A First Step Towards a Unified Proof Checker for QBF

Toni Jussila[1], Armin Biere[1], Carsten Sinz[2], Daniel Kröning[3], and Christoph M. Wintersteiger[3]

[1] Formal Models and Verification, Johannes Kepler University, Linz
[2] Wilhelm-Schickard-Institute for Computer Science, University of Tübingen
[3] Computer Systems Institute, ETH Zürich

Abstract. Compared to SAT, there is no simple concept of what a solution to a QBF problem is. Furthermore, as the series of QBF evaluations shows, the QBF solvers that are available often disagree. Thus, proof generation for QBF seems to be even more important than for SAT. In this paper we propose a new uniform proof format, which captures refutations and witnesses for a variety of QBF solvers, and is based on a novel extended resolution rule for QBF. Our experiments show the flexibility of this new format. We also identify shortcomings of our format and conjecture that a purely resolution based proof calculus is not powerful enough to trace the most efficient solvers.

1 Introduction

The decision problem for the logic of Quantified Boolean Formulas (QBF) is the canonical PSPACE-complete problem. A vast number of problems can succinctly be formulated in QBF. Every finite two-player game can be modeled in QBF [1, 2]. A multitude of AI planning [3–5], and modal logic problems [6] can be formulated in QBF, but also unbounded and bounded model checking for finite-state systems [7–9], as well as other formal verification problems [10, 5].

There exist many different flavors of QBF solvers, based on DPLL [11–16], Q–Resolution [17], BDDs [18], Skolemization [19] or Hyper Binary Resolution [20]. However, state-of-the-art QBF solvers are not yet reliable enough. The validity of many hard instances at recent QBF competitions had to be 'guessed' by means of a majority vote of the contestants, and often the solvers disagree [21].

Certificates for the decision problem of propositional logic (SAT) are easy to define. For satisfiable instances, solvers provide a satisfying assignment, and in case of an unsatisfiable instance, a resolution proof is computed. In both cases, the output can be verified easily by means of efficient (polynomial) and easy-to-inspect proof checkers. The satisfying assignment, or the resolution proof, correspond to a *certificate* of the correctness of the result. In case of SAT, these certificates serve many additional purposes: for example, satisfying assignments are often used as counterexamples. Resolution proofs are often used as an input for other algorithms. As an example, the computation of Craig interpolants relies on a resolution proof.

A certificate in the context of QBF refers to a *proof* of validity or invalidity of a formula. As in the case of unquantified SAT, certificates can serve dual purposes: first, they establish trust in the correctness of the result. The second motivation is to use certificates as input to other algorithms. When formulating a two–player game as a QBF, we wish to determine if there exists a winning strategy. If so, we are also interested in the strategy itself. Besides knowing that we *can* win the game, we also want to know *how* to win. The same holds for invalid formulas: not only do we want to know that a formula is invalid, we also want to find out *why* there is no solution and we wish to convince ourselves of that fact in a, preferably, concise way.

As shown by Tseitin [22], allowing definitions of fresh variables in Boolean formulas can exponentially shorten refutations. In this paper, we extend Tseitin's result to the quantified setting. We apply it in our proposal for a certificate format, which allows variable definitions of predefined structure to simplify the extension of QBF solvers with certificate generation code.

To the best of our knowledge, there exist only two suggestions for QBF certificates: One in the form of an "inference log", from which a BDD–based model or a refutation for a QBF can be reconstructed [23], and a method to generate unsatisfiable cores from traces of search–based solvers [24]. (In the QBF setting—similar to propositional logic—an unsatisfiable core is a subset of the clauses of a QBF formula in prenex normal form, which is still unsatisfiable.)

The first approach probably does not scale well because of the growth of the BDDs. For some instances it takes considerably more time to reconstruct the model from the inference log than it took to generate the model in the first place [23]. The inference log that is provided can serve as a refutation trace. However, due to five different inference strategies that the solver can choose, there are many different kinds of instructions in the inference logs. Among them are context switches between inference styles, explicit and symbolic variants of inference rules, such as resolution, substitution, and assignment, but also rollback and commit operations to undo earlier operations, as well as "other control information" [23]. This set of instructions, tailored to keep the overhead of the solver as small as possible, results in the need for a heavyweight proof checker.

The second approach provides only an unsatisfiable core and depends on the particular QBF solver used. In essence, both approaches just "trace", what the solver is doing and are thus far from a unified format that could be used to certify the computation of QBF solvers based on totally different algorithms.

However, our proposal is only a first step towards a uniform format. It clearly lacks the ability to capture important features of certain QBF solvers, such as *long distance resolution* [12], and *expansion* [17, 19]. We conjecture that the extension rule is not enough to linearly trace the proof process of such solvers.

Our extension rule is described in Section 2, and we discuss how to apply it to the major QBF solving algorithms. In order to evaluate the overhead of generating certificates, we extend three different solvers with this capability.

These implementations are described in Sec. 3. We provide the results of the evaluation in Sec. 4.

2 Theory

We consider closed Quantified Boolean Formulas (QBF) in prenex normal form. Thus, a formula is the concatenation of a quantifier prefix and a matrix of clauses. Let V be an infinite set of variables and let $\Omega\colon V \to \{\exists, \forall\}$ be a function that marks variables either as existential ($\exists$) or as universal ($\forall$). We define an order $<$ over V such that $x_1 < x_2$ for $x_1, x_2 \in V$ iff x_2 is in the scope of x_1, i.e., 'larger' variables are in the scope of 'smaller' variables.

The set of literals contains all variables and their negations $\neg v$ with $v \in V$. As usual, we extend the notion of negation to literals and identify $\neg\neg v$ with v. We also extend the order $<$ and Ω to literals in the natural way (without ordering the two literals of a variable). A *clause* is a disjunction of literals. A clause is trivial if it contains a literal and its negation. An empty clause is a clause without literals. A quantified formula in conjunctive normal form (CNF) is a conjunction of clauses. We assume the quantifier prefix Ω to be implicitly given, and do not mention it explicitly. In the following we just use the term formula to denote a quantified formula in CNF. A formula without clauses is called the empty formula.

A variable of a formula is called the innermost (resp. the outermost) variable if it is maximal (resp. minimal) among all variables of the formula with respect to the order $<$. The semantics $[\![f]\!]$ of a formula f is defined recursively by expanding the outermost variable x of a formula f as follows. If $\Omega(x) = \exists$ then define $[\![f]\!]$ as $[\![f\{x \leftarrow 0\}]\!] \vee [\![f\{x \leftarrow 1\}]\!]$, where the cofactor $f\{x \leftarrow c\}$ of f is f in which every occurrence of x is replaced by the boolean constant c. More precisely, for $c = 0$, clauses containing $\neg x$ are deleted and x is removed from all clauses, and similarly for $c = 1$. If $\Omega(x) = \forall$, let $[\![f]\!] = [\![f\{x \leftarrow 0\}]\!] \wedge [\![f\{x \leftarrow 1\}]\!]$. Note that empty clauses (resp. formulas) are equivalent to the boolean constant 0 (resp. 1) and are thus invalid (resp. valid).

Two non-trivial clauses C and D can be *resolved* iff C contains a literal l and D its negation $\neg l$. The result of a resolution is the *resolvent* clause, which is obtained as a disjunction of all literals in C and D except l and $\neg l$. This is the same definition as for unquantified formulas. If a clause C contains a universal literal l that is larger than all existential literals in C, then it can be removed from C. The process of removing literals according to this rule is called *forall-reduction* [25, 17] and the result obtained from repeated application until no more literals can be removed is called the *forall-reduct* of C. A *Q-resolution* step consists of a resolution step followed by forall-reduction of the resolvent.

The calculus described above is able to simulate the resolution-based, sound, and complete refutation calculus of [25]. This also allows tracing refutation proofs of QBF solvers based on DPLL [15] efficiently, even with further optimizations such as trivial falsity [15], SAT and QBF based learning [5, 12, 16], and hyper binary resolution [20].

We conjecture that in order to trace other algorithms – for instance, algorithms that are structural or BDD-based – a much stronger proof system is necessary. In order to obtain such a stronger proof system, which is one of our main contributions in this paper, we add the following quantified extension rule. It is an adaptation of the classical extension rule [22] to the quantified setting.

Definition 1 (Quantified Extension Rule). *Let y be a fresh variable, which does not occur in formula f, and let g be a formula that may only contain y and variables in f. Furthermore, we demand that for any assignment to variables in f there exists an assignment to y that satisfies g. We also require $\Omega(y) = \exists$ and $z \leq y$ for all variables z in g. Then the Quantified Extension Rule extends f by g, i.e. it adds g conjunctively to f.*

We then call y in Def. 1 a *defined variable* and g a *definition* for y with respect to f. Note that g does not need to enforce a functional dependency of y on other variables in g. As a relation, g has to be total, e.g., it cannot define a partial function, but it can be non-deterministic, e.g., the value of y does not need to be unique. This is a slight generalization of the classical extension rule for propositional logic [22].

In adapting the extension rule to the QBF setting, the crucial question is where to "put" new variables, e.g., how defined variables are ordered with respect to variables that already occur in the formula. It seems intuitive that new variables can only be existential. It is also clear that they cannot be moved further out than the innermost variable on which they depend. In the experimental section we show that we actually need this freedom to move defined variables as far out as possible. Keeping them in the innermost existential scope, as for instance in the Tseitin encoding of a non–CNF QBF formula, is insufficient.

To enforce that proofs are polynomially checkable, one can fall back to the original idea of Tseitin [22] and restrict the set of functions that can be defined and the way they are encoded in to clauses. This is what we suggest to use in practice. Useful functions are the constants, equality, negation, if-then-else and conjunction. Conjunction is sufficient:

Definition 2 (Restricted Quantified Extension Rule). *Let l, r be two literals over variables in the formula f, and y be a fresh variable, which does not occur in f. Let g be the conjunction of the following 3 clauses $(\overline{y} \vee l)$, $(\overline{y} \vee r)$, and $(y \vee \overline{l} \vee \overline{r})$. We also require $\Omega(y) = \exists$ and $z \leq y$ for all variables z in g. The formula f can be* extended *by adding g.*

The soundness of the restricted rule follows from the soundness of the generic rule, which is proved next. It turns out that this rule is enough to produce proofs from refutations of our BDD-based QBF solver EBDDRES in linear time. Refer to Sec. 4 for details.

For the proof of the following theorem we need the distributivity of substitution (resp. cofactoring) over boolean operators, which we formulate for conjunctions as follows without proof:

Lemma 1. $(f \wedge g)\{x \leftarrow c\} \equiv f\{x \leftarrow c\} \wedge g\{x \leftarrow c\}$

Note that the post-fix operator for substitution has greater binding power than boolean operators. The right hand side of the equivalence in the Lemma is thus read as $(f\{x \leftarrow c\}) \wedge (g\{x \leftarrow c\})$. In the following we will omit parenthesis as in Lemma 1 whenever possible. The next theorem shows that adding definitions is sound and does not change the semantics of a formula.

Theorem 1 (Soundness of Quantified Extension Rule). *Let g be a definition for y with respect to f. Then $[\![f]\!] = [\![f \wedge g]\!]$.*

Proof. The proof is by induction on the number of variables in $f \wedge g$. Let x be the outermost variable in $f \wedge g$. First assume that x is different from y, the variable defined by g, and $\Omega(x) = \exists$. The definitions and the lemma imply:

$$\begin{aligned}
[\![f \wedge g]\!] &= [\![(f \wedge g)\{x \leftarrow 0\}]\!] \vee [\![(f \wedge g)\{x \leftarrow 1\}]\!] \\
&= [\![f\{x \leftarrow 0\} \wedge g\{x \leftarrow 0\}]\!] \vee [\![f\{x \leftarrow 1\} \wedge g\{x \leftarrow 1\}]\!] \\
&= [\![f\{x \leftarrow 0\}]\!] \vee [\![f\{x \leftarrow 1\}]\!] \;=\; [\![f]\!]
\end{aligned}$$

In the next to last step we have to apply the induction hypothesis twice for the definitions $g\{x \leftarrow c\}$ with respect to $f\{x \leftarrow c\}$. The second case with $\Omega(x) = \forall$ is identical, except that all the disjunctions '$\vee$' are replaced by conjunctions '$\wedge$'.

In the base case we assume that $x = y$. From the definition of the extension rule, we know that $\Omega(x) = \exists$, and that x does not occur in f. Since all variables z in g have to be smaller or equal to y, g is either constant or only contains y as variable. Therefore g is either equivalent[4] to the boolean constant 1, to the literal $\neg x$, or to the literal x. Otherwise, it is impossible to satisfy g by assigning a value to x. If $g \equiv 1$, then $[\![f \wedge g]\!] = [\![f \wedge 1]\!] = [\![f]\!]$. Without loss of generality, assume $g \equiv x$. Then

$$\begin{aligned}
[\![f \wedge g]\!] &= [\![(f \wedge g)\{x \leftarrow 0\}]\!] \vee [\![(f \wedge g)\{x \leftarrow 1\}]\!] \\
&= [\![f\{x \leftarrow 0\} \wedge g\{x \leftarrow 0\}]\!] \vee [\![f\{x \leftarrow 1\} \wedge g\{x \leftarrow 1\}]\!] \\
&= [\![f \wedge g\{x \leftarrow 0\}]\!] \vee [\![f \wedge g\{x \leftarrow 1\}]\!] \\
&= [\![f \wedge 0]\!] \vee [\![f \wedge 1]\!] \;=\; [\![f]\!]
\end{aligned}$$

□

Expansions as in Quantor [17] and long distance resolution as in Quaffle [12] allow some kind of reasoning across quantifier alternations. With expansions it is in principle possible to resolve clauses on outer variables that stem from different copies of the expanded clauses. Similarly, long distance resolution allows to compactly capture in a learned clause the effect of propagating information from an outer existential scope through an universal quantifier into an inner existential scope and back to the outer existential scope.

Currently we do not know how to efficiently simulate expansions and long distance resolution in our proof format. We conjecture that an even stronger proof system is necessary for this purpose.

[4] with respect to equivalence of propositional unquantified formulas [16].

Definition 3 (Model). *Let V_i be the set of variables in a formula that have a quantification level less than or equal to i and let E_i and A_i be the sets of existentially resp. universally quantified variables in V_i, i.e., $E_i \cup A_i = V_i$. A model M of the formula is then a set of functions*

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

where every f_{v_k} depends exactly on the $k-1$ variables from V_{k-1}.[5]

This definition is essentially the same as the one used by Kleine Büning [26]. It is also used in sKizzo [27] to certify satisfiable resp. valid instances. The functions f_v are also called *Skolem-functions*. Using our extension rule we can provide a model as a set of extensions to the formula by defining new variables representing the value of the Skolem functions. Our certificate thus contains a list of pairs of the form (v, f_v) where f_v is the fresh variable encoding the Skolem-function for v. With this approach, we have implemented model generation for two solvers, Ebddres and Squolem, presented in detail in Sect. 3. A general discussion for model generation with the extension rule for different QBF solvers is presented below.

As discussed above, new variables have to be ordered carefully. If they are simply quantified in the innermost scope, the corresponding function could depend on variables with higher quantification level than the one that it serves as a model for. For example, in a formula with the quantification sequence $\exists e \forall a \exists g$, the newly defined variable g may depend on the value of a, which may turn the defined function into an invalid model. Therefore our proof checker orders extension variables as low as possible, i.e., right after the variable with the highest quantification level occurring in the extension function. Checking that a model function does not depend on higher quantification levels is then trivial again.

However, checking the *validity* of a model according to Def. 3 is co-NP complete [26, 28]. In general, our approach is to check each clause individually to be tautological, leveraging incremental SAT solver technology. This can be achieved by keeping the model functions in the SAT solver and adding the negation of a clause, i.e., the negations of the clauses literals, as assumptions. The resulting problem must then be unsatisfiable, which means that it is impossible to unsatisfy the given clause with the model provided. It is possible to provide a refutation proof for each of those sub-problems (or, alternatively, for the whole formula instead of separate clauses). The complexity of checking this (annotated) model for validity is then polynomial in the size of the certificate. So far, our certificates only contain the Skolem-functions. In future, we will study how providing a refutation for each clause in the model affects the model generation time and model size, as well as the model verification time.

In Skolemization-based solvers constructing a model is easy, since the solvers basic strategy is to construct a model; it just has to dump the Skolem-functions

[5] It is also possible to let the f_{v_k} only depend on the universally quantified variables of V_{k-1}. However, our definition may result in more compact representations of the functions f_{v_k}.

it generates. DPLL- and search-based solvers split on variables using different strategies. Assuming the formula is valid, the solver always encounters either unit clauses, and propagates this information, or splits on a new variable. Whenever a unit clause (l) occurs under some variable assignment α to other variables, we can read this as $\alpha \Rightarrow l$, which corresponds to one entry in the function table of the model function for the variable of l. This information can immediately be dumped as $x_i = a_1 \wedge ... \wedge a_n$, where x_i is a fresh variable and the a_i are the literals from α. When the evaluation is finished, the final model-function is $f_v = x_1 \vee \cdots \vee x_n$.

In Q-Resolution-based solvers, particularly [16, 20], one can interpret any resolution between two clauses c_1 and c_2 into a resolvent as the generation of a conflicting clause. For instance, if c_1 contains a literal x and c_2 contains $\neg x$, then $\neg(c_1 \backslash \{x\}) \wedge \neg(c_2 \backslash \{\neg x\})$ may not happen, because x would have to be 1 and 0 at the same time to satisfy both clauses. One strategy to record a model is to start with overspecified functions and to refine them whenever resolution is applied. BDD-based solvers implementing the bucket algorithm [18] (and also EBDDRES) store intermediate BDDs for variables to be eliminated. From these, it is possible to dump Skolem-functions using the extension rule. The procedure for EBDDRES is presented in more detail in Sect. 3.

An alternative way of providing a model is to provide a refutation for the negation of a valid formula (which would then be invalid). Experiments that negate a formula by translation of the resulting DNF back to a CNF using the Tseitin-transformation have shown that this approach is infeasible. All of the problems that could be solved within 600 seconds by three different solvers (QUANTOR, SKIZZO, SQUOLEM) could either not be solved within the same time when inverted, or took considerably more time to solve.

3 Implementation

We implemented SQUOLEM, a *new* skolemization-based solver, which generates both models and refutations. SQUOLEM eliminates quantifiers from the inside out by explicitly generating Skolem-functions for existential variables. For each variable that is about to be eliminated, it collects all clauses in which the variable occurs and interprets them as implications, e.g., $(a \vee b)$ is interpreted as $\neg a \rightarrow b$. The set of these implications essentially forms a function, by which the variable is replaced.

In case of conflicting implications, e.g., $\neg a \rightarrow b$ and $\neg a \rightarrow \neg b$, a clause stating that this case cannot happen, e.g., (a), is added. In terms of the resolution calculus, the conflict clause can directly be obtained as a resolvent from the two conflicting implications. The process is iterated until either a complete model has been constructed, or conflicting unit clauses occur. In the first case we output the model, in the second case we output a q-resolution trace constructed from the information about a clause's parents, which we record during model construction.

We have instrumented two other *already existing* solvers, (i) the BDD-based solver EBDDRES [29, 30] and (ii) the search-based solver QUAFFLE [12]. EBDDRES

produces both models and refutation traces whereas with QUAFFLE we are so far limited to refutation traces. Unfortunately, in case of QUAFFLE we also had to disable learning, since we cannot trace long distance resolution steps, as already discussed above. EBDDRES is a BDD-based QBF solver that eliminates variables starting from the innermost scope. In order to eliminate a variable x, EBDDRES first builds a conjunction of all the clauses containing x and then quantifies x using one standard BDD OR- resp. AND-operation if the variable is existential resp. universal. After all variables are eliminated, a constant BDD is obtained. For simplicity, the variable ordering for the BDDs is the reverse of the QBF variable order. Thus, root variables are always eliminated.

EBDDRES produces refutations by introducing a Tseitin variable for each BDD node. This Tseitin variable is defined to be an if-then-else gate. Given a BDD node n, let x be its variable and t, t_0, and t_1 the Tseitin variables introduced for n, its left child, and its right child, respectively. Then the definition of t is $t \Leftrightarrow (x\ ?\ t_1\ :\ t_0)$. Thus, if x is true (false), the Tseitin variable t_1 (t_0) has to be true. The refutation is constructed by first showing that the Tseitin variables for the root nodes of the BDDs for every original clause have to be true. Then the logical operations of the solving algorithm are traced until it is shown that the Tseitin variable for the constant BDD zero has to be true, a contradiction. All the details except for universal quantification can be found in [29, 30]. For universal quantification, the proof rule is as follows. Let x be a universal variable to be eliminated and t the variable corresponding to the root node of the BDD which is the conjunction of all the clauses containing x. We have derived that t must be true. The definition of t introduces (among others) the clause $(\neg t \vee x \vee t_0)$. Resolving this with (t) yields $(x \vee t_0)$ which, based on Def. 1, can be forall-reduced to (t_0) since t_0 is not in the scope of x. The proof for t_1 is similar.

EBDDRES produces models as follows. In the bucket algorithm, when variable x is to be eliminated, a BDD containing precisely all the constraints on x is built. Going from the root to the child where x is true (right child) gives another BDD that encodes all the valuations where x has to be true to satisfy all the constraints. If x is existential, then this is precisely a Skolem-function for x. Thus the certificate consists of definitions of the Tseitin variables for this BDD and the Skolem-function is set to be equivalent to Tseitin variable of the root (analogously, we could have chosen the negation of the left child).

For the search-based solver QUAFFLE, the refutation is constructed as follows. Assume (without loss of generality) that the innermost scope is existential. If the instance is unsatisfiable, both choices for the truth value of an existential variable lead to a conflict. For a universal variable, at least one choice leads to a conflict. Whenever a conflict is reached, a conflict clause can be derived.

We produce refutation proofs from these conflict clauses. Consider for instance the simple search tree example presented in Fig. 1 and let the solid lines denote paths to conflicts. Let the left (right) arrow from a node mean setting the truth value of a variable to false (true). Now, the leftmost path $(\neg x \wedge \neg y \wedge \neg z)$ leads to a conflict producing the conflict clause $(x \vee y \vee z)$. Similarly, the path $(\neg x \wedge \neg y \wedge z)$ produces the clause $(x \vee y \vee \neg z)$. We resolve these, and forall-reduce

the result $(x \vee y)$ and thus obtain the clause (x). Similarly, from the conflict clauses from the right-hand side (where x is true) we get $(\neg x)$ and resolving the two, the empty clause. The above representation is simplified. Our experimental work has revealed that in most cases not a complete conflict clause is obtained but one that subsumes it. The consequence of this is that it is possible to omit some resolution steps.

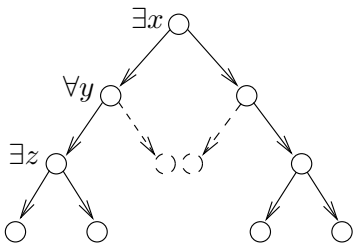

Fig. 1. QUAFFLE search tree.

If a formula has a large alternation depth, our proof-generation algorithm starts by resolving the literals from the innermost existential scope. Subsequently, the new clauses are forall-reduced to eliminate the enclosing universal scope. Then, the proof generation eliminates the second innermost existential scope and thus alternates between resolution and forall-reduction until the outermost scope.

Note that we have only been able to instrument QUAFFLE without learning. So far, it is an open problem how to handle long-distance resolution in the presented framework. Secondly, we are not able to create certificates, only refutation traces.

To verify the certificates that we extract, we implemented QBV, the Quantified Boolean Verifier. The implemented algorithm executes applications of our extension rules and q-resolutions, as listed in the certificate. The last statement in a certificate is a conclusion line that either provides the index of an empty clause (for refutations), or a list of equivalences of variables for a model. As suggested in [27], we check every clause separately against the model. For this purpose we use MINISAT (Version 1.14p) in incremental mode, i.e., we load the model into MINISAT and then add the negation of a clause as an assumption, which must result in an unsatisfiable problem. As stated earlier, this problem is in general Co-NP complete. In a future version we will provide support for refutations for every clause in the original formula, such that the complexity of this step becomes polynomial (with adverse effects on the certificate generation time).

4 Experimental Results

To show that certificate extraction is feasible we have conducted experiments on the 2005 fixed instance and the 2006 preliminary QBF-Eval[6] datasets, in total 445 test cases. We used EBDDRES, QUAFFLE, and SQUOLEM to generate certificates, and verified them using QBV.

The tests for EBDDRES and QUAFFLE were run on a cluster of Intel Pentium IV PCs (3 GHz) with 2 GB RAM each. We set a time limit of 600 seconds and a memory limit of 1 GB. EBDDRES is able to create a model for 80 instances

[6] `http://www.qbflib.org/`

and a refutation trace for 86 instances. For QUAFFLE (without learning), 26 instances could be refuted. We first compare the time required to solve an in-

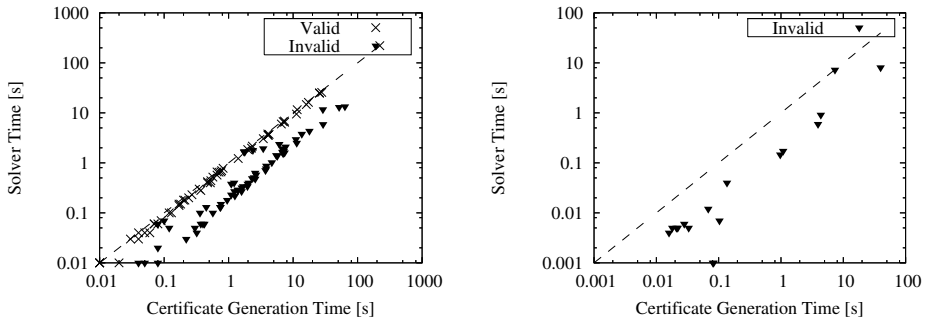

Fig. 2. Certificate Generation Time vs. Solver Time for (a) EBDDRES and (b) QUAFFLE

stance to the time needed to create the certificate. The results for EBDDRES and QUAFFLE are shown in Figs. 2(a) and 2(b), respectively. They show that for both solvers, generating a certificate takes longer than solving the instance. For EBDDRES the overhead for models is about 15% and for refutations 440%. This behavior is probably due to the fact that the refutation files for EBDDRES are so large, that merely saving them takes a lot of time. The same overhead is also observed in the propositional case [29, 30].

For QUAFFLE the overhead is even higher, 1440%. However, this result is not illustrative since we were only able to solve very few instances and these instances belong to only 6 families. For one instance, 'k_lin_p-4', the trace generation overhead is only 1.6%.

We also compare the time needed to validate certificates to the time it takes to generate them. The results for EBDDRES and QUAFFLE are shown in Figs. 3(a) and 3(b). Our experimental results show that, for EBDDRES, it takes on average longer to validate a certificate; again the overhead depends heavily on whether the instance is valid or invalid. Validating models, takes on average over 100 times longer than to generate them. QBV actually times out on 15 instances (time limit 600 seconds) where model generation is feasible. Refutations, on the other hand, can be verified quicker, the ratio is 5.4. For QUAFFLE, the trace validation times are neglible (on average < 0.04 seconds), as Fig. 3(b) shows. The observed behavior is in line with the fact that verifying a model is in general Co-NP complete whereas a resolution trace can be verified polynomially.

The tests for SQUOLEM were run on an Intel Xeon 3.0 GHz machine with 4 GB RAM. Out of 445 instances in the original dataset, our solver finishes on 142 within 600 seconds and a memory limit of 1 GB; 73 instances were found to be valid, 69 invalid.

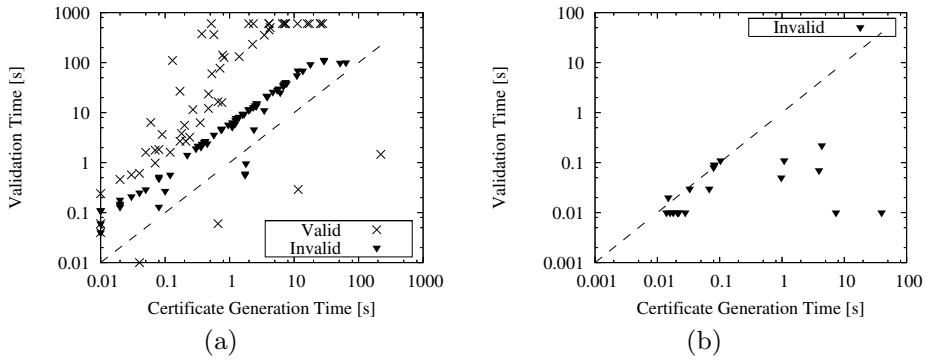

Fig. 3. (a) Certificate Generation Time vs. Validation Time EBDDRES and (b) QUAFFLE

First we compare the time to solve an instance to the time needed to generate a certificate. Naturally, in a purely Skolemization-based solver, there is a small difference in those times. Fig. 4(a) shows that the overhead of certificate generation is usually very small (on average 3.5% for models and 4.5% for refutations, with respect to solving time).

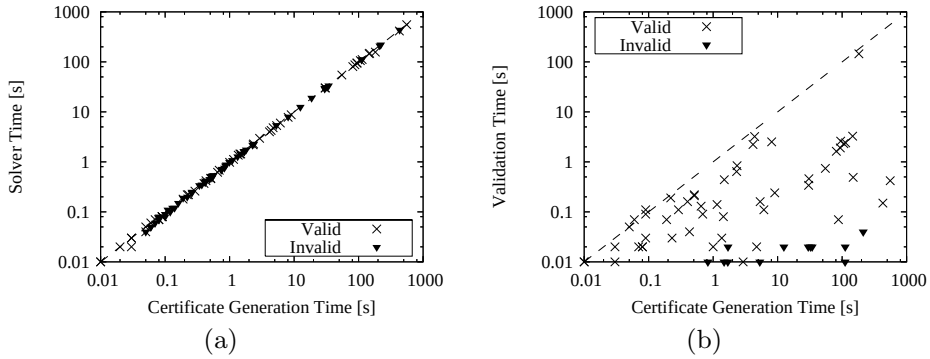

Fig. 4. (a) Certificate Generation Time vs. Solver Time and (b) Certificate Generation Time vs. Validation Time for SQUOLEM

As a second experiment, we investigate the time taken to validate a certificate. Fig. 4(b) shows that the time taken to validate a refutation is negligibly small (on average < 0.01 seconds); certificates of validity, on the other hand, take on average 2.38 seconds to validate. The main contribution to this time is the instance 'qshifter_7', which takes 144 seconds to validate. Excluding this instance, the average validation time is 0.4 seconds. The reason for this extraor-

dinary high runtime on just this single instance is that MiniSAT actually runs into a hard problem: 97% of the runtime is spent on the final model validation.

An interesting property of certificates is the size of the generated files. Table 1 gives an overview of the relative size of the generated certificates, with respect to the size of the original formula file. We present the traces in the original ASCII format as well as compressed with `gzip`. The data indicates that the certificates generated by EBDDRES are very large compared to QUAFFLE and SQUOLEM. Furthermore, its refutations (tracing complex BDD operations) are larger than its models. For SQUOLEM, on the other hand, the numbers suggest that models are considerably larger than refutations. The certificate format is not optimized for file size, but as the data indicates, the files compress well with `gzip`, if smaller files are required.

		Normal			Compressed		
Solver	Type	Best	Avg.	Worst	Best	Avg.	Worst
EBDDRES	Valid	<0.1	210.9	3304.3	<0.1	52.3	784.8
	Invalid	1.0	6857.6	145414.0	1.0	1594.3	31948.3
QUAFFLE	Valid	-	-	-	-	-	-
	Invalid	<0.1	3.4	17.0	<0.1	1.3	5.0
SQUOLEM	Valid	0.7	10.1	175.6	0.2	2.8	49.6
	Invalid	<0.1	3.3	55.0	<0.1	0.8	10.4

Table 1. Relative size of the generated certificates.

Finally, we compare our solvers with two state-of-the-art solvers, QUANTOR and sKIZZO. As QUAFFLE only produces refutation traces, we have no data on valid instances for this solver. Thus, we chose to provide the number of invalid instances that each solver is able to solve within 600 seconds and a 1 GB memory limit, from the total 205 in the data set. The numbers in Tbl. 2 indicate that our solvers together can solve about half as many instances.

	EBDDRES	QUAFFLE	SQUOLEM	QUANTOR	sKIZZO
Solved Instances	80	26	69	153	175
Solved Inst. (Union)	90			178	

Table 2. The number of solved instances in the test set.

5 Reference Implementations

EBDDRES, the instrumented version of QUAFFLE, and the exprimental data can be downloaded from `http://fmv.jku.at/ebddres`. SQUOLEM, the certificate

validator QBV, the experimental data, and a formal specification of the supported certificate format are available at `http://www.verify.ethz.ch/qbv`.

6 Conclusion

We show that it is possible to define a proof format for QBF that is applicable to a wide range of different QBF solvers. Nevertheless, important common features of other QBF solvers cannot be traced efficiently in this format. Even though it seems that existential expansion steps of structural QBF solvers can be traced with our format, we do not know how to trace universal expansion steps with the current set of rules. The same applies to long distance resolution. This is in contrast to SAT, where just adding the extension rule generates a proof calculus, which is as powerful as any other known propositional proof system.

References

1. Stockmeyer, L.J.: The polynomial–time hierarchy. TCS **3** (1976) 1–22
2. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
3. Rintanen, J.: Constructing conditional plans by a theorem-prover. Journal of Artificial Intelligence Research **10** (1999) 323–352
4. Otwell, C., Remshagen, A., Truemper, K.: An effective QBF solver for planning problems. In: MSV/AMCS, CSREA Press (2004) 311–316
5. Giunchiglia, E., Narizzano, M., Tacchella, A.: QBF reasoning on real–world instances. In: Proc. of SAT. LNCS 3542, Springer (2004) 105–121
6. Ladner, R.E.: The computational complexity of provability in systems of modal propositional logic. SIAM Journal on Computing **6**(3) (1977) 467–480
7. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: Proc. of TACAS. LNCS 1579, Springer (1999) 193–207
8. Dershowitz, N., Hanna, Z., Katz, J.: Bounded model checking with QBF. In: Proc. of SAT. LNCS 3569, Springer (2005) 408–414
9. Jussila, T., Biere, A.: Compressing BMC encodings with QBF. In: Proc. 4th Intl. Work. on Bounded Model Checking (BMC). To be published in ENTCS, Elsevier (2006)
10. Benedetti, M.: Experimenting with QBF-based formal verification. In: Proc. of the 3rd International Workshop on Constraints in Formal Verification (CFV). To be published in ENTCS, Elsevier (2005)
11. Plaisted, D.A., Biere, A., Zhu, Y.: A satisfiability procedure for quantified boolean formulae. Discrete Appl. Math. **130**(2) (2003) 291–328
12. Zhang, L., Malik, S.: Towards a symmetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. In: Proc. of CP. LNCS 2470, Springer (2002) 200–215
13. Letz, R.: Lemma and model caching in decision procedures for quantified boolean formulas. In: Proc. of TABLEAUX. LNCS 2381 (2002) 160–175
14. Giunchiglia, E., Narizzano, M., Tacchella, A.: QUBE: A system for deciding quantified boolean formulas satisfiability. In: Proc. of IJCAR. LNCS 2083, Springer (2001) 364–369

15. Cadoli, M., Giovanardi, A., Schaerf, M.: An algorithm to evaluate quantified boolean formulae. In: Proc. of AAAI/IAAI, AAAI (1998) 262–267
16. Samulowitz, H., Bacchus, F.: Using SAT in QBF. In: Proc. of CP. LNCS 3709, Springer (2005) 578–592
17. Biere, A.: Resolve and expand. In: Proc. of SAT. LNCS 3542, Springer (2004) 59–70
18. Pan, G., Vardi, M.Y.: Symbolic decision procedures for QBF. In: Proc. of CP. LNCS 3258, Springer (2004) 453–467
19. Benedetti, M.: Evaluating QBFs via symbolic skolemization. In: Proc. of LPAR. LNCS 3452. Springer (2005) 285–300
20. Samulowitz, H., Bacchus, F.: Binary clause reasoning in QBF. In: Proc. of SAT. LNCS 4121, Springer (2006)
21. Narizzano, M., Tacchella, A., Pulina, L.: Report of the third QBF solvers evaluation. JSAT **2** (2006) 145–164
22. Tseitin, G.: On the complexity of derivation in propositional calculus. Studies in Constructive Mathematics and Mathematical Logic **2** (1968) 115–125
23. Benedetti, M.: sKizzo: A suite to evaluate and certify QBFs. In: Proc. of CADE. LNCS 3632, Springer (2005) 369–376
24. Yu, Y., Malik, S.: Validating the result of a quantified boolean formula (QBF) solver: theory and practice. In: Proc. of ASP-DAC, ACM Press (2005) 1047–1051
25. Kleine Büning, H., Karpinski, M., Flügel, A.: Resolution for quantified boolean formulas. Inf. Comput. **117**(1) (1995) 12–18
26. Kleine Büning, H., Zhao, X.: On models for quantified boolean formulas. In: Logic versus Approximation. LNCS 3075, Springer (2004) 18–32
27. Benedetti, M.: Extracting certificates from quantified boolean formulas. In: Proc. of IJCAI. (2005) 47–53
28. Büning, H.K., Subramani, K., Zhao, X.: On boolean models for quantified boolean horn formulas. In: Proc. of SAT. LNCS 2919, Springer (2003) 93–104
29. Sinz, C., Biere, A.: Extended resolution proofs for conjoining BDDs. In: Proc. of the 1st Intl. Computer Science Symp. in Russia (CSR 2006). LNCS 3967, Springer (2006) 600–611
30. Jussila, T., Sinz, C., Biere, A.: Extended resolution proofs for symbolic SAT solving with quantification. In: Proc. of SAT. LNCS 4121, Springer (2006) 54–60