) \mathbin{\bar{+}} \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \bar{c}\bar{a}_i\bar{b}_j \mathbin{\bar{+}} \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \bar{c}\bar{a}_i\bar{b}_{j+n} \mathbin{\bar{+}} \sum_{i=1}^{n} \bar{c}\bar{a}_i\bar{b}_{2n+1} \mathbin{\bar{+}} \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \bar{c}\bar{a}_{i+n}\bar{b}_j \mathbin{\bar{+}} \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \bar{c}\bar{a}_{i+n}\bar{b}_{j+n} \mathbin{\bar{+}} \sum_{i=1}^{n} \bar{c}\bar{a}_{i+n}\bar{b}_{2n+1} + \bar{c}\bar{a}_{2n+1}\bar{b}_{2n+1}$

Note that $\epsilon_\pm$ is propagated from *unknown* sources, then $\epsilon_\pm$ is propagated by applying multiplication of $\epsilon_\pm$ coefficient and $\bar{c} = [-1, 1]$.