Validating QBF Invalidity in HOL4

Tjark Weber*

University of Cambridge
Computer Laboratory
tw333@cam.ac.uk

Abstract. The Quantified Boolean Formulae (QBF) solver Squolem can generate certificates of invalidity, based on Q-resolution. We present independent checking of these certificates in the HOL4 theorem prover. This enables HOL4 users to benefit from Squolem's automation for QBF problems, and provides high correctness assurances for Squolem's results. Detailed performance data shows that LCF-style certificate checking is feasible even for large QBF instances. Our work prompted improvements to HOL4's inference kernel.

1 Introduction

Deciding the validity of Quantified Boolean Formulae (QBF) is an extension of the well-known Boolean satisfiability problem (SAT). In addition to the usual connectives of propositional logic, QBF may contain universal and existential quantifiers over Boolean variables. As a simple example, consider the formula

$$\exists x \, \forall y \, \exists z. \; x \wedge (y \vee z) \wedge (y \vee \neg z). \tag{1}$$

QBF have applications in adversarial planning and formal verification [1,2,3]. They are also interesting from a theoretical viewpoint: QBF is the canonical PSPACE-complete problem [4]. Whether QBF is harder than SAT is an open problem, but it is widely believed that Boolean quantifiers allow to give exponentially more succinct encodings for certain problems than propositional logic alone.

QBF solvers automatically decide validity of such formulae. (For closed QBF, satisfiability is equivalent to validity, and unsatisfiability is equivalent to invalidity.) In addition, certain QBF solvers can produce certificates for their answers that can be checked independently [5]. Squolem is a state-of-the-art QBF solver that generates Q-resolution [6] based certificates for invalid formulae [7].

In this paper, we present independent checking of Squolem's certificates for invalid QBF in the HOL4 [8] theorem prover. HOL4 is a popular interactive theorem prover for higher-order logic [9]. It is based on a small LCF-style [10,11] kernel that provides an abstract data type of theorems, equipped with a fixed set of constructor functions (corresponding to the axiom schemata and inference rules of higher-order logic). Derived rules (such as Q-resolution) that are not

* This work was supported by the British EPSRC under grant EP/F067909/1.

provided by this kernel must be implemented by composing existing rules. This provides high correctness assurances: derived rules cannot produce inconsistent theorems, as long as the theorem data type itself is implemented correctly. On the other hand, it makes an efficient implementation of derived rules challenging.

The motivation for our work is twofold. First, interactive theorem provers like Coq [12], HOL4, Isabelle [13] and PVS [14] can greatly benefit from the reasoning power of automated tools. Consequently, researchers have on various occasions integrated automated first-order provers [15,16,17], SAT solvers [18], and more recently SMT solvers [19,20] with interactive provers. Our integration of a QBF solver with HOL4 fills a small, but not insignificant gap in this long line of research. It enables HOL4 users to benefit from Squolem's automation for QBF problems, and since the results are checked by HOL4's inference kernel, no trust needs to be put in the QBF solver.

Second, QBF solvers are complex software tools. Similar to state-of-the-art SAT solvers, they typically employ various heuristics and optimizations to achieve competitive performance [21,22]. Correctness is hard to establish, and different QBF solvers frequently disagree on the status of individual benchmarks. QBF-Eval competitions in previous years resolved disagreements by majority vote [23]. This rather unsatisfactory approach confirms the importance of QBF benchmark certification. HOL4's inference kernel has been carefully scrutinized by dozens of researchers for over two decades. By using HOL4 as an independent checker, we obtain high correctness assurances for Squolem's results.

We review related work in Section 2. Relevant background material is introduced in Section 3. Section 4 presents our main contribution: an approach to QBF certificate checking, and in particular an efficient implementation of Q-resolution, in HOL4. Experimental results are given in Section 5. Section 6 concludes.

2 Related Work

To our knowledge, this paper is the first to consider the integration of a QBF solver with an interactive theorem prover. Related work can be classified into two distinct areas: (i) the integration of automated solvers with LCF-style theorem provers, and (ii) certificate checking for QBF solvers.

Integrating automated solvers with interactive theorem provers, just like our work, is typically motivated by a need for increased automation in the interactive system. First attempts were already made in the early 90s [15]. Since then, a long line of related research has developed. Integrations have been proposed for first-order provers [16,17], for model checkers [24], computer algebra systems [25,26], SAT solvers [18], and more recently for SMT solvers [19,20], to name just a few. The approach presented in this paper especially draws on ideas from [18] for efficient LCF-style propositional resolution (see Section 4).

Q-resolution based certificates for Squolem were proposed by Jussila et al. [7]. Other proof formats for QBF solvers have been suggested: e.g., BDD-based traces for sKizzo [27], and inference logs for yQuaffle [28]. Narizzano et al. [5] give an

overview and compare different certificate formats. Squolem's certificates show competitive performance, and they are relatively simple. Thus, implementing a checker is probably easier than for the other formats.

Unsurprisingly, stand-alone proof checkers for QBF are typically much more efficient [5] than the LCF-style proof checker presented in this paper. From the HOL4 point of view, however, a stand-alone checker would become part of the trusted code base (i.e., bugs in the checker—or in the integration—could lead to inconsistent theorems in HOL4). In contrast, the checker presented here cannot draw an unsound inference: any attempt to do so will be prevented by HOL4's trusted inference kernel.

3 Background and Theory

We now introduce relevant definitions and notation in more detail. Our terminology is entirely standard. The reader is expected to be familiar with propositional logic.

3.1 Quantified Boolean Formulae

We assume an infinite set of Boolean variables. A *literal* is a possibly negated Boolean variable. We extend negation to literals and identify $\neg\neg v$ with v. A *clause* is a disjunction of literals. A clause is *trivial* if it contains both a variable and its negation. We say that a propositional formula is in *conjunctive normal form (CNF)* if it is a conjunction of clauses.

Definition 1 (Quantified Boolean Formula). *A* Quantified Boolean Formula (QBF) *is of the form*

$$Q_1 x_1 \ldots Q_n x_n . \phi,$$

where $n \geq 0$, each x_i is a Boolean variable, each Q_i is either $\forall$ or $\exists$, and ϕ is a propositional formula in CNF.

$Q_1 x_1 \ldots Q_n x_n$ is called the *quantifier prefix*, and ϕ is called the *matrix*. Without loss of generality, we consider QBF in this *prenex form* only. Any formula involving only propositional connectives and quantifiers over Boolean variables can be transformed into prenex form through syntactic manipulations. (We have not yet implemented such a transformation. Note that a HOL4 implementation, aside from producing an equivalent QBF in prenex form, would also have to produce a proof of the equivalence. Doing so efficiently can easily be a challenge for large formulae, but is beyond the scope of this paper.)

We define an order $<$ on variables such that $x_i < x_j$ iff $i < j$, i.e., larger variables are in the scope of smaller variables. x_1 is called the *outermost*, x_n the *innermost* variable of the above QBF.

The QDIMACS format [29] is the standard input format of QBF solvers. It provides a textual means of encoding QBF in prenex form. It is a backward-compatible extension of the DIMACS format [30], the standard input format

of SAT solvers. We have implemented a translation from (the QBF subset of) HOL4 terms into QDIMACS format, and a simple recursive-descent parser for QDIMACS files that returns the corresponding QBF as a HOL4 term (see Section 3.4).

The QDIMACS format imposes further restrictions: all variables x_i must be distinct, all variables must appear in the matrix, and the innermost quantifier must be existential (i.e., $Q_n = \exists$). Note that an innermost universal quantifier can be eliminated by removing all occurrences of the bound variable from the matrix: if a non-trivial clause $v \vee \phi$ is true for all values of v, then ϕ must be true (and likewise for a clause $\neg v \vee \phi$). This inference is called *forall-reduction* [6]. Applying it as often as possible (to eliminate all universal variables that are larger than the largest existential variable), one obtains the *forall-reduct* of the original clause.

We further require all variables that appear in the matrix to be bound by some quantifier, i.e., we consider *closed* QBF only. This is to avoid confusion: in the QDIMACS format, free variables have existential semantics (to retain backward compatibility with the DIMACS format), while in HOL4, free variables in theorems have universal semantics (to permit instantiation). Therefore, if a QBF has free variables, we consider its existential closure instead.

The semantics of closed QBF is defined recursively, by expanding the outermost variable: $[\![\forall x.\, \phi]\!] = [\![\phi[x \mapsto \top] \wedge \phi[x \mapsto \bot]]\!]$, and similarly $[\![\exists x.\, \phi]\!] = [\![\phi[x \mapsto \top] \vee \phi[x \mapsto \bot]]\!]$. (Here $\phi[x \mapsto y]$ denotes substitution of y for all free occurrences of x in ϕ.) A QBF is called *invalid* if its semantics is $\bot$ (i.e., false).

3.2 Q-Resolution

QBF of interest typically contain several dozen or even hundreds of quantifiers. A naive recursive computation of their semantics, which would be exponential in the number of quantifiers, is not feasible. Therefore, Squolem takes a different approach. To show that a QBF is invalid, Squolem proves that it entails $\bot$. Squolem's certificates of invalidity are based on a single inference rule that is known as *Q-resolution* [6]. Q-resolution employs propositional resolution followed by forall-reduction to eliminate universal quantifiers.

Let ϕ and ψ be two clauses. We say that ϕ and ψ can be *resolved* if some variable v occurs positively in ϕ and negatively in ψ. (v is called the *pivot* variable.) Propositional resolution then derives the clause $\phi' \vee \psi'$, where ϕ' is ϕ with v removed, and ψ' is ψ with $\neg v$ removed. This clause is called the *resolvent* of ϕ and ψ.

The resolvent of non-trivial clauses no longer contains the pivot variable. If the pivot was existential, the resolvent's largest variable may be universal, thereby enabling forall-reductions.

Definition 2 (Q-resolution). *Let ϕ and ψ be two clauses of a QBF that can be resolved. Their resolvent's forall-reduct is called the* Q-resolvent *of ϕ and ψ.*

Q-resolution, just like resolution for propositional clauses, is sound and refutation-complete for QBF in prenex form [6]. Thus, given any invalid QBF, we can derive $\bot$ by repeated application of Q-resolution to suitable clauses.

As a simple example, consider (1). This QBF is invalid. To derive $\bot$ using Q-resolution, we resolve $y \vee z$ with $y \vee \neg z$ to obtain y. This clause no longer contains z, the QBF's innermost variable. Thus forall-reduction removes y, which is universally quantified, and we obtain the empty clause, i.e., $\bot$.

3.3 Squolem's Certificates of Invalidity

Squolem's certificate format is described in detail in [31]. The format is ASCII-based. Clauses and variables are referenced by positive integers. Negative values stand for negated variables, i.e., integer negation denotes propositional negation.

Certificates of invalidity contain a log of Q-resolution inferences, concluded by a final line that gives the identifier of the empty clause. For each Q-resolvent, the log contains a line stating its assigned clause identifier, the literals that the Q-resolvent contains, and the clauses that it was derived from. Original clauses (from the QBF's matrix) are numbered consecutively, starting from 1.

For instance, mapping x, $y \vee z$ and $y \vee \neg z$ to clause identifiers 1, 2 and 3, respectively, a certificate for (1) might look as follows:

```
QBCertificate
4 0 2 3 0
CONCLUDE INVALID 4
```

Thus, the empty clause (with identifier 4) is obtained by Q-resolving clauses 2 and 3. Note that forall-reduction is part of a Q-resolution inference. 0 is merely used as a separator.

We have written a simple recursive-descent parser for this certificate format that returns the encoded information as a value in Standard ML.

3.4 Higher-Order Logic

HOL4 is a popular LCF-style [10,11] theorem prover for polymorphic higher-order logic [9]. It is based on Church's simple type theory [32] extended with Hindley-Milner style polymorphism [33]. Higher-order logic (HOL) contains a type of Booleans, propositional connectives, and quantifiers over arbitrary types. Hence, quantified propositional logic can straightforwardly be embedded into HOL.

HOL4 implements a natural-deduction calculus. Theorems represent *sequents* $\Gamma \vdash \phi$, where Γ is a finite set of *hypotheses*, and ϕ is the sequent's *conclusion*. Instead of $\emptyset \vdash \phi$, we simply write $\vdash \phi$. Internally, the set of hypotheses is given by a red-black tree (for efficient search, insertion and deletion), with terms treated modulo α-equivalence.

Like other LCF-style provers, HOL4 has a small inference kernel. Theorems are implemented as an abstract data type, and new theorems can be constructed

only through a fixed set of functions provided by this data type. These functions directly correspond to the axiom schemata and inference rules of higher-order logic. Figure 1 shows the rules of HOL that we use to validate Squolem's certificates.

$$\frac{}{\{\phi\} \vdash \phi}\ \textsc{Assume} \qquad \frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \phi}\ \textsc{Conj1} \qquad \frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \psi}\ \textsc{Conj2}$$

$$\frac{\Gamma \vdash \phi \vee \psi \qquad \Delta_1 \cup \{\phi\} \vdash \theta \qquad \Delta_2 \cup \{\psi\} \vdash \theta}{\Gamma \cup \Delta_1 \cup \Delta_2 \vdash \theta}\ \textsc{DisjCases}$$

$$\frac{\Gamma \vdash \phi \implies \bot}{\Gamma \vdash \neg\phi}\ \textsc{NotIntro} \qquad \frac{\Gamma \vdash \neg\phi}{\Gamma \vdash \phi \implies \bot}\ \textsc{NotElim}$$

$$\frac{\Gamma \vdash \psi}{\Gamma \setminus \{\phi\} \vdash \phi \implies \psi}\ \textsc{Disch} \qquad \frac{\Gamma \vdash \phi \implies \psi \qquad \Delta \vdash \phi}{\Gamma \cup \Delta \vdash \psi}\ \textsc{MP}$$

$$\frac{\Gamma \vdash \phi}{\Gamma\,\theta \vdash \phi\,\theta}\ \textsc{Inst}_\theta \qquad \frac{\Gamma \vdash \forall x.\, \phi}{\Gamma \vdash \phi[x \mapsto t]}\ \textsc{Spec}_t$$

$$\frac{\Gamma \vdash \exists x.\, \phi \qquad \Delta \cup \{\phi[x \mapsto v]\} \vdash \psi}{\Gamma \cup \Delta \vdash \psi}\ \textsc{Choose}_v \text{ (v not free in Γ, Δ or ψ)}$$

Fig. 1. Selected HOL inference rules

The LCF-style architecture greatly reduces the trusted code base. Proof procedures, although they may implement arbitrarily complex algorithms, cannot produce unsound theorems, as long as the implementation of the theorem data type is correct. HOL4 is written in Standard ML [34], a type-safe functional language (with impure features, e.g., references) that has an advanced module system. To benefit from HOL4's LCF-style architecture, we must implement proof reconstruction in this language.

On top of its LCF-style inference kernel, HOL4 offers various automated proof procedures: e.g., a simplifier, which performs term rewriting, a decision procedure for propositional logic, and various first-order provers. However, the performance of these procedures is hard to control. To achieve optimal performance, we do not employ them for certificate checking, but instead combine primitive inference rules directly (see Section 4).

4 Checking Squolem's Certificates in HOL4

4.1 Preliminaries

Given a QBF $\varphi = Q_1 x_1 \ldots Q_n x_n.\, \phi$ and a certificate of its invalidity, our goal is to derive $\{\varphi\} \vdash \bot$ as a HOL4 theorem. We start by assuming the QBF's

matrix, ϕ, thereby obtaining $\{\phi\} \vdash \phi$. We then use a combination of forward and backward reasoning: the former to transform the sequent's conclusion into $\bot$, and the latter to introduce quantifiers into the hypothesis, thereby transforming it into φ. This enables a clear separation of propositional and quantifier reasoning.

Suppose that $\phi = C_1 \wedge \cdots \wedge C_k$, where each C_i is a clause of the original QBF. Repeatedly applying inference rules Conj1 and Conj2, we split $\{\phi\} \vdash \phi$ into k separate theorems $\{\phi\} \vdash C_1, \ldots, \{\phi\} \vdash C_k$. This eliminates all conjunctions from the conclusion. Therefore, we do not have to use associativity or commutativity of conjunction in order to resolve clauses. Note that this step does not consume significant amounts of memory: although ϕ may be huge, existing Standard ML implementations employ sharing and store ϕ in memory only once.

We use a similar idea to eliminate disjunctions. Suppose that $C_i = l^i_1 \vee \cdots \vee l^i_{m_i}$, where each l^i_j is a literal. Following [18], we use a combination of Disj-Cases, Disch and MP to transform $\{\phi\} \vdash C_i$ into $\{\phi, \neg l^i_1, \ldots, \neg l^i_{m_i}\} \vdash \bot$. This allows us to benefit from HOL4's relatively efficient management of hypotheses (which are stored in a red-black tree internally) when manipulating literals during resolution, rather than having to use associativity and commutativity of disjunction.

4.2 General Proof Structure

After transformation into this *sequent form*, each clause theorem is stored in a dictionary (implemented by a red-black tree for logarithmic time access), indexed by its numeric clause identifier. Along with each clause, we store the quantifier prefix that is missing from the clause's hypothesis. Since we started by assuming the matrix, this is the entire prefix initially, i.e., $Q_1 x_1 \ldots Q_n x_n$. During certificate validation, we will successively introduce these quantifiers again, until we arrive at the original QBF.

Squolem's certificates of invalidity encode a directed acyclic graph. The empty clause is the root, and each node is connected to the premises from which it is derived by Q-resolution. We perform a depth-first post-order traversal of this graph, starting at the root node, and adding new clauses to the clause dictionary as they are derived. This approach, which is also adopted from [18], has two benefits. First, if there are Q-resolution inferences in the certificate that do not contribute to the derivation of the final $\bot$, these are never checked in HOL4. Second, clauses must be derived in HOL4 only once, even if they are used several times in the proof. Later, they are simply retrieved from the dictionary.

4.3 Q-Resolution

Every node of the proof graph corresponds to a Q-resolution inference, which we have to perform in HOL4 (unless the node does not contribute to the derivation of the final $\bot$, see above). Q-resolution consists of propositional resolution followed by forall-reduction.

Propositional resolution for clauses in sequent form can be implemented efficiently as a combination of primitive HOL inferences [18]:

$$\dfrac{\dfrac{\dfrac{\Gamma \cup \{\neg v\} \vdash \bot}{\Gamma \vdash \neg v \implies \bot}\,\textsc{Disch} \qquad \dfrac{\dfrac{\Delta \cup \{v\} \vdash \bot}{\Delta \vdash v \implies \bot}\,\textsc{Disch}}{\Delta \vdash \neg v}\,\textsc{NotIntro}}{\Gamma \cup \Delta \vdash \bot}\,\text{MP}$$

The resolvent, $\Gamma \cup \Delta \vdash \bot$, is again a clause in sequent form.

It remains to deal with forall-reduction. Let $\{\phi, l_1, \ldots, l_m\} \vdash \bot$ be the resolvent, and let x_i be the largest variable that occurs in it. Since propositional resolution has removed the (existential) pivot variable, x_i may be universal. We must perform forall-reduction if this is the case.

There are two aspects to this task. We successively transform the QBF's matrix, which initially is a hypothesis of each clause, into the original QBF. Thus, we must introduce (existential and universal) quantifiers for variables larger than x_i, which no longer occur in the clause. Second, we must eliminate x_i, exploiting the fact that it is universal.

Suppose the missing quantifier prefix is $Q_1 x_1 \ldots \forall x_i \ldots Q_j x_j$, with $j \geq i$. If $Q_j = \forall$, we derive

$$\dfrac{\dfrac{\{\phi, l_1, \ldots, l_m\} \vdash \bot}{\{l_1, \ldots, l_m\} \vdash \phi \implies \bot}\,\textsc{Disch} \qquad \dfrac{\dfrac{}{\{\forall x_j.\, \phi\} \vdash \forall x_j.\, \phi}\,\textsc{Assume}}{\{\forall x_j.\, \phi\} \vdash \phi}\,\textsc{Spec}_{x_j}}{\{\forall x_j.\, \phi,\, l_1, \ldots, l_m\} \vdash \bot}\,\text{MP}$$

If $Q_j = \exists$, then necessarily $j > i$, and we instead derive

$$\dfrac{\dfrac{}{\{\exists x_j.\, \phi\} \vdash \exists x_j.\, \phi}\,\textsc{Assume} \qquad \{\phi, l_1, \ldots, l_m\} \vdash \bot}{\{\exists x_j.\, \phi,\, l_1, \ldots, l_m\} \vdash \bot}\,\textsc{Choose}_{x_j}$$

The side condition of $\textsc{Choose}_{x_j}$ (see Figure 1) is satisfied because x_j does not occur among $l_1, \ldots, l_m$.

Repeating this step for all missing quantifiers up to $Q_i x_i$, we arrive at $\{Q_i x_i \ldots Q_j x_j.\, \phi,\, l_1, \ldots, l_m\} \vdash \bot$. Note that $Q_i = \forall$, hence our reasoning is sound despite the fact that x_i still occurs in the clause.

Now x_i is bound in $Q_i x_i \ldots Q_j x_j.\, \phi$, and occurs free only in one of the literals $l_1, \ldots, l_m$. We instantiate x_i to $\neg\bot$ if it occurs positively, and to $\bot$ if it occurs negatively. In either case the literal becomes $\neg\bot$ and can be discharged.

We continue to forall-reduce the resulting clause to eliminate further universal variables if possible.

A technicality arises from the interplay of propositional resolution and forall-reduction. Forall-reduction introduces quantifiers (thereby shortening the prefix of missing quantifiers). Therefore, the two clauses used in a resolution step may have different quantifier prefixes around the matrix in their hypotheses: i.e., $Q_i x_i \ldots Q_n x_n.\, \phi$ vs. $Q_j x_j \ldots Q_n x_n.\, \phi$, with $i < j$. The resolvent will contain both formulae as hypotheses. In our bookkeeping, we only keep track of the longer prefix of missing quantifiers, i.e., $Q_1 x_1 \ldots Q_{j-1} x_{j-1}$, and ignore the other hypothesis. Eventually, later forall-reduction steps in the proof will (again) introduce quantifiers $Q_i x_i$ through $Q_{j-1} x_{j-1}$ into the resolvent. At this point both

hypotheses will become identical, and (since hypotheses are implemented by a set) one copy will automatically be discarded by HOL4.

While this could lead to an accumulation of matrix hypotheses (each with a different quantifier prefix) in a clause during certificate validation, clauses with different quantifier prefixes are rarely resolved in practice. In the QBF certificates used for evaluation (see Section 5), clauses derived by Squolem contain at most two matrix hypotheses each.

4.4 Example

Consider (1) again. Let $\phi = x \wedge (y \vee z) \wedge (y \vee \neg z)$ denote its matrix. We assume ϕ to obtain $\{\phi\} \vdash \phi$. Using Conj1 and Conj2, we derive three separate theorems $\{\phi\} \vdash x$, $\{\phi\} \vdash y \vee z$, and $\{\phi\} \vdash y \vee \neg z$. Their respective sequent forms are $\{\phi, \neg x\} \vdash \bot$, $\{\phi, \neg y, \neg z\} \vdash \bot$, and $\{\phi, \neg y, z\} \vdash \bot$ The missing quantifier prefix for each theorem is $\exists x\, \forall y\, \exists z$.

Validating Squolem's proof of invalidity (see Section 3.3), we now Q-resolve the second and the third theorem. Propositional resolution yields $\{\phi, \neg y\} \vdash \bot$. The largest variable that occurs in this clause is y.

Since y is universal, we perform forall-reduction (as detailed in Section 4.3). The innermost missing quantifier is $\exists z$. Thus, we first derive $\{\exists z.\, \phi,\, \neg y\} \vdash \bot$. The next missing quantifier is $\forall y$, so we derive $\{\forall y \exists z.\, \phi,\, \neg y\} \vdash \bot$. Now we eliminate y by instantiating it to $\bot$, thereby obtaining $\{\forall y \exists z.\, \phi,\, \neg \bot\} \vdash \bot$. Discharging $\neg \bot$ yields $\{\forall y \exists z.\, \phi\} \vdash \bot$. The next missing quantifier is $\exists x$, and x does not occur in the clause (except in ϕ). Hence, we finally arrive at $\{\exists x \forall y \exists z.\, \phi\} \vdash \bot$.

4.5 Variable Binding and Substitution

HOL4 comes with two different implementations of its inference kernel: one uses de Bruijn indices (and explicit substitutions) to represent λ-terms [35], the other (by M. Norrish) uses a name-carrying implementation [36]. These implementations differ in the performance (and even complexity) of primitive operations. For instance, λ-abstracting over a variable takes constant time with the name-carrying implementation, but with de Bruijn indices is linear in the size of the abstraction's body (because every occurrence of the newly bound variable in the body must be replaced by an index). Moreover, since the abstraction's body remains unchanged in the name-carrying implementation, there is more potential for memory sharing if the body is also used elsewhere, and hence a potentially smaller memory footprint. Despite these differences, both kernels show similar overall performance on the HOL4 library.

This is no longer true for QBF validation. In higher-order logic, $\forall x.\, \phi$ is syntactic sugar for $\forall(\lambda x.\, \phi)$, and likewise for $\exists x.\, \phi$. Hence, the algorithm for Q-resolution presented in Section 4.3 forms λ-abstractions (and takes them apart again) when introducing quantifiers during forall-reduction. We will see in Section 5 that Norrish's name-carrying implementation, therefore, is significantly faster for QBF validation than the kernel that uses de Bruijn indices internally.

During evaluation, we also observed that the name-carrying implementation spent significant time instantiating variables (to $\bot$ or $\neg\bot$, before they are discharged as part of forall-reduction). Capture-avoiding substitution in a name-carrying implementation may have to rename bound variables away from the free variables in the body of a λ-abstraction. It turned out that in order to collect these free variables, the HOL4 implementation of substitution would unnecessarily descend into the body of a λ-abstraction even when the variable to be instantiated was bound (in which case instantiation would not change the body at all). We achieved an average speed-up of 2.6 (see Section 5) by improving the implementation of capture-avoiding substitution to collect free variables only when they are actually needed for renaming.

One might gain further improvements by using a modified term data structure. The kernel could compute the set of a term's free variables when the term is built, and store it in memory along with the term itself. This might allow for an even more efficient implementation of capture-avoiding substitution.

5 Experimental Results

We have evaluated our implementation on the same set of 69 invalid QBF problems that was previously used (by the Squolem authors) to evaluate the performance of Squolem's certificate generation [7]. The set resulted from applying Squolem to all 445 problems in the *2005 fixed instance* and *2006 preliminary QBF-Eval* data sets. With a time limit of 600 seconds per problem, Squolem solved 142 of these problems; 69 were determined to be invalid.

All experiments were conducted on a Linux system with an Intel Core2 Duo T9300 processor at 2.4 GHz. Memory usage was restricted to 3 GB. HOL4 was running on top of Poly/ML 5.3.

5.1 Run-Times

Table 1 shows our experimental results for the 69 invalid QBF problems. The first column gives the name of the benchmark. The next three columns provide information about the size of the benchmark, giving the number of alternating quantifiers,[1] variables, and clauses, respectively. Column four shows the run-time of Squolem (with certificate generation enabled) to solve the benchmark. Column five shows the number of Q-resolution steps in the resulting certificate. The last three columns finally show the run-time of certificate validation in HOL4, using the de Bruijn kernel, the name-carrying kernel, and our optimized variant of the name-carrying kernel, respectively (see Section 4.5). All run-times are given in seconds (rounded to the nearest tenth of a second). On one benchmark, the de Bruijn kernel ran out of memory (indicated by an M).

[1] Counting successive quantifiers of the same kind, as in $\forall x\, \forall y\, \forall z\, \ldots$, as one quantifier only. The total number of quantifiers in each benchmark is typically identical to the number of variables.

Benchmark name	Quant.	Vars.	Clauses	Squolem (s)	Q-res. steps	de Bruijn (s)	name-carrying (s)	optimized name-c. (s)
Adder2-2-c	7	249	291	8.9	445	6.4	0.6	0.3
adder-2-unsat	3	66	110	13.6	3632	3.3	1.6	0.5
comp.blif_0.10_0.20_0_0_inp_exact	7	310	831	6.0	2612	3.0	0.3	0.1
comp.blif_0.10_1.00_0_0_inp_exact	3	306	842	1.9	3317	1.0	0.1	0.1
flipflop-3-c	3	164	203	0.0	15	0.0	0.0	0.0
flipflop-4-c	3	866	1158	20.5	12	1.4	0.0	0.0
k_d4_p-12	33	755	2234	0.4	276	44.2	1.5	0.6
k_d4_p-16	41	995	2966	0.6	348	95.5	2.7	1.0
k_d4_p-20	49	1235	3698	0.8	420	173.1	4.3	1.7
k_d4_p-21	51	1295	3881	0.8	438	197.4	4.7	2.0
k_d4_p-4	17	275	770	0.1	132	2.8	0.2	0.1
k_d4_p-8	25	515	1502	0.3	204	15.0	0.7	0.3
k_dum_p-12	27	517	1352	0.1	302	17.6	0.8	0.3
k_dum_p-16	35	685	1782	0.1	334	35.1	1.3	0.6
k_dum_p-20	43	853	2212	0.1	366	61.9	1.9	0.8
k_dum_p-21	45	893	2311	0.1	366	68.1	2.1	0.9
k_dum_p-4	15	556	215	0.1	231	2.2	0.2	0.1
k_dum_p-8	19	349	922	0.1	266	6.9	0.4	0.2
k_grz_p-12	17	534	1961	479.2	303	19.1	1.1	0.3
k_grz_p-4	17	318	953	0.1	283	5.3	0.4	0.2
k_grz_p-8	17	419	1406	0.6	286	10.5	0.7	0.2
k_lin_p-4	7	241	840	0.0	28	1.1	0.1	0.0
k_lin_p-8	7	439	1916	242.8	32	4.7	0.2	0.1
k_path_p-12	27	805	2238	0.3	306	40.3	1.4	0.5
k_path_p-16	35	1081	3014	0.4	406	91.9	2.6	1.0
k_path_p-20	43	1357	3790	0.5	506	173.4	4.4	1.9
k_path_p-21	45	1429	3996	0.6	531	200.2	4.9	2.1
k_path_p-4	11	253	686	0.1	106	1.7	0.1	0.1
k_path_p-8	19	529	1462	0.2	206	12.2	0.5	0.2
k_poly_p-12	79	1005	2268	0.4	1072	137.5	3.5	2.0
k_poly_p-16	103	1329	3000	1.6	1375	303.9	6.6	3.8
k_poly_p-20	127	1653	3732	1.8	1711	577.9	11.1	6.6
k_poly_p-21	131	1707	3854	0.9	1771	635.6	12.0	7.1
k_poly_p-4	31	357	804	0.1	377	7.0	0.3	0.2
k_poly_p-8	55	681	1536	0.2	724	44.1	1.5	0.7
k_t4p_p-12	37	979	3040	1.5	394	91.9	2.7	1.0
k_t4p_p-16	45	1529	3936	2.1	474	180.7	4.5	1.8
k_t4p_p-20	53	1539	4832	2.5	554	318.3	6.9	2.7
k_t4p_p-21	55	1609	5056	2.6	574	360.7	7.5	3.0
k_t4p_p-4	21	419	1248	0.5	234	9.3	0.5	0.2
k_t4p_p-8	29	699	2144	1.0	314	37.1	1.4	0.5
s27_d3_u	3	117	254	33.2	309	0.2	0.2	0.0
sat05-561-qd	1	24	61	0.0	158	0.0	0.0	0.0
TOILET2.1.iv.3	3	28	70	0.0	20	0.0	0.0	0.0
toilet_a_08_01.2	3	60	2205	1.0	6	1.7	0.8	0.1
toilet_a_08_01.4	3	112	2429	1.1	44	3.6	1.4	0.2
toilet_a_08_05.2	3	140	2833	124.9	1855	5.7	7.4	4.9
toilet_a_10_01.2	3	74	10455	33.7	6	17.8	8.0	1.0
toilet_a_10_01.3	3	106	10604	35.1	16	24.7	8.5	1.1
toilet_a_10_01.4	3	138	10753	36.3	44	36.5	13.7	1.6
toilet_c_08_01.2	3	55	229	0.0	6	0.1	0.0	0.0
toilet_c_08_01.4	3	107	453	0.1	44	0.2	0.0	0.0
toilet_c_08_05.2	3	135	857	122.4	1855	0.6	0.7	0.7
toilet_c_10_01.2	3	68	325	0.0	6	0.1	0.0	0.0
toilet_c_10_01.4	3	132	623	0.2	44	0.3	0.1	0.0
tree-exa2-10	20	20	12	0.0	18	0.0	0.0	0.0
tree-exa2-15	30	30	17	0.0	28	0.0	0.0	0.0
tree-exa2-20	40	40	22	0.0	38	0.0	0.0	0.0
tree-exa2-25	50	50	27	0.0	48	0.0	0.0	0.0
tree-exa2-30	60	60	32	0.0	58	0.1	0.0	0.0
tree-exa2-35	70	70	37	0.0	68	0.1	0.1	0.0
tree-exa2-40	80	80	42	0.0	78	0.1	0.1	0.1
tree-exa2-45	90	90	47	0.0	88	0.2	0.1	0.1
tree-exa2-50	100	100	52	0.0	98	0.3	0.2	0.1
vonNeumann-ripple-carry-5-c	3	24562	35189	220.3	33	M	1.8	1.5
z4ml.blif_0.10_0.20_0_0_inp_exact	5	65	193	1.8	996	0.2	0.0	0.0
z4ml.blif_0.10_0.20_0_0_out_exact	3	61	185	1.2	1536	1.3	0.3	0.1
z4ml.blif_0.10_1.00_0_0_inp_exact	3	65	198	0.1	588	0.1	0.0	0.0
z4ml.blif_0.10_1.00_0_0_out_exact	3	63	194	0.6	1588	1.2	0.2	0.1

Table 1. Experimental results

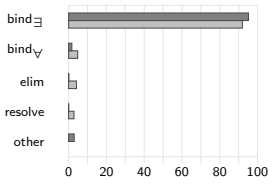

Fig. 2. de Bruijn

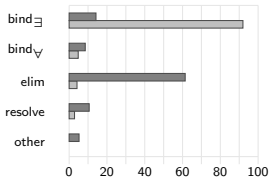

Fig. 3. Name-carrying

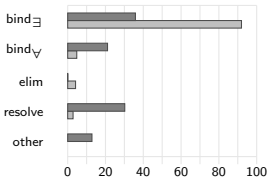

Fig. 4. Optimized

Average run-times are 60.2 seconds for the de Bruijn kernel (not including benchmark vonNeumann-ripple-carry-5-c), 2.1 seconds for the name-carrying kernel, and 0.8 seconds for our optimized variant of the name-carrying kernel. This amounts to speed-up factors of 28.7 (de Bruijn vs. name-carrying) and 2.6 (name-carrying vs. optimized), respectively, for a total speed-up factor of 75.3 (de Bruijn vs. optimized).

For comparison, we have also measured run-times of QBV [7], a stand-alone checker for Squolem's certificates that was developed by the authors of Squolem. QBV validates each of the 69 certificates in less than 0.1 seconds. LCF-style validation in HOL4, using the optimized name-carrying kernel, is one to two orders of magnitude slower. However, for users of HOL4, another comparison might be more relevant: LCF-style validation (using the optimized name-carrying kernel) on average is a factor of 24.5 faster than proof search with Squolem, and at most 8 times slower on individual benchmarks.

5.2 Profiling

To gain deeper insight into these results, we present profiling data for the de Bruijn kernel (Figure 2), the name-carrying kernel (Figure 3), and our optimized variant of the name-carrying kernel (Figure 4).

For each kernel, we show the shares of total run-time (dark bars) and relative number of function calls (light bars) for the following functions: binding of existential quantifiers during forall-reduction ($\mathsf{bind}_\exists$), binding of universal quantifiers during forall-reduction ($\mathsf{bind}_\forall$), elimination of universal variables during forall-reduction (elim), and propositional resolution (resolve) as part of Q-resolution. Additionally, time spent on other aspects of certificate validation, e.g., file parsing and conversion of clauses into sequent form, is shown as well (other). The relative number of function calls (light bars) is the same for each kernel.

We observe that the de Bruijn kernel (Figure 2) spends more than 90% of validation time on the introduction of existential quantifiers. This is in line with the relative frequency of $\mathsf{bind}_\exists$. The name-carrying implementation (Figure 3), however, performs the same operation much more quickly (for the reasons discussed in Section 4.5), reducing its run-time share to less than 20%. On the other hand, time spent on variable elimination (elim) has increased disproportionally, to over 60%. Our optimization of capture-avoiding substitution (see Section 4.5) reduces this time to a negligible fraction again (Figure 4), while the remaining operations take proportionally higher time shares.

6 Conclusions

We have presented LCF-style checking for certificates of QBF invalidity (generated by the QBF solver Squolem) in HOL4. In particular, we have presented an efficient implementation of Q-resolution on top of HOL4's inference kernel for higher-order logic. Detailed performance data shows that LCF-style certificate checking is feasible even for large invalid QBF instances: all 69 benchmark certificates were checked successfully. However, performance very much depends on implementation details of the underlying inference kernel. We have improved HOL4's implementation of capture-avoiding substitution, thereby achieving a speed-up of 75.3 over an implementation based on de Bruijn indices. Our implementation is freely available from the HOL4 repository [36].

Our work has two main applications. First, it enables HOL4 users to benefit from Squolem's automation for QBF problems. These can now be passed from the HOL4 system to Squolem, which will automatically decide their validity. For invalid QBF problems, Squolem's certificate will then be used to derive the QBF's negation as a theorem in HOL4. Second, our work provides high correctness assurances for Squolem's results. Due to HOL4's LCF-style architecture, our proof checker cannot draw unsound inferences (provided HOL4's kernel is correct). Thus, the approach can be used for QBF benchmark certification.

In this paper, we have only considered certificates of invalidity. In principle, one can establish validity of a QBF instance by showing that the negation is invalid. However, this approach is rarely feasible in practice [7]. Squolem can generate certificates of validity directly, based on Skolem functions. LCF-style checking for certificates of validity remains future work.

One could also extend our work to other QBF solvers, which use different certificate formats (see [23] for an overview), and to other interactive theorem provers, e.g., Isabelle or Coq. Because seemingly minor differences in kernel data structures can have significant impact, it is not clear if similar performance can be achieved in these systems.

An alternative approach that might yield better performance than the LCF-style implementation presented in this paper is the use of reflection [37], i.e., implementing and proving correct a checker for Squolem's certificates in the prover's logic, and then executing the verified checker without producing proofs. While this approach still provides relatively high correctness assurances, obtaining a theorem in HOL4 would require enhancing the inference kernel with a reflection rule that allows to trust the result of such a verified computation.

Acknowledgments

The author would like to thank Christoph Wintersteiger for answering various questions about Squolem.

References

1. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: Proceedings of TACAS. Volume 1579 of Lecture Notes in Computer Science., Springer (1999) 193–207
2. Gopalakrishnan, G., Yang, Y., Sivaraj, H.: QB or not QB: An efficient execution verification tool for memory orderings. In: Computer Aided Verification. Volume 3114 of Lecture Notes in Computer Science., Springer (2004) 47–49
3. Hanna, Z., Dershowitz, N., Katz, J.: Bounded model checking with QBF. In: Eight International Conference on Theory and Applications of Satifiability Testing (SAT 2005). Volume 3569 of Lecture Notes in Computer Science., Springer Verlag (2005)
4. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time. In: Proc. 5th Annual ACM Symp. on Theory of Computing. (1973) 1–9
5. Narizzano, M., Peschiera, C., Pulina, L., Tacchella, A.: Evaluating and certifying QBFs: A comparison of state-of-the-art tools. AI Communications **22**(4) (2009) 191–210
6. Büning, H.K., Karpinski, M., Flögel, A.: Resolution for quantified boolean formulas. Information and Computation **117**(1) (1995) 12–18
7. Jussila, T., Biere, A., Sinz, C., Kröning, D., Wintersteiger, C.M.: A first step towards a unified proof checker for QBF. In: Theory and Applications of Satisfiability Testing – SAT 2007. Volume 4501 of Lecture Notes in Computer Science., Springer (2007) 201–214
8. Slind, K., Norrish, M.: A brief overview of HOL4. [38] 28–32
9. 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
10. Gordon, M., Milner, R., Wadsworth, C.P.: Edinburgh LCF: A Mechanised Logic of Computation. Volume 78 of Lecture Notes in Computer Science. Springer (1979)
11. Gordon, M.: From LCF to HOL: a short history. In: Proof, language, and interaction: essays in honour of Robin Milner. MIT Press (2000) 169–185
12. Bertot, Y.: A short presentation of Coq. [38] 12–16
13. Wenzel, M., Paulson, L.C., Nipkow, T.: The Isabelle framework. [38] 33–38
14. Owre, S., Shankar, N.: A brief overview of PVS. [38] 22–27
15. Kumar, R., Kropf, T., Schneider, K.: Integrating a first-order automatic prover in the HOL environment. In Archer, M., Joyce, J.J., Levitt, K.N., Windley, P.J., eds.: Proceedings of the 1991 International Workshop on the HOL Theorem Proving System and its Applications, IEEE Computer Society (1992) 170–176
16. Hurd, J.: An LCF-style interface between HOL and first-order logic. In Voronkov, A., ed.: Proceedings of the 18th International Conference on Automated Deduction (CADE-18). Volume 2392 of Lecture Notes in Artificial Intelligence., Springer (2002) 134–138
17. Meng, J., Paulson, L.C.: Translating higher-order clauses to first-order clauses. Journal of Automated Reasoning **40**(1) (2008) 35–60
18. Weber, T., Amjad, H.: Efficiently checking propositional refutations in HOL theorem provers. Journal of Applied Logic **7**(1) (March 2009) 26–40
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. Böhme, S., Weber, T.: Fast LCF-style proof reconstruction for Z3 (2010) To appear at the International Conference on Interactive Theorem Proving (ITP-10).

21. Letz, R.: Lemma and model caching in decision procedures for quantified boolean formulas. In: Automated Reasoning with Analytic Tableaux and Related Methods. Volume 2381 of Lecture Notes in Computer Science., Springer (2002) 5–15
22. Pulina, L., Tacchella, A.: Learning to integrate deduction and search in reasoning about quantified boolean formulas. In Ghilardi, S., Sebastiani, R., eds.: FroCos. Volume 5749 of Lecture Notes in Computer Science., Springer (2009) 350–365
23. Narizzano, M., Pulina, L., Tacchella, A.: Report of the third QBF solvers evaluation. JSAT **2**(1–4) (2006) 145–164
24. Amjad, H.: Combining model checking and theorem proving. Technical Report UCAM-CL-TR-601, University of Cambridge Computer Laboratory (2004) Ph. D. Thesis.
25. Ballarin, C.: Computer algebra and theorem proving. Technical Report UCAM-CL-TR-473, University of Cambridge Computer Laboratory (1999) Ph. D. Thesis.
26. Boldo, S., Filliâtre, J.C., Melquiond, G.: Combining Coq and Gappa for certifying floating-point programs. In Carette, J., Dixon, L., Coen, C.S., Watt, S.M., eds.: Intelligent Computer Mathematics, 16th Symposium, Calculemus 2009, 8th International Conference, MKM 2009, Proceedings. Volume 5625 of Lecture Notes in Computer Science., Springer (2009) 59–74
27. Benedetti, M.: sKizzo: A suite to evaluate and certify QBFs. In Nieuwenhuis, R., ed.: Automated Deduction - CADE-20, 20th International Conference on Automated Deduction, Tallinn, Estonia, July 22-27, 2005, Proceedings. Volume 3632 of Lecture Notes in Computer Science., Springer (2005) 369–376
28. Yu, Y., Malik, S.: Validating the result of a quantified boolean formula (QBF) solver: theory and practice. In Tang, T., ed.: Proceedings of the 2005 Conference on Asia South Pacific Design Automation, ASP-DAC 2005, Shanghai, China, January 18-21, 2005, ACM Press (2005) 1047–1051
29. : QDIMACS standard version 1.1 (2005) Released on December 21, 2005. Retrieved January 22, 2010 from http://www.qbflib.org/qdimacs.html.
30. : DIMACS satisfiability suggested format (1993) Retrieved January 22, 2010 from ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/doc.
31. Kroening, D., Wintersteiger, C.M.: A file format for QBF certificates (2007) Retrieved September 20, 2009 from http://www.verify.ethz.ch/qbv/download/qbcformat.pdf.
32. Church, A.: A formulation of the simple theory of types. Journal of Symbolic Logic **5** (1940) 56–68
33. Damas, L., Milner, R.: Principal type-schemes for functional programs. In: POPL. (1982) 207–212
34. Milner, R., Tofte, M., Harper, R., MacQueen, D.: The Definition of Standard ML – Revised. MIT Press (1997)
35. Barras, B.: Programming and computing in HOL. In Aagaard, M., Harrison, J., eds.: TPHOLs. Volume 1869 of Lecture Notes in Computer Science., Springer (2000) 17–37
36. HOL4 contributors: HOL4 Kananaskis 5 source code (2010) Retrieved January 22, 2010 from http://hol.sourceforge.net/.
37. Harrison, J.: Metatheory and reflection in theorem proving: A survey and critique. Technical Report CRC-053, SRI Cambridge (1995) Retrieved April 8, 2010 from http://www.cl.cam.ac.uk/~jrh13/papers/reflect.dvi.gz.
38. 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. Volume 5170 of Lecture Notes in Computer Science., Springer (2008)