Towards automating proofs for model-based software engineering

D. Déharbe[1] , P. Fontaine[2] , A. Martins Moreira[1] , S. Merz[2] , A. Santana de Oliveira[3]

[1]Universidade Federal do Rio Grande do Norte
Natal, RN, Brazil

[2]LORIA–INRIA
Nancy, France

[3]Universidade Federal Regional do Semi-Árido
Mossoró, RN, Brazil

{anamaria,anderson,david}@dimap.ufrn.br,

{Pascal.Fontaine,Stephan.Merz}@loria.fr

Abstract. *Model-based approaches are commonly used to engineering software for safety-critical applications. Several artifacts, such as abstract specifications, formal refinements, and assertions at the implementation level require computational support for formal reasoning activities. Delivering formal verification tools that address such reasoning activities in an automated, trustable and flexible fashion remains a scientific challenge. Researchers from LORIA and UFRN have been collaborating to address this challenge and are now developing an automated theorem prover targeted at trustable software verification efforts.*

1. Introduction

The software industry needs effective technologies to assist the design and development of software for safety-critical applications. This domain of application has its own safety-related standards, where strict requirements impose mathematical rigor and traceability to the development activities.

Model-driven design approaches such as the B method [1] have been used by the industry for more than a decade to address these needs. Such methods require that the designers produce a formal model of the requirements. The satisfiability and coherence of this model is then verified using theorem provers to establish the validity of verification conditions. This first model may then be used as a reference to establish the correctness of more concrete design artifacts, known as refinements. Theorem provers again play an essential role in this step, they ensure the overall soundness of the development with respect to the initial requirements. Support for these methods comes in the form of fully or partially automated provers, with varying degrees of efficiency, depending on the expressiveness of the specification language, as well as the maturity of the employed theorem proving technologies.

For the last few years, the collaboration between UFRN and LORIA has been focusing on the design of a state-of-the-art theorem prover that caters to the needs of such model-based software engineering methods (Section 2 of the paper gives a brief account

of the history the collaborations between the institutions). Details on the theorem prover currently under development are then given in Section 3. Finally, Section 4 provides a perspective on the scientific challenges that the authors aim to tackle in the future.

2. A short account of previous collaborations

Collaboration between UFRN and LORIA (encompassing the Cassis, Mosel, and Protheo/Pareo research teams) in the area of formal approaches to software development started formally in 2001 with the FERUS project, coordinated by Anamaria Martins Moreira on the Brazilian side and by Hélène Kirchner on the French side. This project was proposed in response to a joint call of CNPq and INRIA and addressed the reuse of software specification components in the algebraic realm. In addition to short scientific missions of the different partners, during the FERUS project UFRN lecturer Anamaria Martins Moreira realized a one-year long sabbatical at LORIA and her then master student Anderson Santana de Oliveira also realized a four-month stay at LORIA. Anderson Santana was later the recipient of a grant from CAPES to realize a full doctoral study at LORIA, starting in 2004. In 2008, Anderson Santana defended his thesis and after a post-doc stay at UFRN was hired as a lecturer by UFERSA.

In 2002, UFRN lecturer David Déharbe visited LORIA for a sabbatical stay with Michael Rusinowitch. This stay initiated a fruitful collaboration in the area of automatic theorem proving, involving Silvio Ranise, Christophe Ringeissen, and recently in particular Pascal Fontaine. This collaboration was formalized in the second joint UFRN/LORIA project under the sponsorship of CNPq and INRIA: Da Capo (2005-2008). Since then several former UFRN students initiated their doctoral studies at LORIA: Judson Santos Santiago (Cassis group, concluded[1]), Cláudia Fernanda Oliveira Kiermes Tavares (Pareo group, ongoing), Diego Caminho Barbosa de Oliveira (Mosel group, ongoing).

3. The veriT solver

The collaboration between Loria and UFRN in the area of automatic theorem proving has led to the active development of a common tool through which our research results are validated and made available to the larger community. Initially known as haRVey [6] whose first version was made available in 2002, it was essentially a research platform for combining reasoning tools. Under the DaCapo project, we began in 2007 to re-factor the code of haRVey and greatly enhance its capabilities. The new tool is called veriT solver; the release of a usable and stable version is imminent.

The veriT solver is best classified as a Satisfiability Modulo Theories (SMT) solver. It takes a predicate logic formula as input and checks if the formula is satisfiable or not, that is, if there is an interpretation that makes the formula true. The validity problem is trivially translated to the satisfiability problem by taking the negation of the formula; a formula is valid (always true) if and only if its negation is unsatisfiable. Most formal methods and verification techniques heavily rely on checking the satisfiability or the validity of formulas. In addition to giving a verdict on the satisfiability of its input, veriT is able to produce a proof of its result; such proof may be then checked or reused by external components. This feature is important as it makes it possible to certify the results produced by veriT.

[1] Judson Santos Santiago has also been hired as a lecturer by UFERSA.

The input language of veriT is a first-order language with uninterpreted and interpreted symbols based on the SMT-LIB format [13]. Uninterpreted symbols are particularly useful to model arrays, or functions that are partially or totally unknown, whereas interpreted symbols are essential to model the usual data structures used in computer systems such as integers, rationals, real numbers or lists. As a distinguishing feature, veriT includes a complete first-order generic prover (at the moment, the E-prover [14]; inclusion of SPASS [16] is planned). This feature makes it possible for the user to define his own data structures by writing axioms in logic that characterize the data structures and the operators working on them

Figure 1 depicts the architecture of veriT. First veriT reads the input formula, written in the SMT-LIB format [13]. Currently veriT implements two syntactical extensions to this format: lambda expressions and macros. The parser rewrites the instances of these constructs applying beta reduction and macro expansion. The result of this preprocessing may be output to a file in the SMT-LIB format without these extensions. Second, veriT computes the conjunctive normal form of the formula and stores the corresponding clauses in SAT-solver MINISAT [7], a propositional satisfiability solver, that implements a modern and efficient version of the DPLL algorithm [3]. The SAT-solver attempts to find a propositional model of the formula, i.e. a satisfiable set of literals. If no such model is found, then the formula is unsatisfiable. Otherwise, it is propositionally satisfiable. The set of literals is then incrementally input to the theory reasoning engine. The theory reasoning engine may conclude that the set of literals is satisfiable in the combination of first-order theories; in that case the original formula is satisfiable and the execution stops with this result. Another possible outcome is that the set of literals is unsatisfiable in the theory; in that case, a clause is added to the propositional SAT-solver to discard this set of literals from the set of possible models. Finally, the third possible outcome is that the theory reasoning module cannot reach a conclusion; in that case, it may provide lemmas that are translated to additional clauses that further constrains the set of possible models. In the last two cases, the SAT-solver is applied to the updated set of clauses and the process repeats until no more propositional models can be built (the formula is unsatisfiable), or no more lemmas can be generated (the solver is not able to decide the satisfiability of the input formula). For some logics, veriT is also able to produce a proof of the result, which consists of a sequence of instances of basic deduction rules that can be then checked by a third-party.

The theory reasoning module is itself composed of a decision procedure for the theory of equality with uninterpreted functions, based on congruence closure [12], a decision procedure for the so-called "difference logic", a fragment of real and integer arithmetics [4], quantifier instantiation heuristics, and an automated deduction engine based on resolution and superposition [15]. Such module [5] implements a variant of the Nelson and Oppen combination technique [11], based on the propagation of variable equalities. Moreover, to avoid overflow and underflow errors in the manipulation of arithmetic values, veriT uses the GNU MP [9] libraries.

The language of veriT totally covers several sections of the competition of SMT solvers (SMT-COMP [2]). Although the efficiency of veriT is not yet at the level of tools winning those sections of the competition, its performances are on a par with serious general-purpose SMT solvers. We evaluated veriT, CVC3 and Z3 (both using the lat-

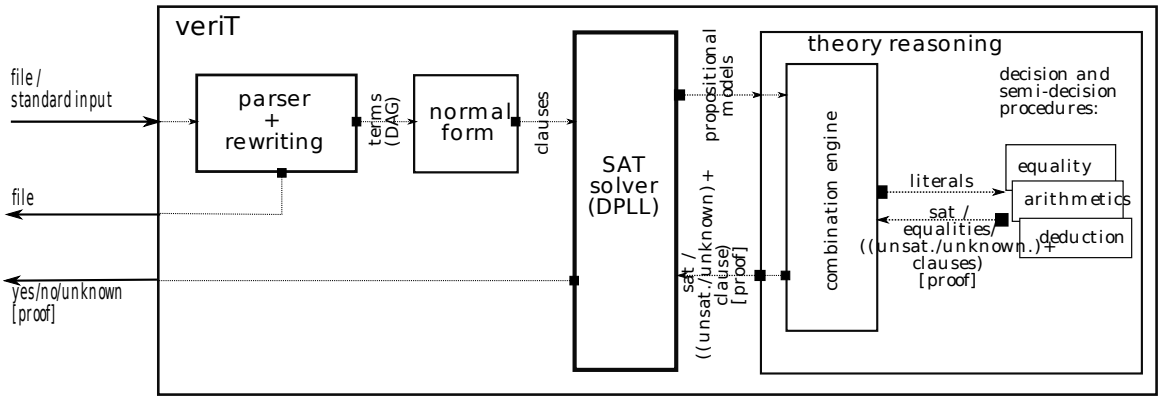

Figure 1. Software architecture of veriT

Solver	QF_UF (6656)	QF_UFIDL (432)	QF_IDL (1673)	QF_RDL (204)	all (8965)
veriT	6323	332	918	100	7673
CVC3	6378	278	802	45	7503
Z3	6608	419	1511	158	8696

Figure 2. Experimental comparison of veriT **with CVC3 and Z3**

est available version in February 2009) against the SMT-LIB benchmarks for QF_IDL, QF_RDL, QF_UF and QF_UFIDL (June 2008 version) using an Intel® Pentium® 4 CPU at 3.00 GHz with 1 GB of RAM and a timeout of 120 seconds. The Figure 2 gives, for each solver, the number of completed benchmarks.

The veriT solver has been successfully used for several verification tasks on lock-free algorithms, and to check refinements in model-based software engineering approach such as B and Circus. The tool is publicly available and distributed under the BSD open-source license. It may be downloaded at `http://www.verit-solver.org`.

Planned enhancements include better quantifier handling, hierarchic theories, new decision procedures for arithmetic fragments, and better integration of the tool within proof assistants.

4. Perspectives and conclusion

Our motivation for the development of the veriT solver is that the formal development of software should be supported by automated reasoning modules. As the solver gains maturity, another important task remains: the automation provided by veriT should be made available inside popular tool platforms supporting formal methods, such as the B method, TLA+, as well as the Isabelle and Coq proof assistants. We have several preliminary works along that line. The veriT solver has already been integrated as a plug-in within the Isabelle proof-assistant [8], some investigation to integrate veriT inside the Coq proof-assistant has been done, and veriT has been used successfully to discharge simple but very tedious proof obligations from B models [10].

In this context it is crucial to provide the users with syntactical criteria to identify the fragment of the language of their favorite formal method for which automatic proof

support is possible. This is not trivial, since syntactical transformations may often be used to convert proof obligations that occur naturally in popular formal methods towards the input language of automated solvers. For example, proof obligations for the B method are typically formulas using set theoretic concepts. A straight translation—using axioms for set theory—into the language of first-order or SMT provers leads to poor results. However veriT is able to handle some set formulas very efficiently, by understanding sets as their characteristic predicates and rewriting the usual operators on sets as boolean operators accordingly. Not every formula can be handled in this way, and notably, quantifiers over sets are not allowed in the original formula. Even in those cases, it is sometimes possible to use clever instantiations to obtain formulas that fall into the fragment handled by veriT, and it would be interesting to automatically recognize such cases. The same recognizers could also be applied to sub-goals in interactive proof assistants, thus simplifying the required expertise. As a last resort, a user aware of this automation capability could guide the manual proof to minimize the number of interactions necessary to prove a given verification condition.

The development of the veriT solver and its use as a component for tool sets supporting formal methods are at the core of the long term fruitful cooperation between the UFRN and LORIA teams.

References

[1] J.-R. Abrial. *The B-Book. Assigning Programs to Meanings*. Cambridge University Press, 1996.

[2] C. Barrett, L. de Moura, and A. Stump. SMT-COMP: Satisfiability Modulo Theories Competition. In *Computer Aided Verification (CAV)*, pages 20–23, 2005.

[3] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[4] D. C. B. de Oliveira. Deciding difference logic in a Nelson-Oppen combination framework. Master's thesis, PPgSC/UFRN, 2007.

[5] D. C. B. de Oliveira, D. Dharbe, and P. Fontaine. Combining decision procedures by (model-)equality propagation. In *Brazilian Symposium on Formal Methods (SBMF2008)*, pages 51–66. Editora Grfica da UFBA, EDUFBA, 2008.

[6] D. Déharbe and S. Ranise. Bdd-driven first-order satisfiability procedures (extended version). Research report 4630, LORIA, 2002.

[7] N. Eén and N. Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing*, volume 2919 of *LNCS*, pages 333–336. Springer, 2003.

[8] P. Fontaine, J.-Y. Marion, S. Merz, L. P. Nieto, and A. Tiu. Expressiveness + automation + soundness: Towards combining SMT solvers and interactive proof assistants. volume 3920 of *Lecture Notes in Computer Science*, pages 167–181. Springer-Verlag, 2006.

[9] T. Granlund. GNU MP: The GNU multiple precision arithmetic library, 4.1.4 ed, 2004. http://gmplib.org/.

[10] E. S. Marinho, V. de Medeiros Jr, D. Déharbe, B. Gomes, and C. Tavares. A ferramenta batcave para a verificação de especificações formais na notação b. In *XXII Simpósio*

Brasileiro de Engenharia de Software. XV Sessão de Ferramentas do SBES., pages 7–12.

[11] G. Nelson and D. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.

[12] G. Nelson and D. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364, 1980.

[13] S. Ranise and C. Tinelli. The SMT-LIB standard : Version 1.2, Aug. 2006.

[14] S. Schulz. System Description: E 0.81. In *Proc. of the 2nd IJCAR, Cork, Ireland*, volume 3097 of *LNAI*, pages 223–228. Springer, 2004.

[15] S. Schulz. System Description: E 0.81. In *Proc. of the 2nd IJCAR, Cork, Ireland*, volume 3097 of *LNAI*, pages 223–228. Springer, 2004.

[16] C. Weidenbach, R. Schmidt, T. Hillenbrand, R. Rusev, and D. Topic. System description: Spass version 3.0. In *Conference on Automated Deduction (CADE)*, volume 4603 of *LNCS*, pages 514–520. Springer, 2007.