SMT-RAT: An SMT-Compliant Nonlinear Real Arithmetic Toolbox (Tool Presentation)

Florian Corzilius, Ulrich Loup, Sebastian Junges, and Erika Ábrahám

RWTH Aachen University, Germany

Abstract. We present `SMT-RAT`, a `C++` toolbox offering theory solver modules for the development of SMT solvers for *nonlinear real arithmetic* (*NRA*). NRA is an important but hard-to-solve theory and only fragments of it can be handled by some of the currently available SMT solvers. Our toolbox contains modules implementing the *virtual substitution* method, the *cylindrical algebraic decomposition* method, a *Gröbner bases simplifier* and a *general simplifier*. These modules can be combined according to a user-defined strategy in order to exploit their advantages.

1 Introduction

The *Satisfiability-Modulo-Theories* (SMT) problem is the problem of checking *SMT formulas*, i.e. Boolean combinations of constraints of one or more theories, for satisfiability. SMT solvers use a SAT solver to find satisfying solutions for the Boolean skeleton of an input SMT formula, which are in turn checked for consistency with other decision procedures for the underlying theories.

The last decade brought great achievements in the field of SMT solving. For instance, the SMT-LIB standard defines a common input format for SMT solvers and provides the community with benchmarks for different theories. In addition, SMT competitions motivate the development and improvement of SMT solvers. Nowadays, different efficient SMT solvers are available for several theories, e.g., for linear real arithmetic. However, only a few solvers support *nonlinear real arithmetic* (*NRA*), the theory of the reals with addition and multiplication.

Nonlinear real arithmetic was shown to be decidable by Tarski [16]. Though the worst-case time complexity of solving real-arithmetic formulas is doubly exponential in the number of variables [18], its existential fragment, which is addressed by SMT solving, can be solved in exponential time [13]. One of the most widely used decision procedures for NRA is the *cylindrical algebraic decomposition* (*CAD*) method [6]. Other well-known methods use, e.g., *Gröbner bases* [17] or the *realization of sign conditions* [2]. Some incomplete methods based on, e.g., *interval constraint propagation* (*ICP*) [12] and *virtual substitution (VS)* [19], can handle significant fragments and, even though they have the same worst-case complexity as the complete methods, they are more efficient in practice. Moreover, they are well-suited for a combination with complete methods, to which they pass reduced sub-problems.

The methods mentioned above are implemented in different tools. For example, `QEPCAD` [4] implements the CAD method, the `Redlog` package [11] of the computer algebra system `Reduce` offers an optimized combination of the VS, the CAD, and Gröbner bases methods, and `RAHD` [15] combines different methods by a user-defined strategy. The strength of these tools lies in solving *conjunctions* of real-arithmetic constraints, but they are not designed for formulas with arbitrary Boolean structures. A natural way to join the advantages of these tools with those of SAT solving suggests their embedding in an SMT solver.

There are some SMT solvers available which support fragments of NRA. `Z3` [20] applies an optimized combination of linear arithmetic decision procedures, ICP and the VS method. `MiniSmt` tries to reduce NRA problems to linear real arithmetic and `HySAT/iSAT` uses ICP. All these SMT solvers are incomplete for NRA, i.e., they can not check satisfiability for all real-arithmetic SMT formulas.

The development of a complete SMT solver for NRA is problematic because the aforementioned algebraic decision procedures are not *SMT-compliant*, i.e., they do not fulfill the requirements for the embedding into an efficient SMT solver. Firstly, in less lazy SMT solving, theory solvers should be able to work *incrementally*, i.e., if they determine the satisfiability of a set of constraints, they should be able to check an extended set of constraints on the basis of the previous result. Secondly, in case a constraint set is unsatisfiable, theory solvers should be able to compute an *infeasible subset* as explanation. Thirdly, they must be able to *backtrack* according to the search of the SAT solver.

In this paper, we present the open-source `C++` toolbox `SMT-RAT`, which implements real-arithmetic constraint simplifier and theory solver modules suited for the embedding into an SMT solver. Besides standard libraries, `SMT-RAT` invokes only the libraries `GiNaC` and `GiNaCRA` [14]. The source code with all modules and a manual with examples can be found at http://smtrat.sourceforge.net/. Our toolbox `SMT-RAT` offers an incremental implementation of the VS method [1,7], which can generate infeasible subsets and supports backtracking. It also provides two incremental implementations of the CAD method. One can handle the multivariate case, whereas the other one is specialized on univariate instances only and can generate infeasible subsets. Furthermore, two simplifier modules are available based on smart simplifications [10] and Gröbner bases, respectively.

This is the first release of our toolbox. At this stage, we do not aim at competing with state-of-the-art solvers in all categories. For example, we do not yet offer extensive simplifiers, ICP, or theory solver modules for linear arithmetic. The main advantages of our toolbox lie in offering (1) *complete* SMT-compliant decision procedures, (2) the possibility to *combine* theory solvers according to a user-defined strategy, and (3) a *modular* and *extendable open-source* implementation. Syntactically, our strategies are more simple than those proposed in [9]. However, we choose a procedure depending on not only the formula but also on the history of solving, offering a very flexible approach.

We use `SMT-RAT` to extend the open-source SMT solver `OpenSMT` [5] by the theory NRA. First experimental results comparing this tool with `Z3` and `CVC3`

[8] indicate that for some highly nonlinear benchmark sets we are able to solve a much larger number of instances, but for some other benchmark sets we still need further improvements.

In the following, we first give some preliminaries in Section 2 and a short introduction to the toolbox design in Section 3. We give some experimental results in Section 4 and conclude the paper in Section 5.

2 Satisfiability Modulo Real Arithmetic

SMT solving denotes an algorithmic framework for solving Boolean combinations of constraints from some theories. SMT solvers combine a SAT solver computing satisfying assignments for the Boolean structure of the SMT formula with procedures to check the consistency of theory constraints. For more details on SMT related topics we refer to [3, Ch. 26].

We consider *NRA formulas* φ, which are Boolean combinations of *constraints* c comparing *polynomials* p to 0. A polynomial p can be a constant, a variable x, or a composition of polynomials by addition, subtraction or multiplication:

$$\begin{array}{rcccccccccccc} p & ::= & 0 & \mid & 1 & \mid & x & \mid & (p+p) & \mid & (p-p) & \mid & (p\cdot p) \\ c & ::= & p=0 & \mid & p<0 & \mid & p>0 & & & & & & \\ \varphi & ::= & c & \mid & (\neg\varphi) & \mid & (\varphi\wedge\varphi) & \mid & (\exists x\varphi) & & & & \end{array}$$

The semantics of NRA formulas is defined as usual.

Given a polynomial $p = a_1 x_1^{e_{1,1}} \cdots x_n^{e_{n,1}} + \cdots + a_k x_1^{e_{1,k}} \cdots x_n^{e_{n,k}}$ in monomial normal form, by $\deg(p) := \max_{1\leq j\leq k}(\sum_{i=1}^{n} e_{i,j})$ we denote the *degree of* p. We call an NRA formula φ *linear* if $\deg(p) \leq 1$ for all polynomials p in φ, and *nonlinear* otherwise. *Linear real arithmetic* (*LRA*) formulas are linear NRA formulas.

3 Toolbox Design

Our toolbox has a modular C++ class design which can be used to compose NRA theory solvers for an SMT embedding in a *dynamic* and *hierarchic* fashion. Our NRA theory solvers are instances of `Manager`, which offers an interface to communicate with the environment and which coordinates the satisfiability check according to a user-defined `Strategy`. Such a strategy combines basic NRA theory solver modules, derived from `Module`. Figure 1 shows an example configuration. Moreover, a `Java`-based graphical user interface can be used for an intuitive and user-friendly specification of strategies (and the automatic generation of a corresponding `Strategy` class). Next, we briefly describe these concepts. For more details we refer to the manual of `SMT-RAT` at http://smtrat.sourceforge.net/manual/manual.pdf.

The `Formula` Class. `Formula` instances contain, besides a sequence of NRA constraints, a bitvector storing some information about the problem and the history of its check. E.g., there is a bit which is 1 if some of the constraints are equations. Also for each module there is a bit which is 1 if the module was

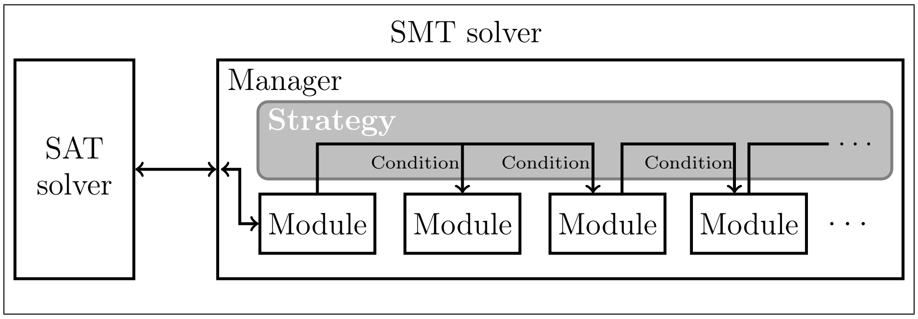

Fig. 1. A snapshot of an `SMT-RAT` composition embedded in an SMT solver

already invoked on the given problem. Such information can be used to specify conditions under which a procedure should be invoked for a certain problem.

The `Module` Class. A *module* is an SMT-compliant implementation of a procedure (e.g., constraint simplifier, an incomplete procedure or a complete decision procedure) which can be used for the satisfiability check of NRA formulas. A module's interface allows to add constraints, to push and pop backtrack points, to check the so far added constraints for consistency and to obtain an infeasible subset of these constraints if they are detected to be inconsistent.

Modules have the opportunity to call other modules (*backends*) on sub-problems. A novel achievement of our toolbox is that this call hierarchy is *dynamic* and guided by a *user-defined strategy*. Currently, we only support sequential execution, parallel solving is planned for later releases.

Inheritance can be used to extend existing modules. Besides the basic type `Module`, our toolbox offers five sub-classes. `SimplifierModule` ($\texttt{SI}_M$) implements smart simplifications [10], while `GroebnerModule` ($\texttt{GS}_M$) simplifies equation systems using Gröbner bases and probably detects inconsistency. The CAD method is implemented in `UnivariateCADModule` ($\texttt{UC}_M$) for the univariate case and in `CADModule` ($\texttt{MC}_M$) for the general multivariate case. The last module class `VSModule` ($\texttt{VS}_M$) implements a version of the VS method as published in [7].

The `Strategy` Class. `SMT-RAT` offers a framework to integrate single modules to powerful and efficient *composed theory solvers*. The composition relies on a user-defined `Strategy` that specifies for each `Formula` instance which module should be used for its check. A strategy is basically a sequence of condition-module pairs. For each `Formula` instance, it determines the first module whose condition evaluates to true on the bitvector of the formula. E.g., the strategy "$c_1\ ?\ (m_1)\ :\ (c_2\ ?\ (m_2)\ :\ (m_3))$" determines m_1 as module type for φ if the bitvector of φ fulfills the condition c_1. If φ does not fulfill c_1 but c_2, then an instance of m_2 is called, otherwise of m_3.

The `Manager` Class. The `Manager` contains references to the available module instances and to the user-defined strategy. It manages, on the one hand, the creation and linking of the modules, and, on the other hand, the communication between them and the environment, e.g., the frontend of an SMT solver.

Table 1. Running times [sec] of $\texttt{Rat}_1$, $\texttt{Rat}_2$, Z3 and CVC3 on four benchmarks

	$\texttt{Rat}_1$		$\texttt{Rat}_2$		Z3		CVC3	
	solved	acc. time	solved	acc. time	solved	acc. time	solved	acc. time
bouncing ball	43/52	4226.24	43/52	424.63	0/52	0.00	0/52	0.00
etcs	2/5	136.15	2/5	135.05	1/5	42.00	1/5	0.11
rectangular pos.	16/22	305.54	16/22	299.54	22/22	27.29	0/22	0.00
zankl	22/166	26.30	22/166	25.81	62/166	1138.96	9/166	2.86

4 Experimental Results

All experiments were performed on an Intel® Core™ i7 CPU at 2.80 GHz with 4 GB RAM with Gentoo Linux. We defined two strategies

$$c_1 \;?\; (\texttt{MC}_M) \;:\; (c_2 \;?\; (\texttt{UC}_M) \;:\; (c_4 \;?\; (\texttt{VS}_M) \;:\; (\texttt{SI}_M)))$$
$$\text{and}\quad c_1 \;?\; (\texttt{MC}_M) \;:\; (c_2 \;?\; (\texttt{UC}_M) \;:\; (c_3 \;?\; (\texttt{VS}_M) \;:\; (c_4 \;?\; (\texttt{GS}_M) \;:\; (\texttt{SI}_M))))$$

where c_1, c_2, c_3 and c_4 hold if the $\texttt{UC}_M$, the $\texttt{VS}_M$, the $\texttt{GS}_M$ and $\texttt{SI}_M$ was invoked and could not solve the given formula, respectively. We embedded two theory solver components using these strategies into OpenSMT, yielding the SMT solvers $\texttt{Rat}_1$ and $\texttt{Rat}_2$, respectively, which we compared to CVC3 2.4 and Z3 3.1, the latter being the winner of last year's SMT competition for NRA.

Table 1 shows the running times in seconds on four benchmark sets with the timeout of 150 seconds. The first one models the nonlinear movement of a bouncing ball which may drop into a hole. The second one is a nonlinear version of the European Train Control System benchmark set. The third one contains problems to checks whether a given set of rectangles fits in a given area. The last benchmark set stems from the SMT competition in 2011.

The results show, that we can solve many examples which Z3 and CVC3 cannot solve. However, Z3 does a better job in the last two benchmark sets, where the major part of the formula is linear. Here it can benefit from its ICP and Simplex solver checking the linear fragment. Nevertheless, the results point out that we can build efficient SMT solvers for NRA using OpenSMT and SMT-RAT. Furthermore, it indicates that extending SMT-RAT by modules, e.g. implementing Simplex or ICP, would lead to significant improvements.

5 Conclusion and Future Work

SMT-RAT is a toolbox contributing several SMT-compliant simplifier and theory solver modules and a framework to combine them according to a user-defined strategy. Experimental results show that an SMT solver enriched by SMT-RAT for solving NRA can compete with state-of-the-art SMT solvers and even solve instances, which they cannot solve.

The design of SMT-RAT aims at modularity, extensibility, and the easy adding of new modules. Moreover, we plan to improve the performance of SMT-RAT compositions by modules implementing, e.g., Simplex and ICP. Furthermore, we want to extend the framework to allow parallel calls of modules, theory propagation, and shared heuristics.

References

1. Ábrahám, E., et al.: A lazy SMT-solver for a non-linear subset of real algebra. In: Proc. of SMT 2010 (2010)
2. Basu, S., Pollack, R., Roy, M.: Algorithms in Real Algebraic Geometry. Springer (2010)
3. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
4. Brown, C.W.: QEPCAD B: A program for computing with semi-algebraic sets using CADs. SIGSAM Bulletin 37(4), 97–108 (2003)
5. Bruttomesso, R., Pek, E., Sharygina, N., Tsitovich, A.: The OpenSMT Solver. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 150–153. Springer, Heidelberg (2010)
6. Collins, G.E.: Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In: Brakhage, H. (ed.) GI-Fachtagung 1975. LNCS, vol. 33, pp. 134–183. Springer, Heidelberg (1975)
7. Corzilius, F., Ábrahám, E.: Virtual Substitution for SMT-Solving. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 360–371. Springer, Heidelberg (2011)
8. http://cs.nyu.edu/acsys/cvc3/
9. de Moura, L., Passmore, G.O.: The strategy challenge in SMT solving, http://research.microsoft.com/en-us/um/people/leonardo/mp-smt-strategy.pdf
10. Dolzmann, A., Sturm, T.: Simplification of quantifier-free formulas over ordered fields. Journal of Symbolic Computation 24, 209–231 (1995)
11. Dolzmann, A., Sturm, T.: REDLOG: Computer algebra meets computer logic. SIGSAM Bulletin 31(2), 2–9 (1997)
12. 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)
13. Heintz, J., Roy, M.F., Solernó, P.: On the theoretical and practical complexity of the existential theory of the reals. The Computer Journal 36(5), 427–431 (1993)
14. Loup, U., Ábrahám, E.: GiNaCRA: A C++ Library for Real Algebraic Computations. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NFM 2011. LNCS, vol. 6617, pp. 512–517. Springer, Heidelberg (2011)
15. Passmore, G.O., Jackson, P.B.: Combined Decision Techniques for the Existential Theory of the Reals. In: Carette, J., Dixon, L., Coen, C.S., Watt, S.M. (eds.) MKM 2009, Held as Part of CICM 2009. LNCS, vol. 5625, pp. 122–137. Springer, Heidelberg (2009)
16. Tarski, A.: A Decision Method for Elementary Algebra and Geometry. University of California Press (1948)

17. Weispfenning, V.: A new approach to quantifier elimination for real algebra. In: Quantifier Elimination and Cylindrical Algebraic Decomposition. Texts and Monographs in Symbolic Computation, pp. 376–392. Springer (1998)
18. Weispfenning, V.: The complexity of linear problems in fields. Journal of Symbolic Computation 5(1-2), 3–27 (1988)
19. Weispfenning, V.: Quantifier elimination for real algebra – The quadratic case and beyond. Applicable Algebra in Engineering, Communication and Computing 8(2), 85–101 (1997)
20. `http://research.microsoft.com/en-us/um/redmond/projects/z3/`