veriT: an open, trustable and efficient SMT-solver

Thomas Bouton[2], Diego Caminha B. de Oliveira[2],
David Déharbe[1], and Pascal Fontaine[2]

[1] Universidade Federal do Rio Grande do Norte, Natal, RN, Brazil
david@dimap.ufrn.br
[2] LORIA–INRIA, Nancy, France
{Thomas.Bouton,Diego.Caminha,Pascal.Fontaine}@loria.fr

Abstract. This article describes the first public version of the satisfiability modulo theory (SMT) solver veriT. It is open-source, proof-producing, and complete for quantifier-free formulas with uninterpreted functions and difference logic on real numbers and integers.

1 Introduction

We present the satisfiability modulo theory (SMT) solver veriT, a joint work of University of Nancy, INRIA (Nancy, France) and Federal University of Rio Grande do Norte (Natal, Brazil). veriT provides an open, trustable and reasonably efficient decision procedure for the logic of unquantified formulas over uninterpreted symbols, difference logic over integer and real numbers, and the combination thereof. This corresponds to the logics identified as QF_IDL, QF_RDL, QF_UF and QF_UFIDL in the SMT-LIB benchmarks [15, 3]. veriT also includes quantifier reasoning capabilities through the integration of a first-order prover and quantifier instantiation heuristics. Finally, veriT has proof-production capabilities; it outputs proofs that may be used or checked by external tools.

veriT is incremental, i.e. after each satisfiability check, new formulas can be added conjunctively to the already checked set of formulas. The input format is the SMT-LIB language [15], but veriT can also be used as a library with an API following the guidelines of [12]. The tool is open-source and distributed under the BSD licence at `http://www.verit-solver.org`. Internally, the solver is organized to be easily extended by plugging new decision procedures in a Nelson-Oppen like combination schema. Although not (yet) as fast as the solvers performing best in the SMT competition [3], veriT has a decent efficiency. We thus claim that it can already be useful in verification platforms where an open-source license, extensibility, and proof certification are important.

Selected features of the veriT solver and an experimental evaluation of its efficiency are presented in Section 2 and 3, respectively. Future developments are described in Section 4.

2 System description

The reasoning core of veriT uses a SAT solver [9] to produce models of the Boolean abstraction of the input formula. Such propositional assignments are

given to a so-called *theory reasoner*, responsible for verifying if they are models in the background theory. This theory reasoner is a fully incremental combination of decision procedures *à la* Nelson and Oppen, where non-convexity of theories is handled using the model-equality propagation technique [7] which integrates model-based guessing [5] in a classical Nelson-Oppen equality exchange. Equality propagation is controlled by the congruence closure algorithm.

The remainder of this section describes some special features of veriT: integration of a third-party first-order prover, extension of the input language with macro definitions, and production of proofs certifying the produced results.

2.1 Integrating a first-order prover

As a particular feature inherited from its predecessor haRVey [8] and, to complement very simple instantiation heuristics, the veriT solver includes a first-order logic (FOL) superposition prover. However, veriT greatly improves the integration of the FOL prover with the other decision procedures, notably with congruence closure. Indeed, the first-order prover is seen within the combination *à la* Nelson-Oppen as a "decision procedure" that takes an arbitrary FOL theory as a parameter. However, due to the cost of running the FOL prover and to its non-incremental nature (when used as a black box), this procedure is called in last resort. A FOL theory is computed from the quantified sub-formulas in the assignment, abstracting ground sub-terms in order to minimize the number of relevant symbols in the theory. In addition, information from congruence closure is used to abstract all subterms in the assignment that do not contain such relevant symbols.

The prover may deduce that the given set of formulas is unsatisfiable. In that case, the deduction tree is parsed to obtain the relevant unsatisfiable subset of the input. A conflict clause is then built using this set and, again, information from the congruence closure data structures. Since the prover is given an upper limit of resources, it always terminates. If the prover terminates without proving the unsatisfaibility of the given set of formulas, ground equalities and deduced ground clauses are identified and propagated back to veriT.

In many cases where the superposition calculus is a decision procedure [2] for the theory represented by the quantified formulas, our technique simulates a Nelson-Oppen combination with on-the-fly purification. It has been shown that first-order generic provers may perform quite well even compared to dedicated decision procedures (see for instance [1]). Currently, the E-prover [16] is used as the first-order prover, and we plan to include Spass [18], which provides better sort handling. Fine-tuning the interplay between the instantiation heuristics and the e-prover is essential for efficiency and remains to be done.

2.2 Macros

The input format for veriT is the SMT-LIB language extended with macro definitions. This syntactic sugar is particularly useful for instance to write formulas containing simple sets constructions (see Figure 1). After β-reduction, and after

rewriting equalities between predicates and functions (for instance, if p and q are unary predicates, $p = q$ is rewritten as $\forall x \,.\, p(x) \equiv q(x)$), the obtained formula is first-order. Such formulas may contain quantifiers but, if no function is used, they belong to the Bernays-Schönfinkel-Ramsey fragment (the Bernays-Schönfinkel class with equality) which is decidable. veriT can use the embedded FOL prover as a decision procedure for this fragment. In many more intricate cases [10], the resulting formula still belongs to a decidable fragment, but the decision procedure may become very expensive. To be practical, such cases require heuristics that have not yet been implemented in veriT.

This macro feature is indeed used in tools (e.g. CRefine [14]) that generate verification conditions for formal developments in set-based modelling languages, such as Circus [4], and that integrate veriT as a verification engine to discharge these proof obligations.

```
(benchmark SET008_3p
  :logic UNKNOWN
  :extrasorts (ELMT)
  :extrapreds ((B ELMT) (C ELMT))
  :extramacros
   ((emptyset (lambda (?v ELMT) . false))
    (intersection (lambda (?p (ELMT boolean)) (?q (ELMT boolean)) .
                     (lambda (?x ELMT) . (and (?p ?x) (?q ?x)))))
    (difference (lambda (?p (ELMT boolean)) (?q (ELMT boolean)) .
                   (lambda (?x ELMT) . (and (?p ?x) (not (?q ?x)))))))
  :formula (not (= (intersection (difference B C) C) emptyset)))
```

Fig. 1. A simple example with the macro capability.

2.3 Proofs

Proof production has two goals. First, this feature increases the confidence in the tool, the proofs being checked by an independent module inside veriT. Second, skeptical proof assistants can use such traces to reconstruct proofs of formulas discharged by veriT (see [11]).

In Figure 2, we give an example of a very simple formula, and the proof output by veriT. Each line states a fact that can be assumed to hold. It is identified by a number, followed by a list starting with a label identifying the rule used to deduce the fact, followed by a clause, and optionally ended by numerical parameters. In our context, a clause is a disjunctive list of formulas (not literals), maybe containing a sole formula. The numerical parameters depend on the rule, and may be either identifiers of previous clauses (e.g. in the resolution rule), or other place information. As an example, the *and* rule (for instance the second line in Figure 2: `(and ((= a c)) 1 0)`) takes two numerical parameters. The first

```
(benchmark example
  :logic QF_UF
  :extrafuns ((a U) (b U) (c U) (f U U))
  :extrapreds ((p U))
  :formula (and (= a c) (= b c)
                (or (not (= (f a) (f b)))
                    (and (p a) (not (p b))))))

1:(input ((and (= a c) (= b c)
               (or (not (= (f a) (f b))) (and (p a) (not (p b)))))))
2:(and ((= a c)) 1 0)
3:(and ((= b c)) 1 1)
4:(and ((or (not (= (f a) (f b))) (and (p a) (not (p b))))) 1 2)
5:(and_pos ((not (and (p a) (not (p b)))) (p a)) 0)
6:(and_pos ((not (and (p a) (not (p b)))) (not (p b))) 1)
7:(or ((not (= (f a) (f b))) (and (p a) (not (p b)))) 4)
8:(eq_congruent ((not (= b a)) (= (f a) (f b))))
9:(eq_transitive ((not (= b c)) (not (= a c)) (= b a)))
10:(resolution ((= (f a) (f b)) (not (= b c)) (not (= a c))) 8 9)
11:(resolution ((= (f a) (f b))) 10 2 3)
12:(resolution ((and (p a) (not (p b)))) 7 11)
13:(resolution ((p a)) 5 12)
14:(resolution ((not (p b))) 6 12)
15:(eq_congruent_pred ((not (= b a)) (p b) (not (p a))))
16:(eq_transitive ((not (= b c)) (not (= a c)) (= b a)))
17:(resolution ((p b) (not (p a)) (not (= b c)) (not (= a c))) 15 16)
18:(resolution () 17 2 3 13 14)
```

Fig. 2. A simple example with its proof.

numerical parameter refers to the clause C in a previous numbered rule (i.e. `1` refers to the clause in the `input` rule, at line 1). This clause C is unit and is hence represented as a list of one formula (the whole input formula in our example), and this formula is a conjunction $a_0 \wedge \cdots \wedge a_n$. Obviously, each subformula $a_0, \ldots a_n$ is a consequence of C, and the second parameter just gives the identifier of the formula in the new clause, i.e. the second numerical parameter in rule at line 2 indicates the formula at position 0 in the input.

veriT already provides proof production for formulas with arbitrary Boolean structure and uninterpreted functions, and is being extended to linear arithmetics. The first line is the input. Every other fact is either a consequence of previous ones, or is a tautology. The input formula being unsatisfiable, the last deduced fact is the empty clause. In the example, lines 2 to 7 account for the conjunctive normal form transformation. Lines 8, 9, 15, and 16 are tautologies related to the theory of equality. The remaining facts are deduced by resolution from the other ones.

Since every proof-related information is handled through a unique module inside veriT, any proof format for SMT (for instance [17, 6]) can be adopted as

soon as it becomes a standard. Although our previous experiments [11] showed that the proof size was not the bottleneck for the cooperation with skeptical proof assistants, the implementation of techniques to greatly reduce the size of our proof traces is planned.

3 Experimental evaluation

We evaluated veriT, CVC3 and Z3 (both using the latest 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(R) Pentium(R) 4 CPU at 3.00 GHz with 1 GiB of RAM and a timeout of 120 seconds. The following table presents, for each solver, the number of completed benchmarks.

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

This clearly shows that, although veriT is not yet as efficient as competition winning tools [3], its efficiency is decent. The proof production capability of veriT does not come at a cost on efficiency.

4 Future work

veriT is a new SMT-solver that provides an open framework to generate certifiable proofs without sacrificing too much efficiency. The future developments will notably include features related to efficiency and expressiveness. Considering efficiency aspects, the tool does not yet implement theory propagation [13], a technique that is known to greatly improve the efficiency of SMT solvers. Concerning expressiveness, the linear arithmetic decision procedure currently only handles difference logic; we are now developing a reasoning engine for linear arithmetic based on the Simplex method, which will extend completeness to full linear arithmetic. Quantifier reasoning will be improved by including new instantiation heuristics, as well as adding support for patterns guiding quantifier instantiations. We also plan to integrate the proof production capabilities of the embedded FOL prover with that of veriT.

Finally, we are working on the application of veriT to formal development efforts, mainly of concurrent systems. In that context, the ability to handle sets is very helpful; a major objective is to improve the support for such constructions.

Acknowledgments: We would like to thank Stephan Merz for his comments and guidance. The ancestor of the veriT solver, haRVey, was initiated by the third author and Silvio Ranise, to whom we are indebted of several ideas used in veriT. We are also grateful to the anonymous reviewers for their remarks.

References

1. A. Armando, M. P. Bonacina, S. Ranise, and S. Schulz. New results on rewrite-based satisfiability procedures. *ACM Trans. Comput. Log.*, 10(1), 2009.
2. A. Armando, S. Ranise, and M. Rusinowitch. A rewriting approach to satisfiability procedures. *Information and Computation*, 183(2):140–164, 2003.
3. C. Barrett, L. de Moura, and A. Stump. SMT-COMP: Satisfiability Modulo Theories Competition. In *Computer Aided Verification (CAV)*, pages 20–23, 2005.
4. A. L. C. Cavalcanti, A. C. A. Sampaio, and J. C. P. Woodcock. A Refinement Strategy for *Circus*. *Formal Aspects of Computing*, 15(2 - 3):146 — 181, 2003.
5. L. de Moura and N. Bjørner. Model-based theory combination. *Electronic Notes in Theoretical Computer Science*, 198(2):37–49, 2008.
6. L. M. de Moura and N. Björner. Proofs and refutations, and Z3. In *LPAR Workshops*, volume 418 of *CEUR Workshop Proceedings*, 2008.
7. D. Déharbe, D. de Oliveira, and P. Fontaine. Combining decision procedures by (model-)equality propagation. In *Brazil. Symp. Formal Methods*, pages 51–66, 2008.
8. D. Déharbe and S. Ranise. Bdd-driven first-order satisfiability procedures (extended version). Research report 4630, LORIA, 2002.
9. 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.
10. P. Fontaine. Combinations of theories and the Bernays-Schönfinkel-Ramsey class. In *4th Int'l Verification Workshop (VERIFY)*, 2007.
11. P. Fontaine, J.-Y. Marion, S. Merz, L. P. Nieto, and A. Tiu. Expressiveness + automation + soundness: Towards combining SMT solvers and interactive proof assistants. In *Tools and Algorithms for Construction and Analysis of Systems*, volume 3920 of *LNCS*, pages 167–181. Springer, 2006.
12. J. Grundy, T. Melham, S. Krstić, and S. McLaughlin. Tool building requirements for an API to first-order solvers. *Electronic Notes in Theoretical Computer Science*, 144:15–26, 2005.
13. R. Nieuwenhuis and A. Oliveras. DPLL(T) with Exhaustive Theory Propagation and its Application to Difference Logic. In *Computer Aided Verification (CAV)*, volume 3576 of *LNCS*, pages 321–334. Springer, 2005.
14. M. Oliveira, C. Gurgel, and A. de Castro. Crefine: Support for the Circus refinement calculus. In *IEEE Intl. Conf. Software Engineering and Formal Methods*, pages 281–290. IEEE Comp. Soc. Press, 2008.
15. S. Ranise and C. Tinelli. The SMT-LIB standard : Version 1.2, Aug. 2006.
16. S. Schulz. System Description: E 0.81. In *Proc. of the 2nd IJCAR, Cork, Ireland*, volume 3097 of *LNAI*, pages 223–228. Springer, 2004.
17. A. Stump. Proof Checking Technology for Satisfiability Modulo Theories. In *Logical Frameworks and Meta-Languages: Theory and Practice*, 2008.
18. 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.