Fast LCF-Style Proof Reconstruction for Z3

Sascha Böhme[1] and Tjark Weber[2,*]

[1] Technische Universität München, boehmes@in.tum.de
[2] University of Cambridge, tw333@cam.ac.uk

Abstract. The Satisfiability Modulo Theories (SMT) solver Z3 can generate proofs of unsatisfiability. We present independent reconstruction of these proofs in the theorem provers Isabelle/HOL and HOL4 with particular focus on efficiency. Our highly optimized implementations outperform previous LCF-style proof checkers for SMT, often by orders of magnitude. Detailed performance data shows that LCF-style proof reconstruction can be faster than proof search in Z3.

1 Introduction

Interactive theorem provers like Isabelle/HOL [1] and HOL4 [2] have become invaluable tools in formal verification. They typically provide rich specification logics, which allow modelling complex systems and their behavior. Despite the merits of user guidance in proving theorems, there is a need for increased proof automation in interactive theorem provers: even proving a simple theorem can be a tedious task.

In recent years, automated theorem provers have emerged for combinations of first-order logic with various background theories, e.g., linear arithmetic, arrays, bit vectors [3]. These provers, called Satisfiability Modulo Theories (SMT) solvers, are of particular value in formal verification, where specifications and verification conditions can often be expressed as SMT formulas [4,5]. SMT solvers also have applications in model checking, constraint solving, and other areas.

Interactive theorem provers can greatly benefit from the reasoning power of SMT solvers. Proof obligations that are SMT formulas can simply be passed to the automated prover, which will solve them without further human guidance. However, almost every SMT solver is known to contain bugs [6]. When integrated naively, the SMT solver (and the integration) become part of the trusted code base: bugs could lead to inconsistent theorems in the interactive prover. For formal verification, where correctness is often paramount, this is undesirable.

The problem can be solved by requiring the SMT solver to produce proofs (of unsatisfiability), and reconstructing these proofs in the interactive prover. Among the proof-producing solvers is Z3 [7], a state-of-the-art SMT solver developed by Microsoft Research. In this paper, we present independent reconstruction for the proofs generated by Z3 in Isabelle/HOL and HOL4. These two

* The first author was supported by the German Federal Ministry of Education and Research under grant 01IS07008. The second author was supported by the British EPSRC under grant EP/F067909/1.

popular LCF-style [8] systems are based on a relatively small trusted kernel (see Sect. 3) that provides a fixed set of simple inference rules. Z3, on the other hand, uses a number of powerful inference rules in its proofs (see Sect. 4). This makes proof reconstruction challenging.

Our LCF-style implementations of proof reconstruction (see Sect. 5) do not extend the trusted code base. Any attempt to perform an unsound inference will be caught by the underlying theorem prover's kernel. In contrast, a stand-alone proof checker for Z3 could be implemented much more efficiently, but would have to be trusted. For utmost reliability, the latter approach is clearly not ideal, also because Z3's proofs are relatively difficult to check: besides first-order reasoning also decision procedures for supported theories (e.g., arrays, linear arithmetic) are required. As a mixture of both worlds, a proof checker obtained by reflection could be much faster than our LCF-style implementation yet be formally verified. However, sparse documentation of Z3's proof rules would heavily complicate this approach, while our implementations just fall back to existing automated proof tools for underspecified cases. Maybe not surprisingly, the only proof checker applicable to all Z3 proofs that existed previous to our work was an older version of Z3 [9]. By using Isabelle/HOL and HOL4 as proof checkers, we in fact discovered previously unknown bugs in Z3.

Driven by the nature of proof obligations commonly seen in formal verification and related domains, we restrict ourselves to reconstruct proofs of first-order logic theorems over the theories of equality and uninterpreted functions, arrays, and linear integer and real arithmetic. In particular, we do not consider proof reconstruction for the theory of bitvectors.

Evaluation of our implementations (see Sect. 6) is performed on SMT-LIB problems [10]. This is because there is a large and diverse library of problems readily available, promising a good coverage of Z3's proof rules. The implicit assumption that such an approach yields a practically useful system for typical goals in Isabelle/HOL or HOL4 has already been confirmed by first users.

2 Related Work

Work on the integration of SMT solvers with LCF-style theorem provers is relatively sparse.

McLaughlin et al. [11] describe a combination of HOL Light and CVC Lite for quantifier-free first-order logic with equality, arrays and linear real arithmetic. Ge and Barrett [12] present the continuation of that work for CVC3, the successor of CVC Lite, supporting also quantified formulas and linear integer arithmetic. CVC Lite's and CVC3's proof rules are much more detailed than the ones used by Z3. For instance, CVC3 employs more than 50 rules for the theory of real linear arithmetic alone. Although one would expect this to allow for more precise (and hence, faster) proof reconstruction, McLaughlin et al. report that their implementation is around six times slower than a decision procedure for linear real arithmetic implemented directly in HOL Light. For an in-depth comparison of our work with proof reconstruction for CVC3 see Sect. 6.

Fontaine et al. [13] describe an integration of the SMT solver haRVey with Isabelle/HOL. Their work is restricted to quantifier-free first-order logic with equality and uninterpreted functions. Hurlin et al. [14] extend this approach to quantified formulas. Skolemization is discussed in great detail. Unlike in our work, background theories (e.g., linear arithmetic, arrays) are not supported.

Recently, the first author [15] presented proof reconstruction for Z3 in the theorem prover Isabelle/HOL. We improve upon that work in both reconstruction speed and completeness (i.e., correct coverage of Z3's inference rules). We discuss similarities and differences in detail in Sect. 5, before comparing performance in Sect. 6.

Common to the above approaches is their relatively poor performance on larger problems. Evaluation is typically done on a few selected, hand-crafted toy examples. Only [12,15] use a significant number of SMT-LIB [10] benchmarks for demonstrating success rates—at the cost of long reconstruction run-times. This paper is the first to focus on efficiency, and consequently, the first to give solid evidence of attainable performance for LCF-style proof reconstruction.

3 LCF-Style Theorem Proving

The term LCF-style [8] describes theorem provers that are based on a small inference kernel. Theorems are implemented as an abstract data type, and the only way to construct new theorems is through a fixed set of functions (corresponding to the underlying logic's axiom schemata and inference rules) provided by this data type. This design greatly reduces the trusted code base. Proof procedures based on an LCF-style kernel cannot produce unsound theorems, as long as the implementation of the theorem data type is correct.

Traditionally, most LCF-style systems implement a natural deduction calculus. Theorems represent *sequents* $\Gamma \vdash \varphi$, where Γ is a finite set of *hypotheses*, and φ is the sequent's *conclusion*. Instead of $\emptyset \vdash \varphi$, we simply write $\vdash \varphi$.

The two incarnations of LCF-style systems that we consider here, i.e., HOL4 and Isabelle/HOL, are popular theorem provers for polymorphic higher-order logic (HOL) [2], based on the simply-typed λ-calculus. Both systems share the implementation language, namely Standard ML.

Although Isabelle/HOL and HOL4 implement the same logic, they differ in their internal data structures, and even in the primitive inference rules provided: some rules that are primitive in one system are derived (i.e., implemented as a combination of primitive rules) in the other. Therefore, optimization is challenging, and performance comparisons must be taken with a grain of salt. Highly optimized proof procedures typically show similar performance in the two systems [16].

On top of their LCF-style inference kernels, both Isabelle/HOL and HOL4 offer various automated proof procedures: notably a simplifier, which performs term rewriting, a decision procedure for propositional logic, tableau- and resolution-based first-order provers, and decision procedures for Presburger arithmetic on integers and real algebra.

Substitution of type and term variables is a primitive inference rule in both Isabelle/HOL and HOL4. Consequently, substitution is typically much faster than (re-)proving the theorem's specific instance. General theorems (which we will call *schematic*) can, therefore, play the role of additional inference rules.

4 Z3: Language and Proof Terms

A detailed and perspicuous description of Z3's language and proof terms has been given in [9,15]. We briefly review the key features necessary to understand this work.

Z3's language is many-sorted first-order logic, based on the SMT-LIB language [10]. Basic sorts include bool, int and real. Interpreted functions include arithmetic operators ($+$, $-$, $\cdot$), Boolean connectives ($\vee$, $\wedge$, $\neg$), constants $\top$ and $\bot$, first-order quantifiers ($\forall$, $\exists$), the distinct predicate, and equality. It is worth noting that the connectives $\wedge$ and $\vee$ are polyadic functions in Z3, i.e., they can take an arbitrary number of arguments.

Z3's proof terms encode natural deduction proofs. The deductive system used by Z3 contains 34 axioms and inference rules. These range from simple rules like **mp** (modus ponens) to rules that abbreviate complex reasoning steps, e.g., **rewrite** for equality reasoning involving interpreted functions, or **th-lemma** for theory-specific reasoning. We discuss selected rules in more detail in Sect. 5.

Z3's proofs are directed acyclic graphs (DAGs). Each node represents application of a single axiom or inference rule. It is labeled with the name of that axiom or inference rule and the proposition to conclude. The edges of a proof graph connect conclusions with their premises. The hypotheses of sequents are not given explicitly. A designated root node concludes $\bot$.

5 Proof Reconstruction

Our work on Z3 proof reconstruction, although heavily optimized (and in large parts developed independently), shares certain features with the approach presented in [15], and more generally also with [14,16]. We thereby confirm that these solutions are of general interest and benefit, beyond a single theorem prover implementation. The similarities with related work are as follows.

Representation of Z3's language in higher-order logic is natural and direct. Basic types and interpreted functions have corresponding counterparts in the HOL implementations considered here. We translate uninterpreted functions and sorts into higher-order logic as variables and type variables, respectively. Arrays are translated as functions. Thus, array updates (store) become function updates, array lookup (select) reduces to function application, and extensionality is an axiom.

Representation of Z3's proofs in Standard ML is via a balanced tree, with lookup in $O(\log n)$, that maps node identifiers to proof nodes. The proof generated by

Z3 is parsed, and a corresponding ML value is built. Proof nodes are given by a disjoint union. Initially, each node contains the information that is recorded explicitly in the Z3 proof. Once the corresponding inference step has been checked in the theorem prover, this information is replaced by the derived theorem. Thus, lemmas only need to be derived once, even if they are used multiple times in the proof. This technique was originally proposed in [16], where efficient LCF-style proof checking for SAT solvers is discussed.

Depth-first (postorder) traversal of the proof, starting from the root node, determines the order in which proof steps are reconstructed. If there are steps in the Z3 proof that do not contribute to the derivation of the final $\bot$, they are never checked. This technique was also adapted from [16].

Assumptions in the Z3 proof can be introduced by three rules: **asserted** and **goal** introduce assumptions made in the input problem (from which $\bot$ is derived eventually), while **hypothesis** introduces arbitrary local assumptions. These must be discharged by the **lemma** rule later. We use the axiom schema $\{\varphi\} \vdash \varphi$ (which is available in both Isabelle/HOL and HOL4) to introduce assumptions, thereby inserting them as hypotheses whenever they are used in the proof. At the very end of proof reconstruction, we check that only assumptions from the input problem remain as hypotheses.

Skolem functions introduced by Z3's proof rule **sk** are given hypothetical definitions in terms of Hilbert's choice operator (see [14,15] for details). This allows to replace the equisatisfiability relation that Z3 uses in its proofs, which has no direct counterpart in higher-order logic, with equivalence.

Local definitions are used by Z3 to introduce abbreviations for formulas. The relevant rules are **intro-def** and **apply-def**. We model locally defined abbreviations by hypothetical definitions, in much the same way as Skolem functions.

5.1 Reconstruction Techniques

Beyond these similarities, however, there are numerous ways in which our approach differs from previous ones. Noticeably, we have spent considerable time on profiling (see Sect. 6.4). This has prompted faster reconstruction techniques for many of Z3's inference rules. We distinguish four different techniques to model a Z3 inference rule in an LCF-style system:

1. as a single primitive inference or schematic theorem,
2. as a combination of primitive inferences and/or schematic theorems,
3. by applying an automated proof procedure,
4. as a combination of the above.

These techniques vary in implementation effort and performance. Primitive inference rules and schematic theorems are typically the preferred choice where possible (because they are easy to use, and as fast as it gets), but they are limited in applicability: the set of primitive rules provided by an LCF-style kernel is fixed, and schematic theorems can only be applied to terms with a fixed structure that

is known in advance. For instance, deriving $\vdash \varphi$ from a conjunction $\vdash \varphi \wedge \psi$ is within the realm of possibility, but a derivation of $\vdash \varphi$ from an arbitrarily nested conjunction $\vdash \ldots \wedge \varphi \wedge \ldots$ already requires a combination of primitive inferences and/or schematic theorem instantiations.

Automated proof procedures, on the other hand, work well for rapid prototyping. About one third of Z3's proof rules merely require propositional reasoning, and another third perform relatively simple first-order reasoning. These rules can, in principle, be implemented by a single application of (1) a fast decision procedure for propositional logic [16], or (2) an automated prover for first-order logic with equality [17], respectively. Even though (1) internally employs a state-of-the-art SAT solver, and (2) has fared well in various CASC competitions [18], the key disadvantage of automated proof procedures is that their performance is hard to control. We have achieved speedups of three to four orders of magnitude by replacing calls to these automated tools with specialized implementations (using combinations of primitive inferences and/or schematic theorems) that perform the *specific* reasoning steps required to model Z3's proof rules.

Table 1 gives an overview of the reconstruction techniques currently used for the different proof rules of Z3. We apply automated proof procedures only to theory-specific rules and to approximate four rules that Z3 never used in our extensive evaluation. If performance for the latter rules was important, they might as well be implemented using the second category of reconstruction techniques.

We now describe relevant optimizations in more detail. Since primitive inference rules in different theorem provers often show different performance relative to each other, there is not necessarily one (prover-independent) optimal approach. We discuss alternatives where appropriate.

Reconstruction technique	Proof rules
Primitive inference or schematic theorem	**asserted**, **commutativity**, **goal**, **hypothesis**, **iff-false**, **iff-true**, **iff**$\sim$, **mp**, **mp**$\sim$, **refl**, **symm**, **trans**, **true**
Combination of primitive inferences and/or schematic theorems	**and-elim**, **apply-def**, **def-axiom**, **elim-unused**, **intro-def**, **lemma**, **monotonicity**, **nnf-neg**, **nnf-pos**, **not-or-elim**, **quant-inst**, **quant-intro**, **sk**, **unit-resolution**
Automated proof procedures	**der**, **distributivity**, **pull-quant**, **push-quant**
Combination of the above	**rewrite**, **rewrite***, **th-lemma**

Table 1. Proof rules and reconstruction techniques

5.2 Propositional and First-Order Reasoning

Nested conjunctions. A recurring task in Z3's proofs is to establish equivalence of two (arbitrarily parenthesized) conjunctions $\varphi \equiv p_1 \wedge \cdots \wedge p_n$ and $\psi \equiv q_1 \wedge \cdots \wedge q_n$, where $\{p_i \mid 1 \leq i \leq n\} = \{q_i \mid 1 \leq i \leq n\}$. Such permutated conjunctions arise for two reasons. First, conjunction in Z3 is polyadic, i.e., it can take an arbitrary

number of arguments, while in Isabelle/HOL and HOL4, conjunction is a binary (right-associative) operator. For instance, $\varphi_1 \wedge \varphi_2 \wedge \varphi_3$ in Isabelle/HOL is really short for $\varphi_1 \wedge (\varphi_2 \wedge \varphi_3)$. Second, unfolding of the distinct predicate leads to conjoined inequalities: e.g., $\mathsf{distinct}\ [x, y, z] \equiv x \neq y \wedge x \neq z \wedge y \neq z$.

A rewriting-based approach (using associativity, commutativity and idempotence of conjunction) turns out to be far too slow. Instead, we perform the required re-ordering not at the term level, but using ML data structures. We first derive theorems $\{\varphi\} \vdash p_i$ (for each p_i) by assuming φ and recursively applying conjunction elimination. These intermediate theorems are stored in a balanced tree, indexed by their conclusion. From them, we derive $\{\varphi\} \vdash \psi$ by recursion over the structure of ψ, using conjunction introduction. In a similar way, we get $\{\psi\} \vdash \varphi$. Combining both theorems, we obtain $\vdash \varphi \Leftrightarrow \psi$.

This implementation has complexity $O(n \log n)$. It improves over an implementation with quadratic complexity that had been part of the HOL4 theorem prover since 1991 [19].

Nested disjunctions are treated dual to nested conjunctions, but our implementations deviate from each other due to differences in the primitive inference rules available.

In HOL4, we first show that $\psi \equiv q_1 \vee \cdots \vee q_n$ follows from each of its disjuncts. Then we recurse over the structure of the premise $\varphi \equiv p_1 \vee \cdots \vee p_n$, using disjunction elimination to show that since each disjunct p_i implies ψ, we have $\{\varphi\} \vdash \psi$. Deriving ψ from each of its disjuncts is not completely straightforward. We achieve complexity $O(n \log n)$ by assuming ψ (thereby obtaining $\{\psi\} \vdash \psi$), and then recursively deriving $\{\vartheta_1\} \vdash \psi$ and $\{\vartheta_2\} \vdash \psi$ from $\{\vartheta_1 \vee \vartheta_2\} \vdash \psi$. This inference step is not provided as a primitive rule by the HOL4 kernel. We found an implementation that uses a combination of primitive rules to be roughly twice as fast as one that instantiates a schematic theorem $\vdash (\vartheta_1 \vee \vartheta_2 \Rightarrow \psi) \Rightarrow \vartheta_1 \Rightarrow \psi$ (and a similar theorem for ϑ_2).

In Isabelle/HOL, we show equivalence of φ and ψ by contraposition, i.e., by showing that $\neg\varphi$ and $\neg\psi$ are equivalent. This can be done in analogy to the case of conjunctions, only that instead of conjunction elimination and conjunction introduction, dual theorems for negated disjunctions must be applied. The complexity of this approach is again $O(n \log n)$.

Unit resolution implements the following inference rule, which strengthens a disjunction by removing disjuncts that have been disproved:

$$\frac{\Gamma \vdash \bigvee_{i \in I} \varphi_i \quad \langle \Gamma_i \vdash \neg\varphi_i \rangle_{i \in J}}{\Gamma \cup \bigcup_{i \in J} \Gamma_i \vdash \bigvee_{i \in I \setminus J} \varphi_i}\ \textbf{unit-resolution}$$

We model this inference rule in HOL4 by extending the technique for nested disjunctions described previously. For $i \in I \setminus J$, deriving the conclusion $\bigvee_{i \in I \setminus J} \varphi_i$ from each φ_i is done exactly as before. For $i \in J$, on the other hand, we use the fact that φ_i has been disproved, and that anything (in particular, the desired conclusion) follows from $\bot$. We found this implementation to be about 30% faster in HOL4 than the approach detailed in [16], which is more efficient only when proofs contain many successive resolution steps.

In Isabelle/HOL, we again use contraposition to model unit resolution. Assume that $\neg \bigvee_{i \in I \setminus J} \varphi_i$ holds and show that this together with the facts $\neg\varphi_i$ (for $i \in J$) implies $\neg \bigvee_{i \in I} \varphi_i$. Hence the premise $\bigvee_{i \in I} \varphi_i$ implies the conclusion $\bigvee_{i \in I \setminus J} \varphi_i$. The main step of this deduction employs the contraposition-based technique for nested disjunctions described previously.

Literal memoization. The Z3 proof rule **and-elim** deduces a conjunct from a polyadic conjunction. In an LCF-style system where conjunction is binary, this amounts to repeated application of conjunction elimination. Since **and-elim** is commonly applied to the same premise $\vdash \varphi$ several times, deducing a different conjunct each time, it is more efficient to explode $\vdash \varphi$ once and for all instead of repeatedly extracting a single conjunct. The resulting conjuncts are stored in the proof node that derived $\vdash \varphi$, indexed by a balanced tree for efficient lookup. This memoization technique applies dually to the rule **not-or-elim**.

Quantifier instantiations in Z3's proof rules can be determined by (first-order) term matching. No first-order proof search is necessary. We avoid the automated first-order provers that are built into Isabelle/HOL and HOL4: they are unnecessarily powerful, but relatively slow. Instead, we perform the required combinations of primitive inferences directly.

5.3 Theory-Specific Reasoning

With these optimizations in place, Z3's propositional and first-order inference steps are checked with reasonable efficiency. The one remaining performance hog is theory-specific reasoning, involving interpreted functions (e.g., linear arithmetic). This is performed by three proof rules in Z3: **rewrite**, **rewrite***[3] and **th-lemma**. We implement these rules by sequentially trying schematic theorems, exploiting associativity, commutativity and idempotence of conjunction and disjunction, trying the simplifier, and applying decision procedures for linear integer and real arithmetic.

When done naively, the automated tools, i.e., the simplifier and the decision procedures for linear arithmetic, dominate run-time. We were able to improve performance by reducing the number of proof obligations that are passed to these tools. In our current implementation the simplifier only rewrites array updates, but not Boolean or arithmetic operators: these are handled through schematic theorems or specialized proof procedures. We employ the following optimization techniques for theory-specific reasoning.

Schematic theorems. Matching a theorem's conclusion against a given term and, if successful, instantiating the theorem accordingly is typically much faster than deriving the instance again. By studying the actual usage of **rewrite** in Z3's proofs, we identified more than 230 useful schematic theorems. These include propositional tautologies such as $\vdash (p \Rightarrow q) \Leftrightarrow (q \vee \neg p)$, theorems about equality, e.g., $\vdash (x = y) \Leftrightarrow (y = x)$, and theorems of linear integer and real arithmetic,

[3] Since **rewrite*** is a variant of **rewrite**, we implicitly include the former when referring to the latter in the remainder of this section.

e.g., $\vdash x + 0 = x$. Together, these theorems allow about 76% of all terms given to **rewrite** to be proved by instantiation alone. Because of their generality, our schematic theorems should be useful for a wide range of benchmarks. We store all schematic theorems in a term net to allow faster search for a match.

To a smaller extent, we also use schematic theorems in the implementations of Z3's proof rules **def-axiom** and **th-lemma**.

Theorem memoization. Isabelle/HOL and HOL4 allow instantiating free variables in a theorem, while Z3 has to re-derive theorems that differ in their uninterpreted functions. Hence, there is more potential for theorem re-use in these provers than in Z3. We exploit this by storing theorems of linear arithmetic that are proved by **rewrite** or **th-lemma** in a term net. Since every theorem is also stored in a proof node anyway, this increases memory requirements only slightly (namely by the memory required for the net's indexing structure). Before invoking a decision procedure for linear arithmetic on a proof obligation, we attempt to retrieve a matching theorem from the net. However, proof obligations that occur frequently are often available as schematic theorems already. Therefore, with an extensive list of schematic theorems in place, the performance gained by theorem memoization is relatively small.

Generalization. We generalize proof obligations by replacing sub-terms that are outside the fragment of linear arithmetic with variables, before passing the proof obligation to the arithmetic decision procedures. This has two benefits. First, it makes theorem memoization more useful, since more general theorems potentially can be re-used more often. Second, it avoids expensive preprocessing inside the arithmetic decision procedures. For instance, HOL4's arithmetic decision procedures perform case splitting of if-then-else expressions. This could lead to an exponential number of cases. Z3's proof rule **th-lemma**, however, does not require the linear arithmetic reasoner to know about if-then-else: if necessary, conditionals are split using one of the other proof rules before Z3 solves the problem by linear arithmetic. Therefore, proof obligations are provable even with all conditionals treated as atomic.

6 Experimental Results

We evaluated our implementations in four ways. First, we measured success rates and run-times of proof reconstruction for 1273 SMT-LIB benchmarks drawn from the latest SMT-COMP [20], an annual competition of SMT solvers. Second, a selection of these benchmarks was taken to compare our implementations with proof reconstruction for CVC3 in HOL Light [11,12] (CH). Third, we contrasted our work with previous work on proof reconstruction for Z3 [15] (ZI). Finally, we measured profiling data to give a deeper insight into our results.

Evaluation was performed on problems comprising first-order formulas (partly quantifier-free, QF, partly with (+p) or without (-p) quantifier patterns) over (combinations of) the theories of equality and uninterpreted functions (UF), arrays (A), linear integer arithmetic (LIA), linear real arithmetic (LRA), combined linear arithmetic (LIRA), integer difference logic (IDL), real difference

logic (RDL). SMT-LIB logic names are formed by concatenation of the theory abbreviations given in parentheses.

We obtained all figures[4] on a Linux system with an Intel Core2 Duo T7700 processor, running at 2.4 GHz—the same machine that had been used to evaluate ZI. Measurements were conducted with Z3 2.3 and CVC3 2.2. As underlying ML environment, we used Poly/ML 5.3.0 for both Isabelle/HOL and HOL4. For comparability with ZI, we restricted proof search to two minutes and proof reconstruction to five minutes, and limited memory usage for both steps to 4 GB. All measured times are CPU times (with garbage collection in Poly/ML excluded).

Run-times for Isabelle/HOL are typically within a factor of 1–2 of HOL4 run-times. This is because we have fully implemented some of the optimizations described in this paper only for HOL4 so far. It should not be taken as an indication that HOL4 is more efficient than Isabelle/HOL per se.

6.1 SMT-COMP Benchmarks

Table 2 shows our results for Isabelle/HOL. For every SMT-LIB logic, we measured for Z3 the average time to find a proof and the average proof size, and for our implementation the average time to reconstruct a proof (timeouts are counted as 300 s). Additionally, we give success and timeout rates for proof reconstruction and the ratio of reconstruction time to solving time (*R-time*).

Our reconstruction succeeds on 75% of all problems solved by Z3. Failures are mostly due to timeouts (19%), but also due to shortcomings of Z3 and in a few cases of our reconstruction.[5] Note that low success rates, which are in most cases caused by timeouts, occur mainly in logics dominated by arithmetic. Closer analysis (of individual examples and profiling data, see Sect. 6.4) suggests that the theory-specific proof rules **rewrite** and **th-lemma** are to blame for this deficiency.

Proofs produced by Z3 may be extremely large. Our implementations are nevertheless able to reconstruct huge proofs within the given timeout. The largest proof successfully reconstructed had a size of 168 MB and comprised more than 3 million Z3 proof rules.

Reconstruction for individual logics is at least 2.7 times slower than proof search; on average the ratio lies at 18.5 despite our extensive optimizations. A thorough study of our measurements, however, reveals that for several problem classes, the picture is different: e.g., in case of AUFLIA−p, AUFLIA+p and AUFLIRA, our reconstruction is faster than Z3 on 11-34% of all problems.

6.2 Comparison with CH

In the implementation of CH we tested, proof reconstruction was tuned for logics including uninterpreted functions, arrays and linear integer arithmetic. Thus,

[4] Our data is available at `http://www4.in.tum.de/~boehmes/fast_proof_rec.html`.

[5] Z3 discovers injectivity of functions and uses this property for rewriting; reconstruction would require yet another special case for the already complex **rewrite** rule, which we have not implemented so far.

Logic	Solved (Z3)			Reconstructed		Rates		
	#	Time	Size	#	Time	Success	Timeout	R-time
AUFLIA+p	187	0.095 s	64 KB	187	0.413 s	100%	0%	4.34
AUFLIA−p	192	0.117 s	81 KB	190	1.962 s	98%	0%	16.72
AUFLIRA	189	0.292 s	366 KB	144	0.794 s	76%	0%	2.72
QF_AUFLIA	92	0.158 s	694 KB	49	136.498 s	53%	42%	863.85
QF_IDL	40	2.322 s	12 MB	19	173.875 s	47%	52%	74.89
QF_LIA	100	17.154 s	77 MB	26	208.713 s	26%	65%	12.17
QF_LRA	88	4.849 s	10 MB	55	142.351 s	62%	36%	29.36
QF_RDL	52	9.773 s	16 MB	26	173.953 s	50%	50%	17.80
QF_UF	87	16.131 s	62 MB	73	73.242 s	83%	16%	4.54
QF_UFIDL	55	4.511 s	12 MB	8	260.351 s	14%	85%	57.72
QF_UFLIA	91	1.543 s	4 MB	85	29.086 s	93%	6%	18.85
QF_UFLRA	100	0.086 s	914 KB	100	3.916 s	100%	0%	45.68
Total	1273	3.656 s	13 MB	962	67.785 s	75%	19%	18.54

Table 2. Experimental results (Isabelle/HOL) for selected SMT-COMP logics

Logic	Solved			Reconstructed			
	#	Time		Rate		Time	
		Z3	CVC3	I	H	I	H
AUFLIA+p	182	0.043 s	0.485 s	100%	84%	0.294 s	12.369 s
AUFLIA−p	173	0.050 s	0.149 s	99%	80%	0.165 s	5.791 s
QF_AUFLIA	89	0.053 s	3.150 s	52%	68%	17.197 s	4.068 s
QF_IDL	34	0.483 s	10.063 s	52%	41%	20.776 s	87.772 s
QF_LIA	18	0.398 s	25.587 s	55%	0%	33.870 s	n/a
QF_UF	58	1.531 s	7.920 s	100%	86%	13.465 s	14.645 s
QF_UFIDL	38	0.301 s	6.525 s	18%	18%	13.017 s	27.120 s
QF_UFLIA	82	0.034 s	4.114 s	100%	9%	4.897 s	18.866 s
Total	674	0.219 s	3.326 s	85%	64%	4.994 s	12.147 s

Table 3. Comparison between Z3/Isabelle/HOL and CVC3/HOL Light

Logic	Solved (Z3)		Reconstructed			Failed		Ratio
	#	Time	#	Time	Size	#T	#Z	R-time
AUFLIA	100	0.180 s	100	0.450 s	206 KB	0	0	2.5
AUFLIRA	100	0.051 s	97	0.034 s	16 KB	0	3	0.7
QF_UF	96	2.992 s	74	16.199 s	16 MB	1	21	5.4
QF_UFLIA	99	0.534 s	92	6.948 s	194 KB	7	0	13.0
QF_UFLRA	100	0.189 s	100	1.705 s	1 MB	0	0	9.0

Table 4. Experimental results (HOL4) for selected SMT-LIB logics

we only compare relevant results of the previous section with CH. Table 3 shows for each considered logic the number of problems solved by both Z3 and CVC3, their average run-time, the success rate of reconstruction for Isabelle/HOL (I) and HOL Light (H), and the average run-time of reconstruction (timeouts are counted as 300 s). Measuring these figures stimulated improvements to the implementation of CH; we only give the newest (and best) results for CH.

Clearly, our implementations can reconstruct more problems. More importantly, our reconstruction is on average more than two times faster than CH (and up to 42 times faster in the case of AUFLIA+p), even though CH does not have any parsing overhead (it uses a binary interface to CVC3, not a file-based one like our implementations).

6.3 Comparison with ZI

For comparability, we evaluated our implementations on the same set of SMT-LIB benchmarks (and, in fact, on the very same Z3 proofs) that were used to evaluate ZI [15]. Table 4 summarizes our experimental results for HOL4. For each SMT-LIB logic, the table shows the number of problems that Z3 determined to be unsatisfiable, the average run-time of Z3 (as reported in [15]), the number of successful proof reconstructions along with average HOL4 run-time and average Z3 proof size, and the number of failed proof reconstructions (due to timeouts #T and confirmed (and by now fixed) Z3 bugs #Z). The rightmost column of Tab. 4 shows the ratio of reconstruction time to solving time (R-time, cf. Sect. 6.1). We observe that this ratio is less than 1 for the AUFLIRA logic: LCF-style proof reconstruction for this logic is faster than proof search in Z3.

A more detailed comparison revealed that our highly optimized implementations (both in Isabelle/HOL and HOL4) outperform ZI on every problem of the AUFLIA, AUFLIRA, QF_UF and QF_UFLRA logics. Often, the performance gain is several orders of magnitude. Only the QF_UFLIA logic contains 18 problems for which reconstruction in HOL4, despite our optimizations, is slower than reported in [15]. We conclude that there is still potential for optimization in HOL4's decision procedure for integer arithmetic [21].

The figure to the right shows the average speedups of Isabelle/HOL (left) and HOL4 (right) over the run-times measured in [15]. Shaded bars give the maximum speedups achieved on individual problems (3,437 in the case of AUFLIRA for our HOL4 implementation!). Note that the figure uses a logarithmic scale. The overall speedup is 13.9 for HOL4 and 11.7 for Isabelle/HOL.

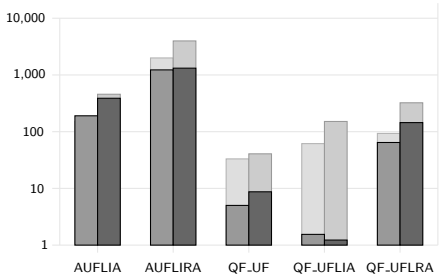

Fig. 1. Speedup factors

6.4 Profiling

To further understand these results and to identify potential for future optimization, we present relevant profiling data for our HOL4 implementation. (Isabelle/HOL profiling data is roughly similar.) Figures 2 to 6 show bar graphs that indicate the four[6] most time-consuming proof rules of Z3 for the respective SMT-LIB logic, and their percentaged shares of total run-time (dark bars). Additionally, time spent on parsing proof files is shown as well (see Tab. 4 for average proof sizes). We contrast each proof rule's relative run-time with the mean frequency of that rule (light bars).

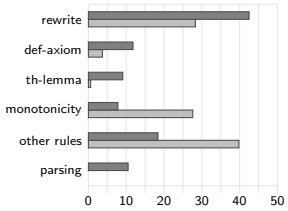

Fig. 2. AUFLIA

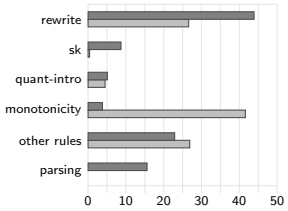

Fig. 3. AUFLIRA

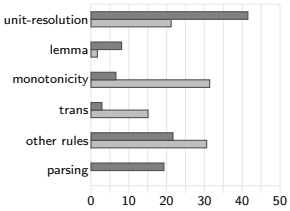

Fig. 4. QF_UF

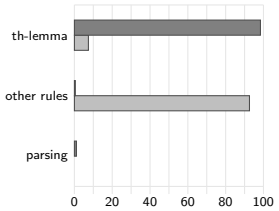

Fig. 5. QF_UFLIA

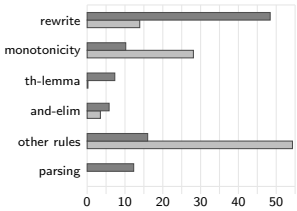

Fig. 6. QF_UFLRA

We see that after extensive optimization, proof reconstruction times are relatively well-balanced between Z3's different proof rules for most logics, although the **rewrite** rule still accounts for almost half of the total run-time on the AUFLIA, AUFLIRA, and QF_UFLRA benchmarks. For QF_UF, on the other hand, rewriting is relatively unimportant, but proofs contain many (in fact, over 5 million) **unit-resolution** inferences. Checking these consequently requires more than 41% of the run-time.

For these four logics, merely parsing Z3's proof files accounts for 11% (AUFLIA) to 19% (QF_UF) of the total run-time. Note that parsing does not involve the LCF-style inference kernel. Hence, there are limits to future performance gains that can be achieved through further optimization of LCF-style reasoning.

The picture looks different for the QF_UFLIA logic, where run-time is dominated almost entirely by the **th-lemma** rule. Parsing and other proof rules

[6] For QF_UFLIA, we only show **th-lemma** separately and combine all other rules.

of Z3 account for less than 2% of reconstruction time. Z3 internally uses the Simplex algorithm to decide linear arithmetic [9] and applies a branch and cut strategy for integers [22]. However, any information about how a decision is found is kept private: the **th-lemma** rule only represents the statement that a system of linear inequations is inconsistent. Consequently, reconstructing this proof rule amounts to finding the refutation again, with (probably) far less optimized decision procedures in the case of Isabelle/HOL and HOL4. Instead of making those faster, we conjecture that enriching **th-lemma** with the necessary information (which is already available in Z3) would improve efficiency of proof reconstruction considerably.

7 Conclusions

We have presented LCF-style proof reconstruction for Z3 in the theorem provers Isabelle/HOL and HOL4. In comparison to a recent implementation in Isabelle/HOL [15], our implementations are significantly faster on most benchmarks, often by orders of magnitude. Moreover, we have modeled the proof rules of Z3 in Isabelle/HOL and HOL4 with unprecedented accuracy, thereby achieving almost full (except for very few exotic corner cases) proof coverage. We also outperform a related implementation of proof reconstruction for CVC3 in HOL Light [12]. We have three main conclusions.

LCF-style proof checking for SMT is feasible. Our implementations give evidence that LCF-style proof checking for SMT solvers is not only possible in principle, but also that it is feasible. Clearly there is a steep price (in terms of performance) that one has to pay for checking proofs in a general-purpose LCF-style theorem prover. However, even for proofs with millions of inferences, LCF-style proof checking can be as fast as (or even faster than) proof search in Z3. This confirms a similar observation made in [16] regarding the feasibility of LCF-style proof checking for large SAT-solver generated proofs.

Specialized implementations can be significantly faster than generic proof procedures in LCF-style provers. The speedup that we achieved over [15] shows the importance of profiling when performance is an issue. We achieved speedups of several orders of magnitude by replacing calls to generic automated proof procedures with specialized implementations that perform the specific inferences required to check Z3's proof rules. This is despite the fact that some of these automated procedures employ state-of-the-art algorithms internally. Of course, writing fast specialized proof procedures requires much more familiarity with the theorem prover than simply calling automated proof procedures.

Z3's proof format could be easier to check. Conceptually, we only had to overcome minor hurdles to implement proof reconstruction for Z3's natural-deduction style proofs in the considered LCF-style theorem provers. That the conclusion of each inference step is given explicitly proved tremendously helpful. Proof rules **rewrite** and **th-lemma**, however, seem overly complex, and despite substantial optimization efforts, they still dominate run-time in our implementations.

We encourage the Z3 authors to (1) replace **rewrite** by a collection of simpler rules with clear semantics and less reconstruction effort, ideally covering specific rewriting steps of at most one theory, and (2) enrich **th-lemma** with additional easily-checkable certificates or trace information guiding refutations to avoid invocations of expensive (arithmetic) decision procedures. Currently the theoretical complexity of proof checking for background theories is the same as for proof search. We also hope that our experience will influence the design of a future SMT proof standard.

We have integrated proof reconstruction as an automated proof procedure in both HOL4 and Isabelle/HOL. If φ is an SMT formula, the user can invoke this proof procedure to have it pass $\neg\varphi$ to Z3, reconstruct Z3's proof of unsatisfiability (if one is found) to obtain $\{\neg\varphi\} \vdash \bot$, and from this $\vdash \varphi$ is derived by contradiction. Our implementations are freely available[7] and already in use.

There are numerous differences in internal data structures between HOL4, Isabelle/HOL and other LCF-style theorem provers, but based on previous experience [16] we have little doubt that the optimization techniques presented in this paper can be used to achieve similar performance in other theorem provers.

Future work includes (1) proof reconstruction for other SMT-LIB theories, e.g., bit vectors, (2) evaluation of proof reconstruction for typical goals of Isabelle/HOL or HOL4, (3) parallel proof reconstruction [23], by checking independent paths in the proof DAG concurrently, and (4) investigations into proof compression [24] for SMT proofs.

Acknowledgments

The authors are grateful to Nikolaj Bjørner and Leonardo de Moura for their help with Z3, to Yeting Ge for his help on proof reconstruction for CVC3, and to Alexander Krauss and Lukas Bulwahn for commenting on an earlier draft of this paper. Additionally, the second author would like to thank Hasan Amjad and Mike Gordon for their support.

References

1. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic. Volume 2283 of Lect. Notes in Comp. Sci. Springer (2002)
2. Gordon, M.J.C., Pitts, A.M.: The HOL logic and system. In: Towards Verified Systems. Volume 2 of Real-Time Safety Critical Systems Series. Elsevier (1994) 49–70
3. Kroening, D., Strichman, O.: Decision Procedures – An Algorithmic Point of View. Springer (2008)
4. Collavizza, H., Gordon, M.: Integration of theorem-proving and constraint programming for software verification. Technical report, Laboratoire d'Informatique, Signaux et Systèmes de Sophia-Antipolis (2008)
5. Böhme, S., Moskal, M., Schulte, W., Wolff, B.: HOL-Boogie — An Interactive Prover-Backend for the Verifying C Compiler. J. Automated Reasoning **44**(1–2) (February 2010) 111–114

[7] See `http://hol.sourceforge.net` and `http://isabelle.in.tum.de`.

6. Brummayer, R., Biere, A.: Fuzzing and delta-debugging SMT solvers. In: 7th International Workshop on Satisfiability Modulo Theories (SMT '09). (2009)
7. de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS '08). Volume 4963 of Lect. Notes in Comp. Sci., Springer (2008) 337–340
8. Gordon, M., Milner, R., Wadsworth, C.P.: Edinburgh LCF: A Mechanised Logic of Computation. Volume 78 of Lect. Notes in Comp. Sci. Springer (1979)
9. de Moura, L.M., Bjørner, N.: 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)
10. Ranise, S., Tinelli, C.: The SMT-LIB standard: Version 1.2 (August 2006) Retrieved January 21, 2010 from http://combination.cs.uiowa.edu/smtlib/papers/format-v1.2-r06.08.30.pdf.
11. McLaughlin, S., Barrett, C., Ge, Y.: Cooperating theorem provers: A case study combining HOL-Light and CVC Lite. Electronic Notes in Theoretical Computer Science **144**(2) (2006) 43–51
12. Ge, Y., Barrett, C.: Proof translation and SMT-LIB benchmark certification: A preliminary report. In: 6th International Workshop on Satisfiability Modulo Theories (SMT '08). (2008)
13. Fontaine, P., Marion, J.Y., Merz, S., Nieto, L.P., Tiu, A.: Expressiveness + automation + soundness: Towards combining SMT solvers and interactive proof assistants. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS '06). Volume 3920 of Lect. Notes in Comp. Sci., Springer (2006) 167–181
14. Hurlin, C., Chaieb, A., Fontaine, P., Merz, S., Weber, T.: Practical proof reconstruction for first-order logic and set-theoretical constructions. In: Proceedings of the Isabelle Workshop 2007, Bremen, Germany (July 2007) 2–13
15. Böhme, S.: Proof reconstruction for Z3 in Isabelle/HOL. In: 7th International Workshop on Satisfiability Modulo Theories (SMT '09). (2009)
16. Weber, T., Amjad, H.: Efficiently checking propositional refutations in HOL theorem provers. J. Applied Logic **7**(1) (March 2009) 26–40
17. Hurd, J.: First-order proof tactics in higher-order logic theorem provers. In: Design and Application of Strategies/Tactics in Higher Order Logics (STRATA '03). Number NASA/CP-2003-212448 in NASA Technical Reports (2003) 56–68
18. Hurd, J.: Metis performance benchmarks (Retrieved January 21, 2010) http://www.gilith.com/software/metis/performance.html.
19. HOL88 contributors: HOL88 source code (Retrieved January 21, 2010) http://www.ftp.cl.cam.ac.uk/ftp/hvg/hol88/holsys.tar.gz.
20. Barrett, C., Deters, M., Oliveras, A., Stump, A.: 5th Annual Satisfiability Modulo Theories Competition (SMT-COMP '09) (2009) http://www.smtcomp.org/2009/.
21. Norrish, M.: Complete integer decision procedures as derived rules in HOL. In: Theorem Proving in Higher Order Logics (TPHOLs 2003). Volume 2758 of Lect. Notes in Comp. Sci., Springer (2003) 71–86
22. Dutertre, B., de Moura, L.M.: A fast linear-arithmetic solver for DPLL(T). In: Conference on Computer Aided Verification (CAV). Volume 4144 of Lect. Notes in Comp. Sci., Springer (2006) 81–94
23. Wenzel, M.: Parallel proof checking in Isabelle/Isar. In: ACM SIGSAM 2009 International Workshop on Programming Languages for Mechanized Mathematics Systems. (2009)
24. Amjad, H.: Data compression for proof replay. J. Automated Reasoning **41**(3–4) (2008) 193–218