raSAT : an SMT Solver for Polynomial Constraints

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

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

Abstract. This paper presents the **raSAT** SMT solver for polynomial constraints, which aims to handle them over both reals and integers with simple unified methodologies: (1) **raSAT** *loop* for inequalities, which extends the *interval constraint propagation* with testing to accelerate SAT detection, and (2) a non-constructive reasoning for equations over reals, based on the generalized intermediate value theorem.

1 Introduction

Polynomial constraint solving is to find an instance that satisfies a given system of polynomial inequalities/equations. Various techniques for solving such a constraint are implemented in SMT solvers, e.g., **Cylindrical algebraic decomposition** (RAHD [19, 18], Z3 4.3 [13]), **Virtual substitution** (SMT-RAT [5], Z3 3.1), **Interval constraint propagation** [2] (iSAT3 [7], dReal [10, 9], RSolver [20], RealPaver [11]), and **CORDIC** (CORD [8]). For integers, **Bit-blasting** (MiniSmt [23]) and **Linearization** (Barcelogic [3]) can be used.

This paper presents the **raSAT** SMT solver[3] for polynomial constraints over reals. For inequalities, it applies a simple iterative approximation refinement, **raSAT** *loop*, which extends the interval constraint propagation (ICP) with testing to boost SAT detection (Section 3). For equations, a non-constructive reasoning based on the generalized intermediate value theorem [17] is applied (Section 4). Implementation with soundness guarantee and optimizing strategies is evaluated by experiments (Section 5).

Although **raSAT** has been developed for constraints over reals, constraints over integers are easily adopted, e.g., by stopping interval decompositions when the width becomes smaller than 1, and generating integer-valued test instances.

raSAT has participated SMT Competition 2015, in two categories of main tracks, *QF_NRA* and *QF_NIA*. The results, in which **Z3 4.4** is a reference, are,

- 3^{rd} in *QF_NRA*, **raSAT** solved 7952 over 10184 (where **Z3 4.4**, **Yices-NL** and **SMT-RAT** solved 10000, 9854 and 8759, respectively.)

[3] Available at http://www.jaist.ac.jp/~s1310007/raSAT/index.html

– 2^{nd} in *QF_NIA*, **raSAT** solved 7917 over 8475 (where **Z3 4.4** and **AProVE** solved 8459 and 8270, respectively).

A preliminary version of **raSAT** was orally presented at *SMT workshop 2014* [22].

2 SMT solver for polynomial constraints

Definition 1. *A polynomial constraint ψ is defined as follow*

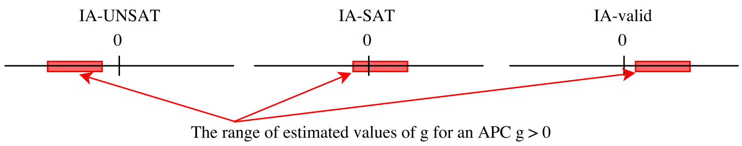
$$\psi ::= g(x_1, ..., x_n) \diamond 0 \mid \psi \wedge \psi \mid \psi \vee \psi \mid \neg\psi \tag{1}$$

where ($\diamond \in \{>, \geq, <, \leq, =, \neq\}$) and $g(x_1, \cdots, x_n)$ is a polynomial with integer coefficients over variables $x_1, \cdots, x_n$. We call $g(x_1, \cdots, x_n) \diamond 0$ an atomic polynomial constraint (APC). When $x_1, \cdots, x_n$ are clear from the context, we denote g for $g(x_1, \cdots, x_n)$, and $var(g)$ for the set of variables appearing in g.

An SMT solver decides whether ψ is satisfiable (SAT), i.e., whether there exists an assignment of reals (resp. integers) to variables that makes ψ *true*. We organize the **raSAT** SMT solver in a very lazy approach for an arithmetic theory T over reals (resp. integers). As a preprocessing, **raSAT** converts a polynomial constraint into conjunctive normal form (CNF) by Tseitin conversion [21]. In addition, the APCs are preprocessed so that the constraint becomes a CNF containing only $>$ and $=$. Then, first, each APC is assigned a Boolean value ($true$ or $false$) by an SAT solver such that ψ is evaluated to $true$. Second, the boolean assignment is checked for consistency against the theory T.

raSAT is one of the interval constraint propagation (ICP) based SMT solvers, as well as **iSAT** [7] and **dReal** [10]. In ICP [2], *interval arithmetic* (IA) [16] plays a central role. **raSAT** implements Classical Interval (CI) [16] and four kinds of Affine Intervals (AI) [4, 14]. We fix their notations. Let $\mathbb{R}$ be the set of real numbers and $\mathbb{R}^\infty = \mathbb{R} \cup \{-\infty, \infty\}$. We naturally extend the standard arithmetic operations on $\mathbb{R}$ to those on $\mathbb{R}^\infty$ as in [16]. The set of all intervals is denoted by $\mathbb{I} = \{[l, h] \mid l \leq h \in \mathbb{R}^\infty\}$. A *box* for a sequence of variables $x_1, \cdots, x_n$ is $B = I_1 \times \cdots \times I_n$ for $I_1, \cdots, I_n \in \mathbb{I}$.

A conjunction φ of APCs is *IA-valid* (resp. *IA-UNSAT*) in a box B if φ is evaluated to *true* (resp. *false*) by IA over B. In this case, B is called a *IA-valid* (resp. *IA-UNSAT*) box with respect to φ. Since IA is an over approximation of arithmetical results, IA-valid (resp. IA-UNSAT) in B implies valid (resp. UNSAT) in B. If neither of them holds, we call *IA-SAT* (as shown below), which cannot decide the satisfiability at the moment. Note that if φ is IA-valid in B, φ is SAT.

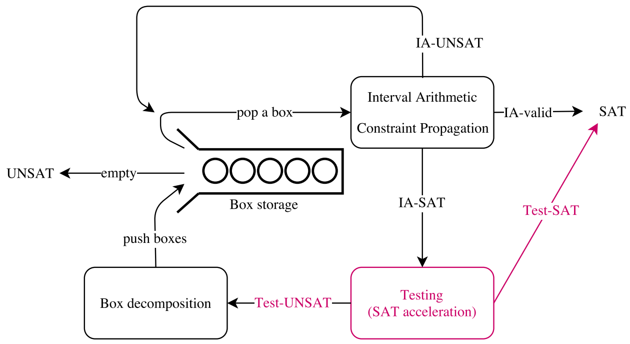


3 ICP and raSAT loop for inequality

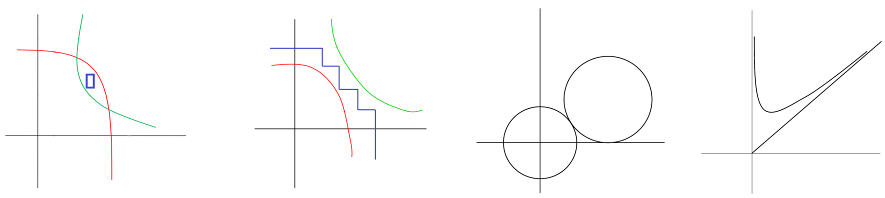
Since ICP is based on IA, which is an over-approximation, it can be applied to decide SAT/UNSAT of inequalities and UNSAT of equalities, but not for SAT of equalities. We first explain ICP for (a conjunction of) inequalities and then extend it as a **raSAT** loop for SAT detection acceleration. Handling the presence of equations will be shown in Section 4.

Starting with a box B ($(-\infty, \infty)^n$ by default), ICP [2] tries to detect SAT of φ in B by iteratively contracting boxes (by backward propagation of interval constraints) and decomposing boxes (when neither IA-valid nor IA-UNSAT detected) until either an IA-valid box is found or no boxes remain to explore.

The **raSAT** loop [14] intends to accelerate ICP for SAT detection by testing. Figure below illustrates the **raSAT** loop, in which "*Test-SAT*" in B means that a satisfiable instance is found by testing in B, and "*Test-UNSAT*", otherwise.



Limitation of ICP and raSAT loop for inequality ICP concludes SAT when it identifies a valid box by IA. Although the number of boxes may be exponential, if $I_1, \cdots, I_n$ are bounded, ICP always detects SAT of the inequalities ψ as Fig.(a) and detects UNSAT of ψ if not touching as illustrated in Fig.(b,c). If $I_1, \cdots, I_n$ are not bounded, adding to touching cases, a typical case of failure in UNSAT detection is a converging case as Fig.(d).



(a) SAT detection (b) UNSAT detection (c) Touching case (d) Convergent case

4 Generalized intermediate value theorem for equations

Handle equations in **raSAT** is illustrated by the *intermediate value theorem* (IVT) for a single equation $g(x) = 0$. If we find t_1, t_2 with $g(t_1) > 0$ and $g(t_2) < 0$,

$g = 0$ holds in between. For multi-variant equations, we apply a custom version (Theorem 1) of the generalized IVT [17, Theorem 5.3.7].

4.1 Generalized intermediate value theorem

Let $B = [l_1, h_1] \times \cdots [l_n, h_n]$ be a box over $V = \{x_1, \cdots, x_n\}$, and let $V' = \{x_{i_1}, \cdots, x_{i_k}\}$ be a subset of V. We denote $B{\downarrow}_{V'} = \{(r_1, \cdots, r_n) \in B \mid r_i = l_i \text{ for } i = i_1, ..., i_k\}$ and $B{\uparrow}_{V'} = \{(r_1, \cdots, r_n) \in B \mid r_i = h_i \text{ for } i = i_1, ..., i_k\}$. Given an assignment $\theta : V' \mapsto \mathbb{R}$, which assigns a real value to each variable in V', $B|_\theta = \{(r_1, \cdots, r_n) \in B \mid r_i = \theta(x_i) \text{ if } x_i \in V'\}$.

Definition 2. *Let* $\bigwedge\limits_{j=1}^{m} g_j = 0$ *be a conjunction of equations over* V*. A sequence* $(V_1, \cdots, V_m)$ *is a* check basis *of* $(g_1, \cdots, g_m)$ *in* B*, if, for each* $j, j' \leq m$*,*

1. $\emptyset \neq V_j \subseteq var(g_j)$*,*
2. $V_j \cap V_{j'} = \emptyset$ *if* $j \neq j'$*, and*
3. *either* $g_j < 0$ *on* $B{\uparrow}_{V_j}$ *and* $g_j > 0$ *on* $B{\downarrow}_{V_j}$*, or* $g_j < 0$ *on* $B{\uparrow}_{V_j}$ *and* $g_j > 0$ *on* $B{\downarrow}_{V_j}$*.*

Theorem 1. *For a conjunction of polynomial inequalities/equations*

$$\varphi = \bigwedge_{j=1}^{m} g_j > 0 \wedge \bigwedge_{j=m+1}^{m'} g_j = 0$$

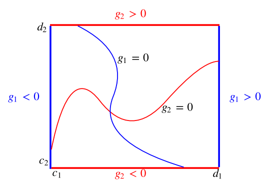
and $B = [l_1, h_1] \times \cdots [l_n, h_n]$*, assume that the followings hold.*

1. *For* $\varphi_1 \wedge \varphi_2 = \bigwedge\limits_{j=1}^{m} g_j > 0$*,* φ_1 *is IA-valid in* B *and* φ_2 *is Test-SAT in* B *with an assignment* $\theta_{\varphi_2} : V_{\varphi_2} \mapsto \mathbb{R}$ *such that* $\theta_{\varphi_2}(x_i) \in [l_i, h_i]$ *for each* $x_i \in V_{\varphi_2}$*, where* V_{φ_2} *is the set of variables in* φ_2*.*
2. *A* check basis $(V_{m+1}, \cdots, V_{m'})$ *over* $V \setminus V_{\varphi_2}$ *of* $(g_{m+1}, \cdots, g_{m'})$ *in* $B|_{\theta_{\varphi_2}}$ *exists.*

Then, φ *has a SAT instance in* B*.*

Example 1 illustrates Theorem 1 for $V = \{x, y\}$ with $m = 0$ and $m' = n = 2$.

Example 1. Given two equations $g_1(x, y) = 0$ and $g_2(x, y) = 0$. Assuming that there exists a box $B = [c_1, d_1] \times [c_2, d_2]$ such that

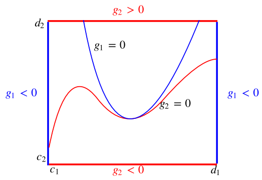


- $g_1(c_1, y) < 0$ for $y \in [c_2, d_2]$, $g_1(d_1, y) > 0$ for $y \in [c_2, d_2]$, and
- $g_2(x, c2) < 0$ for $x \in [c_1, d_1]$, $g_2(x, d2) > 0$ for $x \in [c_1, d_1]$.

Thus, $g_1(x, y) = 0$ and $g_2(x, y) = 0$ share a root in B.

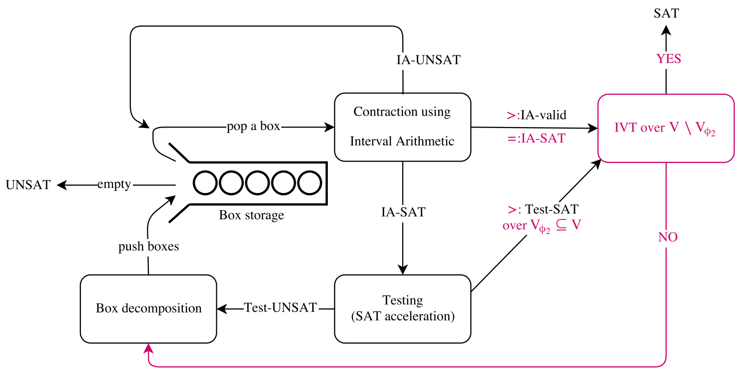
Limitation of the generalized IVT for equality
There are two limitations on applying Theorem 1.

– The number of variables (dimensions) must be no less than the number of equations.
– Trajectories of equations must be crossing. For instance, it may fail to show SAT if two equation $g_1 = 0$, $g_2 = 0$ are touching, as in the right figure.



4.2 raSAT loop with generalized IVT

Theorem 1 is added into the **raSAT** loop as in Figure below. We borrow notations φ, φ_1, and φ_2 from Theorem 1. The label "$>$: IA-valid" means that the conjunction of inequalities appearing in the input is IA-valid. Similar for "$=$: IA-SAT" and "$>$: Test-SAT". The label "Test-SAT over $V_{\varphi_2} \subseteq V$" means that a test instance to conclude Test-SAT of φ_2 is generated on V_{φ_2} and the generalized IVT is applied over $V \setminus V_{\varphi_2}$ in the box $B|_{\theta_{\varphi_2}}$ (described by "IVT over $V \setminus V_{\varphi_2}$").



Example 2. Suppose φ is $g_1 > 0 \wedge g_2 = 0 \wedge g_3 = 0$ where $g_1 = cd - d$, $g_2 = a - c - 2$, and $g_3 = bc - ad - 2$. The initial box storage contains only $B = [-2, 3.5] \times [-5, 0] \times [0, 1.5] \times [-5, -0.5]$ as the initial range of (a, b, c, d).

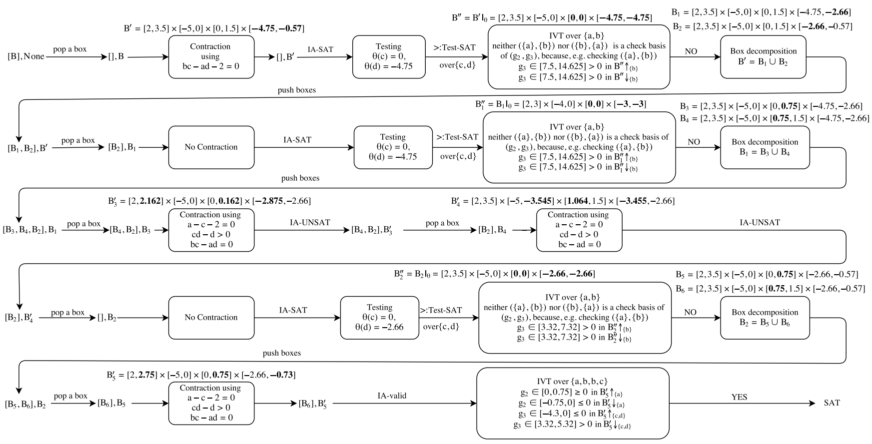


Figure above shows the flow of the **raSAT** loop with IVT, where a label $[...], B$ is for a pair of a box storage and a currently exploring box B, and θ for a test instance. The backward interval constraint propagation reduces B, B_1, and B_3 to B', B'_1, and B'_3, respectively.

5 Implementation and Experiments

5.1 Implementation of raSAT

In **raSAT** implementation, the SAT solver miniSAT [6] manages the Boolean part of the DPLL procedure. There are several notable features of **raSAT**.

Soundness **raSAT** uses the floating point arithmetic, and round-off errors may violate the soundness. To get rid of such pitfalls, **raSAT** integrates an IA library [1] which applies outward rounding [12] of intervals. For the soundness of Test-SAT, **iRRAM**[4], which guarantees the round-off error bounds, confirms that a SAT instance found by the floating point arithmetic is indeed SAT.

Affine interval Various IAs, including Classical Interval (CI) [16] and 4 variations, AF_1, AF_2, EAI, CAI, of Affine Intervals (AI) [4, 15, 14], are implemented as a part of **raSAT**. At the moment, AF2 and CI are used by default, and the choice option will be prepared in the future releases.

AI introduces noise symbols ϵ's, which are interpreted as values in $[-1, 1]$. Variations of AIs come from how to (over) approximate the multiplication of noise symbols in a linear formula. Although the precision is incomparable, AI partially preserve the dependency among values, which is lost in CI. For instance, let $x \in [2, 4] = 3 + \epsilon$. Then, $x - x$ is evaluated to $[-2, 2]$ by CI, but $[0, 0]$ by AI. The example below shows the value dependency. Let $h(x, y) = x^3 - 2xy$ for $x = [0, 2] = 1 + \epsilon_1$ and $y = [1, 3] = 2 + \epsilon_2$. CI estimates $h(x, y)$ as $[-12, 8]$, and AF_2 does as $-3 - \epsilon_1 - 2\epsilon_2 + 3\epsilon_+ + 3\epsilon_\pm$ (evaluated to $[-9, 6]$). Such information is used to design SAT-directed heuristics for choosing a variable at a box decomposition.

SAT-directed heuristics The variable selection strategy is, (1) select the least likely satisfiable APC with respect to *SAT-likelihood*, and (2) choose the most likely influential variable in the APC with respect to the *sensitivity*.

Suppose AI estimates the range $range(g, B)$ of a polynomial g in a box B as $[c_1, d_1]\epsilon_1 + \cdots + [c_n, d_n]\epsilon_n$, which is evaluated by instantiating $[-1, 1]$ to ϵ_i.

- The *SAT-likelihood* of an APC $g > 0$ is $|range(g, B) \cap (0, \infty)|/|range(g, B)|$.
- The *sensitivity* of a variable x_i in $g > 0$ is $max(|c_i|, |d_i|)$.

For instance, the SAT-likelihood of $h(x, y)$ above is $0.4 = \frac{6}{9-(-6)}$ by AF_2 and the sensitivity of x and y are 1 and 2 by AF_2, respectively.

When selecting a box, **raSAT** adopts the largest *SAT-likelihood*, where the *SAT-likelihood* of a box is the least *SAT-likelihood* among APCs on it. Thus, the box storage in the **raSAT** loop with IVT is implemented as a priority queue.

[4] Available at `http://irram.uni-trier.de`

The effect of the heuristics is examined with 18 combinations of the least, largest (with respect to measures), and random variable/box choices. Among them, only the combination above shows visible differences from the random choices, especially on SAT detection for quite large problems, such that it detects 11 SAT (including 5 problems marked "*unknown*") in Zankl/Matrix2∼5, whereas others detect at most 5 SAT (with at most 1 problem marked "*unknown*").

5.2 Experiments

Comparison with other SMT solvers Our comparison has two views, (1) ICP-based solvers, e.g., **iSAT3** and **dReal**, and (2) other SMT-solvers, which are superior than **raSAT** at the SMT competition 2015, e.g., **Z3 4.4** and **SMT-RAT 2.0**[5]. After the competition, **raSAT** has been improved on the backward interval constraint propagation [2]. They are compared on SMT-LIB benchmarks 2015-06-01[6] with timeout of 2500 seconds on an Intel Xeon E7-8837 2.66GHz and 8GB RAM. Note that

- **iSAT3** requires bounded intervals, and its bound of variables is set to $[-1000, 1000]$. For other tools (including **raSAT**), it is kept $(-\infty, \infty)$.
- **dReal** decides δ-SAT, instead of SAT, which allows δ-deviation on the evaluation of polynomials for some $\delta > 0$. Note that δ-SAT does not imply SAT. δ for **dReal** is set to its default value (0.001).

Table 1 shows the numbers of solved problems in each benchmark of the QF_NRA category in SMT-LIB. The "Time" row shows the cumulative running time of successful cases. In the "Benchmark" column, the numbers of SAT/UNSAT problems are associated if already known. "*" means δ-SAT.

Unknown Problems in SMT-LIB In SMT-LIB benchmark, many problems are marked "*unknown*". Among such unknown inequality problems, **raSAT** solves 15 (5 SAT, 10 UNSAT), **Z3 4.4** solves 36 (13 SAT, 23 UNSAT), and **SMT-RAT 2.0** solves 15 (3 SAT, 12 UNSAT). For problems with equations, **raSAT** and **SMT-RAT 2.0** solve 3 UNSAT problems, and **Z3 4.4** solves 492 (276 SAT, 216 UNSAT). For large problems, UNSAT can be detected by finding a small UNSAT core among APCs, whereas SAT detection requires to check all APCs.

For unknown problems, SAT results are easy to check. Although **Z3 4.4** outperforms others, it is worth mentioning that **raSAT** also detects SAT on several quite large problems in Zankl/Matrix-2∼5, which often have more than 50 variables (Meta-Tarski and Matrix-1 have mostly less than 10 and 30 variables, respectively). For instance, **Z3 4.4** solely solves Matrix-3-7, 4-12, and 5-6 (which have 75, 200, and 258 variables), and **raSAT** solely solves Matrix-2-3, 2-8, 3-5, 4-3, and 4-9 (which have 57, 17, 81, 139, and 193 variables). **SMT-RAT 2.0** shows no new SAT detection in Zankl/Matrix-2∼5.

[5] https://github.com/smtrat/smtrat/releases/download/v2.0/rat1_linux64.zip
[6] http://smtlib.cs.uiowa.edu/benchmarks.shtml

Benchmark (inequality only)	**raSAT**	**iSAT3**	**dReal**	**Z3 4.4**	**SMT-RAT**
zankl (SAT)	28	16	103^*	**54**	15
zankl (UNSAT)	10	12	0	**23**	13
meti-Tarski (SAT)(3220)	2940	2774	3534^*	**3220**	3055
meti-Tarski (UNSAT)(1526)	1138	1242	1172	**1523**	1298
hong (UNSAT)(20)	**20**	**20**	**20**	8	3
Total	4136	4064	1192	**4828**	4384
Time(s)	12363.34	1823.83	11145.23	64634.91	124823.17

Benchmark (with equations)	**raSAT**	**iSAT3**	**dReal**	**Z3 4.4**	**SMT-RAT**
zankl (SAT)(11)	**11**	0	11^*	**11**	**11**
zankl (UNSAT)(4)	**4**	**4**	**4**	**4**	**4**
meti-Tarski (SAT)(1805)	1313	1	1994^*	**1805**	1767
meti-Tarski (UNSAT)(1162)	1011	1075	965	**1162**	1114
kissing (SAT)(42)	6	0	18^*	**36**	7
kissing (UNSAT)(3)	0	0	**1**	0	0
hycomp (SAT)	0	0	317^*	**254**	33
hycomp (UNSAT)	1931	**2279**	2130	2200	1410
LassoRanker (SAT)	0	16	0^*	**120**	0
LassoRanker (UNSAT)	0	27	0	**118**	0
Total	4276	3750	3100	**5710**	4346
Time(s)	5978.58	4522.84	32376.47	124960.95	102940.90

Table 1: Comparison among SMT solvers on SMT-LIB benchmark ($* = \delta$-SAT)

6 Conclusion

This paper presented an SMT solver **raSAT** for polynomial constraints over reals using simple techniques, i.e., *interval arithmetic* and *the generalized intermediate value theorem*. Among ICP based SMT solvers, **iSAT3** requires bounded intervals for inputs and SAT detection of equations is limited (e.g., a SAT instance in integers). **dReal** handles only δ-SAT. **raSAT** pursues the theoretical limitation of SAT/UNSAT detection based on ICP.

ICP-based techniques have essential limitations on completeness. These limitations often appear with multiple roots and/or 0-dimensional ideals, and our next step is to combine computer algebraic techniques as a last resort. For instance, we observe during experiments that **raSAT** fails the touching cases with generally a rapid convergence until a box cannot be decomposed further (e.g., a box becomes smaller than the roundoff error limit). When such a box is detected, we plan to apply an existing package of Gröbner basis.

Acknowledgements

The authors would like to thank Pascal Fontain and Nobert Müller for valuable comments and kind instructions on **veriT** and iRRAM. They also thank to anonymous referees for fruitful suggestions. This work is supported by JSPS KAKENHI Grant-in-Aid for Scientific Research(B) (23300005,15H02684).

References

[1] Alliot, J.M., Gotteland, J.B., Vanaret, C., Durand, N., Gianazza, D.: Implementing an interval computation library for OCaml on x86/amd64 architectures. ICFP. (2012)

[2] Benhamou, F., Granvilliers, L.: Continuous and interval constraints. Handbook of Constraint Programming, 571–604. (2006)

[3] Bofill, M., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: The Barcelogic SMT solver. CAV 2008, LNCS 5123, 294–298. (2008)

[4] Comba, J.L.D., Stolfi, J.: Affine arithmetic and its applications to computer graphics. SIBGRAPI93, 9–18. (1993)

[5] Corzilius, F., Loup, U., Junges, S., Ábrahám, E.: SMT-RAT: An SMT-compliant nonlinear real arithmetic toolbox. SAT 2012, LNCS 7317, 442–448 (2012)

[6] En, N., Srensson, N.: An extensible SAT-solver. SAT 2003, LNCS 2919, 502–518. (2004)

[7] Fränzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. JSAT 1, 209–236 (2007)

[8] Ganai, M., Ivancic, F.: Efficient decision procedure for non-linear arithmetic constraints using cordic. FMCAD 2009, 61–68. (2009)

[9] Gao, S., Kong, S., Clarke, E.M.: Satisfiability modulo ODEs. FMCAD 2013, 105–112 (2013)

[10] Gao, S., Kong, S., Clarke, E.: dReal: An SMT solver for nonlinear theories over the reals. CADE-24, LNCS 7898, 208–214. (2013)

[11] Granvilliers, L., Benhamou, F.: Realpaver: An interval solver using constraint satisfaction techniques. ACM TOMS 32, 138–156. (2006)

[12] Hickey, T., Ju, Q., Van Emden, M.H.: Interval arithmetic: From principles to implementation. J. ACM 48(5), 1038–1068. (2001)

[13] Jovanovi, D., de Moura, L.: Solving non-linear arithmetic. IJCAR 2012, LNCS 7364, pp. 339–354. (2012)

[14] Khanh, T.V., Ogawa, M.: SMT for polynomial constraints on real numbers. TAPAS' 2012, ENTCS 289, 27 – 40. (2012).

[15] Messine, F.: Extentions of affine arithmetic: Application to unconstrained global optimization. J. UCS 8(11), 992–1015. (2002)

[16] Moore, R.: Interval analysis. Prentice-Hall series in automatic computation, Prentice-Hall (1966)

[17] Neumaier, A.: Interval Methods for Systems of Equations. Cambridge Middle East Library, Cambridge University Press (1990)

[18] Passmore, G.O.: Combined decision procedures for nonlinear arithmetics, real and complex. dissertation, School of Informatics, University of Edinburgh (2011)

[19] Passmore, G.O., Jackson, P.B.: Combined decision techniques for the existential theory of the reals. CALCULEMUS, LNCS 5625, 122–137. (2009)

[20] Ratschan, S.: Efficient solving of quantified inequality constraints over the real numbers. ACM TOCL 7(4), 723–748 (2006)

[21] Tseitin, G.: On the complexity of derivation in propositional calculus. Automation of Reasoning. Symbolic Computation. (1983)

[22] Tung, V.X., Khanh, T.V., Ogawa, M.: raSAT: SMT for polynomial inequality. SMT workshop 2014, 67 (2014)

[23] Zankl, H., Middeldorp, A.: Satisfiability of non-linear (ir)rational arithmetic. LPAR-16 . LNCS 6355, 481–500. (2010)