Proof Reconstruction for Z3 in Isabelle/HOL

Sascha Böhme
Technische Universität München
boehmes@in.tum.de

Abstract

Currently, only a few Satisfiability Modulo Theories (SMT) solvers are able to produce proof objects, although there is a strong incentive: Proof objects can be reconstructed in a different system to the check soundness of an SMT solver. We present proof reconstruction for the SMT solver Z3 in Isabelle/HOL and give experimental results of its application.

1 Introduction

Current SMT solvers are complex systems and constantly get more and more complex due to the addition of features, new theories and new decision procedures. Consequently, there is reason to doubt their soundness. A well-known approach in this case is to let the solver provide, along with its decision, a certificate (or proof term), which can be checked in a different system. Despite this incentive, this method is not widely adopted in the SMT community. Only recently, the SMT solver Z3 [4] has been enhanced to generate proof terms for the case when a set of assertions is found to be unsatisfiable [3].

So far, proofs emitted by Z3 have been used to find implementation bugs, by checking them in a previous version of Z3. We present a more rigorous approach of reconstructing proofs in a completely different system based on a secure proof kernel, namely the proof assistant Isabelle/HOL [10]. That means, we translate Z3's proofs into a different logic — from many-sorted first-order logic to higher-order logic (HOL) — and apply different (implementations of) decision procedures. We concentrate here on the theories of uninterpreted functions and arrays as well as linear integer and real arithmetic, even though Z3 also supports recursive datatypes and fixed-size bit-vectors.

The proof format of Z3 has the advantage of being fairly small (there are altogether 38 different proof rules), but this small set of proof rules comes at a price: Proof reconstruction gets involved, because many proof rules require several steps of reasoning in Isabelle/HOL. In most cases, these steps follow a fixed scheme, but for some high-level proof rules, an elaborate proof search is required. This makes proof reconstruction challenging.

The contribution of this work is two-fold: On the one hand, we present Z3's proof terms with a high level of detail, especially showing the majority of

its proof rules in a formal way; on the other hand, we explain details of how we implemented proof reconstruction for Z3. We take the first point as a prerequisite for the second one.

2 Isabelle/HOL

Isabelle is an interactive theorem prover based on the so-called LCF approach (going back to Edinburgh LCF [6]). This means that theorems are represented by an abstract datatype, and the only means to create them are the basic inference rules of the underlying core logic. Isabelle's core logic is a fragment of intuitionistic higher-order logic providing implication (written as $P_1 \Longrightarrow P_2$), equality, and universal quantification. On top of this generic layer, various object logics are implemented; the one we use here is HOL [7].

Isabelle accepts a proof only if it can be expressed in terms of the basic inference rules. As a consequence, all proof tools built for Isabelle can essentially be reduced to using only this set of basic rules.

There is a wide range of proof tools provided with Isabelle, notably the simplifier (for term rewriting), the classical reasoner, and Metis (a first-order resolution-based prover). Additionally, there are linear arithmetic decision procedures for refuting sets of inequalities and for quantifier elimination.

3 Proof Terms of Z3

3.1 Terms and Formulas

The language of Z3 is many-sorted first-order logic. A *term*, denoted by t or s with optional subscripts, is either a (sorted) variable (represented by the letter x, y, or z), an application of a (sorted) function symbol (denoted by f) to terms, or a universal or existential quantification (where we use Q to denote either $\forall$ or $\exists$). As usual, we consider constants and numbers as nullary function symbols. We call f the *head symbol* of t, if t has the form $f\, t_1 \ldots t_n$. Z3 allows a quantifier to bind several variables at once; instead of writing $Qx_1 \ldots x_n.\, t$, we use the shorter notation $Q\overline{x}.\, t$. In a similar fashion, we abbreviate a list of terms $t_1 \ldots t_n$ as $\bar{t}$.

Although we do not consider sorts and well-sorted terms in detail, we assume that there is a set of primitive sorts including the distinguished sort *bool* and that the terms under consideration have exactly one sort.

Terms of sort *bool* are called *formulas*. We use P, possibly with subscript, to denote a formula, and $\top$, $\bot$, $\neg$, $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$ with the usual meaning. Additionally, Z3 uses the special binary symbol $\sim$ to represent equisatisfiability in proof objects: Two formulas are equisatisfiable if their existential closure is equivalent. For example, the formula $(\neg x \vee \bot) \sim (\neg y)$ is equivalent to $(\exists x.\, \neg x \vee \bot) \leftrightarrow (\exists y.\, \neg y)$. We use L, possibly with subscript, to denote a formula for which negation never creates a doubly-negated formula; such a

formula is called *literal*. Especially, if L is an already negated formula $\neg P$, then $\neg L$ stands for the formula P.

3.2 Proof Terms

The proof terms of Z3 are designed as natural deduction style proofs. Before describing them, we fix the following terminology. A *sequent*, denoted by $\Gamma \vdash P$, consists of a set of formulas Γ, the *hypotheses*, and a single formula P, the *proposition*. A list of sequents $\Gamma_{i_1} \vdash P_{i_1} \ \ldots \ \Gamma_{i_n} \vdash P_{i_n}$ for a given index set $I = \{i_1, \ldots, i_n\}$ is abbreviated as $\langle i \in I \mid \Gamma_i \vdash P_i \rangle$. A *proof rule* is a schema

$$\frac{S_1 \ \ldots \ S_n}{S}$$

consisting of schematic sequents $S_1, \ldots S_n$, the *assumptions*, and a schematic sequent S, the *conclusion*. A proof rule without assumptions is called an *axiom*.

A derivation tree over a set of proof rules is called a *proof term*. The proof of unsatisfiability of a given set of formulas (called *assertions*) derives $\bot$ from them. Therefore, a *valid proof term* is a proof term where the final sequent, that is, the root of the derivation tree, is $\Gamma \vdash \bot$ with Γ being a subset of the assertions.

In contrast to our formal presentation, Z3 represents proof terms by giving, for each derivation step, the name of the proof rule applied, the proofs of the assumptions, and the proposition of the conclusion; hypotheses are only implicit in Z3 proof terms.

3.3 Proof Rules

We consider here only a subset of 33 of the altogether 38 proof rules of Z3; the remaining five proof rules represent coarse-grained steps compressing one or more applications of the other (fine-grained) proof rules. Figure 1 shows a majority of Z3's proof rules. The symbol $\rightsquigarrow$ stands for either $\rightarrow$, $\leftrightarrow$, or $\sim$; the symbol $\simeq$ denotes one of the three congruence relations $=$, $\leftrightarrow$, and $\sim$; and by $\diamond$ we mean any commutative binary function (for example, addition $+$ or equality $=$). Compared to the proof rules implemented in Z3, we slightly shortened some rule names[1] and dropped the proof rule **goal** (as it is syntactically and semantically equivalent to **asserted**).

We refrain from discussing semantics of the presented proof rules, except for **asserted** and **hyp**. Although they are syntactically equivalent, there is a semantic difference between them: The **asserted** rule may only assume one of the assertions given to the solver for refutation. In contrast, the **hyp** rule may assume any proposition, which, however, must be explicitly discharged later on in the proof using **lemma**.

[1] See Appendix A for details.

$$\frac{}{\emptyset \vdash \top}\,\mathbf{true} \qquad \frac{\Gamma \vdash P}{\Gamma \vdash P \leftrightarrow \top}\,\mathbf{iff}_{\top} \qquad \frac{\Gamma \vdash \neg P}{\Gamma \vdash P \leftrightarrow \bot}\,\mathbf{iff}_{\bot} \qquad \frac{\Gamma \vdash P_1 \leftrightarrow P_2}{\Gamma \vdash P_1 \sim P_2}\,\mathbf{iff}_{\sim}$$

$$\frac{\Gamma \vdash L_1 \wedge \ldots \wedge L_n}{\Gamma \vdash L_i}\,\mathbf{elim}_{\wedge} \qquad \frac{\Gamma \vdash \neg(L_1 \vee \ldots \vee L_n)}{\Gamma \vdash \neg L_i}\,\mathbf{elim}_{\neg\vee}$$

$$\frac{}{\{P\} \vdash P}\,\mathbf{asserted} \quad (P \text{ is an assertion}) \qquad \frac{\Gamma_1 \vdash P_1 \qquad \Gamma_2 \vdash P_1 \rightsquigarrow P_2}{\Gamma_1 \cup \Gamma_2 \vdash P_2}\,\mathbf{mp}_{\rightsquigarrow}$$

$$\frac{}{\{P\} \vdash P}\,\mathbf{hyp} \qquad \frac{\Gamma \cup \{L_1, \ldots, L_n\} \vdash \bot}{\Gamma \setminus \{L_1, \ldots, L_n\} \vdash \neg L_1 \vee \ldots \vee \neg L_n}\,\mathbf{lemma}$$

$$\frac{\Gamma \vdash \bigvee_{i \in I} L_i \qquad \langle i \in I_s \mid \Gamma_i \vdash \neg L_i \rangle}{\Gamma \cup \bigcup_{i \in I_s} \Gamma_i \vdash \bigvee_{i \in I \setminus I_s} L_i}\,\mathbf{unit} \quad I = \{1, \ldots, n\},\ I_s \subseteq I$$

$$\frac{}{\emptyset \vdash t \simeq t}\,\mathbf{refl}_{\simeq} \qquad \frac{\Gamma_1 \vdash t_1 \simeq t_2 \qquad \Gamma_2 \vdash t_2 \simeq t3}{\Gamma_1 \cup \Gamma_2 \vdash t_1 \simeq t_3}\,\mathbf{trans}_{\simeq}$$

$$\frac{\Gamma \vdash t_1 \simeq t_2}{\Gamma \vdash t_2 \simeq t_1}\,\mathbf{symm}_{\simeq} \qquad \frac{\Gamma \vdash (t_1 \diamond t_2 \simeq s_1 \diamond s_2)}{\Gamma \vdash (t_2 \diamond t_1 \simeq s_2 \diamond s_1)}\,\mathbf{comm}_{\simeq\diamond}$$

$$\frac{\langle i \in I_s \mid \Gamma_i \vdash t_i \simeq s_i \rangle}{\bigcup_{i \in I_s} \Gamma_i \vdash f\ t_1 \ldots t_n \simeq f\ s_1 \ldots s_n}\,\mathbf{mono}_{\simeq} \quad \begin{array}{l} I = \{1, \ldots, n\},\ I_s \subseteq I, \text{ and} \\ \emptyset \vdash t_i \simeq s_i \text{ for every } i \in I \setminus I_s \end{array}$$

$$\frac{}{\emptyset \vdash \neg(\forall \overline{x}.\, P\ \overline{x}) \vee P\ \overline{t}}\,\mathbf{inst}_{\forall} \qquad \frac{\Gamma \vdash P_1\ \overline{x} \sim P_2\ \overline{x}}{\Gamma \vdash (Q\overline{x}.\, P_1\ \overline{x}) \sim (Q\overline{x}.\, P_2\ \overline{x})}\,\mathbf{intro}_Q$$

$$\frac{}{\emptyset \vdash (\forall \overline{x}.\, \neg(x_1 = t_1) \vee \ldots \vee \neg(x_n = t_n) \vee P\ \overline{x}) = P\ \overline{t}}\,\mathbf{der} \quad \begin{array}{l} (x_i \text{ does not} \\ \text{occur in } t_i) \end{array}$$

$$\frac{}{\emptyset \vdash (\exists \overline{x}.\, P\ \overline{y}\ \overline{x}) \sim P\ \overline{y}\ [\overline{f}\ \overline{y}]}\,\mathbf{sk}_{\exists} \qquad \frac{}{\emptyset \vdash \neg(\forall \overline{x}.\, P\ \overline{y}\ \overline{x}) \sim \neg P\ \overline{y}\ [\overline{f}\ \overline{y}]}\,\mathbf{sk}_{\forall}$$

($[\overline{f}\ \overline{y}]$ is a list of fresh function symbols, each applied to $\overline{y}$)

Figure 1: Z3 proof rules.

In contrast to the succinct notations of the already given proof rules, the remaining ones are more complex. Therefore, we just indicate their semantics and give examples for some of them. Except where stated otherwise, these rules are axioms.

distrib distributes conjuncts over disjunctions or disjuncts over conjunctions.

def-axiom Tseitin's axioms for (definitional) conjunctive normal form, for example, $\neg(P_1 \wedge P_2) \vee P_1$.

nnf-pos, nnf-neg are used for the conversion of formulas into negation normal form and may have any number of assumptions which, like the conclusion, are equisatisfiability propositions.

def-intro, def-apply handle local definitions to avoid explosion of formula size. **def-intro** locally defines a constant to be equisatisfiable with a formula, and **def-apply** unfolds this definition in a given assumption.

elim$_Q$ eliminates unused quantified variables from the left-hand side of an

equivalence. A typical instance of this rule is $(\forall x.\, \top) \leftrightarrow \top$.

push$_Q$, pull$_Q$ distribute quantifiers over conjunctions and disjunctions. An example for **push$_Q$** is $(\forall x.\, P_1\ x \wedge P_2\ x) \leftrightarrow ((\forall x.\, P_1\ x) \wedge (\forall x.\, P_2\ x))$.

rewrite expresses simplifications as an equality $t_1 = t_2$ or as an equivalence $P_1 \leftrightarrow P_2$ where the right-hand side is obtained from the left-hand side by application of built-in (theory-specific) laws of the head symbol of t_1 or P_1, respectively. Examples are $3 + 0 = 3$ and $(x < y \wedge 0 \leq z) \leftrightarrow (0 \leq z \wedge x < y)$.

th-lemma tags theory-specific propositions, for example a disjunction of inequalities for the theory of linear arithmetic. Dually, this proof rule may also occur inside proofs, that is, non-axiomatically, and then deduce $\bot$ from an arbitrary number of assumptions.

4 Proof Reconstruction in Isabelle/HOL

We reconstruct proofs in a bottom-up manner starting from the axioms and going towards the final sequent. Every reconstruction step, that is, every application of a proof rule, yields an Isabelle theorem, which encodes both the hypotheses and the proposition of a sequent; and due to the LCF architecture, Isabelle ensures that every such deduction is sound. After every step, we compare the inferred theorem's proposition with the proposition given in the proof term. Any mismatch indicates a potential bug either in our reconstruction or in Z3's proof terms. In the final step, we check that the resulting theorem (corresponding to the final sequent) complies with the above condition for a valid proof term, that is, its hypotheses are a subset of the assertions given to the solver.

It is worth noting that our proof reconstruction for Z3 is sound, but possibly incomplete: We guarantee that, whenever the reconstruction succeeds, the underlying proof term is correct. However, the failure of proof reconstruction may either mean that the given proof term is invalid or that there is a bug in our reconstruction method. With the help of a large test bed, we hope to eliminate the latter source of failures.

4.1 Z3 Terms and Formulas in Isabelle/HOL

There is a natural mapping of many-sorted first-order terms and formulas as presented in Subsection 3.1 to higher-order terms of Isabelle/HOL, except for one issue: There is no notion of equisatisfiability in Isabelle/HOL. However, there is no need to provide a formalization for it: Careful study of Z3's proof rules reveals that nearly all occurrences of equisatisfiability can also be replaced by equivalences (see, for example, **iff$_\sim$**, **refl$_\simeq$**, or **intro$_Q$** in Figure 1). To maintain soundness, we only need to interpret the proof rule **sk$_Q$** in a special way (see also the paragraph on Skolemization in the next subsection).

Technique	Proof rules
basic inference rules	**asserted**, **hyp**
single theorem	**true**, $\mathbf{iff}_{\top}$, $\mathbf{iff}_{\bot}$, $\mathbf{iff}_{\sim}$, $\mathbf{mp}_{\rightsquigarrow}$, $\mathbf{refl}_{\simeq}$, $\mathbf{symm}_{\simeq}$, $\mathbf{trans}_{\simeq}$, $\mathbf{comm}_{\simeq\diamond}$
several theorems	$\mathbf{elim}_{\wedge}$, $\mathbf{elim}_{\neg\vee}$, **lemma**, **unit**, $\mathbf{mono}_{\simeq}$, $\mathbf{intro}_{Q}$
tactics	$\mathbf{inst}_{\forall}$, **der**, **distrib**, **def-axiom**, **nnf-pos**, **nnf-neg**, $\mathbf{elim}_{Q}$, $\mathbf{push}_{Q}$, $\mathbf{push}_{Q}$
specialized methods	**def-intro**, **apply-def**, $\mathbf{sk}_{Q}$, **rewrite**, **th-lemma**

Table 1: Z3 proof rules and their reconstruction techniques.

4.2 Z3 Proof Rules in Isabelle/HOL

We use five different techniques to model Z3 proof rules in Isabelle/HOL:

1. basic inference rules of Isabelle;
2. a single theorem, which is the translation of a Z3 proof rule into Isabelle/HOL;
3. a composition of several theorems, controlled by the structure of the assumptions or of the conclusion; this is essentially an iterative application of the previous technique;
4. proof tactics of Isabelle;
5. specialized methods, if none of the other techniques applies.

Table 1 gives a condensed overview of our proof reconstruction implementation by mapping every technique to the proof rules modeled with it. Let us now consider in more detail the second and third techniques and some of the specialized methods.

Proof Rules as Theorems. Isabelle/HOL allows us to model simple proof rules directly as (schematic) theorems. To illustrate this idea, let us have a look at the proof rule $\mathbf{mp}_{\rightarrow}$:

$$\frac{\Gamma_1 \vdash P_1 \qquad \Gamma_2 \vdash P_1 \rightarrow P_2}{\Gamma_1 \cup \Gamma_2 \vdash P_2}\ \mathbf{mp}_{\rightarrow}$$

Mapping this proof rule to Isabelle/HOL, we get the following theorem:

$$?P_1 \Longrightarrow ?P_1 \longrightarrow ?P_2 \Longrightarrow ?P_2$$

With concrete theorems to instantiate $?P_1$ and $?P_1 \longrightarrow ?P_2$ with, we can deduce the theorem corresponding to the conclusion $?P_2$.

This technique can only be applied to proof rules with a fixed format. Proof rules with a varying number of assumptions, for example, cannot be directly expressed as theorems in Isabelle/HOL. Still, in some cases, a variant of this

idea can be applied. For example, consider the proof rule $\mathbf{elim}_\wedge$ and the following two theorems:

$$?P \wedge ?Q \Longrightarrow ?P \qquad ?P \wedge ?Q \Longrightarrow ?Q$$

Given a concrete conjunction as assumption and one of its conjuncts as conclusion of $\mathbf{elim}_\wedge$, iterative application of the above rules, guided by the structure of the conjunction, yields the conjunct.

Skolemization. Isabelle/HOL provides an axiomatization of the Hilbert choice operator ε: If $\exists x.\, P\, x$ holds, then the term $\varepsilon x.\, P\, x$ is equivalent to one witness x such that $P\, x$ holds. With this operator, it is straightforward to model the $\mathbf{sk}_Q$ proof rules. Consider, for example, the sequent

$$\emptyset \vdash (\exists x.\, y < x) \sim (y < f\ y)$$

where f is a Skolem function. With the choice operator, we can write this sequent equivalently as follows:

$$\emptyset \vdash (\exists x.\, y < x) \sim (y < (\lambda y.\ \varepsilon x.\, y < x)\ y)$$

This naïve approach of replacing Skolem functions by choice operators has one drawback: The size of propositions grows exponentially. Therefore, we treat Skolem functions as locally defined functions with hypothetical definitions:

$$\{f = (\lambda y.\ \varepsilon x.\, y < x)\} \vdash (\exists x.\, y < x) \sim (y < f\ y)$$

Note that this way, the proposition is identical to the original proposition above. As a consequence, our special treatment of the $\mathbf{sk}_Q$ rules requires no changes to other proof rules.

The additionally introduced hypotheses are discharged at the end of the reconstruction. For example, from the final sequent

$$\{f = (\lambda y.\ \varepsilon x.\, y < x), \ldots\} \vdash \bot$$

we infer a theorem with proposition $f = (\lambda y.\ \varepsilon x.\, y < x) \Longrightarrow \bot$, instantiate f (which does not occur elsewhere) with the right-hand side of its definition and apply the following theorem:

$$(?t = ?t \Longrightarrow \mathit{False}) \Longrightarrow \mathit{False}$$

Our treatment of the $\mathbf{sk}_Q$ rule has the benefit that, due to the additional hypotheses, we can soundly replace equisatisfiability by logical equivalence.

Theory Lemmas. Z3 handles all theory-specific properties uniformly by the proof rule **th-lemma**. Since there is no direct hint as to which theory the proposition under consideration belongs to, we need to sequentially try different decision procedures: a linear arithmetic solver and, for propositions of the array theory, the simplifier instrumented with the usual axiomatization for extensional arrays.

Logic	Solved		Reconstructed		Failed			Factor
	#	Time	#	Time	#T	#Z	#R	
QF_UF	96	2.992 s	33	19.52 s	36	27	0	6.5
QF_UFLIA	99	0.534 s	93	15.80 s	6	0	0	29.6
QF_UFLRA	100	0.189 s	43	105.51 s	57	0	0	558.3
AUFLIA	100	0.180 s	50	14.64 s	19	0	31	81.3
AUFLIRA	100	0.051 s	81	1.24 s	13	0	6	24.3

Table 2: Results of proof reconstruction for selected SMT-LIB logics.

Rewrite Rules. Similar to theory lemmas, the proof rule **rewrite** is generic; that is, concrete propositions may stem from any of the supported theories or from the propositional logic core. There is no explicit hint of the source of such propositions (except by analyzing the head function symbol of the proposition's left-hand side). Consequently, we need to (sequentially) try different means to reconstruct applications of this proof rule. Our set of reconstruction tools consists of a list of (schematic) theorems (see second technique above) expressing basic rewrite steps, a special tactic exploiting associativity and commutativity of conjunction and disjunction, the simplifier, and a linear arithmetic decision procedure.

5 Experimental Results

We tested the presented proof reconstruction for Z3 on the database of SMT-LIB benchmarks [1]. From each of the logics QF_UF, QF_UFLIA, QF_LRA, AUFLIA, and AUFLIRA, we randomly chose 100 unsatisfiable benchmarks and applied Z3 with a timeout of 2 minutes to them. To all proofs found by Z3, we applied reconstruction with a timeout of 5 minutes. Table 2 summarizes our experimental results. For every SMT-LIB logic, the table gives the number of benchmarks solved by Z3 along with the corresponding average run-time, the number of successfully reconstructed proofs along with the corresponding average run-time, the number of failed reconstructions (broken down into the number of timeouts #T, Z3 bugs #Z, and reconstruction bugs #R), and the ratio between the run-times of reconstruction and proof finding. The experiments were conducted on a Linux system running on an Intel Core 2 Duo processor with 2.4 GHz and 4 GB RAM.

The first observation is that proof reconstruction takes much longer than proof finding, and even times out in many cases. There are at least two reasons to this: (1) Our implementation is not (yet) optimized for speed. (2) Proof terms of Z3 contain high-level proof steps requiring expensive proof search in Isabelle (for example, for **th-lemma**). Additional information accompanying the proof rules (indicating the theory for **th-lemma** or giving the kind of simplification for **rewrite**) would be helpful to improve the second point.

The second observation is that proof reconstruction does not always succeed due to reconstruction bugs; we still need further tuning of the applied tactics.

Finally, we found four different bugs in Z3 related to the creation of proof terms leading to several failures in the logic QF_UF. Furthermore, while developing our reconstruction, we have found a few inconsistencies of Z3's proofs with respect to the available documentation[2]; more precisely, the proof rules' documentation is more restrictive than their instances in proof terms. We reported these issues, and they have been fixed.

6 Related Work

Proof reconstruction for SMT solvers in higher-order theorem provers is not a new idea. There is work on reconstructing haRVey's proofs using Isabelle/HOL [8], but it is restricted to quantified formulas over uninterpreted functions, that is, first-order logic with equality; in particular, it does not consider linear arithmetic. The reconstruction of CVC3's proofs using HOL Light [5] differs from our work due to the fact that the CVC3 proof format follows a different philosophy: CVC3 proof terms provide a much finer level of detail especially concerning theories. For example, instead of two inference rules bundling all theory-specific reasoning in Z3, there are approximately 60 rules in CVC3 only for the theory of linear arithmetic. Reconstructing such fine-grained proof terms requires some more work, but implementing particular CVC3 inference rules can also be simpler than implementing one complex Z3 rule. Although comparative measures showing the advantage of either approach are hard to take, we think that a small set of rules (like in Z3) with additional information to guide proof reconstruction (for theory-specific rules like **rewrite** or **th-lemma**) are a good compromise between the two philosophies.

In contrast to using a higher-order theorem prover, other approaches to proof checking for SMT solvers have also been studied, notably based on a rewrite system [9] and on the LF logical framework [11].

7 Conclusion

We presented the proof format and the proof rules of the SMT solver Z3 and described a method to reconstruct (and thus check) proofs found by Z3 in the theorem prover Isabelle/HOL.[3] To the best of our knowledge, this is the first independent work to exploit Z3's proof terms.

Future work. Our main goal is to eliminate the discussed short-comings and to improve the performance of proof reconstruction.

We are also in progress to incorporate our work into Isabelle/HOL (see [2] for

[2] This refers to the API documentation provided with Z3.

[3] Our work is publicly accessible from http://www4.in.tum.de/~boehmes.

preliminary work in this direction), similar to an already existing SAT solver integration [12], with the goal to combine the power of automated proof search with the trustworthiness of Isabelle/HOL proof checking. Eventually, this will result in a proof procedure for Isabelle based on Z3.

Finally, we would like to apply proof reconstruction to problems involving bit-vectors which will require to enhance at least our treatmeant of the proof rules **rewrite** and **th-lemma**.

Acknowledgments. The author thanks Leonardo de Moura and Nikolaj Bjørner for their support and fruitful discussions, as well as Alexander Krauss, Jasmin Blanchette, and the referees for comments on this work.

This work was supported by the German Federal Ministry of Education and Research under grant 01IS07008.

References

[1] C. Barrett, S. Ranise, A. Stump, and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). http://www.smt-lib.org, 2008.

[2] S. Böhme, M. Moskal, W. Schulte, and B. Wolff. HOL-Boogie — An Interactive Prover-Backend for the Verifying C Compiler. *Journal of Automated Reasoning*, 2009. To appear.

[3] L. de Moura and N. Bjørner. Proofs and Refutations, and Z3. In *Proceedings of the LPAR 2008 Workshops, Knowledge Exchange: Automated Provers and Proof Assistants, and the 7th International Workshop on the Implementation of Logics*, volume 418 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[4] L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[5] Y. Ge and C. Barrett. Proof Translation and SMT-LIB Benchmark Certification: A Preliminary Report. In *Workshop on Satisfiability Modulo Theories*, Aug. 2008. Extended abstract.

[6] M. Gordon, R. Milner, and C. P. Wadsworth. Edinburgh LCF: A Mechanised Logic of Computation. *LNCS*, 78, 1979.

[7] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press, 1993.

[8] C. Hurlin, A. Chaieb, P. Fontaine, S. Merz, and T. Weber. Practical Proof Reconstruction for First-Order Logic and Set-Theoretical Constructions. In *Proceedings of the Isabelle Workshop 2007*, pages 2–13, July 2007.

[9] M. Moskal. Rocket-Fast Proof Checking for SMT Solvers. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4963 of *Lecture Notes in Computer Science*, pages 486–500. Springer, 2008.

[10] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[11] A. Stump. Proof Checking Technology for Satisfiability Modulo Theories. *Electronic Notes in Theoretical Computer Science*, 228:121–133, 2009.

[12] T. Weber. Integrating a SAT Solver with an LCF-style Theorem Prover. In *Proceedings of the Third Workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR 2005)*, volume 144(2) of *Electronic Notes in Theoretical Computer Science*, pages 67–78. Elsevier, Jan. 2006.

A Z3 Proof Rule Names

The following table maps the short proof rule names used in this work to the names used by Z3.

short name	Z3 name	short name	Z3 name
true	`true`	**der**	`der`
asserted	`asserted`	**inst**	`quant-inst`
goal	`goal`	**hyp**	`hypothesis`
mp$_{\rightarrow}$	`mp`	**lemma**	`lemma`
mp$_{\leftrightarrow}$	`mp`	**unit**	`unit-resolution`
refl	`refl`	**iff**$_{\top}$	`iff-true`
symm	`symm`	**iff**$_{\bot}$	`iff-false`
trans	`trans`	**comm**$_{\simeq\diamond}$	`commutativity`
mono	`monotonicity`	**def-axiom**	`def-axiom`
intro$_Q$	`quant-intro`	**def-intro**	`def-intro`
distrib	`distributivity`	**apply-def**	`apply-def`
elim$_{\wedge}$	`and-elim`	**iff**$_{\sim}$	`iff~`
elim$_{\neg\vee}$	`not-or-elim`	**nnf-pos**	`nnf-pos`
rewrite	`rewrite`	**nnf-neg**	`nnf-neg`
pull$_Q$	`pull-quant`	**sk**$_Q$	`sk`
push$_Q$	`push-quant`	**mp**$_{\sim}$	`mp~`
elim$_Q$	`elim-unused-vars`	**th-lemma**	`th-lemma`