Reconstruction of Z3's Bit-Vector Proofs in HOL4 and Isabelle/HOL

Sascha Böhme[1], Anthony C. J. Fox[2], Thomas Sewell[3], Tjark Weber[2]

[1] Fakultät für Informatik, TU München
boehmes@in.tum.de
[2] Computer Laboratory, University of Cambridge
{acjf3,tw333}@cam.ac.uk
[3] National ICT Australia
thomas.sewell@nicta.com.au

Abstract. The Satisfiability Modulo Theories (SMT) solver Z3 can generate proofs of unsatisfiability. We present independent reconstruction of unsatisfiability proofs for bit-vector theories in the theorem provers HOL4 and Isabelle/HOL. Our work shows that LCF-style proof reconstruction for the theory of fixed-size bit-vectors, although difficult because Z3's proofs provide limited detail, is often possible. We thereby obtain high correctness assurances for Z3's results, and increase the degree of proof automation for bit-vector problems in HOL4 and Isabelle/HOL.

1 Introduction

Interactive theorem provers, such as Isabelle/HOL [30] and HOL4 [21], have become powerful and trusted tools in formal verification. They typically provide rich specification logics that are suited to modelling the behaviour of complex systems. Deep theorems can be proved through user guidance. However, without the appropriate tool support, proving even simple theorems can be a tedious task when using interactive provers. Despite the merits of user guidance in proving theorems, there is a clear need for increased proof automation in interactive theorem provers.

In recent years, automated theorem provers have emerged for combinations of first-order logic with various background theories, e.g., linear arithmetic, arrays and bit-vectors. An overview of decision procedures for these domains can be found in [25]. These automated 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 [11,7]. Interactive theorem provers can greatly benefit from the reasoning power of SMT solvers: proof obligations that are SMT formulas can be passed to the automated prover, which will solve them without further human guidance [5].

This paper focuses on the theory of bit-vectors. This is an important theory, since bit-vector problems often occur during hardware and software verification, e.g., arising from loop invariants, ranking functions, and from code/circuits that

involve machine arithmetic. Isabelle/HOL and HOL4 have internal decision procedures for solving bit-vector problems, however, their capabilities are exceeded by those of SMT solvers such as Z3 (see [34]), which is a state-of-the-art SMT solver developed by Microsoft Research, see [29]. However, there is a critical distinction in the design philosophies of these provers: interactive provers are highly conservative, placing proof soundness above efficiency/coverage, whereas SMT solvers are generally more liberal and place high emphasis upon performance. Almost every SMT solver is known to contain bugs [10]. When integrated naively, the SMT solver (and the integration) becomes 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.

This soundness problem can be solved by requiring the SMT solver to produce proofs (of unsatisfiability), and reconstructing these proofs in the interactive prover. In this paper, we present independent reconstruction of unsatisfiability proofs for bit-vector theories generated by Z3 in Isabelle/HOL and HOL4. LCF-style [20] theorem provers implement a relatively small trusted kernel (see Sect. 3), which provides a fixed set of simple inference rules. In contrast, Z3 uses a number of powerful inference rules in its proofs (see Sect. 4) and this makes proof reconstruction challenging. In this paper, we extend a previous implementation of proof reconstruction for Z3 [8] to the theory of fixed-size bit-vectors (as defined in the FIXED_SIZE_BITVECTORS theory of SMT-LIB [2]).

The motivation for our work is twofold. First, we increase proof automation in HOL4 and Isabelle/HOL by using Z3 as an automated prover back-end. Second, we obtain a high degree of confidence in Z3's results. Due to the LCF-style architecture of HOL4 and Isabelle/HOL, the trusted code base consists only of their relatively small inference kernels. In particular, there is no need to trust our (much more complex) proof checker. Any error in a proof will be uncovered during reconstruction. Thus our checker can be used to identify bugs in Z3, and to certify the status of unsatisfiable SMT-LIB benchmarks.

We describe our implementation in detail in Sect. 6. Evaluation is performed on a large number of SMT-LIB benchmarks from the QF_AUFBV, QF_BV, and QF_UFBV logics (see Sect. 7). Section 8 concludes.

2 Related Work

SMT solvers have been an active research topic for the past few years, and an integration with interactive theorem provers has been pursued by a number of researchers.

In oracle style integrations, see [14,31], the client interactive theorem prover simply trusts the SMT solver's results. While this allows for a fast and relatively simple integration, a bug in the SMT solver (or in the integration) could lead to inconsistent theorems in the interactive prover. Closer to our work are integrations that perform proof reconstruction.

McLaughlin et al. [26] 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 [19] present the continuation of that work for CVC3 [3], 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.

Conchon et al. [12] integrated their prover Ergo with the Coq [4] interactive theorem prover. Unlike most SMT solvers, Ergo supports polymorphic first-order logic. Proof reconstruction, however, is restricted to congruence closure and linear arithmetic.

Fontaine et al. [15] describe an integration of the SMT solver haRVey with Isabelle/HOL [30]. Their work is restricted to quantifier-free first-order logic with equality and uninterpreted functions. Hurlin et al. [24] extend this approach to quantified formulas. Background theories (e.g., linear arithmetic, arrays) are not supported.

At SMT'09, Böhme [6] presented proof reconstruction for Z3 in Isabelle/HOL. Böhme and Weber [8] recently extended this to HOL4, improving both reconstruction speed and completeness (i.e., correct coverage of Z3's inference rules). Their highly optimized implementation supports uninterpreted functions with equality, quantifiers, arrays, linear integer and real arithmetic.

Common to the above approaches is their lack of support for bit-vector operations. To our knowledge, this paper is the first to address LCF-style proof reconstruction for the background theory of fixed-size bit-vectors.

3 LCF-Style Theorem Proving

The term LCF-style [20] 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 LCF-style systems that we consider here, HOL4 and Isabelle/HOL, are popular theorem provers for polymorphic higher-order logic (HOL) [21], based on the simply-typed λ-calculus. Isabelle's type system is more sophisticated than HOL4's [22], but we do not require any of the advanced features for this work.

On top of their LCF-style inference kernels, HOL4 and Isabelle/HOL 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 and real algebra. We particularly use a recent decision procedure for bit-vectors based on bit-blasting (see Sect. 5).

The implementation language of HOL4 and Isabelle/HOL is Standard ML [27]. To benefit from the LCF-style design of these provers and the reasoning tools built on top of their inference kernels, we must use this language to implement proof reconstruction.

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

4 Z3: Language and Proof Terms

Succinct descriptions of Z3's language and proof terms have been given in [28,6]. We briefly review the key features, expanding on previous descriptions where necessary.

Z3's language is many-sorted first-order logic, based on the SMT-LIB language [2]. 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 ($\forall$, $\exists$), array operations select and store, the distinct predicate and equality. Proof reconstruction for these has been described before [8].

The present paper focuses on the theory of fixed-width bit-vectors. This adds basic sorts BitVec m for every $m > 0$, bit-vector constants like #b0, and various operations on bit-vectors: concatenation (concat), sub-vector extraction (extract), bit-wise logical operations (bvnot, bvand, bvor), arithmetic operations (bvneg, bvadd, bvmul, bvudiv, bvurem), shift operations (bvshl, bvlshr), unsigned comparison (bvult), and several derived operations. The theory is described in full detail in the `Fixed_Size_BitVectors` and `QF_BV` files[4] of SMT-LIB.

Z3's proof terms encode natural deduction proofs. The deductive system used by Z3 contains 16 axioms and inference rules.[5] These range from simple rules like **mp** (modus ponens) to rules that abbreviate complex reasoning steps. To adapt our previous implementations of proof reconstruction [8] to the theory of bit-vectors, we need to look at two rules in particular: **rewrite** for equality reasoning involving interpreted functions, and **th-lemma-bv** for arbitrary lemmas specific to the theory of bit-vectors. We discuss these in more detail in Sect. 6.

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

[4] Available at `http://combination.cs.uiowa.edu/smtlib/logics/QF_BV.smt2`.

[5] Another 18 rules are described in the Z3 documentation, but were not exercised in any of the benchmarks used for evaluation (see Sect. 7). Although we omit these rules from our presentation, our implementations can handle them as well [8].

In 2010, version 2 of the SMT-LIB language was introduced [2]. It is worth noting that Z3's concrete syntax for proofs of SMT-LIB 2 benchmarks is vastly different from its syntax for proofs of SMT-LIB 1.2 benchmarks. While the SMT-LIB 1.2 proof syntax was line-based (with one inference or term definition per line), the SMT-LIB 2 proof syntax of Z3 resembles the SMT-LIB 2 benchmark syntax and represents proofs as S-expressions.

We have written a recursive-descent parser for (a large subset of) the SMT-LIB 2 language in Standard ML. The parser translates formulas in SMT-LIB syntax into formulas in higher-order logic.[6] We have also written a Standard ML parser for the new Z3 proof format that utilizes our SMT-LIB 2 benchmark parser internally.

At present, Z3's new proof syntax still contains a few quirks and incompatibilities with SMT-LIB 2 (e.g., different names for certain constants, missing parentheses around argument lists). We hope that these syntax issues, which currently complicate proof parsing, will be addressed in a future version of Z3.

5 Bit-Vectors in Higher-Order Logic

Isabelle/HOL's latest theory of machine words (bit-vectors) was developed by Dawson [13], and is based on constructing an isomorphic type for the finite set $\{0, \ldots, 2^n - 1\}$.

As of 2005 HOL4's theory of bit-vectors utilises Harrison's technique for modelling the n-dimensional Euclidean space in HOL Light, see [23].[7] Harrison's approach is based on considering the function space $N \to A$, where N is constrained to be finite. For bit-vectors we consider a Boolean co-domain, i.e., $A = \mathbb{B}$. Isabelle/HOL has a more advanced type system than HOL4 and HOL Light; however, they all support parametric polymorphism and this is sufficient to provide a workable theory of bit-vectors.

To aid clarity, this section will simply focus on HOL4's bit-vector library. We give an overview of this library from the end-user perspective.

Bit-vectors in HOL4 are represented by the type $\alpha\,\mathsf{word}$. For example, 8-bit words have type $\mathsf{8\,word}$, which can also be written as $\mathsf{word8}$. The numeric type $\mathsf{8}$ has exactly eight members, which gives us the required word length. The function $\mathsf{dimindex}$ returns the word length, and $\mathsf{dimword}$ returns the number of elements, e.g., $\mathsf{dimindex}(:\mathsf{8}) = 8$ and $\mathsf{dimword}(:\mathsf{8}) = 256$. The bit-vector library supports a broad collection of standard operations, including, but not limited to:

1. *Signed and unsigned arithmetic operations.* Examples include bit-vector negation, addition, subtraction, multiplication and less-than.
2. *Bitwise/logical operations.* Examples include complement, bitwise-and (&&), bitwise-or (!!), as well as various shifts and rotations.

[6] Our parser identified numerous conformance issues in SMT-LIB 2 benchmarks. We have reported these to the benchmark maintainers.

[7] Prior to this a quotient type construction was used in HOL4.

3. *Signed and unsigned casting maps.* Examples include an embedding from naturals to words (n2w), zero-extension (w2w), sign-extension, word extraction ($><$), and word concatenation.

Bit-vector literals are denoted with a 'w' suffix. Standard number bases are supported, for example 0xAw, 0b1010w and 10w all denote a word literal with value ten.

Importantly, all SMT-LIB bit-vector operations have corresponding definitions in the bit-vector libraries of HOL4 and Isabelle/HOL.

The word library provides a number of simplification sets (*simp-sets*) (which control the behaviour of the simplifier), *conversions* (which construct an equivalence theorem for an input term) and semi-decision procedures. These provide the building blocks for carrying out interactive proofs and for developing further (more powerful) tools. Most simp-sets primarily consist of collections of conditional rewrite rules, but they may also apply conversions, decision procedures and implement other functionality. The main bit-vector simp-set carries out basic algebraic simplification, such as associative-commutative (AC) rewriting, covering the arithmetic and bitwise operations.

Two main bit-vector semi-decision procedures are available:

- `WORD_DECIDE`. This procedure has somewhat limited coverage but it does provide a fairly quick means to discharge basic bit-vector problems. There are three stages: algebraic simplification, bit expansion for non-arithmetic operations, and finally propositional and bounds-based reasoning. By default the final stage makes use of HOL4's standard natural number decision procedure, which enables it to prove, for instance,

$$\vdash\ \forall a : \mathsf{word8}.\ a >_+ 253\mathrm{w} \implies (a = 254\mathrm{w}) \vee (a = 255\mathrm{w})$$

  by utilizing the constraint $0 \leq n < 256$, where n is the numeric value of the bit-vector a.
- `BBLAST`. This is a semi-decision procedure that offers better coverage than `WORD_DECIDE`. However, it is still essentially propositional in nature, covering pure bit-vector problems of the form: $\forall\, w_1 \ldots w_n.\ P(w_1, \ldots, w_n)$ or $\exists\, w_1 \ldots w_n.\ P(w_1, \ldots, w_n)$. As before, the procedure starts by applying algebraic simplifications, but this time the second stage also carries out bit-expansion for addition (which in turn subsumes subtraction and the word orderings). The final stage involves calling a SAT solver. One advantage of this approach is that counterexamples can be provided when goals are invalid. The main limitations are that the procedure does not handle nested quantification (or, more generally, first-order reasoning), and goals that require non-trivial reasoning about multiplication/division. `BBLAST` is described in more detail in [17].

When carrying out interactive proofs, human guidance and additional tools (such as first-order provers) provide the means to tackle goals that are more complex than these individual semi-decision procedures can handle on their own.

6 Proof Reconstruction

Proof reconstruction for the theory of bit-vectors extends our previous work on LCF-style proof reconstruction for Z3 [8]. The general setup remains the same. We translate (negated) proof goals from HOL4 or Isabelle/HOL into SMT-LIB 2 syntax and apply Z3. If Z3 determines the negated goal to be unsatisfiable, we then parse its proof (using our parser for the SMT-LIB 2 language and its extension to the Z3 proof language, see Sect. 4) to obtain the encoded information as a value in Standard ML.[8]

Proofs are represented by a balanced tree (with lookup in $O(\log n)$ time) that maps node identifiers to proof nodes. Proof nodes are given by a disjoint union. Initially, each node contains the information that is recorded explicitly in the Z3 proof: the axiom or inference rule used at the node, the node identifiers of premises, and the rule's conclusion. Once the inference step has been checked in HOL4 or Isabelle/HOL, this information is replaced by a corresponding theorem.

To reconstruct a proof, we start at its designated root node and perform a (depth-first) post-order traversal of the proof DAG. Each node's premises are derived as theorems in HOL4 or Isabelle/HOL. Then these theorems are used to derive the node's conclusion. Ultimately, the root node's inference step, which derives $\bot$ from the root's premises, is reconstructed. We obtain a theorem that proves $\bot$ from the given assumptions, i.e., that shows unsatisfiability of the negated HOL4 or Isabelle/HOL proof goal.

Out of the 16 axioms and inference rules used by Z3, 14 perform propositional and first-order reasoning. These rules are independent of any background theory. Proof reconstruction for them has been described in [8]. It is intricate, but does not require adaptation for the theory of bit-vectors.

Only two rules—incidentally, the most complicated ones in Z3's deductive system—involve bit-vector reasoning: **rewrite** and **th-lemma-bv**. The former is used for equality reasoning about interpreted functions (including not just bit-vector operations, but also logical operators and other interpreted functions). The latter is used to derive arbitrary theory-specific lemmas. It is this rather vague specification of their semantics—and the fact that neither rule provides additional justifications, e.g., trace information—that makes proof reconstruction challenging. We now discuss our implementations of proof reconstruction for **rewrite** and **th-lemma-bv** in detail.

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 specific instance from first principles. By studying the actual usage of **rewrite** in Z3's proofs, we identified about 20 useful schematic theorems for bit-vector reasoning.[9] Examples include associativity and commutativity of

[8] It is important that this round trip from higher-order logic to SMT-LIB 2 and back constitutes an identity transformation. Otherwise, proof reconstruction would derive the wrong formula.

[9] This is in addition to over 230 schematic theorems for propositional and arithmetic reasoning identified earlier [8].

bit-wise operations, e.g., $(x \mathbin{\&\&} y) \mathbin{\&\&} z = x \mathbin{\&\&} (y \mathbin{\&\&} z)$, $x \mathbin{\&\&} y = y \mathbin{\&\&} x$, neutrality of 0w for bit-wise disjunction, $0\mathsf{w} \mathbin{!!} x = x$, and simplification rules for bit extraction, e.g., $(7 \gg< 0)(x : \mathsf{word8}) = x$. We store all schematic theorems in a term net to allow faster search for a match.

Schematic theorems are, in fact, our main workhorse for **rewrite**. On the benchmarks used for evaluation (see Sect. 7), **rewrite** is invoked more than 1 million times. 92.5 % of the proof obligations presented to **rewrite** are solved by instantiation of a schematic theorem.

The theory of fixed-size bit-vectors, unlike other background theories considered in earlier work [8], requires conditional schematic theorems. For instance, converting a bit-vector literal x from type $\alpha\,\mathsf{word}$ to $\beta\,\mathsf{word}$ yields essentially the same literal, provided the literal could be represented in type $\alpha\,\mathsf{word}$ in the first place:

$$\vdash x < \mathsf{dimword}(:\alpha) \implies \mathsf{w2w}\,(\mathsf{n2w}\,x : \alpha\,\mathsf{word}) = (\mathsf{n2w}\,x : \beta\,\mathsf{word}).$$

We prove these conditions by recursive instantiation of (unconditional) schematic theorems, e.g., $\vdash 1 < \mathsf{dimword}(:\alpha)$, and in many cases by simplification: terms such as $\mathsf{dimindex}(:\alpha)$ and $\mathsf{dimword}(:\alpha)$ can be evaluated for numeric types α.

We also use schematic theorems in the implementation of **th-lemma-bv**, but there the impact is much smaller. **th-lemma-bv** is called over 50 million times on the benchmarks used for evaluation, but less than 0.1% of its proof obligations are solved by instantiation. We could increase this percentage by adding more schematic theorems (at the cost of increased memory usage and start-up time), but essentially the lemmas proved by **th-lemma-bv** are more diverse and benchmark dependent than those proved by **rewrite**. For **th-lemma-bv**, schematic theorems are mostly useful to prove corner cases not covered by one of the automated decision procedures discussed below.

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 Isabelle/HOL and HOL4 than in Z3. We exploit this by storing theorems that **rewrite** or **th-lemma-bv** prove via computationally expensive bit-vector decision procedures (see below) 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 on a proof obligation, we attempt to retrieve a matching theorem from the net. This succeeds for 4.5 % of all proof obligations presented to **rewrite**, and for an impressive 99.3 % of proof obligations presented to **th-lemma-bv**. Here we see that schematic theorems and theorem memoization largely complement each other. For **rewrite**, proof obligations that occur frequently are often available as schematic theorems already. For **th-lemma-bv**, however, few proof obligations seemed sufficiently generic to be included as schematic theorems, but individual benchmarks still prove instances of the same proof obligation many times. Therefore, theorem memoization shines.

Strategy selection. Schematic theorems can only prove formulas that have a specific (anticipated) structure. Theorem memoization is successful only when a matching lemma was added to the term net earlier. Initially, bit-vector proof obligations must be proved by other means.

We rely on HOL4's and Isabelle/HOL's existing automation for bit-vector logic (see Sect. 5). Both provers provide a toolbox of semi-decision procedures for bit-vector proof obligations. Further procedures may be programmed in Standard ML. This leads to an unbounded number of proof procedures, which will typically succeed on different (not necessarily disjoint) sets of bit-vector formulas, and exhibit vastly different timing behaviours both in success and failure cases. For instance, proving $x+y = y+x$ by rewriting is trivial if commutativity of bit-vector addition is available as a rewrite rule. Proving the same theorem strictly by bit-blasting alone is possible, but may take significantly longer if the number of bits in x and y is large.

Our current implementations use only four different proof procedures for **rewrite** and **th-lemma-bv**. For **rewrite**, we first try a simplification-based approach, expressing many word operations in terms of !! (disjunction), << (left shift) and $\gg$ (word extract), and then unfolding the definition of bit-wise operators, i.e., considering each bit position separately. This is followed by the application of arithmetic rewrites, an evaluation mechanism for ground arithmetic terms, and a decision procedure for linear arithmetic. This powerful approach solves 98 % of bit-vector goals presented to **rewrite** that are not handled by schematic theorems or memoization. The remaining 2 % are solved by a decision procedure that converts word arithmetic expressions into a canonical form. In particular, we need to fix the sign of word equalities: for instance, $-x = y \iff x + y = 0\text{w}$.

For **th-lemma-bv**, we first use simplification with a relatively large set of standard rewrite rules for (arithmetic and logical) word expressions, including unfolding of bit-wise operators. Over 99 % of goals presented to **th-lemma-bv** are thereby reduced to propositional tautologies. The remaining goals are solved by bit-blasting.

This choice of proof procedures is the result of careful optimization. Starting from a set of about 10 different proof procedures, applied in a hand-chosen order, we independently optimized our implementations of **rewrite** and **th-lemma-bv** using a greedy approach: based on detailed profiling data (see Sect. 7), we modified the order in which these proof procedures were applied to try those that had the shortest average runtime (per solved goal) first. We iterated this process until the number of timeouts was not reduced any further, and tried several different initial orders to avoid local optima. Each iteration required several days of CPU time.

Clearly, more sophisticated approaches than this variant of random-restart hill climbing could be employed. If wall time is considered to be more important than CPU time, we could simply apply a number of proof procedures in parallel, taking advantage of modern multi-core architectures. We could also devise a heuristic hardness model that analyses each proof goal to predict the proof pro-

cedure that is most likely to find a proof quickly. The SATzilla solver successfully uses a similar approach to decide propositional satisfiability [35].

However, one should keep in mind that this optimization problem is ultimately caused by a lack of detail in Z3's proofs for bit-vector theorems. Rather than devoting large amounts of resources to tuning the HOL4 and Isabelle/HOL implementations of bit-blasting and other bit-vector decision procedures, it would seem more worthwhile to modify Z3 itself to print more detailed certificates for the theory of bit-vectors.

7 Experimental Results

Evaluation was performed on SMT-LIB [2] problems comprising quantifier-free (QF) first-order formulas over (combinations of) the theories of arrays (A), equality and uninterpreted functions (UF), and bit-vectors (BV). SMT-LIB logic names are formed by concatenation of the theory abbreviations given in parentheses. We evaluated our implementations on all unsatisfiable bit-vector benchmarks in SMT-LIB.[10] At the time of writing, this comprises 4974 benchmarks from three logics: QF_AUFBV, QF_BV, and QF_UFBV. These benchmarks originate from a variety of sources. They constitute a diverse and well-balanced problem set for evaluation.

We obtained all figures[11] on a 64-bit Linux system with an Intel Core i7 X920 processor, running at 2 GHz. Measurements were conducted with Z3 2.19, the latest version of Z3 at the time of writing. As underlying ML environment, we used Poly/ML 5.4.1 for both Isabelle/HOL and HOL4. For comparability with earlier work [6,8], 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).

Beyond measuring success rates and runtimes of proof reconstruction, we also measured the performance of HOL4 bit-blasting for comparison, and we provide profiling data to give a deeper insight into our results. For space reasons, we do not show Isabelle/HOL results in detail, but they are roughly similar to the HOL4 results discussed below.

7.1 Proof Generation with Z3

Table 1 shows the results obtained from applying Z3 2.19 to all unsatisfiable bit-vector benchmarks in SMT-LIB. For every SMT-LIB logic, we show the number of benchmarks and the average benchmark size. We then measured the number of errors (e.g., segmentation faults), timeouts, and proofs generated by Z3. We also show the average solving time (excluding errors and timeouts), and the average size of generated proofs. We invoked Z3 with option `PROOF_MODE=2`, which enables proof generation.

[10] These benchmarks were obtained from `http://smtexec.org/exec/smtlib2_benchmarks.php` on June 13, 2011, using the query `logic~BV & status=unsat`.

[11] Our data is available at `http://www.cl.cam.ac.uk/~tw333/bit-vectors/`.

Proofs are typically much larger than the original SMT-LIB benchmark—almost 53 times as large on average. The total size of generated proofs is 34.9 GB, and the total CPU time for Z3 on all benchmarks (including errors and timeouts) is around 29.5 hours.

Logic	Benchmarks		Errors		Timeouts		Proofs			
	#	Size (avg)	#	Ratio	#	Ratio	#	Ratio	Time (avg)	Size (avg)
QF_AUFBV	3566	93 kB	0	0.0 %	118	3.3 %	3448	96.7 %	0.6 s	1.2 MB
QF_BV	1377	322 kB	12	0.9 %	630	45.8 %	735	53.4 %	17.3 s	41.1 MB
QF_UFBV	31	343 kB	0	0.0 %	15	48.4 %	16	51.6 %	37.1 s	29.2 MB
Total	4974	158 kB	12	0.2 %	763	15.3 %	4202	84.5 %	4.2 s	8.3 MB

Table 1. Experimental results (Z3 2.19) for selected SMT-LIB logics

7.2 Bit-Blasting in HOL4

Next, we show the results of bit-blasting in HOL4 for comparison. We used our SMT-LIB 2 parser (see Sect. 4) to translate benchmarks into higher-order logic. We then applied HOL4's `BBLAST` tactic to the same set of SMT-LIB benchmarks previously presented to Z3 (i.e., the number and size of benchmarks in Tab. 2 is the same as before). We used a timeout of five minutes per benchmark. Similar to before, we show the number of errors, timeouts, and proofs found by `BBLAST`, as well as the average solving time (excluding errors and timeouts). Every inference performed by `BBLAST` is checked by HOL4's inference kernel, but no persistent proof objects are generated. Therefore, there is no column for proof size in Tab. 2.

Logic	Benchmarks		Errors		Timeouts		Proofs		
	#	Size (avg)	#	Ratio	#	Ratio	#	Ratio	Time (avg)
QF_AUFBV	3566	93 kB	1089	30.5 %	474	13.3 %	2003	56.2 %	26.2 s
QF_BV	1377	322 kB	745	54.1 %	504	36.6 %	128	9.3 %	56.2 s
QF_UFBV	31	343 kB	31	100.0 %	0	0.0 %	0	0.0 %	—
Total	4974	158 kB	1865	37.5 %	978	19.7 %	2131	42.8 %	28.0 s

Table 2. Experimental results (`BBLAST`) for selected SMT-LIB logics

Errors mostly indicate that `BBLAST` gave up on the benchmark. To prove unsatisfiability, many benchmarks require combinations of bit-blasting and equality reasoning (e.g., congruence closure), which `BBLAST` is not capable of, or reasoning about specific bit-vector operations in ways not supported by `BBLAST`. Our results, therefore, show that Z3 is not only much faster than `BBLAST`, but also that it can solve a wider range of problems.

7.3 Proof Reconstruction in HOL4

We checked all proofs generated by Z3 in the HOL4 theorem prover, using a timeout of five minutes per benchmark. Table 3 shows our results. We present the number of errors, timeouts (including out-of-memory results), and successfully checked proofs, along with average HOL4 runtime for the latter. We also show total HOL4 runtime (including errors and timeouts) for each logic.

Errors are caused by unsound inferences in proofs, and in many cases by bugs in Z3's proof pretty-printer,[12] but also by shortcomings in our implementation of proof reconstruction, which fails on some corner cases.

Logic	Proofs	Errors		Timeouts		Success			Overall time (approx)
	#	#	Ratio	#	Ratio	#	Ratio	Time (avg)	
QF_AUFBV	3448	587	17.0 %	54	1.6 %	2807	81.4 %	1.4 s	5.4 hrs
QF_BV	735	96	13.1 %	356	48.4 %	283	38.5 %	18.8 s	31.0 hrs
QF_UFBV	16	0	0.0 %	16	100.0 %	0	0.0 %	—	1.2 hrs
Total	4202	683	16.3 %	426	10.1 %	3090	73.5 %	2.6 s	37.6 hrs

Table 3. Experimental results (HOL4 proof reconstruction) for Z3's proofs

Although HOL4 achieves an overall success rate of 73.5 %, we see that this rate varies significantly with the SMT-LIB logic. QF_AUFBV contains a large number of relatively easy benchmarks, which can be solved quickly by Z3, have small proofs, and consequently can (in most cases) be checked successfully in HOL4. Table 1 indicates that QF_BV and QF_UFBV contain significantly harder problems. This is reflected by the performance of HOL4 on these logics, which can check a mere 38.5 % of benchmarks in QF_BV within the given time limit, and times out for all 16 proofs in QF_UFBV. However, proof reconstruction is more than an order of magnitude faster than `BBLAST`, and can solve 1.5 times as many SMT-LIB problems. Proof generation with Z3 is typically one to two orders of magnitude faster than proof reconstruction in HOL4.

7.4 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 1 to 3 show bar graphs that indicate the percentage shares of total runtime (dark bars) for **rewrite**, **th-lemma-bv**, and Z3's other proof rules. Additionally, time spent on parsing proof files is shown as well (see Tab. 1 for average proof sizes). We contrast each proof rule's relative runtime with the mean frequency of that rule (light bars).

[12] We have notified the Z3 authors of the problems that we found. Additionally, we corrected some obvious syntax errors in proofs, e.g., unbalanced parentheses.

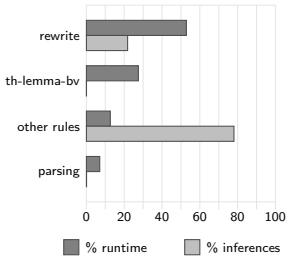

Fig. 1. QF_AUFBV

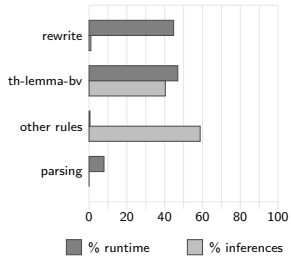

Fig. 2. QF_BV

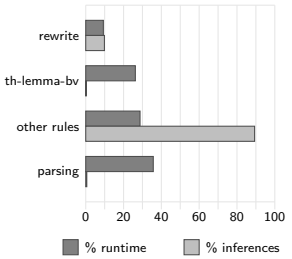

Fig. 3. QF_UFBV

We see that despite extensive optimization, proof reconstruction times are still dominated by **rewrite** and **th-lemma-bv**. Although less than 1% of all inferences in QF_AUFBV and QF_UFBV are applications of **th-lemma-bv**, checking these consumes over 26% of runtime. Even more extremely, **rewrite** in QF_BV accounts for less than 1% of inferences, but almost 45% of proof reconstruction time. In contrast, all other rules combined constitute the majority of proof inferences (between 59% for QF_BV and 89% for QF_UFBV), but they can be checked much more quickly: in 29% (for QF_UFBV) or less of total runtime.

Proof parsing takes less than 8% of total runtime for QF_AUFBV and QF_BV, but 36% for QF_UFBV. It times out on the largest proofs. Proofs for QF_BV are larger than proofs for QF_UFBV on average (see Tab. 1), but QF_BV contains many small proofs that can be parsed relatively quickly. The variation in proof size is much smaller for QF_UFBV. Median proof sizes are 3.7 MB for QF_BV and 22.5 MB for QF_UFBV, respectively.

8 Conclusions

Bit-vectors play an important role in hardware and software verification. They naturally show up in the verification of, e.g., 32- and 64-bit architectures and machine data types [18]. In this paper, we have extended a previous implementation of LCF-style proof reconstruction for Z3 [8] to the theory of fixed-size bit-vectors. To our knowledge, this paper is the first to consider independent checking of SMT solver proofs for bit-vector theories.

Even though Z3's proofs provide little detail about theory-specific reasoning, our experimental results (Sect. 7) show that LCF-style proof reconstruction for the theory of fixed-size bit-vectors is often possible. We have achieved an overall success rate of 73.5% on SMT-LIB benchmarks. We thereby obtain high correctness assurances for Z3's results. Checking Z3's proofs also significantly increases the degree of proof automation for bit-vector problems in HOL4 and Isabelle/HOL. Proof reconstruction is more powerful in scope and performance

than built-in decision procedures, such as `BBLAST`, previously offered by these provers. Our implementations are freely available[13] and already in use [5].

Z3's proof rules **rewrite** and **th-lemma-bv** seem overly complex. Despite substantial optimization efforts, they still dominate runtime in our implementations. Proof reconstruction currently needs to duplicate proof search that Z3 has performed before, to re-obtain essential information that was computed by Z3 internally, but not included in the proof.

More work could be done on the checker side: for instance, we could attempt to re-implement Z3's decision procedure for bit-vectors [34] on top of HOL4's or Isabelle's LCF-style inference kernel. However, instead of duplicating Z3's highly tuned decision procedures in our proof checker, it would seem more sensible to modify Z3's proof format to include all relevant information [9].

Unfortunately, we could not do this ourselves because Z3 is closed source. We again [8] 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-bv** with additional easily-checkable certificates or trace information guiding refutations to avoid invocations of expensive decision procedures (e.g., bit-blasting) in the checker.

Based on previous experience [32] we are confident that the techniques presented in this paper can be used to achieve similar performance for bit-vector reasoning in other LCF-style theorem provers for higher-order logic.

Future work should aim for improved reconstruction coverage (i.e., fewer errors) and improved performance, possibly after Z3's proof format has been modified as suggested above. We also intend to evaluate proof reconstruction for typical goals of Isabelle/HOL or HOL4; to implement parallel proof reconstruction [33], by checking independent paths in the proof DAG concurrently; and to investigate proof compression [1,16] for SMT proofs.

Acknowledgments

This research was partially funded by EPSRC grant EP/F067909/1. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. The authors are grateful to Nikolaj Bjørner and Leonardo de Moura for their help with Z3.

References

1. Amjad, H.: Data compression for proof replay. Journal of Automated Reasoning 41(3–4), 193–218 (2008)
2. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB Standard: Version 2.0. In: Gupta, A., Kroening, D. (eds.) Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, England) (2010)

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

3. Barrett, C., Tinelli, C.: CVC3. In: Damm, W., Hermanns, H. (eds.) Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4590, pp. 298–302. Springer (2007)
4. Bertot, Y.: A short presentation of Coq. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5170, pp. 12–16. Springer (2008)
5. Blanchette, J.C., Böhme, S., Paulson, L.C.: Extending Sledgehammer with SMT solvers. In: Automated Deduction (2011), to appear
6. Böhme, S.: Proof reconstruction for Z3 in Isabelle/HOL. In: 7th International Workshop on Satisfiability Modulo Theories (SMT '09) (2009)
7. Böhme, S., Moskal, M., Schulte, W., Wolff, B.: HOL-Boogie — An Interactive Prover-Backend for the Verifying C Compiler. Journal of Automated Reasoning 44(1–2), 111–114 (2010)
8. Böhme, S., Weber, T.: Fast LCF-style proof reconstruction for Z3. In: Kaufmann, M., Paulson, L.C. (eds.) Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6172, pp. 179–194. Springer (2010)
9. Böhme, S., Weber, T.: Designing proof formats: A user's perspective. In: First Workshop on Proof Exchange for Theorem Proving (2011), to appear
10. Brummayer, R., Biere, A.: Fuzzing and delta-debugging SMT solvers. In: 7th International Workshop on Satisfiability Modulo Theories (SMT '09) (2009)
11. Collavizza, H., Gordon, M.: Integration of theorem-proving and constraint programming for software verification. Tech. rep., Laboratoire d'Informatique, Signaux et Systèmes de Sophia-Antipolis (2008)
12. Conchon, S., Contejean, E., Kanig, J., Lescuyer, S.: Lightweight integration of the Ergo theorem prover inside a proof assistant. In: AFM '07: Proceedings of the Second Workshop on Automated Formal Methods. pp. 55–59. ACM Press (2007)
13. Dawson, J.: Isabelle theories for machine words. Electronic Notes in Theoretical Computer Science 250(1), 55–70 (2009), Proceedings of the Seventh International Workshop on Automated Verification of Critical Systems (AVoCS 2007)
14. Erkök, L., Matthews, J.: Using Yices as an automated solver in Isabelle/HOL. In: AFM '08: Proceedings of the Third Workshop on Automated Formal Methods. pp. 3–13. ACM Press (2008)
15. 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). Lecture Notes in Computer Science, vol. 3920, pp. 167–181. Springer (2006)
16. Fontaine, P., Merz, S., Paleo, B.W.: Compression of propositional resolution proofs via partial regularization. In: Automated Deduction (2011), to appear
17. Fox, A.: LCF-style bit-blasting in HOL4. In: Second International Conference on Interactive Theorem Proving. Lecture Notes in Computer Science, vol. 6898. Springer (2011), to appear
18. Fox, A.C.J., Gordon, M.J.C., Myreen, M.O.: Specification and verification of ARM hardware and software. In: Hardin, D.S. (ed.) Design and Verification of Microprocessor Systems for High-Assurance Applications, pp. 221–248. Springer (2010)
19. 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)

20. Gordon, M., Milner, R., Wadsworth, C.P.: Edinburgh LCF: A Mechanised Logic of Computation, Lecture Notes in Computer Science, vol. 78. Springer (1979)
21. Gordon, M.J.C., Pitts, A.M.: The HOL logic and system. In: Towards Verified Systems, Real-Time Safety Critical Systems Series, vol. 2, chap. 3, pp. 49–70. Elsevier (1994)
22. Haftmann, F., Wenzel, M.: Constructive type classes in Isabelle. In: Altenkirch, T., McBride, C. (eds.) Types for Proofs and Programs, International Workshop, TYPES 2006, Nottingham, UK, April 18-21, 2006, Revised Selected Papers. Lecture Notes in Computer Science, vol. 4502, pp. 160–174 (2007)
23. Harrison, J.: A HOL theory of Euclidean Space. In: Hurd, J., Melham, T.F. (eds.) Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford, UK, August 22-25, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3603, pp. 114–129. Springer (2005)
24. 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. pp. 2–13. Bremen, Germany (Jul 2007)
25. Kroening, D., Strichman, O.: Decision Procedures – An Algorithmic Point of View. Springer (2008)
26. 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), 43–51 (2006)
27. Milner, R., Tofte, M., Harper, R., MacQueen, D.: The Definition of Standard ML – Revised. MIT Press (1997)
28. 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. CEUR Workshop Proceedings, vol. 418. CEUR-WS.org (2008)
29. de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS '08). Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008)
30. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science, vol. 2283. Springer (2002)
31. Weber, T.: SMT solvers: New oracles for the HOL theorem prover. International Journal on Software Tools for Technology Transfer (2011), to appear
32. Weber, T., Amjad, H.: Efficiently checking propositional refutations in HOL theorem provers. Journal of Applied Logic 7(1), 26–40 (2009)
33. Wenzel, M.: Parallel proof checking in Isabelle/Isar. In: ACM SIGSAM 2009 International Workshop on Programming Languages for Mechanized Mathematics Systems (2009)
34. Wintersteiger, C.M., Hamadi, Y., de Moura, L.M.: Efficiently solving quantified bit-vector formulas. In: Bloem, R., Sharygina, N. (eds.) Proceedings of the 10th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2010, Lugano, Switzerland, October 20-23. pp. 239–246. IEEE (2010)
35. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: Portfolio-based algorithm selection for SAT. J. Artif. Intell. Res. (JAIR) 32, 565–606 (2008)