A Uniform Approach for Generating Proofs and Strategies for Both True and False QBF Formulas

Alexandra Goultiaeva*
Department of Computer Science,
University of Toronto, Canada.
alexia@cs.toronto.edu

Allen Van Gelder
University of California,
Santa Cruz, CA, USA.
www.cse.ucsc.edu/∼avg

Fahiem Bacchus*
Department of Computer Science,
University of Toronto, Canada.
fbacchus@cs.toronto.edu

Abstract

Many important problems can be compactly represented as quantified boolean formulas (QBF) and solved by general QBF solvers. To date QBF solvers have mainly focused on determining whether or not the input QBF is true or false. However, additional important information about an application can be gathered from its QBF formulation. In this paper we demonstrate that a circuit-based QBF solver can be exploited to obtain a Q-Resolution proof of the truth *or* the falsity of a QBF. QBFs have a natural interpretation as a two person game and our main result is to show how, via a simple computation, the moves for the winning player can be computed directly from these proofs. This result shows that the proof is a representation of the winning strategy. In previous approaches the winning strategy has often been represented in a way that makes it hard to verify. In our approach the correctness of the strategy follows directly from the correctness of the proof, which is relatively easy to verify.

1 Introduction

Quantified Boolean Formulas (QBFs) are a powerful generalization of satisfiability (SAT) in which variables can be universally as well as existentially quantified. While any problem in NP can be compactly encoded in SAT, QBF allows us to compactly encode any problem in PSPACE. Hence QBF solvers have a much wider range of potential application areas as there are many problems with compact QBF encoding but whose SAT encodings will necessarily be exponentially larger (unless NP=PSPACE). As a result QBF solvers have already seen application in a range of areas including automated planning [Rintanen, 2007; Giunchiglia *et al.*, 2004], ontological reasoning [Kontchakov *et al.*, 2009], and formal verification [Mangassarian *et al.*, 2007; Biere *et al.*, 1999].

To date, QBF solvers have mainly focused on techniques for more efficiently solving the decision problem, i.e., on determining whether or not the input QBF is *true* or *false*. In SAT, on the other hand, a range of useful applications have been made accessible by obtaining from the solver either a satisfying model or a proof of UNSAT. Such certificates verify the result and supply additional valuable information about the application. Certificates for QBF can be valuable in similar ways.

Previous work on generating certificates for QBF, e.g., Narizzano *et al.* [2009], Jussila *et al.* [2007], has treated the cases of true and false formulas differently. Q-Resolution refutations are usually used to certify false formulas. Q-Resolution [Kleine Büning *et al.*, 1995] is a well studied and simple proof system for QBF, and refutations in this proof system thus form natural and easily checked certificates. Previous methods for verifying true formulas, however, have used certificates that are very hard to verify, or have used certificates that require an additional non-standard proof theory based on term resolution.

In previous work it has been demonstrated how a QBF solver utilizing a circuit representation [Goultiaeva and Bacchus, 2010] can attain complete duality in its treatment of existential and universal quantifiers using a technique called dual propagation. In this paper we show how this duality can be exploited to yield Q-resolution proofs for both *true* and *false* QBF. Thus we obtain a uniform treatment for both types of QBF with the same natural and well studied certificate.

Another critical piece of information that can be obtained about a QBF comes from its natural interpretation as a two-player game. This means that (for a closed QBF) there always exists a winning strategy for one of the players. In applications that are naturally viewed as games, the utility of knowing the winning strategy is clear. But strategies can be useful in other contexts as well. For example, Staber and Bloem [2007] demonstrate how repairs for a sequential circuit can be computed from a strategy.

In this paper, our main contribution is to show that a Q-resolution proof is in fact a representation of a winning strategy. In particular, we show how the winning moves of the strategy can be easily computed from the proof. This has two consequences. First, since we can generate proofs for both *true* and *false* QBFs, our technique can be used to realize a winning strategy for either player. Previous work on obtaining strategies [Benedetti, 2005; Jussila *et al.*, 2007] has generated strategies only for true QBF. Second, the correctness of the strategy follows directly from the correctness of our eas-

*Supported by Natural Sciences and Engineering Research Council of Canada

ily checked proofs. Previous work yields strategies that can be hard to verify and the suggested verification techniques are error prone.

After presenting sufficient background (Section 2) to formalize the problem, we explain how circuit-based QBF solvers can generate proofs for both *true* and *false* QBF formulas and present our main contribution: a simple algorithm for computing the winning moves specified by the proof (Section 3). The fact that such an algorithm exists shows that the proof actually represents a strategy. We then look in more detail at previous work (Section 4). Empirical results are presented next. We have instrumented a circuit solver CirQit to produce certificates, and we provide some results showing that generating proofs does not impose a large overhead over simply determining truth or falsity, and that the requirement of a circuit based representation to obtain proofs for true formulas is not a practical impediment (Section 5). Then we show how an implemented version of our algorithm for extracting winning moves from a proof can be utilized to generate winning lines of play from the certificates.

2 Background

A quantified boolean formula (QBF) has the form $\mathbf{Q}.\phi$ where $\mathbf{Q}$ is a sequence of universally ($\forall$) or existentially ($\exists$) quantified variables, and ϕ is a propositional formula over those variables. Let $\mathbf{Q}.\phi\lceil_{x=0}$ ($\mathbf{Q}.\phi\lceil_{x=1}$) be the new QBF that is the restriction of $\mathbf{Q}.\phi$ by $x = 0$ ($x = 1$). This restriction is computed by replacing the variable x in ϕ by the constant $0 =$ *false* (or $1 =$ *true*) followed by standard boolean simplification, and the removal from $\mathbf{Q}$ of the variable x. In a **closed QBF** all of the variables are bound by a quantifier.

The truth of a closed QBF is defined recursively: $\exists x\mathbf{Q}.\phi$ is true iff either $\mathbf{Q}.\phi\lceil_{x=0}$ or $\mathbf{Q}.\phi\lceil_{x=1}$ is true, and $\forall x\mathbf{Q}.\phi$ is true iff both $\mathbf{Q}.\phi\lceil_{x=0}$ and $\mathbf{Q}.\phi\lceil_{x=1}$ are true. Hence, when deciding the truth of $\mathbf{Q}.\phi$, ϕ will eventually be reduced to the constant 1 or 0: $\mathbf{Q}.1$ is always true and $\mathbf{Q}.0$ is always false.

Most current QBF solvers utilize a conjunctive normal form representation (CNF). For the QBF $\mathbf{Q}.\phi$ this means that ϕ is in CNF, i.e., it is a conjunction of clauses each of which is a disjunction of literals each of which is either a propositional variable, e.g., x, or its negation, $\overline{x}$, where overbar is used to denote negation. We write clauses as sets of literals enclosed in square brackets, e.g., $[p, q, \overline{r}]$, and use $\bot$ to denote the empty clause $[\,]$. We say that a set of literals S is **tautological** if there exists a literal l such that $\{l, \overline{l}\} \subseteq S$. Otherwise S is **non-tautological**. The restriction of a CNF encoded QBF is particularly simple to compute. For example, $\mathbf{Q}.\phi\lceil_{x=0}$ is computed by simply removing all clauses of ϕ that contain $\overline{x}$ and then removing x from all remaining clauses.

A key feature of the CNF representation is that it supports a well known and simple QBF proof system known as **Q-resolution** [Kleine Büning *et al.*, 1995]. Q-resolution is a sound inference system that is complete for proving that a CNF-encoded QBF is *false*. It is formally attractive because it is quite simple and is a natural generalization of the well studied resolution proof system for SAT. Q-resolution contains two rules of inference, **resolution** and **universal reduction**.

Definition 1 (resolution, universal reduction) Let $C_1 = [q, \alpha]$ and $C_2 = [\overline{q}, \beta]$ be two clauses, where $\alpha \cup \beta$ is non-tautological (α and or β can be empty), q is an **existentially** quantified literal, and neither q nor $\overline{q}$ appear in $\alpha \cup \beta$. Resolution infers the new clause $\mathtt{res}_q(C_1, C_2) = [\alpha, \beta]$ which is called the **resolvent**.

Let $C = [q, \alpha]$ be a non-tautological clause, and q be a **universally** quantified literal that is *tailing* in C. A universal literal q is said to be **tailing** in C if there is no existential literal in C that is scoped by q (q comes after all of these variables in the quantifier prefix). Universal reduction infers the new clause $\mathtt{unrd}_q(C) = [\alpha]$. □

Definition 2 (Q-derivation, Q-refutation) A *Q-derivation* is a directed acyclic graph (DAG) in which each node is either an input clause (a DAG leaf), or a Q-Resolution inference step (an internal node). A sample Q-Resolution is shown in Figure 2a). The inference operation at a node n is $\mathtt{res}_q(n_1, n_2)$ or $\mathtt{unrd}_q(n_1)$, and is said to generate the clause that results from that operation. The literal being resolved upon or reduced is specified, and must have the right sign for the operation. The outgoing edge(s) from n go to the operand node(s) n_1 and possibly n_2, whose clauses are used for the operation at n.

The root node generates the derived clause C that is the result of the Q-derivation. If C is the empty clause, $\bot$, then the Q-derivation is called a *Q-refutation*.

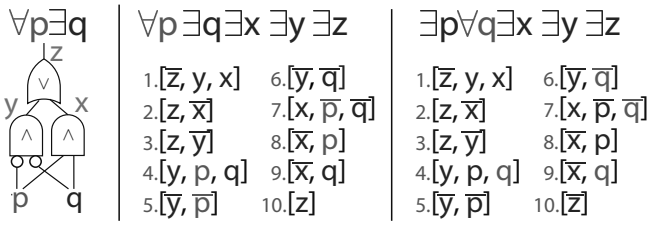
The derivation and all of the clauses generated by the internal nodes are fully specified by the leaf input clauses. Thus one DAG can represent multiple actual derivations, depending on the clauses in its leaves. □

A Q-resolution proof can often be extracted from the inferences made by a QBF solver, yielding a natural and useful certificate verifying when a QBF is false. However, when the QBF is true, there is no refutation to extract. As a result previous work has utilized other kinds of certificates for true QBF. This problem arises in part because these solvers use different inference steps to show that a QBF is true, steps that are not so easily modeled as Q-resolution steps.

In Goultiaeva and Bacchus [2010], however, a solver was presented that used a circuit representation rather than a CNF representation. The paper showed that by exploiting the circuit representation the solver could treat true and false QBF formulas in a completely symmetric manner.

A **circuit representation** of $\mathbf{Q}.\phi$ involves representing ϕ as a boolean circuit utilizing AND, OR and NOT gates. The quantified variables $\mathbf{Q}$ (i.e., the variables of ϕ) are the inputs to the circuit, and the circuit outputs 0 or 1 for each setting of these inputs depending on whether or not ϕ is true or false under this setting of its variables. Figure 1a) shows a circuit representation of the formula $\mathbf{Q}.\psi = \exists p \forall q.(\overline{p} \wedge \overline{q}) \vee (p \wedge q)$. In general, circuit representations can be more compact than propositional formulas since any repeated sub-formula needs to be represented by only a single sub-circuit.

A circuit-encoded QBF $\mathbf{Q}.\phi$ can easily be converted to CNF, and in fact most verification problems are initially encoded as circuits or general propositional formulas and then converted to CNF. This is standardly accomplished using the Tseitin encoding [Tseitin, 1983; Plaisted and Greenbaum,



(a) $\mathbf{Q}.\psi$ (b) $\mathrm{CNF}[\mathbf{Q}.\psi]$ (c) $\mathrm{CNF}[\neg\mathbf{Q}.\psi]$

Figure 1: The running example: a circuit representation of the formula, and the Tseitin encoding of it and its negation

1986] which introduces new variables to ensure only a polynomial increase in the size of the formula. The encoding works by introducing a new existentially quantified variable for every gate output in the circuit along with clauses that force this variable to have a truth value equal to the gate output given the values of the gate inputs.

Figure 1b) shows a Tseitin encoding of the formula given in 1a). New variables x, y and z are introduced. Clauses 1-3 (4-6, 7-9) constrain the variables z (y,x) to agree with their assigned gate outputs. The last clause represents the requirement that the formula is *true*. (Later we will consider $\neg(\mathbf{Q}.\psi)$, and discuss its refutation in Example 6)

Each newly introduced variable is the output of some subcircuit. These new variables must be existential since they are functionally determined by the inputs that feed into their subcircuit. Furthermore, they can be introduced into the quantifier $\mathbf{Q}$ right after all of the variables that form their inputs. However, for this paper is it sufficient to simply place all these new variables at the end of $\mathbf{Q}$. Given a circuit represented QBF $\mathbf{Q}.\phi$ we use $\mathrm{CNF}[\mathbf{Q}.\phi]$ to denote its conversion to CNF via the Tseitin encoding. That is, $\mathrm{CNF}[\mathbf{Q}.\phi] = \mathbf{Q}\exists g_1,\ldots,g_k.\mathrm{CNF}[\phi]$, where $\mathrm{CNF}[\phi]$ is the Tseitin encoding of ϕ and $g_1,\ldots,g_n$ are the newly introduced variables inserted (for simplicity) at the end of $\mathbf{Q}$. We call the clauses in $\mathrm{CNF}[\phi]$ the **input clauses**.

3 Proofs and Strategies

It well known that backtracking DPLL search generates a resolution proof, and this connection has been exploited in SAT solvers to develop the critical technique of clause learning and to allow such proofs to be extracted from the solver for UNSAT instances. Similarly it can be shown that a Q-refutation can be extracted from DPLL search based QBF solvers for *false* formulas.

DPLL search can also be used with a circuit representation, as described in Goultiaeva *et al.* [2009]. The inferences used in such a search can be mapped to Q-resolution steps making it possible to extract a Q-refutation from such solvers even though they are not working directly with a CNF representation. In particular, if the circuit based formula $\mathbf{Q}.\phi$ is being solved and is found to be *false* then a Q-refutation can be extracted. This Q-refutation certifies that $\mathrm{CNF}[\mathbf{Q}.\phi]$ is false; it does not apply directly to the non-CNF formula $\mathbf{Q}.\phi$. However, one can quite safely take this Q-refutation as also certifying $\mathbf{Q}.\phi$ since it is easy to see that $\mathrm{CNF}[\mathbf{Q}.\phi]$ and $\mathbf{Q}.\phi$ are equivalent.

For our purposes, however, the most important feature of a circuit representation is that it supports a completely symmetric treatment of *true* and *false* formulas. In particular, the solver CirQit described in Goultiaeva and Bacchus [2010] simultaneously attempts to solve both $\mathbf{Q}.\phi$ and $\neg\mathbf{Q}.\phi$ by propagating two different truth values along the same circuit representation for $\mathbf{Q}.\phi$. It will terminate with either the conclusion that $\mathbf{Q}.\phi$ is false from which a Q-refutation for $\mathrm{CNF}[\mathbf{Q}.\phi]$ can be extracted, or with the conclusion that $\neg\mathbf{Q}.\phi$ is false from which a Q-refutation for $\neg\mathbf{Q}.\phi$ can be extracted.

This can be accomplished because in a circuit representation $\mathbf{Q}.\phi$ and $\neg\mathbf{Q}.\phi \equiv \overline{\mathbf{Q}}.\neg\phi$, where $\overline{\mathbf{Q}}$ is $\mathbf{Q}$ with every quantifier switched, turn out empirically to be solvable with similar amounts of effort. Figure 1c) shows a Tseitin encoding of $\neg\mathbf{Q}.\psi$. The encoding is different from Figure 1b) in the quantifier, and in Clause 10, which now encodes the condition that the formula must be false. All other clauses are unchanged. In contrast, if ϕ is in CNF, converting $\neg\phi$ to CNF often results in a much harder problem that is often not feasible to solve [Jussila *et al.*, 2007].

There are various technical challenges that arise when trying to instrument a solver so as to extract a proof. Some of these challenges have to do with managing the potentially very large sized proofs, and some have to do with dealing with "special" optimizations that are often embedded in real solvers. These special optimizations often involve inferences that are difficult to map to Q-resolution steps. Fortunately, CirQit solver does not utilize any hard to translate specialized inference rules. For example, the technique of long distance resolution [Zhang and Malik, 2002] used in many search-based solvers can generate and use tautological clauses. Solver specific arguments are needed to show that the solver remains sound even when making inferences from such clauses [Zhang, 2003]. CirQit employs a dynamic reordering technique during clause learning that allows it to always learn non-tautological clauses via a legal sequence of Q-resolution steps.

As a result we were able to instrument CirQit so that it would output a Q-refutation for both *true* and *false* formulas: a Q-refutation of $\mathbf{Q}.\phi$ shows that the input is *false*, while a Q-refutation of $\neg\mathbf{Q}.\phi$ shows that the input is *true*. The Q-refutation can then be verified by simply checking that all input clauses are correct (i.e., that they would be present in the Tseitin encoding of the circuit), and that all resolution steps are correct. Checking the correctness of the input clauses requires consulting the circuit representation.

Our aim was to be able to use any third party verifier to check these refutations. Such a verifier would not in general know anything about circuit representations. So we could not assume that it would be able to verify the input clauses against an inputted circuit. To solve this problem we instead generated and output $\mathrm{CNF}[\mathbf{Q}.\phi]$ for *false* formulas and $\mathrm{CNF}[\neg\mathbf{Q}.\phi]$ for *true* formulas from CirQit (note CirQit has to solve the decision problem first to determine which CNF to output) along with the relevant Q-refutation. We were then able to use the previously developed verifier tool QBV [Jussila *et al.*, 2007] to verify that the Q-refutations was in fact a proof of $\mathrm{CNF}[\mathbf{Q}.\phi]$ or $\mathrm{CNF}[\neg\mathbf{Q}.\phi]$, thus verifying the truth

value of $\mathbf{Q}.\phi$.

3.1 Proofs represent Strategies

The main innovation of this paper was to realize that Q-refutations are sufficient. In particular, in this section we demonstrate that a Q-refutation represents a strategy: the moves mandated by the strategy can be extracted from a refutation via a simple computation and the correctness of the strategy follows directly from the correctness of the refutation. Previous works have used other representations of strategies (e.g., a set of Skolem functions) that make verifying their correctness considerably more difficult.

To understand the connection between QBFs and strategy it should be noted that a closed QBF formula $\mathbf{Q}.\phi$ has a natural interpretation a two-player game with players E (*existential*) and A (*universal*), whose "moves" involve setting their boolean variables. The variables are set in quantifier-prefix order, from outer to inner scopes. The quantifier prefix can be partitioned into *blocks* of alternating quantifier type. The *quantifier depth* of a block begins at 1 for the outermost block and increases by 1 with each alternation. The variable order within a quantifier block is immaterial to the value of the QBF formula, so we may assume that each player sets all the variables in one block in a single turn. Thus turns are taken in order of increasing quantifier depth, and the quantifier type of the outermost unset block of variables determines whose turn it is to play. By the time all variables have been set, ϕ, the body of the QBF, will be reduced to a formula that evaluates either to *true*, in which case E is the winner, or to *false*, in which case A is the winner, for this particular "play".

For every *true* QBF there exists a *strategy* for E, with which E can win every play of the game, no matter how A plays. In particular, the strategy tells E how to play given the previous moves played. If E follows the strategy for all its moves, it will set its variables in such a way that the body of the QBF must evaluate to *true*. Hence, the strategy serves to counter all of A's moves ensuring that E wins. Similarly, a winning strategy exists for A for every *false* QBF.

A Q-refutation of $\mathbf{Q}.\phi$ verifies that it is *false*. This means that there is a winning strategy for A. As stated previously, a Q-refutation itself is a representation of a strategy. This means that it can be used to extract the moves for A that are guaranteed to make the formula *false*.

Algorithm 1 presents a method for extracting the moves for A from a Q-refutation. Algorithm 1 can be equally well applied to a Q-refutation of $\neg\mathbf{Q}.\phi$ (verifying that $\mathbf{Q}.\phi$ is *true*) to obtain a winning strategy for E. In particular, in $\neg\mathbf{Q}.\phi$, E has become the universal player, since the quantifiers have been switched. Hence, E can win $\mathbf{Q}.\phi$ by playing universal's winning strategy for $\neg\mathbf{Q}.\phi$.

This is exactly the case in our example, since $\mathbf{Q}.\psi$ is true. Then, to generate a winning strategy for E, we start with a Q-refutation of $\neg(\mathbf{Q}.\psi)$, which is shown in Figure 2a).

Our algorithm produces an assignment for the universal variables u that A has to play each time it is A's turn to play. These assignments constitute winning moves for A given the previous moves of the game.

The algorithm takes as input the CNF encoded QBF formula $\mathbf{Q}.\phi$ (which must be false for A to have a winning strategy) along with a Q-refutation π verifying that the formula is false. It works its way through the blocks of $\mathbf{Q}$ processing the outermost block first. If the outermost block is existential, it queries E for its move. E must set all the variables of this block. These moves are represented in a partial assignment σ, and then applied to restrict the formula $\mathbf{Q}.\phi \rightarrow (\mathbf{Q}.\phi)\lceil_\sigma$ and the proof $\pi \rightarrow \pi\lceil_\sigma$. The technique for restricting a proof is given below. The algorithm will then return to the outermost while loop with a shorter $\mathbf{Q}$ that is either empty or with a universal outermost block.

If the outermost block is universal a search of π is conducted in reverse topological order (children of a node are examined before the node is) for the first node D whose clause has no existential variables. (This ordering implies that the inputs to an inference step are examined before the output of the inference step). If multiple nodes qualify, the first node encountered is selected. Some node of the current proof π must satisfy this condition, as the root of π must be the empty clause $\perp$ at this stage of the algorithm. The clause generated at node D by π thus contains only universal variables (or is empty). Any universal literals from the outermost block B contained in D are then set to *false* in A's winning moves. The other variables of B can be set by A to an arbitrary value—A will still win.

Finally, we modify the proof and the formula in preparation for the next round of the game. In particular, we make D the new root of π and remove the other parts of π which are now redundant. We then add a sequence of universal reduction steps to derive the empty clause from D (this is possible since D contains only universal variables). The new node deriving the empty clause then becomes the new root of π. This restores π's status as a Q-refutation (a derivation of $\perp$). Finally, we restrict π by A's move, τ. It can be shown that τ cannot satisfy any clause of the new π, so π remains a Q-derivation of $\perp$ after this step and is ready for processing the next round of the game.

Algorithm 1: Compute Strategies

```
1  strategy_For_A (Q.φ, π)
2  while Q is not empty do
3      Let B be outermost block of Q
4      if B is existential then
5          σ = Query existential player E for their move
               /* σ must assign every variable in B.   */
6          π = π⌈_σ
7          Q.φ = (Q.φ)⌈_σ
8      else B is universal and it is A's move
9          Search nodes of π in reverse topological order
10         D = first node with no existential literals
11         τ = {¬q | q ∈ D ∧ q ∈ B}
12         if τ does not set every variable in B then
13             Add to τ an arbitrary value for each variable of B
                 it does not set
14         Output τ (this is A's next move)
15         π = D ∪ {all nodes reachable from D}
16         π = π ∪ {unrd generating ⊥ leading to D}
17         π = π⌈_τ
18         Q.φ = (Q.φ)⌈_τ
```

Table 1: Extensions to `res` and `unrd` to accommodate restrictions. For interpreting conditions, $\top$ is considered to contain every literal.

	If ...	then $\texttt{res}_q(D_1, D_2) = \ldots$
(1)	$q \in D_1$ and $D_2 = \top$	$\top$
(2)	$D_1 = \top$ and $\overline{q} \in D_2$	$\top$
(3)	$q \notin D_1$ and $\overline{q} \in D_2$	D_1
(4)	$q \in D_1$ and $\overline{q} \notin D_2$	D_2
(5)	$q \notin D_1$ and $\overline{q} \notin D_2$	narrower$^{(*)}$ of D_1, D_2

	If ...	then $\texttt{unrd}_q(D_1) = \ldots$
(6)	$D_1 = \top$	$\top$
(7)	$q \notin D_1$	D_1

$^{(*)}$ Narrower means shorter. The most narrow clause is $\bot$, the least narrow is $\top$. Break ties by any fixed order on clauses.

Now we describe how the restriction of the proof by the players' moves is computed, completing the technical details of the algorithm. As specified above a proof π consists of a DAG with leaf nodes labeled by input clauses and internal nodes computed by applying the inference step they specify to the clauses computed by their children. We want to restrict π by an assignment of truth values σ to some of the variables. We usually represent σ as the set of literals it maps to *true*.

To compute $\pi\lceil_\sigma$ we first restrict all of π's input clauses at its leaf nodes using the standard technique. For a clause C, $C\lceil_\sigma = \top$ if σ contains a literal of C, otherwise we remove the negation of all literals in σ from C. Once the leaf nodes have new clauses they are marked as processed. Once an internal node n's children have been marked as processed we compute n's new clause by applying the inference step specified at n to the new clauses that have been computed for its children.

Now, however, the standard inference steps of resolution and universal reduction might no longer apply to the clauses supplied to n by its children. One or more of these clauses might have been reduced to $\top$ by σ. Similarly, because of previous reductions n might be trying to resolve on a literal q such that q or $\overline{q}$ has vanished from the clauses it is to apply the resolution step to. Hence, we have to extend the inference rules `res` and `unrd` to deal with these cases. The extensions are summarized in Table 1. The extensions for `res` were previously developed for SAT [Van Gelder, 2005] (although they were part of the proof-theory folklore before this paper). The extension for `unrd` is new, for QBF.

Lemma 3 The extensions to Definition 1 given in Table 1 are sound. That is, any clause generated by the extended rules at some node n is logically entailed by the children of n.

With these extensions to the inference rules the restriction σ of a Q-derivation, π. $\pi' = \pi\lceil_\sigma$, is defined inductively as follows: (1) for a leaf $C'_i = C_i\lceil_\sigma$; (2) if $D = \texttt{res}_q(D_1, D_2)$ and D'_1 and D'_2 have been derived for π', then $D' = \texttt{res}_q(D'_1, D'_2)$; (3) if $D = \texttt{unrd}_q(D_1)$ and D'_1 has been derived for π', then $D' = \texttt{unrd}_q(D'_1)$. Rules (2) and (3) might require Table 1. This shows how to compute $\pi\lceil_\sigma$ at line 6 and 17 of Algorithm 1.

The results of applying a restriction by p and $\overline{p}$ are shown in Figure 2b) and c). Note that, since $\top$ is considered to contain all literals, the rules 3) and 4) can apply when one of the clauses is $\top$. One example is the clause $[\overline{q}, z]$ when the proof

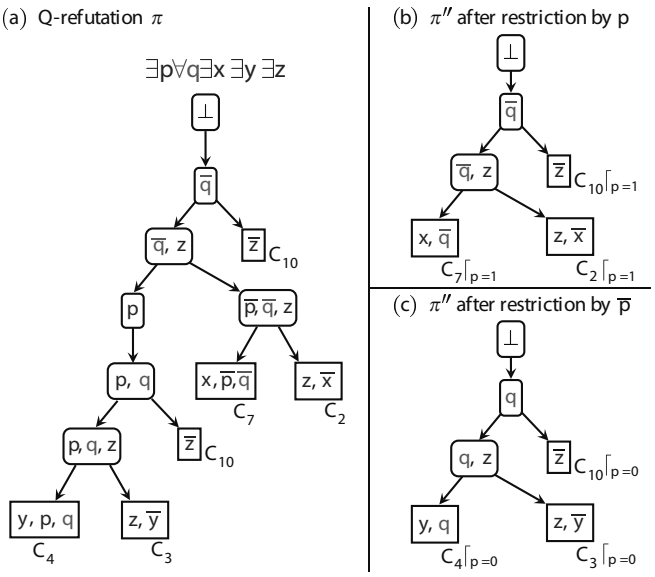


Figure 2: Applying a restriction to an initial Q-refutation based on the E's first choice leads to a Q-refutation in which one node contains exactly those universal literals that A should set to 0 as the winning strategy for that situation.

is reduced by p. In this case, $D_1 = \top$ and $D_2 = [\overline{q}, z]$, and rule 4) applies, yielding D_2. However, these rules can also apply when no $\top$ clause is present. For example, consider a derivation $C_1 = \texttt{res}_p([a, p], [b, \overline{p}])$, $C_2 = \texttt{res}_a(C_1, [\overline{a}])$. Suppose we restrict it by p. By rule 5), $C'_1 = \texttt{res}_p(\top, [b])$ becomes $[b]$. Now, $C'_2 = \texttt{res}_a([b], [\overline{a}])$. Rule 3) applies, and $C'_2 = [b]$.

Lemma 4 Suppose C is the clause generated at some node n in π, and C' is the clause generated at a node n in π'. Suppose further that the Q-derivation of n in π does not contain any universal reduction steps that would reduce some variable $u \in \sigma$. Then $C\lceil_\sigma$ logically entails C'.

Proof: (Sketch) Assume there is no such universal reduction step. Then we can show that $C\lceil_\sigma$ logically entails C' by induction on the structure of the derivation.

Any steps that produce a clause that is not $\top$ would yield C' such that $C' \subseteq C\lceil_\sigma$.

C' is $\top$ only when C contains some literal $p \in \sigma$: we can show this by considering the rules that produce $\top$. Consider rule 1), and let $C = \texttt{res}_q(D_1, D_2)$. $D'_2 = \top$, which means that $p_1 \in D_2$ for some literal $p_1 \in \sigma$. If $p_1 \notin C$, then $p_1 = \overline{q}$. Since $q \in D'_1$, but q is *false* under σ, then $D'_1 = \top$. This means there is some literal $p_2 \in D'_1$ such that $p_2 \in \sigma$ and $p_2 \neq \overline{q}$, which means $p_2 \in C'$. Rule 2) is symmetric. Rule 3) also cannot break the property unless some literal $p \in \sigma$ was universally reduced.

■

Now we show that Algorithm 1 is correct, i.e., it computes a winning strategy for universal.

Theorem 5 Let π be a Q-refutation of $\mathbf{Q}.\phi$. Let $(\mathbf{Q}.\phi)\lceil_\tau$ be as computed at line 18 of Algorithm 1. Then $\pi\lceil_\tau$ as computed at line 17 is a Q-refutation for $(\mathbf{Q}.\phi)\lceil_\tau$.

Proof: (Sketch) Every internal node in $\pi\lceil_\tau$ that is not itself $\top$ has at least one operand that is not $\top$. All such nodes are derived from their operands using Definition 1 and Table 1. So, it suffices to show that $\bot$ is derived in $\pi\lceil_\tau$.

Consider the node D, which is the first node with no existential literals (an *all-universal* node). Let τ be the variable assignment determined by the current iteration of the algorithm. If the Q-derivation of D contained any universal reduction steps on any literal $u \in \tau$, then there must have been a previous all-universal clause D_2 (because all existential variables of smaller blocks have already been assigned). The algorithm uses reverse topological order, so D_2 would have been chosen first. So, $D\lceil_\tau = \bot$, and by Lemma 4 $D' = \bot$. Through rules 3), 4), 5) and 7), $\bot$ would be propagated to the root of $\pi\lceil_\tau$. ■

The proof of correctness for the reduction on line 6 is similar. Here σ only contains existential variables, which cannot be universally reduced.

The formula $(\mathbf{Q}.\phi)\lceil_\tau$ at line 18 characterizes the remaining of the game between E and A after A's latest move τ: it is the original formula restricted by all of the moves played so far. The theorem says $(\mathbf{Q}.\phi)\lceil_\tau$ has a Q-refutation. That is, it is *false*, and thus there is a winning strategy for A in the game that remains after all prior moves. In other words, by playing τ, A is still able to win the game. A will thus remain in a winning position up to the end of the game, and when it has finished all of its moves it must be a winner. That is, τ as computed by the algorithm is a part of a winning strategy.

Example 6 Theorem 5 and the preceding discussion are illustrated in Figure 2. Part (a) shows the initial Q-refutation, π, of $\mathrm{CNF}[\neg(\mathbf{Q}.\psi)]$ which is shown in Figure 1c). Observe that universal reductions on both $\overline{q}$ and q occur on the left branch of this proof. This is normal.

If E chooses p to be 1, then the restriction $p = 1$ is applied to the underlying formula and π is transformed into a new Q-refutation shown in part (b). The lowest all-universal node contains the clause $[\overline{q}]$, which becomes $\bot$ after universal reduction(s). Thus the winning response by A is $q = 1$.

If instead, E chooses p to be 0, then the restriction $p = 0$ is applied to the underlying formula and π is transformed into a new Q-refutation shown in part (c). The lowest all-universal node contains the clause $[q]$, which becomes $\bot$ after universal reduction. Thus the winning response by A is $q = 0$.

All necessary settings for the entire quantifier block are found in one node. □

The solver reported in this paper produces proofs in the QIR format, an announced standard for the 2011 SAT Competition[1] This format permits "linear input" subproofs to be condensed by omitting resolvents that are used only once. This condensation has large practical advantages. For example, the Q-refutation in part (a) of Figure 2 takes only two QIR lines.

Another practical advantage of QIR for strategy extraction is that the algorithm described in this section for full resolution proofs adapts very cleanly to this more abbreviated format, in that it is able to extract the winning strategy without computing the omitted resolvents.

[1] See `http://www.satcompetition.org/2011`.

This proof format differs from many Q-resolution formats suggested previously in that it restricts the ordering of the clauses in each derivation line. It is not clear if it is sound to leave the ordering up to the verifier to derive, since the "pivoting" method as used in SAT is not sufficient.

4 Related Work

The papers Narizzano *et al.* [2009], Jussila *et al.* [2007] review a number of works on generating certificates from QBF solvers as well as making some original contributions to the subject. In these papers a number of different solvers capable of generating certificates along with tools for verifying these certificates are discussed. None of these solvers/verifiers are, however, capable of generating Q-refutations for both *true* and *false* formulas. Nor has it been previously observed that the refutations serve as representations of a winning strategy.

`qube-cert/checker` generates Q-refutations for *false* QBF, but a specialized term-refutation for *true* QBF. The term-refutations require their own theory and verification tools: techniques standard for Q-refutations cannot be applied. This solver/verifier does not generate strategies. However, Algorithm 1 could be applied to obtain winning moves from the Q-refutation. Furthermore, potentially our method could be extended to deal with term-refutations to extract winning moves for existential when the QBF is *true*.

Both `ebddres/tracecheck` and `Squolem` produce Q-refutations for *false* QBF. `ebddres/tracecheck` does not produce a certificate for *true* QBF.

Both `Squolem` and `sKizzo/ozziks` produce strategies represented as a set of Skolem-functions for *true* QBF. The idea of Skolem-functions is that the value of each universal variable in an optimal game can be defined as a function over the the preceding existential variables. A set of such functions for all the universal variables would constitute a strategy.

Verifying the strategies produced by these solvers is a hard problem (co-NP complete [Kleine Büning and Zhao, 2004]). To verify them Jussila *et al.* [2007] (and Benedetti [2005] before) utilize multiple calls to a SAT solver. In contrast, the correctness of the strategies we generate follows directly from the theorems we presented in Section 3 and the correctness of our relatively easy to verify Q-refutation proofs. Clearly this is a much less error prone process than multiple calls to a SAT solver.

5 Experimental Results

5.1 Proof Generation

We evaluated our implementation of two-sided Q-resolution generation empirically, to study the effect of proof generation on performance, and to see how CirQit fits in generally with other state-of-the-art QBF solvers. Each test was run on a 2.6GHz 2-core AMD processor with 16 GB of memory. Programs were single-thread, and were limited to four GB of memory because they were 32-bit binaries. A 1200 second timeout was used for the solver.

We have used CirQit to generate certificates for the non-CNF benchmarks from QBFEVAL [Peschiera *et al.*, 2010]. The results are summarized in Table 2. We can see that the

Table 2: Statistics of CirQit on the QBFEVAL dataset.

	# Solved		S Time		Prf		% Verified		Ver	
Family	T	F	T	F	T	F	T	F	T	F
assertion (120)	0	45	0	14522	-	1.03	-	100	-	1.08
Core (63)	11	33	2918	7024	1.22	1.08	0	100	0.00	1.55
counter (45)	35	7	28	282	1.23	1.19	100	71	6.00	0.20
dme (11)	11	0	12	0	1.03	-	100	-	1.49	-
possibility (120)	7	54	2877	17132	0.92	1.00	100	100	2.72	1.07
ring(20)	10	10	1	35	1.37	1.21	100	100	3.68	11.59
Umbrella (73)	7	26	914	665	1.16	1.04	0	100	0.00	1.63
semaphore (16)	16	0	2	0	1.46	-	100	-	4.04	-
Seidl (150)	93	57	38	7	1.25	1.41	100	100	4.34	6.61
consistency (10)	7	0	2197	0	1.02	-	100	-	2.95	-
Total (628)	197	232	8986	39666	1.07	1.03	91	99	1.48	1.18

S is solving time in CPU seconds. Prf is ratio of the time producing the proof over the time just solving. % Verified is the percentage of solved problems that was successfully verified by QBV. Ver is the ratio of time verifying the proof to time producing it. The results are segregated into true (T) and false (F) formulas.

overhead of producing the proofs is quite small, and it was able to produce proofs for every problem it solved.

We used an independently written verifier, named `qbv` [Jussila *et al.*, 2007], to verify the Q-resolution proofs produced by CirQit. This procedure helps to prevent the verifier from incorporating some faulty assumptions that might be in the solver, and verifying incorrect output. On 20 problems the verifier ran out of memory. On the remaining problems it accepted the proof produced.

As mentioned previously, the QIR format which CirQit now natively produces is a bit more restrictive than the format of `qbv`. As a result, some information that is lost in translation can make the certificate easier to verify. Currently `qbv` performs constraint propagation as part of the verification process, which is unnecessary for verifying QIR. However, even though it is possible to construct a more efficient verifier, even results with `qbv` demonstrate that it is quite feasible to produce and verify Q-resolution proofs for both *true* and *false* formulas.

A CNF formula can also be represented as a logical circuit. The output gate would be an AND-gate, with one child OR-gate for each clause. However, the resulting circuit is very shallow, and, of course, does not contain any information lost in translating to CNF. Running on CNF formulas CirQit is stripped of most of its advantages, and the specialized data structures of CNF solvers overtake it. However, when given formulas that have not been transformed, CirQit is a robust solver that can compete with the state of the art.

The families Core and Umbrella are encoded in both CNF and non-CNF [Peschiera *et al.*, 2010]. CirQit solved 77 problems in these two families, while `qube-cert` solved 88. However, the certificates of `qube-cert` include tautologous clauses, for which there is no established proof theory. All of the certificates for the true formulas solved by `qube-cert` were rejected by its own verifier.

On the same dataset, `Squolem` was not able to solve a single problem within the timeout. `sKizzo` solved 72 problems, of which 45 were *true*. As expected, verification of those certificates proved prohibitive. It was only able to verify 10, and in all cases verifying an instance took orders of magnitude longer longer than solving it. The certificates produced by CirQit were on average 3.93 times smaller.

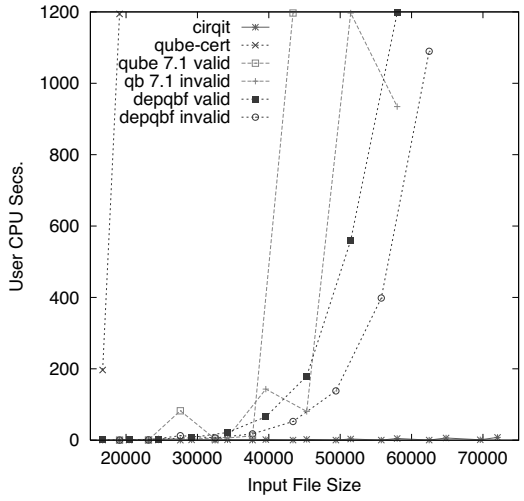


Figure 3: User CPU time comparisons on 22 benchmarks in the marbles family.

CirQit does not generally do well on problems represented in CNF. However, performance varies greatly from family to family, and there are cases where CirQit outperforms other search-based CNF-based solvers, including `depqbf` and `QuBE 7.1`, even on CNF encoded problems. For example, the *marbles* family of benchmarks, available in CNF only, encodes two-person games similar to, but simpler than, nim. Figure 3 displays the results on that family.

These problems pose an interesting challenge for search-based solvers because polynomial-length proofs exist; the preprocessor in Qube 7.1 solves them almost instantly. CirQit succeeds in quadratic time, while several other leading solvers take exponential time.

5.2 Move Extraction

We have implemented Algorithm 1 described in Section 3. Given the CNF problem specification, a refutation in QIR format, and the appropriate settings for the outermost variables, the algorithm determines the settings of the first unassigned block.

We have applied the algorithm to the game of Tic-Tac-Toe. This is a two-player game on a 3-by-3 grid, which is initially empty. Two players, x and o, take turns marking the empty squares. The first one to get three marks in a row wins the game. We used Q-resolution proofs as strategies for both players, and played them against each other using the move generation algorithm, obtaining the first game shown in Figure 4.

For each player p, let ϕ_p be the QBF encoding the question "Does p have a winning strategy?". The moves of x (o) will be existential (universal) in ϕ_o and universal (existential) in ϕ_x. Both ϕ_x and ϕ_o are false: if both players play optimally, there will be a tie. An optimal strategy S_p for the universal player in ϕ_p ensures that player p cannot win.

We used CirQit to generate a certificate for each formula. After that, the lines of play were generated from the certificate alone. No other input was needed to determine the moves.

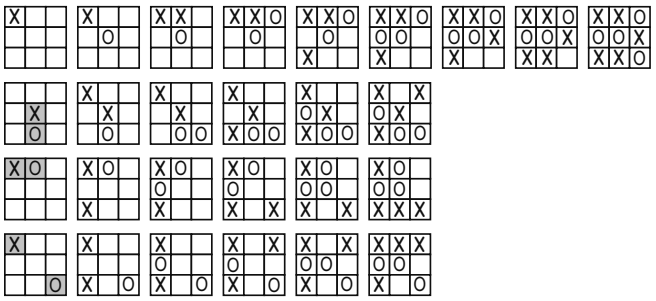


Figure 4: Sample games obtained by strategy generation. Greyed-out moves were predetermined.

The move generator for the x-player provided the settings of variables in the first quantifier block (which represent x's first move). Those settings were then fed into the move generator for the o-player, which determined o's next move. The iterations continued for all quantifier levels, and resulted in the first game shown in Figure 4.

Now, suppose we pre-specify two initial moves so that the o-player is put into a losing position. Let ϕ'_p represent ϕ_p restricted by those two moves. The formula ϕ'_x becomes true, because now x can ensure its victory. Then our move generator can be used to play a winning strategy for x. Note that we cannot obtain a strategy for the losing player, because its moves are irrelevant to the outcome. So, we demonstrate the generated moves against a human player. The remaining games in Figure 4 display sample games with different initial settings.

6 Conclusion and Future Work

We have demonstrated that Q-resolution proofs can, in fact, be viewed as a different representation of the strategy. They are straightforward to generate for both *true* and *false* formulas using a circuit solver. Also, they are much easier to verify than Skolem-function approaches.

The interesting question for future work is to formalize and compare the properties of this representation to those of existing ones. Are there useful queries that are easily answerable in one representation but not the other? What other advantages and limitations are there?

We conjecture that both representations would have the same worst case size complexity. However, it is likely that for some formulas one might be more compact and vice versa. Investigating the potential trade-offs is another interesting avenue for future work.

References

[Benedetti, 2005] M. Benedetti. Extracting certificates from quantified boolean formulas. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[Biere *et al.*, 1999] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proceedings of the International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, pages 193–207, 1999.

[Giunchiglia *et al.*, 2004] E. Giunchiglia, M. Narizzano, and A. Tacchella. QBF reasoning on real-world instances. In *Theory and Applications of Satisfiability Testing (SAT)*, 2004.

[Goultiaeva and Bacchus, 2010] A. Goultiaeva and F. Bacchus. Exploiting QBF duality on a circuit representation. In *Proceedings of the AAAI National Conference (AAAI)*, pages 71–76, 2010.

[Goultiaeva *et al.*, 2009] A. Goultiaeva, V. Iverson, and F. Bacchus. Beyond CNF: A circuit-based QBF solver. In *Theory and Applications of Satisfiability Testing (SAT)*, pages 412–426, 2009.

[Jussila *et al.*, 2007] T. Jussila, A. Biere, C. Sinz, D. Kröning, and C. M. Wintersteiger. A first step towards a unified proof checker for QBF. In *Theory and Applications of Satisfiability Testing (SAT)*, pages 201–214, 2007.

[Kleine Büning and Zhao, 2004] H. Kleine Büning and X. Zhao. On models for quantified boolean formulas. In *Logic versus Approximation, Springer LNCS 3075*, pages 18–32, 2004.

[Kleine Büning *et al.*, 1995] H. Kleine Büning, M. Karpinski, and A. Flögel. Resolution for quantified boolean formulas. *Inf. Comput.*, 117(1):12–18, 1995.

[Kontchakov *et al.*, 2009] R. Kontchakov, L. Pulina, U. Sattler, T. Schneider, P. Selmer, F. Wolter, and M. Zakharyaschev. Minimal module extraction from DL-lite ontologies using QBF solvers. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 836–841, 2009.

[Mangassarian *et al.*, 2007] H. Mangassarian, A. G. Veneris, S. Safarpour, M. Benedetti, and D. Smith. A performance-driven QBF-based iterative logic array representation with applications to verification, debug and test. In *International Conference on Computer-Aided Design (ICCAD)*, pages 240–245, 2007.

[Narizzano *et al.*, 2009] M. Narizzano, C. Peschiera, L. Pulina, and A. Tacchella. Evaluating and certifying QBFs: A comparison of state-of-the-art tools. *AI Commun.*, 22(4):191–210, 2009.

[Peschiera *et al.*, 2010] C. Peschiera, L. Pulina, A. Tacchella, U. Bubeck, O. Kullmann, and I. Lynce. The seventh QBF solvers evaluation (qbfeval'10). In *Theory and Applications of Satisfiability Testing (SAT)*, pages 237–250, 2010.

[Plaisted and Greenbaum, 1986] D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[Rintanen, 2007] J. Rintanen. Asymptotically optimal encodings of conformant planning in QBF. In *Proceedings of the AAAI National Conference (AAAI)*, pages 1045–1050, 2007.

[Staber and Bloem, 2007] S. Staber and R. Bloem. Fault localization and correction with QBF. In *Theory and Applications of Satisfiability Testing (SAT)*, pages 355–368, 2007.

[Tseitin, 1983] G. Tseitin. On the complexity of proofs in propositional logics. In *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*. Springer-Verlag, 1983.

[Van Gelder, 2005] A. Van Gelder. Input distance and lower bounds for propositional resolution proof length. In *Theory and Applications of Satisfiability Testing (SAT)*, 2005.

[Zhang and Malik, 2002] L. Zhang and S. Malik. Conflict driven learning in a quantified boolean satisfiability solver. In *Proceedings of the International Conference on Computer-aided Design (ICCAD)*, pages 442–449, 2002.

[Zhang, 2003] L. Zhang. *Searching for Truth: Techniques for Satisfiability of Boolean Formulas*. PhD thesis, Princeton University, 2003.