Better SMT Proofs for Easier Reconstruction

Haniel Barbosa[1], Jasmin Christian Blanchette[2,3,4], Mathias Fleury[4,5], Pascal Fontaine[3], and Hans-Jörg Schurr[3]

[1] University of Iowa, 52240 Iowa City, USA
haniel-barbosa@uiowa.edu
[2] Vrije Universiteit Amsterdam, 1081 HV Amsterdam, the Netherlands
j.c.blanchette@vu.nl
[3] Université de Lorraine, CNRS, Inria, LORIA, 54000 Nancy, France
{jasmin.blanchette,pascal.fontaine,hans-jorg.schurr}@loria.fr
[4] Max-Planck-Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany
{jasmin.blanchette,mathias.fleury}@mpi-inf.mpg.de
[5] Saarbrücken Graduate School of Computer Science, Saarland Informatics Campus, Saarbrücken, Germany
s8mafleu@stud.uni-saarland.de

Proof assistants are used in verification, formal mathematics, and other areas to provide trustworthy, machine-checkable formal proofs of theorems. Proof automation reduces the burden of proof on users, thereby allowing them to focus on the core of their arguments. A successful approach to automation is to invoke an external automatic theorem prover, such as a satisfiability-modulo-theories (SMT) solver [5], reconstructing any generated proofs using the proof assistant's inference kernel. The success rate of reconstruction, and hence the usefulness of this approach, depends on the quality of the generated proofs.

We report on the experience gained by working on reconstruction of proofs generated by an SMT solver while also improving the solver's output. By doing so, we were able to understand some practical constraints of reconstruction systems and find areas that require attention in the documentation of the proof output. We also discovered bugs in the proof generation code.

Proof generation from SMT solvers has attracted attention in the past [3]. The SMT solvers CVC4 [2] and Z3 [9] produce proofs, but CVC4's output format does not record quantifier reasoning, whereas Z3 does not always produce fine-grained steps, notably for skolemization. The SMT solver we work with, veriT [8], was recently extended with a more fine-grained proof-producing module [1] that records skolemization and other preprocessing steps in a detailed fashion. Proofs produced by veriT [6] take the form of a list of steps with optional annotations for term sharing. The syntax is based on SMT-LIB [4].

Many proof assistants reconstruct proofs generated by automatic theorem provers. Examples include the SMTCoq plugin [11], which reconstructs proofs from CVC4 and veriT inside Coq, and Isabelle's *smt* tactic [7], which reconstructs Z3 proofs. We extended this tactic to support veriT proofs as well. The *smt* tactic first translates the current higher-order proof goal to a first-order SMT problem. Then the external SMT solver is invoked. If the solver reports "unsatisfiable," the tactic will attempt to replay the generated proof in Isabelle.

Our experience emphasizes the importance of complete documentation. When veriT is called with the option `--proof-format-and-exit`, it generates a list of proof rules and a description of their semantics. Furthermore, earlier publications [1, 6, 10] provide background documentation on the proof calculus. Nevertheless, this documentation was lacking, especially concerning implicit steps performed by veriT. To replay the proof, the implicit steps must also be performed on Isabelle's side. Such implicit transformations appear in two places. First, veriT ignores the orientation of equalities in the input. The simple solution was to print the

input assertions after this normalization to allow Isabelle to link atoms with their normalized form. Second, veriT performs some simplifications immediately before printing a proof step. This includes eliminating repeated literals and double negation from clauses. Now that this behaviour is precisely documented, Isabelle can reconstruct these implicit steps most of the time at the cost of some automatic search. We plan to make this implicit normalization optional in future versions of veriT.

The size of the generated proofs is a practical constraint we initially overlooked. During skolemization, veriT introduces Hilbert choice terms in place of skolemized variables. Thus, $\exists x.\, p(x)$ is skolemized to $p(\varepsilon x.\, p(x))$. While this is elegant in theory, the choice term can be prohibitively large, especially when it is repeated in the output, leading to reconstruction failures. A solution is to use term sharing in the generated proofs: veriT adds a name annotation to every term and subsequently uses the name instead of the term. Sharing can have a dramatic impact on size: a 62 MB proof was compressed to 192 KB.

We encountered some difficulties with the replacement of constants by choice terms. Instead of choice terms, for efficiency reasons veriT uses fresh constants during solving. These constants must be replaced by the corresponding choice terms in the proof output. When choice terms were nested, the proof output did not fully replace constants inside choice terms. Since the choice functions are often quite long, such errors are hard to detect by a human reader, but instantly prohibit reconstruction.

We also observed a phenomenon we call *proof rot*. During the development of an automatic prover, we might inadvertently introduce small discrepancies with respect to the documented behaviour. For example, the instantiation rule used by veriT is slightly stronger than published [10] and documented. The documented form is $(\forall x.\, \varphi) \to \varphi[t/x]$, but in practice it sometimes takes the form $(\forall x.\, \varphi_1 \wedge \cdots \wedge \varphi_n) \to \varphi_i[t/x]$. Such changes accumulate and complicate reconstruction. During the implementation of the reconstruction procedure, each change had to be either documented or corrected. Now that it is in place, proof reconstruction serves as a safeguard to prevent such changes from being accidentally reintroduced.

Prospect Proofs are meant to be replayed. Implementing the reconstruction during the development of the proof-producing routines ensures that proofs can be replayed in practice. Given the flexibility of the SMT language, the proofs generated by SMT solvers need to account for a wide variety of theories and language features, which results in complex proofs with many possibilities for errors. These errors can be found by reconstructing proofs.

The quality of veriT's proofs remains unsatisfactory. Simple input problems often produce long, unwieldy proofs; yet, many proof steps are too coarse. A rule for linear arithmetic produces a certificate of the unsatisfiability of a set of inequalities using Farkas's lemma without providing explicit coefficients. This means that reconstruction must rely on a decision procedure to refind the proof, which sometimes fails or is slow. Furthermore, term sharing is required to keep proofs to a reasonable size, but also results in unreadable proofs. A good balance has yet to be found.

A call to an automated prover from inside a proof assistant can fail. Often this is because the prover could not find a proof, but sometimes the proof cannot be reconstructed. This should never happen.

Acknowledgments The work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 713999, Matryoshka). Previous experiments were carried out using the Grid'5000 testbed (https://www.grid5000.fr/), supported by a scientific interest group hosted by Inria and including CNRS, RENATER, and several universities as well as other organizations.

References

[1] Haniel Barbosa, Jasmin Christian Blanchette, Mathias Fleury, and Pascal Fontaine. Scalable fine-grained proofs for formula processing. *J. Automated Reasoning*. To appear.

[2] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *CAV 2011*, volume 6806 of *LNCS*, pages 171–177. Springer, 2011.

[3] Clark Barrett, Leonardo de Moura, and Pascal Fontaine. Proofs in satisfiability modulo theories. In David Delahaye and Bruno Woltzenlogel Paleo, editors, *APPA 2014*, volume 55 of *Mathematical Logic and Foundations*, pages 23–44. College Publications, 2014.

[4] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.

[5] Clark Barrett, Roberto Sebastiani, Sanjit Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 26, pages 825–885. IOS Press, 2009.

[6] Frédéric Besson, Pascal Fontaine, and Laurent Théry. A flexible proof format for SMT: a proposal. In Pascal Fontaine and Aaron Stump, editors, *PxTP 2011*, pages 15–26, 2011.

[7] Sascha Böhme. *Proving Theorems of Higher-Order Logic with SMT Solvers*. PhD thesis, Technische Universität München, 2012.

[8] Thomas Bouton, Diego Caminha B. de Oliveira, David Déharbe, and Pascal Fontaine. veriT: An open, trustable and efficient SMT-solver. In Renate A. Schmidt, editor, *CADE 2009*, volume 5663 of *LNCS*, pages 151–156. Springer, 2009.

[9] Leonardo de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS 2008*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[10] David Déharbe, Pascal Fontaine, and Bruno Woltzenlogel Paleo. Quantifier inference rules for SMT proofs. In Pascal Fontaine and Aaron Stump, editors, *PxTP 2011*, pages 33–39, 2011.

[11] Burak Ekici, Alain Mebsout, Cesare Tinelli, Chantal Keller, Guy Katz, Andrew Reynolds, and Clark Barrett. SMTCoq: A plug-in for integrating SMT solvers into Coq. In Rupak Majumdar and Viktor Kunčak, editors, *CAV 2017*, volume 10426 of *LNCS*, pages 126–133. Springer, 2017.