SMT-RAT: An Open Source C++ Toolbox for Strategic and Parallel SMT Solving*

Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp, and Erika Ábrahám

RWTH Aachen University, Germany

Abstract. During the last decade, popular SMT solvers have been extended step-by-step with a wide range of decision procedures for different theories. Some SMT solvers also support the user-defined tuning and combination of such procedures, typically via command-line options. However, configuring solvers this way is a tedious task with restricted options.
In this paper we present our modular and extensible `C++` library `SMT-RAT`, which offers numerous parameterized procedure modules for different logics. These modules can be configured and combined into an SMT solver using a comprehensible whilst powerful strategy, which can be specified via a graphical user interface. This makes it easier to construct a solver which is tuned for a specific set of problem instances. Compared to a previous version, we have extended our library with a number of new modules and support for parallelization in strategies. An additional contribution is our thread-safe and generic `C++` library `CArL`, offering efficient data structures and basic operations for real arithmetic, which can be used for the fast implementation of new theory-solving procedures.

1 Introduction

The *satisfiability problem* (SAT) poses the question whether a given propositional formula has a solution. *Satisfiability-modulo-theories* (SMT) tackles its natural extension, where we allow *theory constraints* in place of propositions. Lazy SMT solving [33] uses a *SAT solver* to find solutions of the *Boolean skeleton* of an *SMT formula* and invokes dedicated *theory solvers* to check the consistency in the underlying theory. Whereas full lazy approaches search for a complete Boolean solution before invoking theory solvers, *less lazy* techniques consult them more frequently. This cooperation highly benefits from an *SMT-compliant* theory solver, which (1) works *incrementally*, i.e., it should be able to exploit results from previous consistency checks; (2) it can *backtrack* according to the SAT solving; (3) for inconsistent constraint sets, it should be able to find an *infeasible subset* as explanation.

Most activities in the area of SMT solving focus on theories such as bit vectors (BV), uninterpreted functions (UF) or linear arithmetic over the reals (LRA)

* The final publication in the proceedings of SAT 2015 is available at Springer via `http://dx.doi.org/10.1007/978-3-319-24318-4_26`

and integers (LIA) resulting in the SMT solvers, e. g., `CVC4` [3], `MathSAT5` [8], `Yices2` [15] or `OpenSMT2` [6]. However, less activity can be observed for SMT solvers for (the existential fragment of) *non-linear real arithmetic* (NRA): besides some incomplete solvers like `MiniSmt` [38] and `iSAT3` [17,32], we are only aware of one SMT solver `Z3` [28,24] that is *complete* for NRA. Even fewer SMT solvers are available for (the existential fragment of) *non-linear integer arithmetic* (NIA), which is undecidable in general. To the best of our knowledge, only `Z3` and the SMT solving spin-off of `Aprove` [9] can tackle this theory.

One of the most widely used decision procedures for NRA is the *cylindrical algebraic decomposition* (CAD) method [10]. Other well-known methods use, e.g., *Gröbner bases* (GB) [35] or the *realization of sign conditions* [4]. Also some incomplete methods based on, e.g., *interval constraint propagation* (ICP) [17] or the *virtual substitution* (VS) [37] can handle significant fragments. However, the exponential worst-case complexity of solving NRA formulas [36,22] makes it challenging to develop practically feasible solutions. Embedding the above NRA decision procedures in SMT solvers as theory solvers is a promising symbiosis. Highly efficient SAT solvers can handle the Boolean problem structure and learn from previous (SAT and theory) conflicts. The expensive theory consistency checks then only concern conjunctions of theory constraints.

Available implementations of the above decision procedures are seldom available as *libraries*, and even if they are, they are not SMT compliant. Thus, for an SMT embedding, these mathematically complex decision procedures had to be adapted and extended before an SMT-compliant implementation could be realized. For the implementation, an *efficient library for basic computations with polynomials* was needed, which, if we want to have the door open for parallelization, must be additionally *thread-safe*. Furthermore, on a given problem instance there might be significant differences in the running times of different theory solvers. Therefore, we aim at their *strategic combination* [29] to increase usability.

We have developed the `C++` library `SMT-RAT` containing a variety of *modules* implementing SMT-compliant solving procedures. The modular design of `SMT-RAT` facilitates an easy extension by further solving procedures. Modules share a common interface allowing their combination according to a user-defined strategy resulting in an SMT solver. Currently, `SMT-RAT` can solve problems of (the quantifier-free fragments of) LRA, LIA, NRA and NIA. Compared to the previous version of `SMT-RAT` [12], (1) we have extended and optimized the VS module, the GB module (can now handle inequalities and simplify formulas), and the CAD module (can now handle arbitrary instead of only univariate polynomials); (2) we have implemented a Simplex module [15], an ICP module [18], a module embedding a SAT solver and a module simplifying polynomial constraints using non-trivial factorization and sum-of-squares decomposition; (3) we have implemented a general branch-and-bound method for finding integer solutions with NRA modules, where the splitting decisions are lifted to the SAT level; (4) we have extended `SMT-RAT` to support strategies, which compose procedures such that they run in parallel on multiple cores and implemented an easy-to-grasp graphical user interface for the construction of such a strategy; (5) we have extended the module interfaces to support lemma exchange and lightweight invocation, where it is allowed to avoid hard obstacles during solving at the price of possibly not finding a conclusive answer.

2 System Architecture

2.1 Data structures and basic procedures: `CArL` [26]

The current version of `SMT-RAT` integrates custom-designed data structures for SMT formulas and basic functions to manipulate them, bundled in the library `CArL`, which has also been successfully used in the tool `Prophesy` [13].

While there exist `C++` libraries for the manipulation of polynomials such as CoCoA[1] and `GiNaC` [5], these libraries share some common deficits. First of all, they lack customization possibilities and are usually tied to one fixed *representation of numbers*. Secondly, the libraries are often not flexible when it comes to manipulation of *variable* (and polynomial) *orderings*, which is essential for efficient implementations of a CAD or a GB procedure. Thirdly, the libraries are usually not thread-safe, which precludes the design of parallel solvers.

In `CArL`, the data structure for *SMT formulas* is a directed acyclic graph, with Boolean operators as inner nodes and Boolean variables or theory constraints, e. g., polynomial inequalities, as leafs. Essential simplifications and normalizations [14] are applied by default and identical formulas are stored only once.

Polynomials are represented by default as a sum of terms. We mark leading and constant terms and sort all terms only on demand. The data structure is templated in several ways. Amongst others, we can use rational numbers, native numbers, intervals and polynomials as coefficients. Furthermore, we can use different orderings and store additional information with the polynomials with minimal overhead by utilizing policy templates. Besides, `CArL` supports univariate representations of multivariate polynomials, which is essential for, i. a., the CAD.

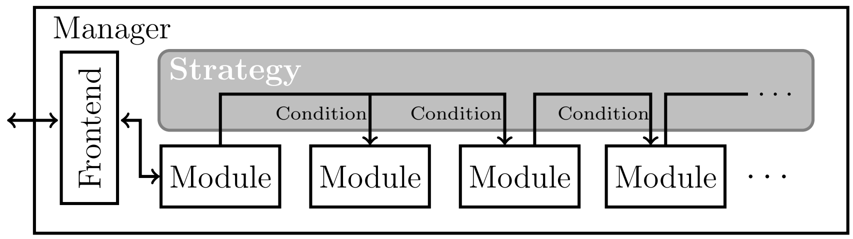
Variables are represented by bit vectors, encoding their identity, their domain and their rank (for support of fast custom-ordering of variables). Additional information is stored in a central pool. For the *representation of rational numbers*, we support `gmp` [20] (thread-safe) and `cln` [21] (faster single-threaded). Algebraic numbers are represented by the interval-isolated root of a univariate polynomial. Intervals in `CArL` are an extension of `boost` intervals also allowing open bounds.

Besides standard arithmetic operations, `CArL` includes the required procedures for CAD, including Sturm sequences and root isolation, and a variant of the Buchberger algorithm to compute Gröbner bases. The implemented methods are specifically tailored towards SMT compliance.

2.2 Interfaces and strategic compositions of procedures: `SMT-RAT` [34]

Based on `CArL`'s data structures and basic functions, a rich set of SMT-compliant implementations of NRA/NIA procedures is provided by `SMT-RAT`. Each procedure is encapsulated in a *module*, which fixes a common interface. Modules can be

Fig. 1: A snapshot of an `SMT-RAT` composition of an SMT solver.



composed to a solver according to a user-defined *strategy*. The *manager* class provides the API, including the parsing of an SMT-LIBv2 input file, and a manager instance maintains the allocation of *solving tasks* to modules according to the strategy. An overview is given in Figure 1.

Modules Each module m has an initially empty *set of received formulas* $C_{rcv}(m)$. We can manipulate $C_{rcv}(m)$ by adding (removing) formulas φ to (from) it with `add(`φ`)` (`remove(`φ`)`). The main function of a module is `check(bool full)`, which either decides whether the conjunction of the received formulas in $C_{rcv}(m)$ is satisfiable or not, returning `sat` or `unsat`, respectively, or returns `unknown`. If the function's argument `full` is set to `false`, the underlying procedure of m is allowed to omit hard obstacles during solving at the cost of returning `unknown` in more cases. Usually, $C_{rcv}(m)$ is only slightly changed between two consecutive `check` calls, hence, the solver's performance can be significantly improved if a module works incrementally and supports backtracking. In case m determines the unsatisfiability of $C_{rcv}(m)$, it can return an *infeasible subset* $C_{inf}(m) \subseteq C_{rcv}(m)$. Moreover, a module can specify *lemmas*, which are valid formulas. They encapsulate information which can be extracted from a module's internal state and propagated among other modules. Furthermore, a module itself can ask other modules for the satisfiability of its *set of passed formulas* denoted by $C_{pas}(m)$, if it invokes the procedure `runBackends(bool full)` (controlled by the manager). It thereby delegates work to modules that may be more suitable for the (sub-)problems in $C_{pas}(m)$.

Strategy `SMT-RAT` supports user-defined *strategies* for the composition of modules. A graphical user interface can be used to specify strategies as directed trees $T := (V, E)$ with a set V of modules as nodes and the transitions $E \subseteq V \times \Omega \times \Sigma \times V$, with Ω being a set of *conditions* and Σ being a set of *priority values*. A condition is an arbitrary Boolean combination of formula properties, such as propositions about the Boolean structure of the formula, e.g., whether it is in conjunctive normal form (CNF), about the constraints, e. g., whether it contains equations, or about the polynomials, e.g., whether they are linear. Furthermore, each edge carries a unique priority value from $\Sigma = \{1, \ldots, |E|\}$.

Fig. 2: Example strategies with `SMT-RAT` ($\top \mathrel{\hat{=}}$ no condition).

$$\texttt{rat}_1\colon\quad \texttt{CNF}_M \xrightarrow{\top,1} \texttt{PP}_M \xrightarrow{\top,2} \texttt{SAT}_M \xrightarrow{\top,3} \texttt{SIM}_M \xrightarrow{\top,4} \texttt{VS}_M \xrightarrow{\top,5} \texttt{CAD}_M$$

$$\texttt{rat}_2\colon\quad \texttt{CNF}_M \xrightarrow{\top,1} \texttt{PP}_M \xrightarrow{\top,2} \texttt{SAT}_M \xrightarrow{\top,3} \texttt{ICP}_M \xrightarrow{\top,4} \texttt{VS}_M \xrightarrow{\top,5} \texttt{CAD}_M$$

$$\texttt{rat}_3\colon\quad \begin{tikzcd} & & & & \texttt{VS}_M \arrow[r, "{\top,5}"] & \texttt{CAD}_M \\ \texttt{CNF}_M \arrow[r, "{\top,1}"] & \texttt{PP}_M \arrow[r, "{\top,2}"] & \texttt{SAT}_M \arrow[r, "{\top,3}"] & \texttt{SIM}_M \arrow[ur, "{\top,4}"] \arrow[dr, "{\top,6}"'] & & \\ & & & & \texttt{CAD}_M & \end{tikzcd}$$

$$\texttt{rat}_4\colon\quad \begin{tikzcd} & & \texttt{SAT}_M \arrow[r, "{\top,3}"] & \texttt{SIM}_M \arrow[r, "{\top,4}"] & \texttt{VS}_M \arrow[r, "{\top,5}"] & \texttt{CAD}_M \\ \texttt{CNF}_M \arrow[r, "{\top,1}"] & \texttt{PP}_M \arrow[ur, "{\top,2}"] \arrow[dr, "{\top,6}"'] & & & & \\ & & \texttt{SAT}_M \arrow[r, "{\top,7}"'] & \texttt{ICP}_M \arrow[r, "{\top,8}"'] & \texttt{VS}_M \arrow[r, "{\top,9}"'] & \texttt{CAD}_M \end{tikzcd}$$

Manager The *manager* holds the strategy $T = (V, E)$ and the SMT solver's input formula C_{input}. Initially, the manager calls the method `check` of the module m_r, being the root of T, with $C_{rcv}(m_r) = C_{input}$. Whenever a module $m \in V$ calls `runBackends`, the manager adds a *solving task* $(\sigma,\ m,\ m')$ to its priority queue Q of solving tasks (ordered by the priority value), if there exists an edge $(m,\ \omega,\ \sigma,\ m') \in E$ such that ω holds for $C_{pas}(m)$. If a processor p on the machine on which `SMT-RAT` is executed is available, the first solving task of Q is assigned to p and popped from Q. The manager thereby starts `check` of m' with $C_{rcv}(m') = C_{pas}(m)$ and passes the result (including infeasible subsets and lemmas) back to m, which can now benefit in its solving and reasoning process from this shared information. Note that a strategy-based composition of modules works incrementally and supports backtracking not just within one module but as a whole. Therefore, each module m stores the subsets of $C_{rcv}(m)$, which form the reasons for a passed formula being added. In order to exploit the incrementality of the modules, all backends executed in parallel terminate in a consistent state (instead of being killed), if one of them finds an answer.

Procedures implemented as modules Usually, a SAT solver forms the heart of an SMT solver. In `SMT-RAT`, the module $\texttt{SAT}_M$ abstracts $C_{rcv}(\texttt{SAT}_M)$ to propositional logic and uses the efficient SAT solver `minisat` [16] to find a satisfying solution for the Boolean abstraction. It invokes `runBackends` where $C_{pas}(\texttt{SAT}_M)$ contains the constraints abstracted by the assigned Boolean variables in a less-lazy fashion [33]. The module $\texttt{SIM}_M$ implements the Simplex method equipped with branch-and-bound and cutting-plane procedures as presented in [15]. We apply it on the linear constraints of any conjunction of NRA/NIA constraints. For a conjunction of nonlinear constraints `SMT-RAT` provides the modules $\texttt{GB}_M$, $\texttt{VS}_M$ and $\texttt{CAD}_M$, implementing GB [25], VS [11] and CAD [27] procedures, respectively. Moreover, the module $\texttt{ICP}_M$ uses ICP similar as presented in [18], lifting splitting decisions and contraction lemmas to a preceding $\texttt{SAT}_M$ and harnessing other modules for

Table 1: Results in seconds (timeout = 200s) obtained on a 2.1 GHz AMD.

Benchmark (#examples)	Z3		$\mathtt{rat}_1$		$\mathtt{rat}_2$		$\mathtt{rat}_3$		$\mathtt{rat}_4$	
	solved	*time*	*solved*	*time*	*solved*	*time*	*solved*	*time*	*solved*	*time*
Hong (20)	50.0%	72.8	15.0%	< 1.0	**100.0%**	**< 1.0**	15.0%	< 1.0	**100.0%**	**< 1.0**
- sat	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0
- unsat	10	72.8	3	< 1.0	20	< 1.0	3	< 1.0	20	< 1.0
Kissing (45)	**68.9%**	**1155.9**	17.8%	50.2	35.6%	375.9	28.9%	26.5	28.9%	54.4
- sat	31	1155.9	8	50.2	16	375.9	13	26.5	13	54.4
- unsat	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0
MetiTarski (7713)	**99.9%**	**370.5**	92.7%	4964.3	92.8%	4658.3	93.2%	3974.8	95.6%	3109.4
- sat	5025	133.7	4766	2180.8	4740	2952.1	4802	1803.8	4815	2290.4
- unsat	2684	236.8	2385	2783.4	2418	1706.2	2388	2170.9	2560	819.0
Keymaera (421)	**99.8%**	**11.5**	97.6%	26.0	96.9%	17.0	96.4%	74.7	98.1%	25.3
- sat	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0
- unsat	420	11.5	411	26.0	408	17.0	406	74.7	413	25.3
Witness (99)	21.2%	107.1	72.7%	2110.9	64.6%	332.2	21.2%	10.9	**75.8%**	**937.9**
- sat	4	75.3	55	2110.6	47	331.9	4	9.8	58	937.6
- unsat	17	31.8	17	< 1.0	17	< 1.0	17	1.1	17	< 1.0
AProve (8829)	**94.0%**	**12011.6**	79.5%	5077.8	80.3%	6128.4	76.6%	10645	80.0%	3886.3
- sat	8014	11090.9	6965	5038.7	7038	5695.5	6698	10181.3	7009	3782.3
- unsat	284	920.7	50	39.1	56	432.9	68	463.6	58	104.0
Calypto (177)	**98.9%**	**11.6**	83.6%	123.3	78.0%	323.9	37.3%	402.1	85.3%	308.3
- sat	79	7.5	64	46.5	59	236.5	21	304.5	67	224.7
- unsat	96	4.1	84	76.7	79	87.4	45	97.7	84	83.6

nonlinear conjunctions of constraints as backends. The module $\mathtt{CNF}_M$ invokes `runBackends` on $C_{pas}(\mathtt{CNF}_M)$ being a formula in CNF which is satisfiability-equivalent to $C_{rcv}(\mathtt{CNF}_M)$. The module $\mathtt{PP}_M$ performs some preprocessing based on factorizations and sum-of-square decompositions of polynomials.

3 Experimental Results and Future Work

We evaluated the four strategies specified in Figure 2 on the five NRA benchmark sets Hong [23], Kissing (both crafted and dimension dependent), MetiTarski [2], Keymaera [30], Witness [31] (generated by theorem proving, counterexample-guided synthesis and formal verification, respectively) and the two NIA benchmark sets AProve [19] and Calypto [7] (generated by automated termination analysis and sequential equivalence checking, respectively). The first two strategies, $\mathtt{rat}_1$ and $\mathtt{rat}_2$, are sequential, using a nested combination of Simplex/ ICP, VS and CAD. The third strategy $\mathtt{rat}_3$ extends the first one by applying CAD in parallel to the nested combination of VS and CAD. The last strategy $\mathtt{rat}_4$ basically runs the first two strategies in parallel.

Table 1 shows the experimental results, which compare the four `SMT-RAT` strategies with the currently fastest SMT solver for these theories, `Z3`, showing that `SMT-RAT` is already competitive. We ran `Z3` sequentially and in parallel and took the best of both real-time performances for each instance. The column "solved" shows the number of solved instances and the column "time" states the accumulated solving time not including timeouts. On Witness, `SMT-RAT` performs even better than `Z3`, as it benefits from the algebraic procedures being tuned for small variable domains as occurring in these examples. It also performs better on Hong, where it highly profits from the ICP module. Even though $\mathtt{rat}_4$

is the best SMT-RAT strategy overall, we observed that both parallel strategies perform worse than expected, which is due to $\mathtt{CAD}_M$ currently not always being able to terminate quickly with a consistent state when called in parallel. We want to extend SMT-RAT with further modules based on linearization, bit-blasting and further preprocessing. More experimental results can be found on our website [34].

References

1. Abbott, J., Bigatti, A.M.: CoCoALib: A C++ library for computations in commutative algebra... and beyond. In: Mathematical Software ICMS 2010, LNCS, vol. 6327, pp. 73–76. Springer (2010)
2. Akbarpour, B., Paulson, L.C.: Metitarski: An automatic theorem prover for real-valued special functions. Journal of Automated Reasoning 44(3), 175–205 (2010)
3. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Proc. of CAV'11. LNCS, vol. 6806, pp. 171–177. Springer (2011)
4. Basu, S., Pollack, R., Roy, M.: Algorithms in Real Algebraic Geometry. Springer (2010)
5. Bauer, C., Frink, A., Kreckel, R.: Introduction to the GiNaC framework for symbolic computation within the C++ programming language. Journal of Symbolic Computation 33(1), 1–12 (2002)
6. Bruttomesso, R., et al.: The OpenSMT solver. In: Proc. of TACAS'10. LNCS, vol. 6015, pp. 150–153. Springer (2010)
7. Chauhan, P., Goyal, D., Hasteer, G., Mathur, A., Sharma, N.: Non-cycle-accurate sequential equivalence checking. In: Proc. of DAC'09. pp. 460–465. ACM Press (2009)
8. Cimatti, A., Griggio, A., Schaafsma, B., Sebastiani, R.: The MathSAT5 SMT solver. In: Proc. TACAS'13, LNCS, vol. 7795, pp. 93–107. Springer (2013)
9. Codish, M., Fekete, Y., Fuhs, C., Giesl, J., Waldmann, J.: Exotic semi-ring constraints. In: Proc. of SMT'12. EPiC Series, vol. 20, pp. 88–97. EasyChair (2013)
10. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Automata Theory and Formal Languages. LNCS, vol. 33, pp. 134–183. Springer (1975)
11. Corzilius, F., Ábrahám, E.: Virtual substitution for SMT solving. In: Proc. of FCT'11. LNCS, vol. 6914, pp. 360–371. Springer (2011)
12. Corzilius, F., Loup, U., Junges, S., Ábrahám, E.: SMT-RAT: An SMT-compliant nonlinear real arithmetic toolbox. In: Proc. of SAT'12. LNCS, vol. 7317, pp. 442–448. Springer (2012)
13. Dehnert, C., Junges, S., Jansen, N., Corzilius, F., Volk, M., Bruintjes, H., Katoen, J.P., Ábrahám, E.: PROPhESY: A PRObabilistic ParamEter SYthesis Tool. In: Proc. of CAV'15. LNCS, vol. 9207. Springer (2015)
14. Dolzmann, A., Sturm, T.: Simplification of quantifier-free formulas over ordered fields. Journal of Symbolic Computation 24, 209–231 (1995)
15. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: Proc. of CAV'06. LNCS, vol. 4144, pp. 81–94. Springer (2006)
16. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Proc. of SAT'03. LNCS, vol. 2919, pp. 502–518. Springer (2004)
17. Fränzle, M., et al.: Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure. Journal on Satisfiability, Boolean Modeling and Computation 1(3-4), 209–236 (2007)

18. Gao, S., Ganai, M.K., Ivancic, F., Gupta, A., Sankaranarayanan, S., Clarke, E.M.: Integrating ICP and LRA solvers for deciding nonlinear real arithmetic problems. In: Proc. of FMCAD'10. pp. 81–89. IEEE (2010)
19. Giesl, J., Brockschmidt, M., Emmes, F., Frohn, F., Fuhs, C., Otto, C., Plücker, M., Schneider-Kamp, P., Ströder, T., Swiderski, S., Thiemann, R.: Proving termination of programs automatically with AProVE. In: Proc. of IJCAR'14. LNAI, vol. 8562, pp. 184–191. Springer (2014)
20. Granlund, T., the GMP development team: GNU MP: The GNU Multiple Precision Arithmetic Library, `http://gmplib.org`
21. Haible, B., Kreckel, R.B.: CLN: Class Library for Numbers. `http://www.ginac.de/CLN`
22. Heintz, J., Roy, M.F., Solerno, P.: On the theoretical and practical complexity of the existential theory of reals. The Computer Journal 36(5), 427–431 (1993)
23. Hong, H.: Comparison of several decision algorithms for the existential theory of the reals. Tech. Rep. 91-41, Research Institute for Symbolic Computation, Johannes Kepler University Linz (1991)
24. Jovanovic, D., de Moura, L.M.: Solving non-linear arithmetic. In: Proc. of IJCAR'12. LNAI, vol. 7364, pp. 339–354. Springer (2012)
25. Junges, S., Loup, U., Corzilius, F., Ábrahám, E.: On Gröbner bases in the context of satisfiability-modulo-theories solving over the real numbers. In: Algebraic Informatics, LNCS, vol. 8080, pp. 186–198. Springer (2013)
26. Kremer, G., Corzilius, F., Junges, S., Schupp, S., Ábrahám, E.: CArL: Computer ARithmetic and Logic Library, `https://github.com/smtrat/carl`
27. Loup, U., Scheibler, K., Corzilius, F., Ábrahám, E., Becker, B.: A symbiosis of interval constraint propagation and cylindrical algebraic decomposition. In: Proc. of CADE-24. LNCS, vol. 7898, pp. 193–207. Springer (2013)
28. de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: Proc. of TACAS'08. LNCS, vol. 4963, pp. 337–340. Springer (2008)
29. de Moura, L., Passmore, G.O.: The strategy challenge in SMT solving. In: Automated Reasoning and Mathematics, pp. 15–44. Springer (2013)
30. Platzer, A., Quesel, J.D., Rümmer, P.: Real world verification. In: Proc. of CADE-22. LNCS, vol. 5663, pp. 485–501. Springer (2009)
31. Ravanbakhsh, H., Sankaranarayanan, S.: Counterexample guided synthesis of switched controllers for reach-while-stay properties. CoRR abs/1505.01180 (2015), `http://arxiv.org/abs/1505.01180`
32. Scheibler, K., Kupferschmid, S., Becker, B.: Recent improvements in the SMT solver iSAT. In: Proc. of MBMV'13. pp. 231–241. Institut für Angewandte Mikroelektronik und Datentechnik, Fakultät für Informatik und Elektrotechnik, Universität Rostock (2013)
33. Sebastiani, R.: Lazy satisfiability modulo theories. Journal on Satisfiability, Boolean Modeling and Computation 3, 141–224 (2007)
34. `https://github.com/smtrat/smtrat/wiki`
35. Weispfenning, V.: A new approach to quantifier elimination for real algebra. In: Quantifier Elimination and Cylindrical Algebraic Decomposition. pp. 376–392. Texts and Monographs in Symbolic Computation, Springer (1998)
36. Weispfenning, V.: The complexity of linear problems in fields. Journal of Symbolic Computation 5(1-2), 3–27 (1988)
37. Weispfenning, V.: Quantifier elimination for real algebra - the quadratic case and beyond. Appl. Algebra Eng. Commun. Comput. 8(2), 85–101 (1997)
38. Zankl, H., Middeldorp, A.: Satisfiability of non-linear (ir)rational arithmetic. In: Proc. of LPAR'10. LNAI, vol. 6355, pp. 481–500. Springer (2010)