Proving Valid Quantified Boolean Formulas in HOL Light

Ondřej Kunčar

Charles University in Prague
Faculty of Mathematics and Physics
Automated Reasoning Group
ondrej.kuncar@mff.cuni.cz

Abstract. This paper describes the integration of Squolem, Quantified Boolean Formulas (QBF) solver, with the interactive theorem prover HOL Light. Squolem generates certificates of validity which are based on witness functions. The certificates are checked in HOL Light by constructing proofs based on these certificates. The presented approach allows HOL Light users to prove larger valid QBF problems than before and provides correctness checking of Squolem's outputs based on the LCF approach. An error in Squolem was discovered thanks to the integration. Experiments show that the feasibility of the integration is very sensitive to implementation of HOL Light and used inferences. This resulted in improvements in HOL Light's inference system.

1 Introduction

Deciding whether Quantifier Boolean Formula (QBF) evaluates to true is the canonical PSPACE-complete problem [20]. This problem can be seen as a generalization of the well-known Boolean satisfiability problem (SAT). QBF can contain universal and existential quantifiers over Boolean variables. Let us introduce a simple example, which is nothing else than a definition of the XOR function:

$$\forall v_1\, \forall v_2\, \exists v_3.\, v_3 \Leftrightarrow ((v_1 \wedge \neg v_2) \vee (\neg v_1 \wedge v_2))\ . \tag{1}$$

Whether the problem of true QBFs is harder than SAT is an open problem. Many problems can be succinctly formulated in QBF – every finite two-player game, many types of planning [7, 22], model checking for finite systems and other formal verification problems [2, 3, 6].

Because we work only with closed formulas, validity and invalidity is the same concept as satisfiability and unsatisfiability respectively. QBF solvers are nowadays powerful tools, which are able to decide validity or invalidity of QBFs automatically. Some of them can generate a certificate that witnesses their output. Squolem [17] is a state-of-the-art QBF solver which is able to generate certificates for valid formulas. These certificates are based on witness functions for existential quantifiers.

In this paper we present how to prove QBF validity in the HOL Light interactive theorem prover [11, 12] using Squolem's certificates of validity. HOL Light, made by John Harrison, is a contemporary interactive theorem prover belonging to the broader family of higher-order logic theorem provers. HOL Light has a very small LCF-style kernel [8] and, moreover, a simplified version of the kernel was proved to be correct [10].

The motivation for our work is twofold. First, interactive theorem provers are nowadays becoming increasingly important thanks to their wide use in areas such as formal specification and verification of complex systems or formalization and verification of mathematics. While these systems often contain a very powerful formalism, their main weakness is that the construction of the proof is often lengthy and requires a considerable human effort. As described in Section 2, many integrations of external tools have been done to increase the amount of automation of interactive theorem provers and to decrease the need for human resources. Each of these integrations resulted in increased strength of the interactive theorem prover – we are talking about situations where formulas that were infeasible to prove using the built-in tactics are proved within a few seconds.

Second, our construction of a proof in HOL Light can serve as another independent check of correctness of Squolem. QBF solvers are generally complex tools with nontrivial implementation in some fast imperative programming language (for example C). This fact causes natural concern about correctness of Squolem. Moreover, it is quite common that QBF solvers disagree on the same inputs. Because HOL Light has a LCF-style kernel, validation of Squolem's certificate in HOL Light lowers significantly the probability that the Squolem's answer was incorrect. We really found a small bug in Squolem due to our system. If a input of Squolem contains tautological clauses, then Squolem 1.0 gives an incorrect answer (and of course an incorrect certificate). Squolem 2.0 gives a correct answer, but still an incorrect certificate. This bug was resolved in the version 2.01 after we pointed out the problem to Christoph Wintersteiger.

Related work is discussed in the next section. In Section 3 we provide necessary definitions and background. We present the main part of our work, how to construct a proof of a valid QBF in HOL Light from Squolem's certificate of validity, in Section 4. We provide experimental results and technical aspects concerning the implementation including optimizations in Section 5. Section 6 concludes this paper and suggests directions for future work.

2 Related Work

The most related work is the paper by Weber [24]. In that paper the author implemented validation of Squolem's certificates of invalidity in another LCF interactive prover HOL4, i.e., it is possible to prove that the given QBF is not valid in HOL4. Squolem's certificates of invalidity are based on a Q-resolution proof of $\bot$. By replaying the resolution proof, a proof of invalidity in HOL4 is established. In principle, it would be possible to prove validity of QBF by using the system by Weber. The method is simple: negate the original formula

(valid) and then prove that the negated formula is invalid. But Jussila et al. [17] demonstrated that QBF solvers often perform significantly worse on negated problems. Thus we are going to use directly Squolem's certificates of validity. In the conclusion section of [24] there is a note that LCF-style checking for certificates of validity remains future work. To our knowledge, our work is the first work concerning this task, i.e., proving *valid* QBFs in an interactive theorem prover using an external QBF solver.

Other related work comes from the research area of automation of interactive theorem provers. One of the first integration of an external tool in a trusted theorem prover was the work by Harrison and Thery [13]. It is important to mention earlier but the essential result of John Harrison, who tried to integrate binary decision diagrams (BDD) directly into the HOL Light system [9]. Harrison found out that performing the BDD operations directly by the LCF kernel is about 100 times slower (after optimization) rather than a direct implementation in C. This observation was probably the main reason why most of the further integrations of decision-making procedures use an external solver (to solve the task), which generates a certificate/witness of its output, and a respective proof is generated from such a certificate in the interactive theorem prover.

Let us name just a few recent papers concerning integration of external tools into interactive theorem provers to increase their automation. For first-order theorem provers it is work done by Hurd [15, 16] and the Sledgehammer system [19, 23]. Weber and Amjad [25] integrated SAT solvers with HOL4, Böhme and Weber integrated the SMT solver Z3 with Isabelle/HOL [4].

Certificates for Squolem were described by Jussila et al. [17]. Other certificates formats were proposed too, an overview can be found in [21]. Authors of Squolem developed a stand-alone checker QBV for Squolem's certificates [17]. It is not surprising that QBV is much more efficient than our approach. On the other hand, QBV would have to become part of the trusted code if users of HOL Light wanted to use it to prove QBFs in HOL Light. Moreover, our system provides a check with much higher assurance than QBV thanks to the LCF-kernel.

3 Theory

3.1 Quantified Boolean Formulas

As usual, we assume that we have an infinite set of Boolean variables. The set of *literals* consists of all variables and their negations. We also extend the notion of negation to literals and identify $\neg\neg v$ with v. A *clause* is a disjunction of literals. A propositional formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses. We say that QBF is in *prenex normal form* if the formula is concatenation of a quantifier part and a quantifier-free part. Without loss of generality, we consider only closed QBF in prenex normal form with a propositional core in CNF – the formal definition is as follows:

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

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

where $n \geq 0$, each x_i is a Boolean variable, each Q_i is the universal $\forall$ or the existential $\exists$ quantifier, and ϕ is a propositional formula in CNF and all of its variables are among $x_1, \dots, x_n$.

$Q_1 x_1 \dots Q_n x_n$ is called a *quantifier prefix* and ϕ is called a *matrix*. We define an order $<$ over variables such that $x_1 < x_2$ if x_2 is in the scope of x_1. We call a variable the *intermost* or the *outermost* variable if it is maximal or minimal among all variables of formula (with respect to the order $<$) respectively. We say that x has a *quantification level* i if it is on the i-th position in the quantifier prefix.

If we consider a quantifier prefix as a finite sequence of quantifiers, we can define two relations on quantifier prefixes $\subseteq$ and $\preceq$. We define $\mathbf{Q}_1 \subseteq \mathbf{Q}_2$ if the quantifier prefix $\mathbf{Q}_1$ is a subsequence of the quantifier prefix $\mathbf{Q}_2$, and $\mathbf{Q}_1 \preceq \mathbf{Q}_2$ if the sequence of the variables in $\mathbf{Q}_1$ is a subsequence of the sequence of the variables in $\mathbf{Q}_2$. In other words, $\mathbf{Q}_1 \preceq \mathbf{Q}_2$ if we omit symbols for quantifiers (only variables left) in $\mathbf{Q}_1$ and $\mathbf{Q}_2$, and then we ask if the former is a subsequence of the latter.

The semantics $[\![f]\!]$ of closed QBF f is defined recursively by expanding the outermost variable x: $[\![\forall x . \phi]\!] = [\![\phi[x \mapsto 1] \wedge \phi[x \mapsto 0]]\!]$, and similarly $[\![\exists x . \phi]\!] = [\![\phi[x \mapsto 1] \vee \phi[x \mapsto 0]]\!]$. Where $\phi[x \mapsto c]$ denotes ϕ in which every free occurrence of x is replaced by the constant c. We call QBF *valid* or *invalid* if its semantics is 1 or 0 respectively.

3.2 QBF Models

Squolem's certificate of validity contains a model of the given QBF. The following general definition of a QBF model is a slightly improved definition used in [5, 17].

Definition 2 (Model). *Let $\Phi = Q_1 x_1 \dots Q_n x_n . \phi$ be a valid closed QBF in prenex normal form. Let V_i be the set of variables of Φ that have their quantification level less than or equal to i and let E_i and A_i be the sets of the existentially and universally quantified variables in V_i respectively, i.e., $E_i \cup A_i = V_i$. Let M be the set of functions*

$$M := \{ f_{v_k} : \{0,1\}^{k-1} \to \{0,1\} \mid v_k \in E_n \},$$

where each f_{v_k} depends exactly on the $k-1$ variables from V_{k-1}. M is said to be a model of Φ if

$$[\![\forall x_{i_1} \dots \forall x_{i_k} . \phi \, [x_{j_1} \mapsto f_{x_{j_1}}(x_1, \dots, x_{j_1 - 1}), \dots, x_{j_l} \mapsto f_{x_{j_l}}(x_1, \dots, x_{j_l - 1})]]\!] = 1,$$

where $\{x_{i_1}, \dots, x_{i_k}\} = A_n$ and $\{x_{j_1}, \dots, x_{j_l}\} = E_n$.

Functions f_v in the definition are nothing else than *witness functions*, which give witnesses based on (potentially) all preceding variables. As is noted in [17], it is also possible to let functions f_{v_k} only depend on the universally quantified variables of V_{k-1} but the authors of [17] claim that the stated definition may result in more compact representations of the functions f_{v_k}. In practice these functions are represented by propositional formulas.

3.3 Squolem's Certificates of Validity

Squolem's certificate format is described in detail in [18], we describe only the relevant part – certificates for valid formulas. The format is text based, variables are represented by positive integers and negated variables are denoted by negative integers, i.e., integer negation expresses propositional negation. The certificate describes a model of a given valid QBF by providing witness functions for existentially quantified variables.

The functions are defined gradually by *extensions*: definitions that introduce new Boolean functions defined by propositional formulas. Each new extension introduces a fresh variable which can be later used for referring to the newly defined Boolean function. It is a reasonable requirement not to allow an arbitrary propositional formula in the definition (which would be too hard to verify), therefore in [18] the authors allow just two special types:

If-Then-Else A new function

$$f(x, y, z) = \textit{if } l_1 \textit{ then } l_2 \textit{ else } l_3$$

is defined as If-Then-Else of three existing variables, where l_1, l_2 and l_3 are literals in variables x, y and z. This function is not actually denoted in the certificate by $f(x, y, z)$, but by a newly introduced fresh variable, let us say w. Then this type of Boolean functions can be represented by the following propositional formula: $w \Leftrightarrow (l_1 \wedge l_2) \vee (\neg l_1 \wedge l_3)$.

And A new function

$$f(x_1, \ldots, x_n) = \bigwedge_{i=1}^{n} l_i$$

is defined as a conjunction of the n literals l_i, which use the variables x_i. The number of conjuncts, n, can be an arbitrary non-negative integer. In the case when $n = 0$ the defined function is the Boolean constant 1. The newly defined function is also actually denoted by a fresh variable and its representation by a Boolean formula is straightforward in this case.

After definitions of all extensions there is a final line containing a list of pairs (v, l_v) for all existentially quantified variables in the given formula. We call the pairs as *witness assignments*. Here v is the existentially quantified variable and l_v is a literal representing an already defined extension, i.e., a possibly negated variable denoting an extension. The corresponding Boolean formula is obvious $v \Leftrightarrow l_v$. This list of witness assignments represents a model in the sense of Definition 2.

Let us conclude this section by an example of Squolem's certificate of validity for formula (1), which is translated into CNF as follows

$$\forall v_1\, \forall v_2\, \exists v_3.\, (v_3 \vee v_1 \vee \neg v_2) \wedge (v_3 \vee v_2 \vee \neg v_1) \wedge (v_1 \vee v_2 \vee \neg v_3) \wedge (\neg v_3 \vee \neg v_1 \vee \neg v_2)\ . \quad (2)$$

Squolem then generates the following certificate:

```
QBCertificate
E 4 A 1 -2 0
E 5 A -1 2 0
E 6 I 4 4 5
CONCLUDE VALID 3 6
```

Lines beginning with `E` represent extensions. These three lines represent three extensions, the first two lines define And extensions and the third line is the If-Then-Else extension. The corresponding Boolean formulas are as follows:

$$\begin{gathered} v_4 \Leftrightarrow v_1 \wedge \neg v_2 \\ v_5 \Leftrightarrow \neg v_1 \wedge v_2 \\ v_6 \Leftrightarrow (v_4 \wedge v_4) \vee (\neg v_4 \wedge v_5) \end{gathered}$$

The last line of the certificate says that the witness function for the existentially quantified variable v_3 is the extension v_6. It is not difficult to see that the extension v_6 together with extensions v_4 and v_5 defines the Boolean function XOR.

4 System Description

The overall structure of our system is as follows: first of all, we preprocess the given formula and serialize it into Squolem's input format. We run Squolem, which generates the corresponding certificate of validity. Then we parse this certificate and finally we construct a proof in HOL Light from information gained during the parsing. In this section we describe preprocessing of the given formula and especially construction of the proof. Other parts of our system contain non-interesting software engineering.

4.1 Preprocessing

As our system supports general closed QBFs and Squolem only works with formulas in CNF and prenex normal form, we had to incorporate a preprocessing phase. We implemented a naïve version of the transformation using conversions already available in HOL Light – `NNFC_CONV`, `CNF_CONV` and `PRENEX_CONV`. The transformation may cause an exponential blowup of the formula. More sophisticated conversions could be implemented as well, but because the main focus of this paper is on proof reconstruction (and our benchmark problems are already in prenex CNF), such techniques are beyond the scope of this paper.

The second preprocessing step that was incorporated is renaming of all variables according to the same scheme. We use the scheme `v_i` where `i` is a number representing quantification level of the variable. The scheme provides a uniform way of mapping variables to integers and vice versa, which is useful for text based communication with Squolem (i.e., serializing input and parsing certificates) and in data structures involving variables.

Thus our preprocessing makes the theorem $\vdash \Phi^* \Leftrightarrow \Phi$ where Φ is the original formula and Φ^* is the preprocessed one, which is in the form $Q_1 v_1 \ldots Q_n v_n . \phi$. Our goal is to prove $\vdash \Phi^*$ as a HOL Light theorem given a Squolem's certificate of its validity. The original formula is then trivially inferred by the `EQ_MP` inference.

4.2 Validating Squolem's Model

The question is how to represent the model contained in the given Squolem certificate. We represent a model as the conjunction of the corresponding Boolean formulas of all extensions and witness assignments. Let us denote this term by $\mathfrak{M}$, and call it a *model term*.

For the certificate of formula (1) the model term is defined as follows

$$\mathfrak{M} = (v_4 \Leftrightarrow v_1 \wedge \neg v_2) \wedge (v_5 \Leftrightarrow \neg v_1 \wedge v_2) \wedge (v_6 \Leftrightarrow (v_4 \wedge v_4) \vee (\neg v_4 \wedge v_5)) \wedge (v_3 \Leftrightarrow v_6) \; .$$

Now we can show how to verify the given model. Let us consider the following formula

$$\mathfrak{M} \Rightarrow \phi \; . \tag{3}$$

We claim that the given model is really a model of Φ^* if and only if (3) is a propositional tautology. It is an easy observation that the value of each variable that represents a witness function is uniquely determined in every satisfying assignments of variables on which the function depends. Let us suppose that (3) is not a propositional tautology then the negation of (3) $\mathfrak{M} \wedge \neg\phi$ has a satisfying assignment. This assignment uniquely determines values of existentially quantified variables in ϕ, but does not satisfy ϕ. Thus we find a counterexample that witnesses that the given model is not actually a model of Φ^*. On the other hand, if (3) is a propositional tautology then every satisfying assignment of $\mathfrak{M}$ has to satisfy ϕ.

We prove (3) by calling an external SAT solver. For this we followed Weber and Amjad [25], who integrated external SAT solvers zChaff and MiniSat with HOL theorem provers including HOL Light. In order to prove a formula to be a propositional tautology they negate it and run a SAT solver. If the formula is really a tautology then there is no satisfying assignment of the negated formula and the SAT solver produces a resolution proof of $\bot$. If we replay the resolution proof in the interactive theorem prover, we get our formula as HOL Light's theorem:

$$\vdash \mathfrak{M} \Rightarrow \phi \; . \tag{4}$$

4.3 Adding Quantifiers

Let us denote the quantifier prefix of the formula Φ^* as $\mathbf{Q}$, thus we have the following equation: $\mathbf{Q} = Q_1 v_1 \dots Q_n v_n$. As was described in 3.3, each And and If-Then-Else extension defines a new fresh variable. We want to define *extended quantifier prefix* $\mathbf{Q_e}$ that correctly incorporates these new fresh variables into $\mathbf{Q}$. We follow [17] and quantify new variables existentially. An important question is how to order the new variables with respect to the variables in the original quantifier prefix – it is clear that they have to be put after the variables on which their extension function depends. But they can't be put too deep because then the corresponding function could depend on variables for which it should serve as a model. Therefore we put each new variable right after the variable with the highest quantification level in the extension function.

This method, however, still doesn't yield a fully correct quantifier prefix. There is a problem with witness assignments. If we have for example a pair (v_i, v_j), the value of v_i depends on v_j. But v_i is not a new fresh variable, it is an existentially quantified variable from the original quantifier prefix $\mathbf{Q}$. Therefore it generally doesn't have to be after the variable v_j. Let us consider an example where $\mathbf{Q} = \forall v_1 \exists v_2 \exists v_3$, and there are the extension $v_4 = v_1 \wedge v_3$ and the witness assignment $v_2 = v_4$. After we incorporate v_4, we have $\mathbf{Q_e} = \forall v_1 \exists v_2 \exists v_3 \exists v_4$. But a value of v_2 depends on a value of v_4, therefore v_2 has to be after v_4. Fortunately, it is logically correct to reorder quantifiers in the block of the same quantifiers, thus we can move v_2 after v_4 in our example. In general, we need to topologically sort each block of existential quantifiers according to their extension dependencies.[1]

Our next step is to prove the formula $\mathbf{Q_e}\mathfrak{M} \Rightarrow \mathbf{Q}\phi$. We prove it from (3) by sequential addition of quantifiers by the following three inferences, which we designed and implemented (see 5.1):

$$\frac{\vdash A \Rightarrow B}{\vdash (\forall x.\, A) \Rightarrow \forall x.\, B} \qquad \frac{\vdash A \Rightarrow B}{\vdash (\exists x.\, A) \Rightarrow B}\ (x \text{ not free in B}) \qquad \frac{\vdash A \Rightarrow B}{\vdash A \Rightarrow \exists x.\, B}$$

We go simultaneously through $\mathbf{Q_e}$ and $\mathbf{Q}$, in a bottom-up fashion, and in each step we use the first rule from the following list that matches:

- $\mathbf{Q} = \dots \exists v_i$ – The whole existential block in $\mathbf{Q}$ will be sequentially added by the third inference. Because $\mathbf{Q_e}$ was made from $\mathbf{Q}$, there has to be the corresponding block in $\mathbf{Q_e}$. It contains the same variables as the corresponding block in $\mathbf{Q}$ plus potentially some fresh variables from extensions. We add this block of $\mathbf{Q_e}$ sequentially by the second inference. The condition 'x not free in B' is satisfied because all common variables from the added blocks of $\mathbf{Q}$ and $\mathbf{Q_e}$ are bounded in B from the first step of this rule.
- $\mathbf{Q_e} = \dots \exists v_i$ – There is an existential block in $\mathbf{Q_e}$ that contains only fresh variables from extensions, therefore we can add this block in $\mathbf{Q_e}$ by the second inference. If the block contained a non-fresh variable, the first rule from this list would match.

[1] This is possible because Squolem never generates circular dependency between extensions and witness assignments.

– $\mathbf{Q_e} = \ldots \forall v_i$ and $\mathbf{Q} = \ldots \forall v_i$ – Let us note that the universally quantified variables in $\mathbf{Q_e}$ and $\mathbf{Q}$ have to be the same because we didn't change order of variables in the universally quantified blocks. Thus we add both quantifiers at once by the first inference.

After this juggling with quantifiers we have the following theorem:

$$\vdash \mathbf{Q_e}\mathfrak{M} \Rightarrow \mathbf{Q}\phi\,. \tag{5}$$

4.4 Proving The Quantified Model Term

If we were able to prove $\mathbf{Q_e}\mathfrak{M}$, we would accomplish our goal because we can derive $\vdash \mathbf{Q}\phi$ from (5) by a call of MP. And because $\mathbf{Q}\phi = \Phi^*$, we would be done.

We start with the proof of each extension and witness assignment Boolean formula. For each such a Boolean formula $v_k = \vartheta(v_{i_1}, \ldots, v_{i_l})$ we prove the following theorem

$$\vdash \forall v_{i_1} \ldots \forall v_{i_l}\, \exists v_k.\, v_k = \vartheta(v_{i_1}, \ldots, v_{i_l}) \tag{6}$$

by the following derivation

$$\dfrac{\dfrac{\dfrac{}{\vdash \vartheta(v_{i_1}, \ldots, v_{i_l}) = \vartheta(v_{i_1}, \ldots, v_{i_l})}\,\mathrm{REFL}_{\vartheta(v_{i_1},\ldots,v_{i_l})}}{\vdash \exists v_k.\, v_k = \vartheta(v_{i_1}, \ldots, v_{i_l}))}\,\mathrm{CHOOSE}_{\exists v_k.\, v_k = \vartheta(v_{i_1},\ldots,v_{i_l}),\ \vartheta(v_{i_1},\ldots,v_{i_l})}}{\vdash \forall v_{i_l}\, \exists v_k.\, v_k = \vartheta(v_{i_1}, \ldots, v_{i_l})}\,\mathrm{GEN}_{v_{i_l}}$$

$$\dfrac{\vdots}{\vdash \forall v_{i_1} \ldots \forall v_{i_l}\, \exists v_k.\, v_k = \vartheta(v_{i_1}, \ldots, v_{i_l})}\,\mathrm{GEN}_{v_{i_1}}.$$

It is very important to note that we ordered the variables $v_{i_1}, \ldots, v_{i_l}$ according to their quantification levels in $\mathbf{Q_e}$. Let $E_1, \ldots, E_N$ be all extension and witness assignment Boolean functions and let $\mathbf{Q}_1, \ldots, \mathbf{Q}_N$ be the quantifier prefixes that we get in expressions (6) for each Boolean function. Because we made $\mathbf{Q_e}$ in 4.3 so that each existential variable goes after the variables on which it depends and because we ordered the variables $v_{i_1}, \ldots, v_{i_l}$ in the same order as they appear in $\mathbf{Q_e}$, we get for all i that $\mathbf{Q}_i \preceq \mathbf{Q_e}$.

We can rewrite expressions (6) as $\mathbf{Q}_i E_i$ for each extension. Let us consider the following formula $(\mathbf{Q}_1 E_1) \wedge \cdots \wedge (\mathbf{Q}_N E_N)$. We show in the rest of this section that we are able to "lift" each $\mathbf{Q}_i$ in front of the big conjunction of E_i's in such a way that we get the following theorem

$$\vdash \mathbf{Q_e}(E_1 \wedge \cdots \wedge E_N)\,, \tag{7}$$

which is nothing else than $\vdash \mathbf{Q_e}\mathfrak{M}$.

We designed and implemented the following inference

$$\dfrac{\mathbf{Q}'A \qquad \mathbf{Q}''B}{\mathbf{Q}'''(A \wedge B)}\,\mathrm{LIFT},$$

which has the following property: if $\mathbf{Q}' \preceq \mathbf{Q_e}$ and $\mathbf{Q}'' \preceq \mathbf{Q_e}$, then $\mathbf{Q}''' \preceq \mathbf{Q_e}$. In addition, these two relations hold unconditionally: $\mathbf{Q}' \preceq \mathbf{Q}'''$ and $\mathbf{Q}'' \preceq \mathbf{Q}'''$. With LIFT it is almost possible (we derive $\mathbf{Q}^*$ instead of $\mathbf{Q_e}$) to derive (7) by $N-1$ calls of LIFT:

$$\dfrac{\dfrac{\dfrac{}{\vdash \mathbf{Q}_{N-2}E_{N-2}} \qquad \dfrac{\vdash \mathbf{Q}_{N-1}E_{N-1} \qquad \vdash \mathbf{Q}_N E_N}{\vdash \mathbf{Q}'(E_{N-1} \wedge E_N)}\,\text{LIFT}}{\vdash \mathbf{Q}''(E_{N-2} \wedge E_{N-1} \wedge E_N)}\,\text{LIFT} \qquad \dfrac{\ddots \qquad \vdots}{\vdash \mathbf{Q}^{'\cdots'}(E_2 \wedge \cdots \wedge E_{N-2} \wedge E_{N-1} \wedge E_N)}\,\text{LIFT}}{\vdash \mathbf{Q}^*(E_1 \wedge E_2 \wedge \cdots \wedge E_{N-2} \wedge E_{N-1} \wedge E_N)}\,\text{LIFT}$$

Now we finish a proof of (7). From the above written properties of LIFT it follows that $\mathbf{Q}^* \preceq \mathbf{Q_e}$. Because we have for every existentially quantified variable the corresponding extension[2], and because LIFT prefers existentially quantified variables during lifting (see next section), each $\mathbf{Q}_i$ contributes by exactly one existentially quantified variable into $\mathbf{Q}^*$. Therefore $\mathbf{Q}^*$ contains the same existentially quantified variables as $\mathbf{Q_e}$. From that follows that $\mathbf{Q}^* \subseteq \mathbf{Q_e}$. This generally does not have to be equality because some universally quantified variables from $\mathbf{Q_e}$ can be missing in $\mathbf{Q}^*$. Those are exactly the variables that weren't present in any extension, i.e., they are not free in $\mathfrak{M}$, and therefore they can be quite easily added in $\mathbf{Q}^*$. Our rule ADD_MISSING_UNIVERSALS does this job – it is a simple use of HOL Light's rewriting conversions:

$$\dfrac{\vdash \mathbf{Q}^*(E_1 \wedge \cdots \wedge E_N)}{\vdash \mathbf{Q_e}(E_1 \wedge \cdots \wedge E_N)}\,\text{ADD\_MISSING\_UNIVERSALS}$$

4.5 LIFT

The main goal of LIFT is to prove the following implication

$$\vdash (\mathbf{Q}'A \wedge \mathbf{Q}''B) \Rightarrow \mathbf{Q}'''(A \wedge B)\,. \tag{8}$$

If we have (8), it is straightforward to derive the conclusion of LIFT by a call of CONJ and MP.

Because $\mathbf{Q}' \preceq \mathbf{Q_e}$ and $\mathbf{Q}'' \preceq \mathbf{Q_e}$, all we need to do is to *merge* $\mathbf{Q}'$ and $\mathbf{Q}''$ together according to quantification levels. If we find items of $\mathbf{Q}'$ and $\mathbf{Q}''$ that have different quantifiers, we prefer existential quantifier. We start with $\vdash A \wedge B \Rightarrow A \wedge B$, which we derive by $\text{ASSUME}_{A \wedge B}$ and DISCH_ALL.

Then we go simultaneously through $\mathbf{Q}'$ and $\mathbf{Q}''$, in a bottom-up fashion, and perform merging using the following inferences, which we implemented:

$$\dfrac{\vdash (A \wedge B) \Rightarrow C}{\vdash ((\forall x.\, A) \wedge \forall x.\, B) \Rightarrow \forall x.\, C}$$

[2] If Squolem doesn't generate a witness function for some existentially quantified variable v_i, we add the following artificial extension $v_i \Leftrightarrow 1$.

$$\frac{\vdash (A \wedge B) \Rightarrow C}{\vdash ((\exists x.\, A) \wedge \forall x.\, B) \Rightarrow \exists x.\, C} \qquad \frac{\vdash (A \wedge B) \Rightarrow C}{\vdash ((\forall x.\, A) \wedge \exists x.\, B) \Rightarrow \exists x.\, C}$$

$$\frac{\vdash (A \wedge B) \Rightarrow C}{\vdash ((\forall x.\, A) \wedge B) \Rightarrow \forall x.\, C}\ x \notin B \qquad \frac{\vdash (A \wedge B) \Rightarrow C}{\vdash ((\exists x.\, A) \wedge B) \Rightarrow \exists x.\, C}\ x \notin B$$

$$\frac{\vdash (A \wedge B) \Rightarrow C}{\vdash (A \wedge \forall x.\, B) \Rightarrow \forall x.\, C}\ x \notin A \qquad \frac{\vdash (A \wedge B) \Rightarrow C}{\vdash (A \wedge \exists x.\, B) \Rightarrow \exists x.\, C}\ x \notin A$$

The notation '$x \notin B$' means 'x is not free in B'. For example, if we encounter two universal quantifiers with the same variables, we use the first rule. On the other hand, if we need to merge two universal quantifiers with different variables and the first variable has higher quantification level than the second, we use the fourth rule and so on.

It is an important observation that we cannot encounter two existential quantifiers with the same variable. As was discussed in 4.4, each $\mathbf{Q}_i$ contributes by exactly one existentially quantified variable and all these variables are different. If we encountered this two-existentials situation, it would be a problem because it generally doesn't hold that $((\exists x.\, A) \wedge \exists x.\, B) \Rightarrow \exists x.\, A \wedge B$.

5 Implementation and Evaluation

5.1 Implementation of Rules

HOL Light has a very simple kernel especially in comparison with HOL4 or Isabelle/HOL. Many rules are not included in HOL Light's kernel and they are derived from primitive rules, including for example the rule MP – modus ponens. It turns out to be one of the sources of inefficiency. We discussed three rules for adding quantifiers into (3) in 4.3. It is natural to implement the second rule by HOL Light's CHOOSE and the third rule by EXISTS.[3] We tried it but this approach turned out to be significantly slower than the following approach: We prove the following schematic theorems

$$\vdash (\forall x.\, A \Rightarrow B) \Rightarrow ((\forall x.\, A) \Rightarrow \forall x.\, B) \qquad \vdash (A \Rightarrow B) \Rightarrow (A \Rightarrow \exists x.\, B)$$

$$\vdash (\forall x.\, A \Rightarrow B) \Rightarrow (\exists x.\, A \Rightarrow B),\ x \text{ not free in } B,$$

and in every call of the corresponding rules from 4.3, we instantiate them properly and by MP derive the consequent. We used similar approach in the implementation of rules used in LIFT.

5.2 Alpha-Equivalence Optimization

After we implemented optimizations described in 5.1, performance was still poor. We did some profiling to gain a deeper insight into this problem, and made a quite surprising discovery. Our system spent 99.4 % of the time in HOL Light's

[3] Both of rules are for example implemented directly in the HOL4 kernel [14].

Table 1. Profiling results

	non-optimized		optimized	
function	relative time (%)	number of calls	relative time (%)	number of calls
`orda`	99.4	668974	16.63	668974
`ordav`	97.84	19786610	2.59	125146
`compare` and `==`	92.33	1225276114	13.67	951183

kernel function `alphaorder`. This function implements the order of HOL Light's terms with the property that alpha-equivalent terms are equal according to this order. This order is among others used to implement the simple test that two terms are alpha-equivalent. The test for alpha-equivalence is a common test in HOL Light's rules; for example, it is used in the MP rule.

The implementation of `alphaorder t1 t2` is as follows: go simultaneously through (up to bottom) the structure of `t1` and `t2` and compare recursively smaller parts. A list of pairs of alpha-equivalent bound variables is maintained during the traversal. This traversal is implemented in the function `orda`. If $\lambda x.\, s_1$ and $\lambda y.\, s_2$ are compared, then a new pair of alpha-equivalent variables (x, y) is added to the front of the list. If we need to compare two variables, we have to check the list of alpha-equivalent variables first, which is done in linear time. The comparison of variables is implemented in the function `ordav`.

This linear-time implementation is ineffective for formulas with many abstractions. Because the test for alpha-equivalence of two variables is linear, the test for the whole formula is quadratic. It seems that this is not a problem in normal use of HOL Light (i.e., if common formulas are used). But we work with formulas which have thousands of variables, and because we work only with closed formulas and each quantifier is encoded by a particular type of abstraction, our formulas have thousands of abstractions.

Our optimization is based on the observation of the problem that alpha-equivalence of two identical terms is still possibly quadratic because the pair (x, y) is added to the list even if x and y are identical variables. Thus our optimization is as follows: we detect if the list of alpha-equivalent variables contains pairs of identical variables. If so, we do not use this list. Thus comparison of two identical formulas is linear and not quadratic because all pairs of variables are only compared, and there is no need to go through the list in linear time. The complexity can be actually improved even more because if we do not take the list of alpha-equivalent variables into consideration, we can compare shared subterms only by comparing two pointers pointing to this shared subterm. And this pointer comparison is a constant time operation.

Thanks to this optimization we get a speed-up factor of 321.0 (see 5.3). Detailed profiling data can be seen in Table 1.

5.3 Run-Times

We performed a set of benchmarks to show performance of our implementation and feasibility of validation of Squolem's certificates in HOL Light. We used a

Table 2. Detailed evaluation results for the time limit of 60 seconds

instance name	qntfs.	vars.	clauses	extensions	non. opt. (s)	SAT (s)	model (s)	total (s)
Adder2-2-s	6	249	292	580	179.7	0.3	0.3	1.3
adder-2-sat	4	64	109	206	16.5	0.3	0.1	0.5
CHAIN12v.13	3	925	4582	1809	∞	3.6	2.0	30.7
CHAIN13v.14	3	1080	5458	2090	∞	4.6	2.6	45.6
comp.blif_0.10_1.00_0_1_inp_exact	3	307	844	4973	∞	4.9	30.6	59.1
counter_2	5	42	103	362	40.3	0.2	0.2	0.5
counter_e_2	5	50	123	740	395.1	0.3	0.5	1.3
counter_r_2	5	50	121	408	56.6	0.2	0.2	0.6
counter_re_2	5	58	141	639	228.6	0.3	0.4	1.1
impl02	5	10	18	22	0.0	0.0	0.0	0.0
impl04	9	18	34	42	0.1	0.0	0.0	0.0
impl06	13	26	50	62	0.4	0.0	0.0	0.1
impl08	17	34	66	82	0.7	0.1	0.0	0.1
impl10	21	42	82	102	1.1	0.1	0.0	0.2
impl12	25	50	98	122	1.9	0.1	0.0	0.2
impl14	29	58	114	142	3.0	0.1	0.0	0.2
impl16	33	66	130	162	4.0	0.1	0.1	0.3
impl18	37	74	146	182	6.6	0.1	0.1	0.4
impl20	41	82	162	202	7.5	0.2	0.1	0.5
k_d4_n-4	17	393	1312	3105	∞	4.9	7.2	26.7
k_dum_n-12	35	620	1594	2911	∞	3.9	5.7	30.1
k_dum_n-16	43	796	2062	3799	∞	9.1	9.5	50.5
k_dum_n-4	19	262	649	1152	1400.7	0.8	1.0	4.6
k_dum_n-8	27	444	1126	2023	∞	1.7	2.7	13.2
k_grz_n-4	17	317	902	1767	∞	1.9	2.3	10.5
k_grz_n-8	17	433	1413	3050	∞	3.9	7.6	29.4
k_path_n-12	29	876	2440	4235	∞	4.3	12.2	58.2
k_path_n-4	13	324	888	1464	2839.2	1.2	1.5	7.1
k_path_n-8	21	600	1664	2846	∞	2.4	5.4	26.6
k_ph_n-4	5	141	411	726	328.7	0.5	0.6	2.1
k_poly_n-4	29	330	743	1513	2956.5	1.4	1.6	7.3
k_poly_n-8	53	654	1475	3097	∞	4.4	6.5	31.1
k_t4p_n-4	27	624	1895	4058	∞	10.3	12.5	55.1
mutex-16-s	2	1378	1779	3523	∞	2.4	7.5	32.1
mutex-2-s	2	104	127	214	10.8	0.1	0.1	0.3
mutex-4-s	2	286	363	612	223.5	0.4	0.4	1.6
mutex-8-s	2	650	835	1652	∞	0.9	1.9	7.3
qshifter_3	2	19	128	128	4.2	0.2	0.1	0.3
qshifter_4	2	36	512	512	194.0	1.3	0.4	2.6
qshifter_5	2	69	2048	2048	∞	8.8	4.1	30.8
s27_d2_s	3	65	142	166	4.7	0.1	0.1	0.2
s298_d2_s	3	699	1895	1469	2526.8	1.5	1.2	12.6
s499_d2_s	3	950	2665	2093	∞	2.9	3.5	27.0
TOILET2.1.iv.4	3	37	99	89	0.8	0.1	0.0	0.1
tree-exa10-10	2	20	18	19	0.0	0.0	0.0	0.0
tree-exa10-15	2	30	28	29	0.1	0.0	0.0	0.0
tree-exa10-20	2	40	38	39	0.2	0.0	0.0	0.1
tree-exa10-25	2	50	48	49	0.3	0.0	0.0	0.1
tree-exa10-30	2	60	58	59	0.5	0.0	0.0	0.1
z4ml.blif_0.10_0.20_0_1_inp_exact	5	65	193	1087	1759.2	0.7	1.3	2.9
z4ml.blif_0.10_0.20_0_1_out_exact	3	61	185	1221	2480.7	0.8	1.5	3.3
z4ml.blif_0.10_1.00_0_1_inp_exact	3	66	200	546	155.1	0.3	0.3	0.9
z4ml.blif_0.10_1.00_0_1_out_exact	3	64	196	1219	2411.1	0.8	1.5	3.3

Table 3. Evaluation results for various time limits

time limit (s)	success rate (%)	average time (s)	quantifier blocks	variables	clauses
5	33	0.9	41	286	649
60	53	12	53	1378	5458
600	81	73	133	3015	17752
3000	94	248	133	11570	19663

similar methodology as in [17, 24]. The authors of Squolem conducted experiments on the *2005 fixed instance* and the *2006 preliminary QBF-Eval* data sets, in total 445 instances of QBFs [17]. We ran Squolem with the time limit of 600 seconds and the memory limit of 1 GB. Squolem solved 100 valid problems within the given limits. We ran our system on these 100 valid QBF problems.

All benchmarks were run on a Linux system with four AMD Phenom II X4 955 processors (3.2 GHz) and with 8 GB RAM. We set time limits to 5, 60, 600 and 3000 seconds and the memory limit to 1.5 GB RAM. We present our results for these time limits in Table 3. One can see in the table that we are able to solve more than half of our instances within the time limit of 60 seconds and the success rate is 94 percents for the time limit of 3000 seconds. Also one can see that we are able to solve instances with thousands of variables.

We have decided to show detailed evaluation results for the time limit of 60 seconds. We present our data in the same format as Weber [24] to allow easy comparison of the results. The data can be found in Table 2. The first column contains the name of a benchmark; the next three columns give a characterization of a formula by providing three size parameters of the formula – the number of the quantifier blocks, variables and clauses. The fifth column gives the size of Squolem's certificate measured by the number of the generated extensions. The next column contains the run-time of our system without the alpha-equivalence optimization. The symbol ∞ denotes the case when the time limit of 3600 seconds was exceeded.

The last three columns contain run-times for validation of Squolem's certificates in HOL Light. The SAT column tells how much time we spent by constructing the proof of (3) using the external SAT solver, i.e., by validating the model (see 4.2). The model column shows the run-time of proving the quantified model term (see 4.4). The last column finally contains the total run-time of our system for the given problem. All run-time columns are given in seconds and rounded to one decimal place. If we consider only instances for which we have data for non-optimized implementation, we get a speed-up factor of 321.0 for non-optimized vs. optimized implementation.

6 Conclusions and Future Work

We have developed and implemented a system that constructs proofs of valid QBFs from Squolem's certificates of validity. Our evaluation showed that this task is feasible – more than half of our benchmarks were solved within the time limit of 60 seconds. We had a similar experience with implementation as in [24, 25], namely that performance is very sensitive to used inferences and to implementation details of the inference kernel. We proposed an optimization in HOL Light's kernel concerning computation of alpha-equivalence of terms, and got a speed-up factor of 321.0. Our implementation is freely available from the following web address: `http://ktiml.mff.cuni.cz/~kuncar/squolem2hollight`.

As was discussed in detail in Section 1, our system has two main applications. First, our system increases the amount of automation of HOL Light, and

allows HOL Light's users to prove QBFs that are beyond the scope of the built-in tactics of HOL Light. Proving these formulas without our work would demand considerable human effort. Second, our approach can be used for validating correctness of Squolem's results because of HOL Light's small LCF-style kernel. A small bug was found and resolved in Squolem due to our work.

An alternative approach to using the LCF-style kernel directly is the use of reflection. This alternative approach requires implementation and a proof of correctness of a checker for Squolem's certificates in the prover's logic. Then this checker is run without producing any proof. In general, reflection provides better performance and still relatively high correctness assurances. To our best knowledge, there has not been done any work on reflectively verifying QBF solvers. There is also no support for reflection in HOL Light, namely one would have to integrate a reflection rule into HOL Light's kernel allowing it to trust the results of such a verified checker.

Some possible directions for future work are as follows: (i) There is still small room for further optimization, but probably not so radical as we presented. (ii) It is possible to implement our approach in other LCF-style interactive theorem provers, namely HOL4 and Isabelle/HOL. One can expect that implementation can differ because of variations in their kernels. (iii) Another direction is to continue in the general research of automation of interactive theorem provers, and integrate other systems. Integration of the system MetiTarski [1] seems to be the next challenging research task.

Acknowledgments. The author would like to thank John Harrison for proposing various optimizations in the code. This research was partially supported by SVV project number 263 314.

References

1. Akbarpour, B., Paulson, L.C.: MetiTarski: An Automatic Theorem Prover for Real-Valued Special Functions. J. Autom. Reasoning 44(3), 175–205 (2010)
2. Benedetti, M., Mangassarian, H.: QBF-Based Formal Verification: Experience and Perspectives. vol. 5, pp. 133–191 (2008)
3. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic Model Checking without BDDs. In: TACAS. LNCS, vol. 1579, pp. 193–207 (1999)
4. Böhme, S., Weber, T.: Fast LCF-Style Proof Reconstruction for Z3. In: Kaufmann, M., Paulson, L.C. (eds.) ITP. Lecture Notes in Computer Science, vol. 6172, pp. 179–194. Springer (2010)
5. Büning, H.K., Zhao, X.: On Models for Quantified Boolean Formulas. In: Lenski, W. (ed.) Logic versus Approximation, Lecture Notes in Computer Science, vol. 3075, pp. 18–32. Springer Berlin / Heidelberg (2004)
6. Dershowitz, N., Hanna, Z., Katz, J.: Bounded Model Checking with QBF. In: Bacchus, F., Walsh, T. (eds.) SAT. Lecture Notes in Computer Science, vol. 3569, pp. 408–414. Springer (2005)
7. Giunchiglia, E., Narizzano, M., Tacchella, A.: QBF Reasoning on Real-World Instances. In: Hoos, H.H., Mitchell, D.G. (eds.) SAT (Selected Papers. Lecture Notes in Computer Science, vol. 3542, pp. 105–121. Springer (2004)

8. Gordon, M.: From LCF to HOL: a short history. In: Plotkin, G.D., Stirling, C., Tofte, M. (eds.) Proof, Language, and Interaction. pp. 169–186. The MIT Press (2000)
9. Harrison, J.: Binary Decision Diagrams as a HOL Derived Rule. Comput. J. 38(2), 162–170 (1995)
10. Harrison, J.: Towards Self-verification of HOL Light. In: Furbach, U., Shankar, N. (eds.) IJCAR. Lecture Notes in Computer Science, vol. 4130, pp. 177–191 (2006)
11. Harrison, J.: The HOL Light theorem prover (2010), `http://www.cl.cam.ac.uk/~jrh13/hol-light/`
12. Harrison, J., Slind, K., Arthan, R.: HOL. In: Wiedijk, F. (ed.) The Seventeen Provers of the World. Lecture Notes in Computer Science, vol. 3600, pp. 11–19. Springer (2006)
13. Harrison, J., Théry, L.: A skeptic's approach to combining HOL and Maple. Journal of Automated Reasoning 21, 279–294 (1998)
14. HOL contributors: HOL4 Kananaskis 6 source code (2010), retreived February, 6 2011 from `http://hol.sourceforge.net`
15. Hurd, J.: An LCF-Style Interface between HOL and First-Order Logic. In: Voronkov, A. (ed.) CADE. Lecture Notes in Computer Science, vol. 2392, pp. 134–138. Springer (2002)
16. Hurd, J.: First-Order Proof Tactics in Higher-Order Logic Theorem Provers. In: Archer, M., Vito, B.D., Muñoz, C. (eds.) Design and Application of Strategies/Tactics in Higher Order Logics (STRATA 2003). pp. 56–68. No. NASA/CP-2003-212448 in NASA Technical Reports (Sep 2003)
17. Jussila, T., Biere, A., Sinz, C., Kröning, D., Wintersteiger, C.: A First Step Towards a Unified Proof Checker for QBF. In: Marques-Silva, J., Sakallah, K. (eds.) Theory and Applications of Satisfiability Testing – SAT 2007, Lecture Notes in Computer Science, vol. 4501, pp. 201–214. Springer Berlin / Heidelberg (2007)
18. Kröning, D., Wintersteiger, C.: A file format for QBF certificates (2007), retreived February, 6 2011 from `http://www.cprover.org/qbv/download/qbcformat.pdf`
19. Meng, J., Paulson, L.C.: Translating Higher-Order Clauses to First-Order Clauses. J. Autom. Reasoning 40(1), 35–60 (2008)
20. Meyer, A., Stockmeyer, L.: Word Problems Requiring Exponential Time. In: Proc. 5th ACM Symp. on the Theory of Computing. pp. 1–9 (1973)
21. Narizzano, M., Peschiera, C., Pulina, L., Tacchella, A.: Evaluating and certifying QBFs: A comparison of state-of-the-art tools. AI Commun. 22(4), 191–210 (2009)
22. Otwell, C., Remshagen, A., Truemper, K.: An Effective QBF Solver for Planning Problems. In: Arabnia, H.R., Joshua, R., Ajwa, I.A., Gravvanis, G.A. (eds.) MSV/AMCS. pp. 311–316. CSREA Press (2004)
23. Paulson, L.C., Susanto, K.W.: Source-Level Proof Reconstruction for Interactive Theorem Proving. In: Schneider, K., Brandt, J. (eds.) TPHOLs. Lecture Notes in Computer Science, vol. 4732, pp. 232–245. Springer (2007)
24. Weber, T.: Validating QBF Invalidity in HOL4. In: Kaufmann, M., Paulson, L. (eds.) Interactive Theorem Proving, Lecture Notes in Computer Science, vol. 6172, pp. 466–480. Springer Berlin / Heidelberg (2010)
25. Weber, T., Amjad, H.: Efficiently checking propositional refutations in HOL theorem provers. Journal of Applied Logic 7(1), 26–40 (2009), special Issue: Empirically Successful Computerized Reasoning