Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science 289 (2012) 27–40

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

SMT for Polynomial Constraints on Real Numbers

To Van Khanh[1] and 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. In addition to existing interval arithmetic, such as classical intervals and affine intervals, we newly design Chebyshev Approximation Intervals, focusing on multiplications of the same variables, like Taylor expansions. When testing cannot find a satisfiable instance, this framework is designed to start 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 [16], termination proving [10], hybrid systems, loop invariant generation, and parameter design of control.

Tarski proved that polynomial constraints over real numbers (algebraic numbers) is decidable [21], and later Collins proposed Quantifier Elimination by Cylindrical Algebraic Decomposition [4], which is nowadays implemented in Mathematica, Maple/SynRac, Reduce/Redlog, and QEPCAD. However, it is DEXPTIME with respect to the number of variables, and works fine in practice up to 5 variables and lower degrees. For instance, eight variables with degree 10 require 20-30 hours by supercomputer.

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

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

doi:10.1016/j.entcs.2012.11.004

SMT (SAT modulo theories) separates case analysis and the core computation in the theory. RAHD [18] is such an example, which applies variations of QE-CAD implementations (e.g., QEPCAD-B, Reduce/Redlog) as a background theory.

An alternative choice of theory is approximation, e.g., iSAT [8], MiniSmt [22], Barcelogic [3], CVC3 [2], and CORD [9], in which bounded binary encoding, CORDIC, and Interval arithmetic are examples of background theories. Among them, MiniSmt and CVC3 have participated in QF_NRA category of SMT-LIB [1].

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. In addition to existing interval arithmetic, 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 [15] for interaction with MiniSat 2.2. Initially, an SAT instance given from SAT solver describes possible combinations of input ranges. If interval arithmetic (IA) reports unsatisfiability (IA_UNSAT), such combinations are removed for next SAT searching by memorizing them as learnt clauses to SAT solver. If IA reports validity (IA_VALID), any instances in the ranges is satisfiable. If IA finds neither validity nor unsatisfiability (IA_SAT), each polynomial is examined by testing.

If testing cannot find a satisfiable instance (Test_UNSAT), such combinations of input ranges can be memorized as a learnt clause by heuristics, and removed from next searching.

When IA decides neither satisfiability nor unsatisfiability, this framework is designed to start 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 [15]. Section 3 explains variations of interval arithmetic and newly proposes Chebyshev Approximation Intervals, CAI1, CAI2. Section 4 describes testing strategies. The framework of our SMT solver is described with examples in section 5. Preliminary experiments on small examples from SMT-LIB benchmarks [1] are reported in section 6. Section 7 discusses some related works, 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 benchmarks).

Definition 2.1 A polynomial inequality constraint is in the form of

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

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 a background 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* [15].

As notation, l and l_i denote *literals*, a *clause* is a set of literals, and a *Conjunctive Normal Form* (CNF) F is a set of clauses. M and M_1 are (partial) *assignments*, which are sequences of literals. $\Longrightarrow$ is a *binary relation* over *states* which are pairs of an assignment M and a CNF F, denoted as $M \parallel F$. A clause C is *true* in M if $C \cap M \neq \emptyset$. M is satisfied on F, denoted as $M \models F$, if all clauses of F are true in M. If $F \cup \neg G$ is *unsatisfiable* in a background theory T which is denoted as $F \models_T G$.

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

$$Ml M_1 \parallel F \Longrightarrow \emptyset \parallel F \wedge (\neg l_1 \vee ... \vee \neg l_n \vee \neg l) \qquad \text{if} \begin{cases} Ml M_1 \models F \\ \{l_1, ..., l_n\} \subseteq M \\ l_1 \wedge ... \wedge l_n \models_T \neg l \end{cases}$$

- *Eager theory propagation* interacts with theory T during DPLL procedure of SAT, and DPLL procedure continues when the theory admits the current decisions.

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

We adopt *very lazy theory learning* on `MiniSat2.2`, which naturally memorizes unsatisfiable combination of input ranges for a polynomial as a learnt clause. Certain combination with *eager theory propagation* would improve the efficiency. However, it is left for future work, since it requires tighter interaction between SAT solver and theory, which needs internal modification of MiniSat.

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

$$C = \exists x_1 \in [l_1, h_1] \cdots x_k \in [l_k, h_k]\ .\ \bigwedge_1^m f_i(x_1, \cdots, x_k) > 0,$$

$f_i^l(x_1, \cdots, x_k)$ and $f_i^u(x_1, \cdots, x_k)$ are lower and upper bounds estimated by IA, we say

- C is *IA_VALID* if $\forall i \in [1, m].\ f_i^l(x_1, \cdots, x_k) > 0$,
- C is *IA_UNSAT* if $\exists i \in [1, m].\ f_i^u(x_1, \cdots, x_k) \leq 0$ and
- C is *IA_SAT* if $\exists j \in [1, m].\ f_j^l(x_1, \cdots, x_k) \leq 0 \wedge (\bigwedge\limits_i f_i^u(x_1, \cdots, x_k) > 0)$.

Note that IA_VALID and IA_UNSAT safely reason SAT and UNSAT, respectively. However, IA_SAT cannot conclude SAT, and treated as *unknown*.

A popular example of IA is Classical Interval (CI) [14], 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 (AF) introduces *noise symbols* ϵ, which is interpreted as a value in $[-1, 1]$ [5,6,7], 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 [5,6],
- $\epsilon\epsilon'$ is replaced with $[-1, 1]\epsilon$ (or $[-1, 1]\epsilon'$), called Extended Affine Interval (EAI) [16], and
- $\epsilon\epsilon'$ is pushed into the fixed error noise symbol $\epsilon_\pm$, denoted AF1 [11].

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.

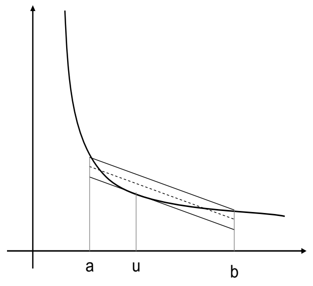

Fig. 1. Chebyshev approximation

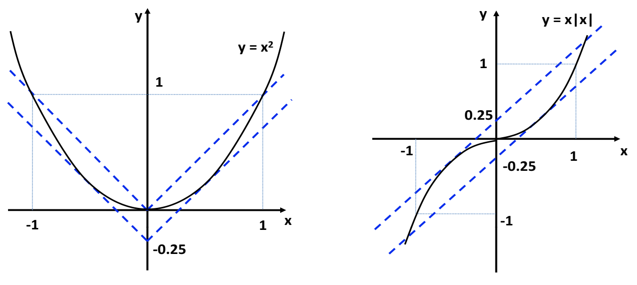

Fig. 2. Chebyshev approximation of x^2 and $x\ |x|$

We newly design Chebyshev Approximation Interval (CAI1, CAI2) and implement Classical Interval (CI), Affine Intervals (AF, AF1, AF2) [11], and ChebyshevApproximation Intervals (CAI1, CAI2). Their forms are, e.g.,

$$\text{AF1} \quad \hat{x} = a_0 + \sum_{i=1}^{n} a_i \epsilon_i + a_{n+1}\epsilon_\pm$$

$$\text{AF2} \quad \ddot{x} = a_0 + \sum_{i=1}^{n} a_i\epsilon_i + a_{n+1}\epsilon_+ + a_{n+2}\epsilon_- + a_{n+3}\epsilon_\pm$$

$$\text{CAI1} \ \mathring{x} = \bar{a}_0 + \sum_{i=1}^{n} \bar{a}_i\epsilon_i + \sum_{i=1}^{n} \bar{a}_{i+n}\epsilon_{i+n} + \bar{a}_{2n+1}\epsilon_\pm$$

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

(i) introduction of noise symbols [5,6,11],

(ii) keeping products of noise symbols up to degree 2 ($\epsilon_i\epsilon_j$) [11] (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 $x \in [-1, 1]$,

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

which are explained in Figure 2. This observation leads symbolic manipulation on products of the same noise symbol ϵ as

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

Remark 3.1 Introduction of Chebyshev approximation is not new. For instance, Stolfi [20] proposed it based on the mean-value theorem, as in the left of Figure 2. Miyajima et al. [13] 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) [12], 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 bounds 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

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

and finite set Θ of substitutions, we denote $\vdash_{test(\Theta)} C$ if $\bigwedge_j f_j(\theta(x_1), \cdots, \theta(x_k)) > 0$ 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 *Test_SAT* if $\vdash_{test_\Theta} C$ and
- C is *Test_UNSAT* if $\nvdash_{test_\Theta} C$.

Test_UNSAT does not imply UNSAT, but we will use its information for computing leanrn clauses as heuristics.

There are two immediate strategies to generate random test cases.

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

- 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 $i \in \{1, ..., k\}$).

Reducing the number of unnecessary test cases is an important task to improve efficiency. For instance, if we consider 10 variables and each has 2 test cases, then we have 2^{10} instances as a total. In solving that problem, we divides constraints into *small groups* (constraints in a group share some variables) and we compute satisfiable test cases for each group first.

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 [17].

Fig 3 describes its design framework.

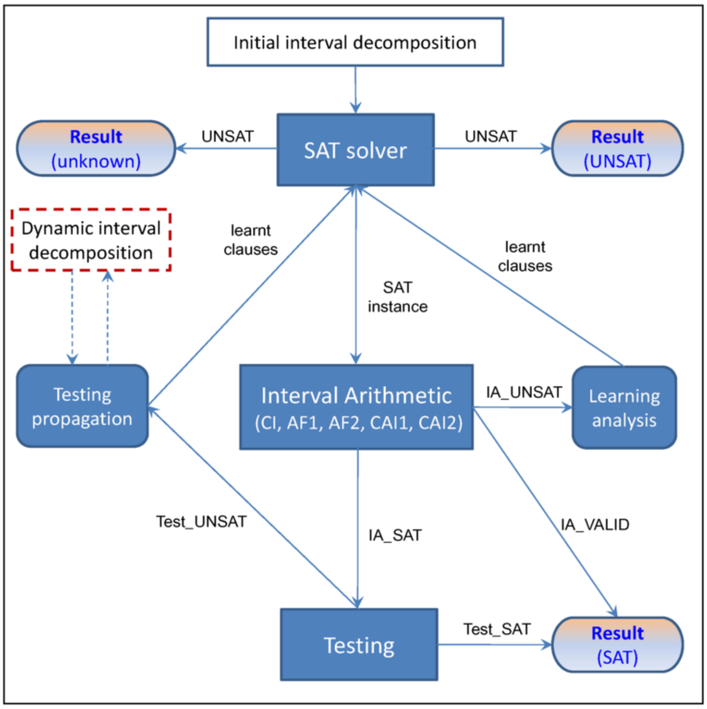

Fig. 3. Framework of SMT solver

- `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 a CNF, which is sent to SAT solver.
- `SAT solver`: We use MiniSat2.2 as a backend SAT solver. The SAT solver finds a satisfiable combination of input ranges of all variables. A satisfiable (SAT) instance is sent to IA for checking. If the SAT solver returns unsatisfiability, we conclude *unknown* if testing is applied, otherwise we conclude *UNSAT* for the final result.
- `Interval Arithmetic (IA)`: We implement CI, AF1, AF2, CAI1 and CAI2 as IA. IA decides each polynomial $f_i(x_1, \cdots, x_k) > 0$ either IA_VALID, IA_UNSAT, or IA_SAT, under given input ranges. If some of them are IA_UNSAT, we return IA_UNSAT and a negation of a combination computed by *Learning analysis* is added to the SAT solver as a learnt clause. If each of them is IA_VALID, we have done. If some of them remain IA_SAT, all IA_SAT polynomials are sent to testing (still memorizing polynomials detected to be IA_VALID).
- `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 all polynomials are Test_SAT for a test case, we have done. If it cannot find a successful test case, it returns Test_UNSAT.

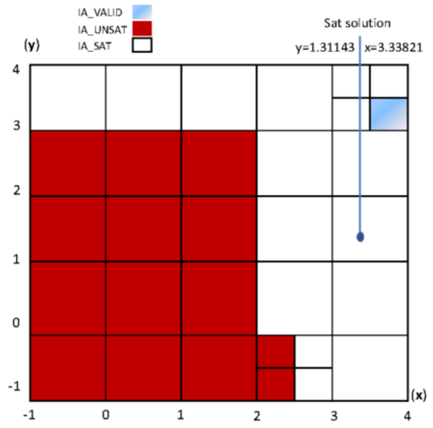

Fig. 4. Solver working on Example 5.1

- `Testing propagation`: When testing of polynomials returns Test_UNSAT, a negation of a combination of input ranges is computed and then it is added to CNF as a learnt clause. This is heuristics to narrow the search and intends to find other SAT instances for next evaluation.

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

Example 5.1 Fig 4 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] and y = [-1,4]
(assert (f = 4x + 3y -xy > 12))
```

First, by *Initial interval decomposition*, the input ranges $[-1, 4]$ of variables x and y are split into 5 small input ranges. By IA, the red areas ($x \in [-1, 2]$ and $y \in [-1, 3]$) are detected to be IA_UNSAT. The remaining areas remain white, which means IA_SAT. 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 IA_UNSAT. Similarly, in the area $x \in [3, 4]$ and $y \in [3, 4]$, the right below quarter is detected to be IA_VALID (light blue) by IA.

6 Preliminary experiment

In this section, we show preliminary results with the problem P1

$$\exists x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10} \in [0, 3]\ x_{11} \in [-3, 4]\ x_{12} \in [-1, 3].$$
$$x_1x_3 - x_1x_3x_7 > 0\ \wedge\ x_1x_2 - x_1x_6 - x_1x_2x_7 > 0\ \wedge x_1x_3 - x_3x_5 > 0\ \wedge$$
$$x_0 + x_1x_2 - x_4 - x_2x_5 > 0\ \wedge\ x_8 - x_2 + x_{10}x_9 - x_{10}x_3 > 0\ \wedge\ x_3x_7 > 1\ \wedge$$

$x_6 + x_2 x_7 > 0 \ \wedge \ x_{11}^3 - 2x_{11}^2(1 + x_{12}^2) - 2x_{12}(x_{11} + x_{12}) + x_{12} - 6.5 > 0,$

P2 (P2 is just changed the input of x_{11} from P1 into $x_{11} \in$ **[-3,2]**) and 18 problems in the division QF_NRA of SMT-LIB [1] benchmarks. We choose problems with up to 20 variables.

P1 was checked by AF1, AF2, CAI1 and CAI2, all of them give SAT results. For the problem P2, while AF1 and AF2 detect *unknown*, CAI1 and CAI2 detect *UNSAT* because CAI1 and CAI2 give better results for over - approximation than AF1 and AF2 in this problem. 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 ≥ 0. In this experiment, we set an upper bound for these variables and evaluate these problems with a range $[0, 2.5]$. The range $[0, 2.5]$ is split into 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 ≤ 15 *(except the problem matrix-1-all-21)* and AF1, AF2 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 choices of theories among SMTs for polynomial constraints, e.g.,

- QE-CAD (Quantifier Elimination by Cylindrical Algebraic Decomposition) [4],
- interval arithmetic (as over approximation),
- bounded binary encoding (as under approximation), and
- reduction to linear constraints, e.g., bounded integer coefficients and CORDIC (COordinate Rotation DIgital Computer).

RAHD [18] separates case analysis and the core computation of QE-CAD originated by Tarski. It applies different versions of QE-CAD implementations such as QEPCAD-B, Reduce/Redlog.

Interval arithmetic is an over-approximation, but sufficiently fine decomposition of input ranges will correctly find satisfiability of polynomial inequalities. Input range decomposition has proposed in RSOLVER [19] and implemented in RSOLVER [19] and iSAT [8]. While RSOLVER develops a pruning algorithm to remove unsatisfied elements, iSAT applies a tight interaction of SAT solver and eager theory propagation. Thus, conflict detection and theory propagation are directly applied for SAT solver to provide new assignments. Our approach is combining testing (as under-approximation) with interval arithmetic. It will supply more opportunity to conclude satisfiability, furthermore it will guide more likely range decomposition. We also apply Chebyshev Affine Intervals, instead of using CI in RSOLVER and

Problem name	**No. variables**	**No. constraints**	**Interval arithmetic**	**Result**	**Time (s)**
P1	13	8	$AF1$	SAT	0.109
P1	13	8	$AF2$	SAT	0.14
P1	13	8	$CAI1$	SAT	1.687
P1	13	8	$CAI2$	SAT	338.593
P2	13	8	$AF1$	unknown	0.125
P2	13	8	$AF2$	unknown	0.046
P2	13	8	$CAI1$	UNSAT	1.062
P2	13	8	$CAI2$	UNSAT	159.546
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	17	$AF1$	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, P2 and QF_NRA

iSAT.

MiniSmt [22] applies bundled bit encoding, which encodes non-linear arithmetic over rational numbers (i.e., pairs of integers) under given bounds, and reduced to SAT solving. To handle limited use of polynomial equality, it introduces the fixed number of algebraic numbers symbolically. MiniSmt can show satisfiability quickly, but due to the bound of the search, it cannot conclude unsatisfiability. CVC3 [2] is also a popular SMT, participating NRA category of SMT-LIB as well as MiniSmt. However, we could not find references that provide its technical details.

Barcelogic [3] assumes finite input ranges on integers, and reduces polynomial constraints to linear ones by instantiating one of arguments in multiplications with finitely many possible integers in bounded ranges. These linear constraints are solved by Yices (Presburger arithmetic over integers).

CORD [9] uses another reduction to linear constraints, called CORDIC (COordinate Rotation DIgital Computer), which translates a non-linear operation to linear forms by *n iterative steps.* One of two arguments of a multiplication is normalized to $(-2, 2)$, then the multiplication is approximated by the sum of n positive/negative shifters, in which the k_{th} shifter corresponding to the half of the $(k-1)_{th}$ shifter. Initially, the first shifter is set to the value of unnormalized argument. Each iterative step of a CORDIC translation is encoded as linear constraints, and some linear constraints are added to account for all inaccuracies in approximation of CORDIC. Finally, these linear constraints are solved by Yices (Presburger arithmetic over real numbers).

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. In addition to existing interval arithmetic, such as classical intervals and affine intervals, we newly designed Chebyshev Approximation Intervals.

Interval arithmetic can indicate *unsatisfiable areas* (IA_UNSAT) and remove these areas from search space. Testing only focus on areas that IA decides neither validity nor unsatisfiability. When testing cannot find a satisfiable instance in an area, *heuristics* is applied to make the solver not to search that area again. The result of preliminary experiments on small examples including SMT-LIB benchmarks is encouraging. Our status is preliminary, and there is much future work to be undertaken.

- `Test data generation strategy`: When the number of variables becomes large, the number of test cases 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 ϵ_i, the coefficient of ϵ_i in the result reflects strength of its influence. We can generate more test cases for such sensitive variables. This was proposed in [17] under the program analysis context and we hope to apply to our SMT.
- `Dynamic interval decomposition` and refinement loop: In Figure 3, dynamic interval composition is connected with dotted lines, which means it is 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 θ would be nearer to satisfiable instances if $f_i(\theta(x_1), \cdots, \theta(x_n))$ is nearer to 0. 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 [17] under the program analysis context and we hope to apply to our SMT.
- Polynomial equality: Currently, we can handle polynomial inequalities only. However, for instance

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

can be decomposed into two phases. First, find some areas $[l_{1k}, h_{1k}] \subseteq$

$[l_1, h_1] \cdots [l_{nk}, h_{nk}] \subseteq [l_n, h_n]$ (by interval arithmetic) such that

$$\forall x_1 \in [l_{1k}, h_{1k}] \cdots x_n \in [l_{nk}, h_{nk}].\ \bigwedge_j f_j(x_1, \cdots, x_n) > 0.$$

and find two instances (by testing) in that areas such that $g(a_1, \cdots, a_n) > 0$ and $g(b_1, \cdots, b_n) < 0$. By Intermediate value theorem, we can conclude $\exists x_1 \in [l_1, h_1] \cdots x_n \in [l_n, h_n].\ g(x_1, \cdots, x_n) = 0$.

- `Scalability and practical experiments`: Scalability is very important in practice, and we expect the partial use of eager theory propagation will improve efficiency.

Acknowledgement

The authors would like to thank Nao Hirokawa and Harald Zankl for fruitful discussions, and anonymous reviewers for their valuable comments. This research is partially supported by Grant-in-Aid for Scientific Research (B) No. 23300005 Mext, Japan.

References

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

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

[3] 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 22nd International Conference on Automated Deduction CADE-22," Lecture Notes in Computer Science, Vol. **5663** (2009), Springer-Verlag, 294–305.

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

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

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

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

[8] 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.

[9] Ganai, M., and Franjo Ivancic, *Effcient decision procedure for non-linear arithmetic constraints using CORDIC*, "Formal Methods in ComputerAided Design," FMCAD 2009, 61-68.

[10] 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.

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

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

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

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

[15] 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.

[16] 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.

[17] Ngoc, D., Mizuhito Ogawa, *Checking Roundoff Errors using Counterexample-Guided Narrowing*, "Proceedings of the IEEE/ACM international conference on Automated software engineering," ASE 2010, ACM, 301–304.

[18] Passmore, G. O., Paul B. Jackson, *Combined Decision Techniques for the Existential Theory of the Reals*, "Proceedings of the 16th Symposium, 8th International Conference. Held as Part of CICM '09 on Intelligent Computer Mathematics," Calculemus '09/MKM '09, Springer-Verlag, 122–137.

[19] Ratschan, S., *Effcient solving of quantified inequality constraints over the real numbers*, ACM Trans. Comput. Logic, Vol. **7** No. 4 (2006), 723-748.

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

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

[22] 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.

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 $\bar{a}\bar{\times}\bar{b}$)*:

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

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

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

  where $\{\bar{+}, \bar{-}, \bar{\times}\}$ are CI arithmetic, and

  · $$K_0 = \bar{a}_0\bar{b}_0 \bar{+} \sum_{i=1}^{n} (\bar{a}_i\bar{b}_i[-\frac{1}{4}, 0] \bar{+} \bar{a}_i\bar{b}_{i+n}[-\frac{1}{4}, \frac{1}{4}] \bar{+} \bar{b}_i\bar{a}_{i+n}[-\frac{1}{4}, \frac{1}{4}] \bar{+} \bar{a}_{i+n}\bar{b}_{i+n}[-\frac{1}{4}, 0])$$

$$\cdot\ K = (\bar{c}\bar{a}_0\bar{b}_{2n+1} \mp \bar{c}\bar{b}_0\bar{a}_{2n+1}) \mp \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\bar{c}\bar{a}_i\bar{b}_j \mp \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\bar{c}\bar{a}_i\bar{b}_{j+n} \mp \sum_{i=1}^{n}\bar{c}\bar{a}_i\bar{b}_{2n+1}$$
$$\mp \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\bar{c}\bar{a}_{i+n}\bar{b}_j \mp \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\bar{c}\bar{a}_{i+n}\bar{b}_{j+n} \mp \sum_{i=1}^{n}\bar{c}\bar{a}_{i+n}\bar{b}_{2n+1} + \bar{c}\bar{a}_{2n+1}\bar{b}_{2n+1}$$

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