CAQE: A Certifying QBF Solver

Markus N. Rabe
University of California, Berkeley
rabe@berkeley.edu

Leander Tentrup
Saarland University
tentrup@cs.uni-saarland.de

***Abstract*—We present a new CEGAR-based algorithm for QBF. The algorithm builds on a decomposition of QBFs into a sequence of propositional formulas, which we call the clausal abstraction. Each of the propositional formulas contains the variables of just one quantifier level and additional variables describing the interaction with adjacent quantifier levels. This decomposition leads to a simpler notion of refinement compared to earlier approaches. We also show how to effectively construct Skolem and Herbrand functions from true, respectively false, QBFs; allowing us to certify the solver result.**

We implemented the algorithm in a solver called CAQE. The experimental evaluation shows that CAQE has competitive performance compared to current QBF solvers and outperforms previous certifying solvers.

I. Introduction

Efficient solving techniques for Boolean theories are an integral part of modern verification and synthesis methods. The ever growing complexity of verification and synthesis problems led to propositional problems of enormous size. To see further advances in these areas, we believe that is necessary to move to more compact representations of these requirements. Quantified Boolean formulas (QBFs) have repeatedly been considered as a candidate theory to compactly encode Boolean problems [1]–[7]. Recent advances in QBF solvers give raise to the hope that QBF may help to increase the scalability of verification and synthesis approaches.

The recent introduction of algorithms based on counter-example guided abstraction refinement (CEGAR) significantly improved the scalability of QBF solving [8], [9]. However, the CEGAR approach shows poor performance for instances with many quantifier alternations, as we show in this paper, and it currently lacks the ability to certify its results. In this work, we present a modification of the CEGAR approach for QBF that tackles these two problems.

Certifying the results is particularly important for QBF, as the pure yes/no answer is of little use. Like the propositional SAT problem [10], [11], QBF can be used to encode objects of interest, like error paths [3], [5] and implementations [4], [12], [13]. While the yes/no answer of a SAT or QBF solver then provides the information about the *existence* of this object, we often want to construct a concrete instance for further use. For the propositional SAT problem the object can typically be extracted as the assignment of the variables, but for QBF the object potentially consists of the Skolem functions for the existential quantifiers or the Herbrand functions for the universal quantifiers. Most current QBF solvers, however, are unable to provide Skolem or Herbrand functions or suffer performance penalties when they do [14], [15].

This work was partially supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center “Automatic Verification and Analysis of Complex Systems” (SFB/TR 14 AVACS) and by NSF grants CCF-1139138 and CCF-1116993. The first author did a part of the work at Saarland University.

Our approach is based on the observation that the only information relevant for the processing of inner quantifier levels is which clauses are satisfied by the outer quantifier levels. Consider the following example:

$$\forall X \exists Y.\ (x_1 \vee \overline{x_2} \vee y_1) \wedge (x_2 \vee \overline{y_1} \vee \overline{y_2}) \wedge (y_1 \vee y_2)$$

where $x_i \in X$ and $y_i \in Y$ for all i.

To determine the truth value of this QBF, we need to show that for every assignment of the variables X, there is an assignment of the variables Y such that the propositional part of the formula above is true. Any assignment of X satisfies a certain set of clauses and thereby requires that the remaining set of clauses is satisfiable by the variables Y. For example the assignment $x_1\overline{x_2}$ satisfies exactly the first clause and any assignment of the variables Y that satisfies the remaining clauses is sufficient for this case. We can thus split the formula into two parts; one for the universally quantified variables X and one for the existentially quantified variables Y:

$$\varphi_X := ((x_1 \vee \overline{x_2}) \to \neg b_1) \wedge (x_2 \to \neg b_2) \wedge (\text{false} \to \neg b_3)\,,$$
$$\varphi_Y := (t_1 \vee y_1) \wedge (t_2 \vee \overline{y_1} \vee \overline{y_2}) \wedge (t_3 \vee y_1 \vee y_2)\,,$$

where the variables $B = \{b_1, b_2, b_3\}$ (<u>b</u>ottom) indicate that the clause is satisfied by the lower quantifier level ($\exists Y$), and the variables $T = \{t_1, t_2, t_3\}$ (<u>t</u>op) indicate that the clause is satisfied by the upper quantifier level ($\forall X$). That is, for every clause φ_X requires that whenever the clause is satisfied by the X variables, it does not have to be satisfied by the Y variables. And φ_Y requires that when the clause is satisfied by some X, it does not have to be satisfied by the variables Y. The problem to determine the truth of the QBF is then equivalent to determining whether for each satisfying assignment of φ_X that includes the assignment $\mathbf{b}$ of B, the formula $\varphi_Y(\mathbf{t_b})$ is satisfiable, where $\mathbf{t_b}$ is the assignment of T that assigns t_i iff b_i is *not* assigned in $\mathbf{b}$ (for all $1 \leq i \leq 3$). We call this decomposition of the QBF the *clausal abstraction*.

Following the CEGAR approach to QBF, we would alternate between the two quantifier levels and determine satisfying assignments of φ_X and φ_Y. When there is no assignment of φ_X left, we conclude that the original formula is true, or when there is no satisfying assignment of φ_Y for a given assignment of X, we have found a counter-example. While the overall approach is similar to the existing CEGAR approaches to 2QBF [9], it lets us rephrase the refinement step in an interesting way: Every satisfying assignment α of φ_Y defines a single clause over the variables B that we can add to the formula φ_X. This excludes all assignments of X for which α satisfies the remaining formula.

The principle of clausal abstractions can be lifted to full QBF and we show that this leads to an algorithm with competitive performance. The evaluation of our implementation CAQE reveals the differences to the previous CEGAR-based QBF solver RAReQS: The algorithm we propose is particularly effective for QBFs with many quantifier alternations, while the previous CEGAR-based approach seems particularly effective for problems with few quantifier alternations.

Our approach can be used to certify the result of a QBF. From the sequence of T assignments and assignments of the quantified variables, we can effectively extract Skolem and Herbrand functions in the form of circuits. We describe a proof format and provide a tool chain for the certification process, which outperforms earlier certifying QBF solvers.

To summarize, the contributions of this paper are twofold:

- We develop a CEGAR algorithm for QBF based on clausal abstractions, and
- we give a method to effectively extract Skolem and Herbrand functions for the certification of the results.

II. Quantified Boolean Formulas

A quantified Boolean formula (QBF) is a propositional formula over a finite set of variables X with domain $\mathbb{B} = \{0, 1\}$ extended with quantification. The syntax is given by the following grammar:

$$\varphi := x \mid \neg\varphi \mid (\varphi) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x.\, \varphi \mid \forall x.\, \varphi \ ,$$

where $x \in X$. For readability, we lift the quantification over variables to the quantification over sets of variables and denote $\forall x_1.\forall x_2.\ldots\forall x_n.\varphi$ with $\forall X.\varphi$ and $\exists x_1.\exists x_2.\ldots\exists x_n.\varphi$ with $\exists X.\varphi$, accordingly, for $X = \{x_1, \ldots, x_n\}$.

Given a subset of variables X, an *assignment* of X is a function $\alpha : X \to \mathbb{B}$ that maps each variable $x \in X$ to either true (1) or false (0). For simplicity we describe assignments also by the subset $\mathbf{x} \subseteq X$ of variables that are assigned 1 (or true). We denote *the set of assignments* of a set of variables X by $\mathcal{A}(X)$.

A quantifier $Q\,x.\,\varphi$ for $Q \in \{\exists, \forall\}$ *binds* the variable x in the *scope* φ. Variables that are not bound by a quantifier are called *free*. We assume the natural semantics of the satisfaction relation $\mathbf{x} \models \varphi$ for QBF φ and assignments $\mathbf{x} \subseteq X$ where X are the free variables of φ. *QBF satisfiability* is the problem to determine, for a given QBF φ, the existence of an assignment for the free variables of φ, such that the formula is true.

The dependency set of an existentially quantified variable y, denoted by $dep(y)$, is the set X of universally quantified variables x such that $\exists y.\,\varphi$ is in the scope of x. A *Skolem function* $f_y : \mathcal{A}(dep(y)) \to \mathbb{B}$ maps an assignment of the dependencies of y to an assignment of y. The truth of a QBF φ is equivalent to the existence of a Skolem function f_y for every variable y of the existentially quantified variables Y, such that $\{y \in Y \mid f_y(\mathbf{x} \cap dep(y))\} \models \varphi$ holds for every assignment $\mathbf{x}$ of the universal variables X.

A *closed* QBF is a formula without free variables. Closed QBFs are either true or false. A formula is in prenex normal form, if the formula consists of a quantifier prefix followed by a propositional formula. Every QBF can be transformed into a closed QBF and into prenex form while maintaining satisfiability. For a $k > 0$, a formula φ is in the kQBF fragment if it is closed, in prenex normal form, and has exactly k alternations between $\exists$ and $\forall$ quantifiers.

A *literal* l is either a variable $x \in X$, or its negation $\neg x$. Given a set of literals $\{l_1, \ldots, l_n\}$, the disjunctive combination $(l_1 \vee \ldots \vee l_n)$ is called a *clause* and the conjunctive combination $(l_1 \wedge \ldots \wedge l_n)$ is called a *cube*. Given a literal l, the polarity of l, $sign(l)$ for short, is 1 if l is positive and 0 otherwise. The variable corresponding to l is defined as $var(l) = x$ where $x = l$ if $sign(l) = 1$ and $x = \neg l$ otherwise.

A QBF is in prenex conjunctive normal form (PCNF) if its propositional formula is a conjunction over clauses, which is called a *matrix*. To simplify the notation, we treat a matrix ψ as a set of clauses $\psi = \{C_1, \ldots, C_n\}$ and a clause C as a set of literals $C = \{l_1, \ldots, l_m\}$ and use standard set operations like intersection and union for their manipulation. Every prenex QBF can be transformed into prenex CNF using the Tseitin transformation [16] with a linear increase in the size of the formula and number of existential variables.

III. Clausal Abstractions

In this section, we present clausal abstractions, a decomposition of QBFs into sequences of propositional formulas—one propositional formula for each quantifier level. Clausal abstractions provide us a new notion of refinement and thereby leads us to a variant of the CEGAR algorithm for QBF.

The example in the introduction intuitively explained how every clause in a QBF with one quantifier alternation can be split into two parts with additional variables that describe their interaction. The main observation was that every assignment for the variables quantified by the inner (existential) quantifier, corresponds to a single cube over the new variables T. In the following, we extend this principle to QBF with more than one quantifier alternation and we consider a closed QBF φ in prenex conjunctive normal form:

$$\varphi := Q_1 X_1.\ \ldots\ Q_n X_n.\ \psi\,,$$

where $\psi = C_1 \wedge \ldots \wedge C_k$ is the matrix of φ with k clauses and each $Q_i X_i$ is a *quantifier block*, i.e., a maximal list of consecutive quantifiers of the same type.

Let's first consider the case that Q_1 is an existential quantifier. Proving φ to be true means to find an assignment $\mathbf{x_1}$ for X_1 such that the remaining formula $Q_2 X_2.\ \ldots\ Q_n X_n.\ \psi(\mathbf{x_1})$ is true. Inspecting $\psi(\mathbf{x_1})$ reveals that the assignment of the variables X_1 eliminated certain clauses (by satisfying them) and removed all occurrences of X_1 literals from the remaining clauses. We can thus split each clause C_i into two parts $C_{i,1}$

and $C_{i,>1}$, where $C_{i,1}$ is the part that can be satisfied by some assignment to X_1 and $C_{i,>1}$ is the part that must be satisfied by some variable in X_i with $1 < i \leq n$. For each clause C_i, we introduce the variables b_i (bottom) in $C_{i,1}$ and t_i (top) $C_{i,>1}$ to indicate by which part C_i is satisfied.

$$\begin{aligned} C_{i,1} &= \left(\bigvee_{l \in C_i \wedge var(l) \in X_1} l\right) \vee b_i \\ C_{i,>1} &= \left(\bigvee_{l \in C_i \wedge var(l) \in \left(\bigcup_{i>1} X_i\right)} l\right) \vee t_i \end{aligned}$$

$C_{i,1}$ contains no variables in $\bigcup_{i>1} X_i$ and $C_{i,>1}$ is free of the variables X_1. We call the conjunction of clauses $\varphi_{\exists X_1} = \bigwedge_{j<k} C_{j,1}$ the *existential clausal abstraction* for X_1.

As the variables B occur only positively in $C_{i,1}$, the existential clausal abstraction is monotone in B. In particular, there is a *unique* minimal assignment $\mathbf{b}_{\min}(\mathbf{x_1})$ to B for every assignment $\mathbf{x_1}$ to X_1. The minimal assignment $\mathbf{b}_{\min}(\mathbf{x_1})$ contains exactly the set B variables whose clauses have to be satisfied by a variable from a quantifier block $i > 1$ when the first quantifier block chooses $\mathbf{x_1}$. Hence it is clear that the formula that results from fixing $\mathbf{x_1}$ in the matrix ψ is the same as the matrix that results from fixing $\mathbf{t} = \{t_i \mid b_i \notin \mathbf{b}_{\min},\, 1 \leq i \leq k\}$ in the remaining matrix $\psi_{>1} = \bigwedge_{i \leq k} C_{i,>1}$.

Now let's turn to the case that the outermost quantifier Q_1 is a universal quantifier. Analogue to the previous case, disproving φ means to show that there is an assignment $\mathbf{x_1}$ of X_1 such that the remaining formula $Q_2 X_2. \ldots Q_n X_n.\ \psi(\mathbf{x_1})$ is false. Now, assignments of X_1 that satisfy *less* clauses are more desirable, as they set more of the variables B to true and therefore make it harder to satisfy the remaining formula. The existential clausal abstraction, however, always allows us to set more of the variables B to true and therefore fails to represent when a clause is *necessarily* fulfilled by an assignment $\mathbf{x_1}$. In other words, the existential clausal abstraction represents the lower bounds on B, while we need the upper bounds on B for the universal case. For the universal case, we therefore propose to use the following clausal abstraction:

$$\begin{aligned} C_{i,1} &= \bigwedge_{l \in C_i \wedge var(l) \in X_1} (\bar{l} \vee \bar{b}_i) \\ C_{i,>1} &= \left(\bigvee_{l \in C_i \wedge var(l) \in \left(\bigcup_{i>1} X_i\right)} l\right) \vee t_i \end{aligned}$$

The *universal clausal abstraction* for X_1 is then the conjunction $\varphi_{\forall X_1} = \bigwedge_{j<k} C_{j,1}$. The universal clausal abstraction guarantees that for every assignment $\mathbf{x_1}$ to X_1 there is a unique *maximal* assignment $\mathbf{b}_{\max}(\mathbf{x_1})$ to B, that is an assignment with a maximal number of variables set to true. The decomposition of each clause C_i into the conjunction $C_{i,1}$ ensures that whenever the clause C_i is satisfied by one of its literals, then the variable b_i must be set to false—representing that the other quantifier blocks do not have to satisfy this clause any more. Again, the formula that results from fixing $\mathbf{x_1}$ in $C_1 \wedge \ldots \wedge C_k$ is the same as the matrix that results from fixing $\mathbf{t} = \{t_i \mid b_i \notin \mathbf{b}_{\max},\, 1 \leq i \leq k\}$ in the remaining formula $\psi_{>1} = C_{1,>1} \wedge \ldots \wedge C_{k,>1}$.

We observe that each clausal abstraction only needs one copy of the T variables and another copy of the B variables. When we build the clausal abstractions $\varphi_{Q_i X_i}$ for all quantifier blocks Q_i, we can thus reuse the sets of variables for T and B and only need to introduce $2k$ fresh variables.

IV. CEGAR with Clausal Abstractions

In this section, we present a CEGAR algorithm for QBF based on clausal abstractions. To formulate the algorithm, we assume a method called SAT to solve propositional formulas. We assume that SAT returns whether the formula is satisfiable (SAT) or unsatisfiable (UNSAT). Further, in case the formula is satisfiable SAT returns a satisfying assignment—potentially only for a subset of the variables like in line 6. In our implementation these queries are solved by a SAT solver. In the following, we use $\Psi_{>1} = Q_2 X_2 \ldots Q_n X_n.\psi_{>1}$ to denote the formula that remains when splitting the clausal abstraction $\varphi_{Q_1 X_1}$ from the QBF Ψ. Expressions of the form $c \;?\; e_1 : e_2$ denote abbreviated if-statements. If the conditional c evaluates to true we return e_1 and otherwise e_2.

The input to the algorithm SOLVE is a QBF $QX.\,\Psi$ and the output is either SAT or UNSAT.

```
1:  procedure SOLVE($QX.\,\Psi$)
2:      if $\Psi$ is propositional then
3:          return SAT($\Psi$)
4:      $\alpha \leftarrow (Q = \exists) \;?\; \varphi_{\exists X} : \varphi_{\forall X}$
5:      while true do
6:          $result$, $\mathbf{b} \leftarrow$ SAT($\alpha$)
7:          if $result$ = UNSAT then
8:              return $(Q = \exists)$ ? UNSAT : SAT
9:          $\mathbf{t} \leftarrow \{t_i \mid b_i \notin \mathbf{b},\, 1 \leq i \leq k\}$
10:         $result \leftarrow$ SOLVE($\Psi_{>1}(\mathbf{t})$)
11:         if $Q = \exists$ and $result$ = UNSAT then
12:             $\alpha \leftarrow \alpha \wedge (\bigvee_{l \in \mathbf{b}} \bar{l})$
13:         else if $Q = \forall$ and $result$ = SAT then
14:             $\alpha \leftarrow \alpha \wedge (\bigvee_{l \notin \mathbf{b}} l)$
15:         else
16:             return $(Q = \exists)$ ? SAT : UNSAT
```

The algorithm generates an assignment $\mathbf{b}$ for the variables B in the clausal abstraction, determines $\mathbf{t} = \{t_i \mid b_i \notin \mathbf{b},\, 1 \leq i \leq k\}$, and then goes into recursion for the remaining formula $Q_2 X_2 \ldots Q_n X_n.\psi_{>1}(\mathbf{t})$. Whenever a recursive call failed for an existential quantifier block, i.e., the remaining formula turned out to be false, we add the clause $\bigvee_{l \in \mathbf{b}} \bar{l}$ to the existential clausal abstraction. Analogously, when a recursive call failed for a universal quantifier block, i.e., the remaining formula turned out to be true, we add the clause $\bigvee_{l \notin \mathbf{b}} l$ to the universal clausal abstraction. The next iteration will either bring up a new assignment for the variables B or fail to do so, in which case the formula is violated in the existential case and satisfied in the universal case, respectively.

A. Reusing Clausal Abstractions and their Refinements

Reusing the state of SAT solvers is critical for performance. Instead of regenerating the clausal abstractions for inner quantifier for every instantiation of the variables of the outer quantifier blocks, we set up one SAT solver per quantifier block that we keep for the complete run of the algorithm. Reusing the SAT solvers for each quantifier level also enables us to efficiently reuse all previous refinements for the same level. The clauses by which we refined stay valid for all times.

```
procedure SOLVE_∃($\exists X.\, \Psi$, $\mathbf{t}$)
    while true do
        $\mathit{result}, \mathbf{b}, \mathit{failed} \leftarrow \textsc{sat}(\varphi_X, \mathbf{t})$
        if $\mathit{result}$ = UNSAT then
            return UNSAT, _, $\mathit{failed}$
        else if $\Psi$ is propositional then
            return SAT, $\mathbf{t}$, _
        $\mathbf{t_b} \leftarrow \{t_i \mid b_i \notin \mathbf{b},\, 1 \le i \le k\}$
        $\mathit{result}, \mathbf{t}', \mathit{failed}' \leftarrow \textsc{solve}_\forall(\Psi, \mathbf{t} \cup \mathbf{t_b})$
        if $\mathit{result}$ = UNSAT then
            $\varphi_X \leftarrow \varphi_X \wedge (\bigvee_{t \in \mathit{failed}'} \neg b_t)$
        else
            return SAT, $\mathbf{t}'$, _
```

```
procedure SOLVE_∀($\forall X.\, \Psi$, $\mathbf{t}$)
    while true do
        $\mathit{result}, \mathbf{t}', \mathit{failed} \leftarrow \textsc{sat}(\varphi_X, \mathbf{t}^+)$
        if $\mathit{result}$ = UNSAT then
            return SAT, $\mathit{failed}$, _
        $\mathit{result}, \mathbf{t}'', \mathit{failed}' \leftarrow \textsc{solve}_\exists(\Psi, \mathbf{t}')$
        if $\mathit{result}$ = SAT then
            $\varphi_X \leftarrow \varphi_X \wedge (\bigvee_{t \in \mathbf{t}''} \neg t)$
        else
            return UNSAT, _, $\mathit{failed}'$
```

Given a QBF with matrix ψ we prepare a SAT solver for each existential quantifier block $\exists X$ with the clauses

$$\varphi_X \coloneqq \bigwedge_{C_i \in \psi} \Big(\big(\bigvee_{l \in C_i \wedge var(l) \in X} l \big) \vee t_i \vee b_i \Big) , \tag{1}$$

and for each universal quantifier block $\forall X$, we prepare a SAT solver with the clauses

$$\varphi_X \coloneqq \bigwedge_{C_i \in \psi} \Big(\bigwedge_{l \in C_i \wedge var(l) \in X} (\bar{l} \vee t_i) \Big) . \tag{2}$$

In the construction of the clausal abstraction for the final level, i.e., φ_{X_k}, we additionally require the bottom literals B to be false, as there is no quantifier level below the current level to which we could pass the proof obligations.

To obtain the clausal abstraction of a QBF $\Psi(\mathbf{t})$ we then simply *assume* the assignment $\mathbf{t}$. Solving formulas under assumptions is a common feature of modern SAT solvers.

Note that formula (2) does not contain B variables. For the universal clausal abstractions it is possible to join the two types of literals and ask for a satisfying assignment of φ_X that assigns a minimal number of T variables to true, under the assumption that at least those that were given in the function call are true. This is merely a measure to reduce the number of variables used in the formula.

B. Algorithm with Optimizations

The input to the algorithm SOLVE consists of a QBF $Q_i X_i.\, \Psi$. Depending on the type of the outermost quantifier, it calls SOLVE$_\exists$ or SOLVE$_\forall$ and fixes an initial assignment $\mathbf{t} = \emptyset$ of T that indicates that no clauses were satisfied so far.

```
procedure SOLVE($\Phi$)
    return $\Phi = \exists X.\, \Psi$ ? SOLVE_∃($\Phi$, $\emptyset$) : SOLVE_∀($\Phi$, $\emptyset$)
```

The algorithm SOLVE$_\exists$ makes use of a feature of modern SAT solvers by *assuming* a partial assignment for a particular call. This feature enables us to deactivate those clauses that are satisfied already. The notation $\textsc{sat}(\varphi_X, \mathbf{t})$ denotes a SAT call for which we additionally assume the assignment $\mathbf{t}$. SAT calls with assumptions either return a satisfying assignment, which is guaranteed to have the sub-assignment $\mathbf{t}$, or in case the formula is unsatisfiable under the assumptions they additionally return a set of *failed assumptions*, denoted by *failed*. The failed assumptions are a subset of the assumptions $\mathbf{t}$ and suffice to make the formula unsatisfiable. In line 11, $\neg b_t$ denotes the variable $b \in B$ that corresponds to the same clause as variable t. The algorithm refines the clausal abstraction only by those variables B that made the recursive call to SOLVE$_\forall$ unsatisfiable. This significantly strengthens the refinement compared to the basic algorithm.

Similar to the algorithm SOLVE$_\exists$, the algorithm SOLVE$_\forall$ makes use of assumptions for SAT calls. Since, we joined the T and B variables, we only assume the positively assigned variables in $\mathbf{t}$, denoted by the call $\textsc{sat}(\varphi_X, \mathbf{t}^+)$. The SAT call then produces an assignment $\mathbf{t}'$ for which it possibly sets further variables T to true. In contrast to the existential case, we can use t' directly for the recursive call. The refinement step only refines by the variables T occurring in the assignment $\mathbf{t}''$ returned by the recursive call. The assignment $\mathbf{t}''$ is a subset of $\mathbf{t}'$ and therefore represents the information that the call to SOLVE$_\exists$ satisfied more clauses than required by the assignment $\mathbf{t}'$.

Theorem 1. *SOLVE(Φ) is correct and terminates.*

The proof is a simple induction over the quantifier quantifier hierarchy.

C. Stronger Refinements

The algorithm above always refines by a single clause. In certain cases, however, we can strengthen the refinement in SOLVE$_\exists$ by excluding a conjunction of clauses $\mathcal{C}$, that are equivalent in the following sense: If some clause C corresponds to a failed assumption, then all other clauses $\mathcal{C} \setminus \{C\}$ would also lead to a failed assumption. Formally, we characterize this criterion by the subset relation between clauses restricted to the lower level literals. Let $\exists_i X_i$ be a quantifier block of a QBF with matrix ψ, let failed' be the failed assumptions returned by the lower level (line 9), and let $t_k \in \mathit{failed}'$ be one of the failed assumptions. If C_k cannot be satisfied by a quantifier block $Q_j X_j$ with $j > i$, then any C_l with $C_l \cap \Big(\bigcup_{j>i} X_j \Big) \subseteq C_k \cap \Big(\bigcup_{j>i} X_j \Big)$, $C_l \subseteq_i C_k$ for short, cannot be satisfied either. Hence, given the failed assumptions failed', we refine

$$\bigvee_{t_k \in \mathit{failed}'} \; \bigwedge_{\substack{C_l \in \psi, \\ C_l \subseteq_i C_k}} \neg b_l \, , \tag{3}$$

```
p cap 3 3
d
d
6 -3
u SAT
d
4 5 3
u SAT
u SAT
1
u SAT
r SAT
```

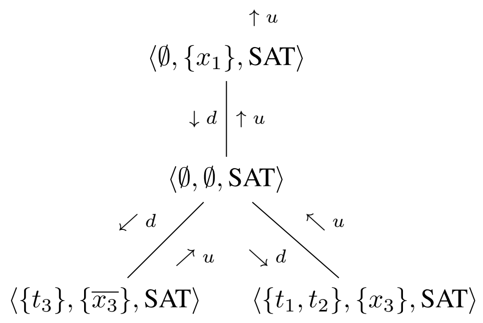
$\uparrow u$
$\langle\emptyset,\{x_1\},\mathrm{SAT}\rangle$
$\downarrow d$
$\uparrow u$
$\langle\emptyset,\emptyset,\mathrm{SAT}\rangle$
$\swarrow d$
$\nwarrow u$
$\nearrow u$
$\searrow d$
$\langle\{t_3\},\{\overline{x_3}\},\mathrm{SAT}\rangle$
$\langle\{t_1,t_2\},\{x_3\},\mathrm{SAT}\rangle$

Fig. 1. A clausal abstraction proof in the CAP format (left) and its tree structure (right).

The refinement has to be transformed to CNF using the Tseitin transformation [16].

Additionally, after we have found an assignment for an existential quantifier block, we have a routine that checks whether the assignment satisfies clauses that are deactivated by the current T assignment. If so, we delete the corresponding literals from the T assignment.

D. Preprocessing Techniques

We use basic preprocessing techniques that can be easily integrated into our certification infrastructure: tautology clauses, pure literals, unit clauses, universal reduction, and miniscoping. For miniscoping, we apply the well known rule $\forall X.\,\exists Y_1, Y_2.\,\varphi(X, Y_1) \wedge \psi(X, Y_2) \equiv (\forall X.\,\exists Y_1.\,\varphi(X, Y_1)) \wedge (\forall X.\,\exists Y_2.\,\psi(X, Y_2))$. That is, we search for a partitioning of the matrix according to the existential variables of the current scope. By applying this rule bottom-up, we get a tree-shaped quantification header. Note that this tree only branches after an existential quantifier, hence, we modify the algorithm to split the current entry according to the partitioning and solve every child individually. For true QBF instances, this transformation can significantly reduce the size of the Skolem functions.

V. Certification

Similar to the QBFCert [15] framework, we propose a two step approach to certification that allows us to keep the certification infrastructure separate from the solver. First, our solver outputs a *clausal abstraction proof* (CAP), that is a sequence of assignments of T variables together with the corresponding assignments of X variables as well as navigation symbols to determine the quantification level. A clausal abstraction proof is essentially a *post*-order linearization of the recursion tree.

Clausal abstraction proofs contain the following elements:

- Header: `p cap` v c where v is the maximal variable number and c is the number of clauses.
- Result: `r` *res* where *res* $\in \{\mathrm{SAT}, \mathrm{UNSAT}\}$.
- Quantifier tree navigation: `d` for **d**own, `u` *res* for **u**p with subtree result *res* $\in \{\mathrm{SAT}, \mathrm{UNSAT}\}$, and `n` for **n**ext sibling (in the case of miniscoping).
- T and variable assignments: $t_1\ t_2\ \ldots\ l_1\ l_2\ \ldots$, with $v < t_i \leq v + c$ for every i and $0 < |l_j| \leq v$ for every j.
- The strengthening instruction `s c` $t_1 \ldots t_n$ representing the cube of T variables used in the strong refinement optimization described by equation (3) in Section IV-C.

As an example, consider the following QBF:

$$\exists x_1 \forall x_2 \exists x_3 : \underbrace{(x_1 \vee x_2 \vee \overline{x_3})}_{t_1 \equiv x_4} \underbrace{(\overline{x_1} \vee x_2 \vee \overline{x_3})}_{t_2 \equiv x_5} \underbrace{(\overline{x_1} \vee \overline{x_2} \vee x_3)}_{t_3 \equiv x_6}$$

Figure 1 shows a clausal abstraction proof for this QBF. After the header, the proof descends to the lowest quantifier (lines `d` and `d`) where the decision $x_3 = 0$ is a satisfying assignment given that clause 3, corresponding to t_6, is satisfied by a higher level quantifier (line `6 -3`). The algorithm presented in Section IV guarantees that there is an assignment of the higher quantifier levels that satisfies the this set of clauses (i.e. clause 3). The proof format, however, delays printing the assignments of higher quantifier levels until it *successfully* ascends to the upper levels of the proof. Omitting assignments for failed proof branches saves a significant amount of space. Next we ascend to the universal quantifier level (line `u SAT`). The previous assignment of the universal quantified variable x_2 can be omitted, as it did not refute the current proof branch. The universal level chooses a new assignment for x_2 and the proof descends again (line `d`). This time clauses 1 and 2, corresponding to t_4 and t_5, are satisfied by a higher level clause and $x_3 = 1$ is a satisfying assignment for the remaining clauses (line `4 5 3`) and we ascend again (line `u SAT`). We exhausted the assignments of x_2 and the proof hence ascends from the universal quantifier level (line `u SAT`). Upon returning successfully from this proof branch, we print the assignment of $x_1 = 1$ (line `1`) that was chosen in the beginning. We return successfully from the outermost existential quantifier level (line `u SAT`) and the QBF is concluded to be true (`r SAT`).

We proceed with the general description of the certification approach. From the clausal abstraction proof, we build a circuit—encoded as an And-Inverter-Graph (AIG)—representing the Skolem or Herbrand function. First, we parse the clausal abstraction proof into a tree structure, where the levels of the tree correspond to the quantifier blocks of the QBF. Since the clausal abstraction proof is a post-order linearization, we can build the proof tree bottom-up. For a quantifier block $Q\,X.\,\Psi$, a node $\langle \mathbf{t}, \mathbf{x}, r\rangle$ in the tree is a tuple consisting of a T assignment $\mathbf{t}$, an X assignment $\mathbf{x}$, and the subtree result $r \in \{\mathrm{SAT}, \mathrm{UNSAT}\}$. Next, we prune the tree according to the result: If the QBF is true, we dismiss universal levels as well as nodes that are labeled as UNSAT. Analogously, if the QBF is false, we dismiss existential levels as well as nodes with $r = \mathrm{SAT}$. All remaining nodes are relevant for the certificate.

Given a quantifier block $Q\,X.\,\Psi$, we first collect the list of nodes $\mathcal{N}_X$ corresponding to this level (according to the navigation commands in the proof trace). For every variable $x \in X$ we then build the Skolem/Herbrand function f_x with the algorithm ConstructFunction, which takes three arguments: the variable x, the list of nodes $\mathcal{N}_X$, and the type of quantifier $Q \in \{\exists, \forall\}$.

```
1: procedure ConstructFunction($x$, $\mathcal{N}$, $Q$)
2:     $f_x \leftarrow$ false
3:     $f_{pre} \leftarrow$ true
4:     for $\langle \mathbf{t}, \mathbf{x}, r \rangle \in \mathcal{N}$ do
5:         if $x \in \mathbf{x}$ then
6:             $f_x \leftarrow f_x \vee (f_{pre} \wedge \text{Precondition}(\mathbf{t}, Q))$
7:         $f_{pre} \leftarrow f_{pre} \wedge \neg\text{Precondition}(\mathbf{t}, Q)$
8:     return $f_x$
```

```
1: procedure Precondition($\mathbf{t}, Q$)
2:     if $Q = \exists$ then
3:         return $\bigwedge_{t \in \mathbf{t}} \left( \bigvee_{l \in C_t \wedge var(l) \in \bigcup_{j<i} X_i} l \right)$
4:     else
5:         return $\bigwedge_{t \in T \setminus \mathbf{t}} \left( \bigwedge_{l \in C_t \wedge var(l) \in \bigcup_{j<i} X_i} \bar{l} \right)$
```

In the algorithm above the formula f_x characterizes when the Skolem/Herbrand function will set x to true and f_{pre} characterizes the cases that are not yet covered by f_x. In each iteration over the list of nodes we extend f_{pre} by the case that is covered by the current node (line 7). If the variable x occurs positively in the assignment, we also extend the function f_x (line 6). Each case is described by the conjunction over the clauses described by the T assignment restricted to the variables of smaller quantifier levels (see algorithm Precondition). For a given $t \in T$, we denote with C_t the clause that corresponds to the t variable. The formulas f_x computed by the algorithm ConstructFunction then define the output signals of the AIG.

In the example of Fig. 1, the list of nodes for x_1 consists of the single node: $\langle \emptyset, \{x_1\}, \text{SAT} \rangle$, indicating that without precondition ($\emptyset$) x_1 is set to true. For x_3 there are two nodes: $\langle \{t_3\}, \{\overline{x_3}\}, \text{SAT} \rangle$ and $\langle \{t_1, t_2\}, \{x_3\}, \text{SAT} \rangle$, indicating that if clause 3 is satisfied, x_3 is set to false, and if clauses 1 and 2 are satisfied, x_3 is set to true. The Skolem function computed by the algorithm above is $f_{x_3} = (x_1 \vee x_2) \wedge (\overline{x_1} \vee x_2)$, which simplifies to $f_{x_3} = x_2$.

For a true (false) QBF, the resulting AIG certificates can be checked by substituting the existential (universal) variables in matrix by applications of their Skolem (Herbrand) functions and then query a SAT solver to ask for an assignment of the variables such that some clause is falsified (all clauses are satisfied). The certificate is valid if, and only if, the SAT solver returns UNSAT and the Skolem/Herbrand functions depend only on variables in their dependency set.

VI. Experimental Evaluation

We implemented the algorithm and its optimizations in a tool named CAQE[1] (Clausal Abstraction for Quantifier Elimination). The tool is written in the programming language C and we use a generic SAT solver interface that can be instantiated with PicoSAT [17] (default) or MiniSat [18]. In this section, we evaluate the implementation on the instances from QBF Gallery 2014 [19] in several categories: the number of solved instances per benchmark family with and without preprocessing, the number of instances solved in certification mode, and the size of the generated certificates. We compare CAQE to the (available) best performing solvers of the QBF Gallery 2014. For our experiments, we used a machine with a 3.6 GHz quad-core Intel Xeon processor and 32 GB of memory. The timeout was set to 10 minutes.

[1]available at http://react.uni-saarland.de/tools/caqe/

A. Solved Instances per Family

Table I shows for each solver how many instances of the QBFGallery benchmark set are solved within 10 minutes. We removed the preprocessing track of QBFGallery and instead ran every solver with and without preprocessing using Bloqqer [20]. For three of the seven families, a configuration of CAQE solved the highest number of instances. Overall, CAQE using PicoSAT ranked second after RAReQS when using the number of solved instances as a measure.

Most notably, CAQE solved an exceptionally high number of problems in the *hardness* family, which consists of bounded model checking queries for incomplete designs [6]. One characteristic of this family is that number of quantifier alternations is relatively high; going up to 60 alternations. Figure 2 shows the performance of all solvers for the 188 instances of the full benchmark set that have a high number of quantifier alternations. The plot suggests that CAQE performs well on instances with a high number of quantifier alternations—in particular compared to other CEGAR-based solvers.

Unsurprisingly, the choice of the underlying SAT solver has a significant impact on the performance of CAQE. In this setting, the variant using PicoSAT performs better, however, more testing and optimization was done for this variant.

The effect of preprocessing is unusual. While preprocessing by Bloqqer improved the performance overall (in particular for CAQE using MiniSat), the analysis per family reveals that Bloqqer decreased the performance of CAQE using PicoSAT for three benchmark families.

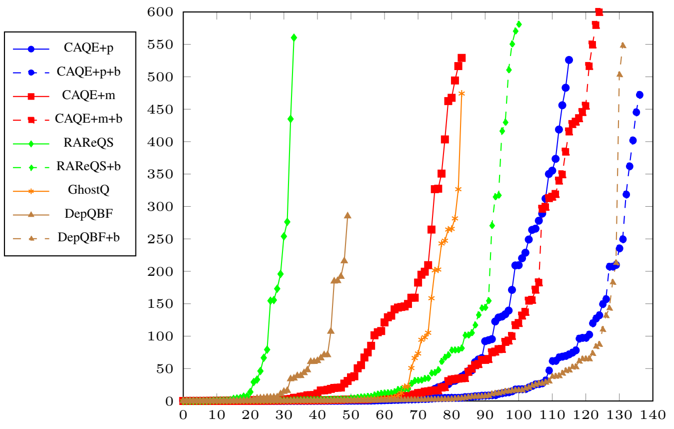

Fig. 2. Number of solved instances within 10 minutes among the 188 instances from QBFGallery 2014 with more than 6 quantifier alternations.

TABLE I
NUMBER OF INSTANCES OF THE QBFGALLERY 2014 BENCHMARK SET SOLVED WITHIN 10 MINUTES.

		CAQE				RAReQS		GhostQ	DepQBF	
Family	total	picosat	picosat+bloqqer	minisat	minisat+bloqqer	rareqs	rareqs+bloqqer	ghostq	depqbf	depqbf+bloqqer
eval2012r2	276	75	112	55	98	81	**129**	124	88	128
bomb	132	**91**	74	75	59	84	82	75	67	80
complexity	104	50	67	60	67	75	**91**	26	49	57
dungeon	107	46	31	22	**69**	57	62	45	44	66
hardness	114	78	**103**	58	94	15	68	57	8	81
planning	147	84	79	50	55	**146**	135	31	57	47
testing	131	54	77	25	84	36	92	**102**	57	76
all	1011	478	543	345	526	494	**659**	460	370	535

B. Certificates

Table II shows the number of instances of the QBF Gallery 2014 benchmark set that was solved within 10 minutes in certifying mode (#solved) and the number of certificates that was verified within 10 minutes (#verified). The evaluation also shows that CAQE can certify more results than DepQBF and provides certificates that are only half the size in average. The size of the certificates is measured in terms of AND-gates in the AIGER file that encodes the Skolem or Herbrand function after minimization with the DFRAIG algorithm of ABC [21].

Compared to the non-certifying run in Table I, the solvers solved between 84% (DepQBF) and 90% (CAQE) of the instances in certifying mode. This can be traced back to two factors. First, certain optimizations have to be disabled in certification mode, and second, there is a significant amount of time spent writing the proofs to disk. Furthermore, not all of the instances solved in certification mode could be verified within the given amount of verification time.

Lastly, we compare the size of the certificates of CAQE and DepQBF on the commonly solved instances in Fig. 3. The variance of the relative certificate sizes is high, but the number of instances where CAQE generated certificates of significantly smaller size than DepQBF is larger than the number of instances where DepQBF generated certificates of significantly smaller size than CAQE.

TABLE II
CERTIFYING RUNS OF DEPQBF AND CAQE.

Solver	# solved	# verified	# unique	avg. size
CAQE	428	340	146	3138
DepQBF	312	239	44	7447
virtual best	468	384	-	5357

VII. Related Work

Various techniques have been proposed for QBF, including expansion [22], [23], BDDs [24], [25], (DPLL-like) search [14], [26], and CEGAR [9]. The CEGAR approach has been first explored in the context of model checking [27]. Janota and Silva successfully applied CEGAR to 2QBF [9].

RAReQS: Subsequently, Janota et al. extended the CEGAR approach to full QBF [8]. They implemented the approach in the tool RAReQS (and to a certain extend also in GhostQ), which lead to significant performance gains for several problem families. To evaluate the truth of a QBF $\varphi = Q_1 X_1 \dots Q_n X_n . \varphi$ with n quantifier alternations, the algorithm picks an assignment $\mathbf{x}_1$ to X_1 and recursively determines the truth of $\varphi'(\mathbf{x}_1) = Q_2 X_2 \dots Q_n X_n . \varphi(\mathbf{x}_1)$. If that call returns a counter-example, that is an assignment $\mathbf{x}_2$ to X_2, then φ is refined by the formula $\varphi'' = Q_1 X_1 . Q_3 X_3 \dots Q_n X_n . \varphi(\mathbf{x}_2)$, which has two quantifier alternations fewer than φ. Before checking $\varphi'(\mathbf{x}_1')$ for other assignments $\mathbf{x}_1'$, it is first checked whether the assignment is already excluded by $\mathbf{x}_1'$. (That is, we first check $\varphi''(\mathbf{x}_1')$.) In this way, the assignment $\mathbf{x_1}$ cannot occur a second time as a counter-example. A potential problem with this approach is that the formula φ'' is itself a QBF, and may itself be refined with other QBFs in later iterations. We may therefore have to check each new assignment $\mathbf{x}_1'$ for a *tree* of counter-examples that each are QBFs, and the size of the tree of counter-examples may grow exponentially with the quantifier alternation depth. Our experiments suggest that this problem actually occurs in practice: while being very effective for low quantifier alternation depths, RAReQS solves only few instances with a higher number quantifier alternations.

In this work we propose an alternative CEGAR algorithm in which we refine only by single clauses. This notion of refinement coincides with the RAReQS refinement step in the case of 2QBF, but for two or more quantifier alternations it is *weaker*: Assignments that lead to a counter-example may reappear later. This explains why RAReQS outperforms CAQE on benchmarks with a low number of quantifier alternations. The weaker notion of refinement in CAQE, however, avoids the need for the tree of counter-examples and therefore scales well to instances with many quantifier alternations.

Clause selection: Very recently and independently from this work, Janota and Marques-Silva proposed *clause selection* and implemented the approach in the tool Qesto [28]. Similar to clausal abstractions, they reason about the satisfaction of sets of clauses and the algorithms have a similar structure. The encodings of the individual quantifier blocks, however, reveal interesting differences in the execution of this idea:

- Qesto uses equality constraints for the variables connecting the quantification levels while CAQE uses implications, requiring fewer clauses in the encoding.
- For universally quantified levels, CAQE needs to add only one variable per clause, while Qesto needs two.
- Qesto considers the clauses for existential and universal

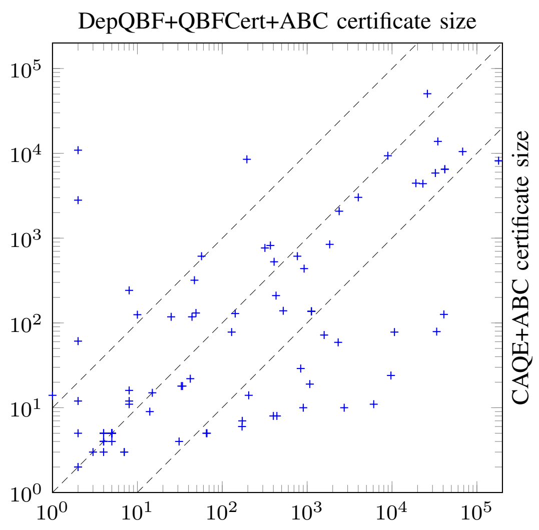

Fig. 3. The size of certificates computed by CAQE and DepQBF.

levels in a negated form, while CAQE does not negate existential levels.

It will be interesting to see whether CAQE and Qesto share the same runtime characteristics or whether these are particular to our encoding, and whether Qesto can be extended to certification as well.

Certifying QBF: Certifying QBF solvers enable a rich set of applications like encodings of bounded model checking [3]–[6] and synthesis that use the certificates as implementations [12]. Previous certifying QBF solvers were based on a DPLL-like search [14], [15] or expansion [29]. Our work shows how to enable certification for the CEGAR approach.

There has been work on certifying QBF preprocessing techniques based on QRAT proofs in Bloqqer [30], [31]. It may be possible to integrate CAQE in combination with Bloqqer in a similar setting as in [32].

VIII. Conclusions and Future work

We presented clausal abstractions, a decomposition of QBFs into sequences of propositional formulas, and a new CEGAR algorithm for QBF. The overall performance is competitive and our experiments suggest that the new algorithm is effective for instances with a high number of quantifier alternations. We showed how to certify the results of the algorithm and the evaluation shows that significantly more instances can be certified with our solver compared to the state-of-the-art.

In the future, we plan to consider joining the two notions of refinement used in RAReQS and CAQE and to integrate our algorithm in a certification framework like [32] to enable its use together with certified preprocessing.

References

[1] M. Benedetti and H. Mangassarian, “QBF-based formal verification: Experience and perspectives,” *JSAT*, vol. 5, no. 1-4, pp. 133–191, 2008.
[2] W. Zhang, “QBF encoding of temporal properties and QBF-based verification.” in *IJCAR*, 2014, pp. 224–239.
[3] T. Jussila and A. Biere, “Compressing BMC encodings with QBF.” *Electr. Notes Theor. Comput. Sci.*, pp. 45–56, 2007.
[4] B. Finkbeiner and L. Tentrup, “Fast DQBF refutation,” in *Proceedings of SAT*, 2014, pp. 243–251.
[5] N. Dershowitz, Z. Hanna, and J. Katz, “Bounded model checking with QBF,” in *Proceedings of SAT*, 2005, pp. 408–414.
[6] C. Miller, C. Scholl, and B. Becker, “Proving QBF-hardness in bounded model checking for incomplete designs,” in *Proceedings of MTV*, 2013, pp. 23–28.
[7] M. N. Rabe, C. M. Wintersteiger, H. Kugler, B. Yordanov, and Y. Hamadi, “Symbolic approximation of the bounded reachability probability in large Markov chains,” in *Proceedings of QEST*. Springer, 2014, pp. 388–403.
[8] M. Janota, W. Klieber, J. Marques-Silva, and E. M. Clarke, “Solving QBF with counterexample guided refinement.” in *Proceedings of SAT*, 2012, pp. 114–128.
[9] M. Janota and J. P. M. Silva, “Abstraction-based algorithm for 2QBF,” in *Proceedings of SAT*, 2011, pp. 230–244.
[10] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, “Symbolic model checking without BDDs,” in *Proceedings of TACAS*, 1999, pp. 193–207.
[11] A. Solar-Lezama, R. M. Rabbah, R. Bodík, and K. Ebcioglu, “Programming by sketching for bit-streaming programs,” in *Proceedings of PLDI*, 2005, pp. 281–294.
[12] R. Bloem, U. Egly, P. Klampfl, R. Könighofer, and F. Lonsing, “SAT-based methods for circuit synthesis,” in *Proceedings of FMCAD*, 2014, pp. 31–34.
[13] B. Finkbeiner and L. Tentrup, “Detecting unrealizable specifications of distributed systems,” in *Proceedings of TACAS*, 2014, pp. 78–92.
[14] F. Lonsing and A. Biere, “DepQBF: A dependency-aware QBF solver,” *JSAT*, vol. 7, no. 2-3, pp. 71–76, 2010.
[15] A. Niemetz, M. Preiner, F. Lonsing, M. Seidl, and A. Biere, “Resolution-based certificate extraction for QBF - (tool presentation),” in *Proceedings of SAT*, 2012, pp. 430–435.
[16] G. S. Tseitin, “On the complexity of derivation in propositional calculus,” *Studies in constructive mathematics and mathematical logic*, vol. 2, no. 115-125, pp. 10–13, 1968.
[17] A. Biere, “PicoSAT essentials,” *JSAT*, vol. 4, no. 2-4, pp. 75–97, 2008.
[18] N. Eén and N. Sörensson, “An extensible SAT-solver,” in *Proceedings of SAT*, 2003, pp. 502–518.
[19] “QBF Gallery - Joint Evaluation of Quantified Boolean Formulas,” 2014, http://qbf.satisfiability.org/gallery/.
[20] A. Biere, F. Lonsing, and M. Seidl, “Blocked clause elimination for QBF,” in *Proceedings of CADE-23*, 2011, pp. 101–115.
[21] R. K. Brayton and A. Mishchenko, “ABC: An academic industrial-strength verification tool,” in *Proceedings of CAV*, 2010, pp. 24–40.
[22] A. Biere, “Resolve and expand,” in *Proceedings of SAT*, 2004.
[23] F. Lonsing and A. Biere, “Nenofex: Expanding NNF for QBF solving,” in *Proceedings of SAT*, 2008, pp. 196–210.
[24] G. Audemard and L. Sais, “A symbolic search based approach for quantified boolean formulas,” in *Proceedings of SAT*, 2005, pp. 16–30.
[25] O. Olivo and E. A. Emerson, “A more efficient BDD-based QBF solver,” in *Proceedings of CP*, 2011, pp. 675–690.
[26] E. Giunchiglia, M. Narizzano, and A. Tacchella, “QuBE: A system for deciding quantified boolean formulas satisfiability.” in *Proceedings of IJCAR*, 2001, pp. 364–369.
[27] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, “Counterexample-guided abstraction refinement for symbolic model checking,” *J. ACM*, vol. 50, no. 5, pp. 752–794, 2003.
[28] M. Janota and J. Marques-Silva, “Solving QBF by clause selection,” in *Proceedings of IJCAI*. AAAI Press, 2015, pp. 325–331.
[29] A. Goultiaeva, A. Van Gelder, and F. Bacchus, “A uniform approach for generating proofs and strategies for both true and false QBF formulas,” in *Proceedings of IJCAI*, 2011, pp. 546–553.
[30] M. Heule, M. Seidl, and A. Biere, “A unified proof system for QBF preprocessing,” in *Proceedings of IJCAR*, ser. LNCS, vol. 8562. Springer, 2014, pp. 91–106.
[31] ——, “Efficient extraction of skolem functions from QRAT proofs,” in *Proc. of FMCAD*, 2014, pp. 107–114.
[32] M. Janota, R. Grigore, and J. Marques-Silva, “On QBF proofs and preprocessing,” in *Proceedings of LPAR*, 2013, pp. 473–489.