On Structural Information and the Experimental Evaluation of SMT Tools

Najet Boughanmi, Silvio Ranise, Christophe Ringeissen

On Structural Information and the Experimental Evaluation of SMT Tools

Najet Boughanmi, Silvio Ranise, and Christophe Ringeissen

LORIA & INRIA-Lorraine

Abstract. We are interested in studying the impact of various pre-processing transformations of ground formulae on the performances of Satisfiability Modulo Theory tools (based on the integration between an enumerator of truth assignments and a satisfiability procedure for conjunction of literals in some theory) which are supposed to discharge them. We briefly discuss our preliminary experiences.

Context. Reasoning tools (such as MathSAT [4], DLSAT [5], DPLL(T) [10], TSAT [1], ICS [9], CVC-Lite [3], **haRVey** [6], *Simplify* [8], or Zapato [2]) based on the Satisfiability Modulo Theory (SMT) approach rely on the integration between an enumerator of truth assignments and a satisfiability procedure for conjunction of literals in some theory (such as the theory of equality, various fragments of Linear Arithmetic, and their combinations). Recently, such tools have received a lot of attention for their capability of efficiently discharging large proof obligations obtained in the application of formal methods (such as verification of hardware systems, software model checking, and so on).

Since many SMT tools are available and have different characteristics that may improve performances on certain types of proof obligations, it is very useful to have experimental analyses which allow one to choose the best tool for his own problems. Also, the availability of benchmarks may facilitate the evaluation and the comparison of SMT tools, and advance the state of the art in the field, in the same way as, for instance, the TPTP library[1] has done for theorem proving, or the SATLIB library[2] has done for propositional satisfiability. This is the main aim of the SMT-LIB initiative which also proposes a common format to facilitate the exchange of benchmarks.[3] While the SMT-LIB initiative was launched, the paper [7] presented an extensive experimental evaluation of six SMT tools (among which *Simplify* and ICS) on four suites of benchmarks.

Motivation. In the last section of [7], it is observed that "*When developing the translators for these experiments, we noticed that the performance of most solvers heavily depends on the way problems are encoded.*" Stimulated by this observation, we have started assessing the impact of different kind of translations on the performances of SMT tools on the set of benchmarks used in [7]. The goal

[1] `http://www.tptp.org/`

[2] `http://www.satlib.org/`

[3] See `http://combination.cs.uiowa.edu/smtlib` for details on SMT and its common format.

of this on-going work is to understand which kind of translations can be used to make problems easier for SMT tools as done for resolution/paramodulation based theorem provers e.g., in [11]. We hope that this will benefit also the construction of translators from the SMT-LIB common format to the existing SMT tools. In particular, we intend to use these techniques in combination with **haRVey**,[4] the SMT tool which we are currently developing.

Benchmarks. To begin our study, we have considered the SAL benchmark suite of [7]. The proof obligations in this suite are derived from bounded model checking of timed automata and linear hybrid systems, and from test-case generation for embedded controllers. The formulae are represented in non-clausal form and they must be checked for satisfiability modulo Linear Arithmetic (LA). Since **haRVey**'s input syntax is identical (up to renaming of keywords) to that of *Simplify* (at least on the sub-set used in the suite), we have used this instance for our experiments. The formulae in the suite in *Simplify*'s syntax have the following form:

$$(\bigwedge_{j=1}^{n}\bigwedge_{i=1}^{m_j} b_i^j \Leftrightarrow \phi_i^j) \wedge (\bigwedge_{k=0}^{p} \psi_k \wedge \bigwedge_{l=1}^{m_1} b_l^1) \tag{1}$$

where the b_i^j are propositional letters, the ϕ_i^j are arbitrary Boolean combinations of propositional letters b_i^0 $(i = 1, ..., n_0)$, ground atoms of LA, and propositional letters b_i^k for $k < j$, and the ψ_k are arbitrary Boolean combinations of ground atoms of LA and b_i^j $(i = 0, ..., m_j)$. We notice that each sub-formulae in (1) is associated with a propositional letter b_i^j for $i > 1$. The formulae have been obtained by translation from ICS. In the ICS version of the suite, the propositional letters b_i^j $(j > 0)$ are introduced by the construct `prop`, which is similar to the 'let' of many functional programming languages.

The problem. By inspection of (1), it is clear that, for $j > 0$, the b_i^j's can be considered as "pointers" to ease references to complex formulae. We conjecture that ICS is capable to exploit this kind of structural information so to speed up its performances. Unfortunately, as already observed in [7], other SMT tools may not be able to exploit such a structural information after the translation. For example, *Simplify* does not have a let-like construct which may play the same role of ICS's `prop` so that translating "`prop` b_i^j ϕ_i^j" by "$b_i^j \Leftrightarrow \phi_i^j$" seems to be the best possible choice.[5] As a consequence, formulae in the form (1) are surprisingly difficult for *Simplify*, which is reported [7] to time out (with a time limit of 1 hour) for 99 benchmarks out of 217 in the suite. Even worse, **haRVey**

[4] `http://www.loria.fr/equipes/cassis/softwares/haRVey`

[5] The SMT-LIB common format provides a suitable let construct so that translations to the input formats of various SMT tools can exploit this structural information. Indeed, such translations must be designed to exploit the available structure and this work may help in this task.

runs out of memory for almost all but the smallest problems in the suite when using BDDs to enumerate truth assignments and it performs poorly compared to ICS when using a SAT solver.

Preliminary experiments. To enable *Simplify* and **haRVey** to significantly ameliorate their performances on the SAL benchmark suite, we have decided to implement the obvious preliminary transformations (i.e. $(b \Leftrightarrow \Phi) \wedge \Psi[b]$ rewrites to $\Psi[b/\Phi]$ when the propositional letter b does not occur in the formula Φ) that allow us to eliminate the b_i^j's in (1) for $j > 0$ and transformed all the problems in the suite. So far, we have run both *Simplify* and **haRVey** on around 25% of the problems in the suite and we have obtained encouraging results:

- *Simplify* is one order of magnitude faster on the transformed formulae than on the original ones in [7]. It is also capable of solving in a time comparable to that of ICS a problem on which it timed-out before.
- The performances of **haRVey** are also greatly ameliorated: using only the BDD package, it is already capable of solving most of the problems considered so far in a time comparable to that of ICS. We expect that for larger problems, the use of the available SAT solver will be necessary but we think that the timings will be comparable to that of ICS.

We plan to perform extensive experimental results and to study the impact of more simplification rules on the performances of SMT tools.

References

1. A. Armando, C. Castellini, E. Giunchiglia, and M. Maratea. A SAT-based Decision Procedure for the Boolean Combination of Difference Constraints. In *Proc. SAT'04*, 2004.
2. T. Ball, B. Cook, S.K. Lahiri, and L. Zhang. Zapato: Automatic Theorem Proving for Predicate Abstraction Refinement. In *CAV*, volume 3114 of *LNCS*, 2004.
3. C.L. Barrett and S. Berezin. CVC Lite: A New Implementation of the Cooperating Validity Checker. In *Proc. CAV'04*, volume 3114 of *LNCS*. Springer, 2004.
4. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P.van Rossum, S. Schulz, and R. Sebastiani. An incremental and Layered Procedure for the Satisfiability of Linear Arithmetic Logic. In *Proc. TACAS'05 (to appear)*, 2005.
5. S. Cotton, E. Asarin, O. Maler, and P. Niebert. Some Progress in Satisfiability Checking for Difference Logic. In *Proc. FORMATS-FTRTFT 2004*, 2004.
6. D. Deharbe and S. Ranise. Light-Weight Theorem Proving for Debugging and Verifying Units of Code. In *Proc. SEFM'03)*. IEEE Computer Society Press, 2003.
7. L. de Moura and H. Ruess. An Experimental Evaluation of Ground Decision Procedures. In: Proceedings of Computer-Aided Verification, CAV'04.
8. D. L. Detlefs, G. Nelson, and J. B. Saxe. Simplify: a Theorem Prover for Program Checking. Technical Report 148, HP Labs, 2003.
9. J.-C. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS: Integrated Canonizer and Solver. In *Proc. CAV'01*, volume 2102 of *LNCS*, pages 246–249, 2001.
10. H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In *Proc. CAV'04*, volume 3114 of *LNCS*, pages 175–188. Springer, 2004.
11. C. Weidenbach and A. Nonnengart. Computing Small Clause Normal Forms. In Robinson, A. and Voronkov, A., editors, *Hand. of Automated Reasoning*, volume 1. Elsevier Science, 2001.