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 it 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 rise 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].

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:

\[ \forall x \exists y. \left( x_1 \lor \neg x_2 \lor y_1 \right) \land \left( x_2 \lor \neg y_1 \lor y_2 \right) \land \left( y_1 \lor y_2 \right) \]

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 \( x_1 \neg x_2 \) 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 \( Y \):

\[ \varphi_X := \left( \left( x_1 \lor \neg x_2 \right) \rightarrow \neg b_2 \right) \land \left( x_2 \rightarrow \neg b_2 \right) \land \left( \text{false} \rightarrow \neg b_3 \right) \],

\[ \varphi_Y := \left( t_1 \lor y_1 \right) \land \left( t_2 \lor \neg y_1 \lor y_2 \right) \land \left( t_3 \lor y_1 \lor y_2 \right) \],

where the variables \( B = \{ b_1, b_2, b_3 \} \) (bottom) indicate that the clause is satisfied by the lower quantifier level (\( \forall X \)), and the variables \( T = \{ t_1, t_2, t_3 \} \) (top) indicate that the clause is satisfied by the upper quantifier level (\( \forall Y \)). That is, for every clause \( \varphi_X \) requires that whenever the clause is satisfied by the \( X \) variables, it does not have to be satisfied by the \( Y \) variables. And \( \varphi_Y \) 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 \( \varphi_X \) that includes the assignment \( b \) of \( B \), the formula \( \varphi_Y (t_h) \) is satisfiable, where \( t_h \) is the assignment of \( T \) that assigns \( t_i \) iff \( b_i \) is not assigned in \( b \) (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 \( \varphi_X \) and \( \varphi_Y \). When there is no assignment of \( \varphi_X \) left, we conclude that the original formula is true, or when there is no satisfying assignment of \( \varphi_Y \) 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 $\alpha$ of $\varphi_Y$ defines a single clause over the variables $B$ that we can add to the formula $\varphi_X$. This excludes all assignments of $X$ for which $\alpha$ 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:

$$
\varphi ::= x \mid \neg \varphi \mid (\varphi) \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid \exists x. \varphi \mid \forall x. \varphi,
$$

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 $x \subseteq X$ of variables that are assigned 1 (or true). We denote the set of assignments of a set of variables $X$ by $A(X)$.

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

The dependency set of an existentially quantified variable $y$, denoted by $\text{dep}(y)$, is the set $X$ of universally quantified variables $x$ such that $\exists y. \varphi$ is in the scope of $x$. A Skolem function $f_y : \mathcal{A(\text{dep}(y))} \to \mathbb{B}$ maps an assignment of the dependencies of $y$ to an assignment of $y$. The truth of a QBF $\varphi$ is equivalent to the existence of a Skolem function $f_y$ for every variable $y$ of the existentially quantified variables $Y$, such that $\{y \in Y \mid f_y(x \cap \text{dep}(y))\} \models \varphi$ holds for every assignment $x$ 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 $\varphi$ is in the $k$QBF 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 $(l_1 \lor \ldots \lor l_n)$ is called a clause and the conjunctive combination $(l_1 \land \ldots \land l_n)$ is called a cube. Given a literal $l$, the polarity of $l$, $\text{sign}(l)$ for short, is 1 if $l$ is positive and 0 otherwise. The variable corresponding to $l$ is defined as $\text{var}(l) = x$ where $x = l$ if $\text{sign}(l) = 1$ 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 $\psi$ as a set of clauses $\psi = \{C_1, \ldots, C_n\}$ and a clause $C$ as a set of literals $C = \{l_1, \ldots, l_n\}$ 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 $\varphi$ in prenex conjunctive normal form:

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

where $\psi = C_1 \land \ldots \land C_k$ is the matrix of $\varphi$ 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 $\varphi$ to be true means to find an assignment $x_1$ for $X_1$ such that the remaining formula $Q_2 X_2 \ldots Q_n X_n. \psi(x_1)$ is true. Inspecting $\psi(x_1)$ 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 $C_{i,1}$
When we build the clausal abstraction matrix that results from fixing an assignment to a maximal number of variables set to true. The maximal assignment is the set of variables \( i \), exactly the set \( \{ b \mid b \in b_{\min}(x_1) \} \) in the remaining matrix \( \psi_{\geq 1} = \bigwedge_{l<k} C_{l,1} \).

Now let’s turn to the case that the outermost quantifier is a universal quantifier. Analogous to the previous case, disproving \( \varphi \) means to show that there is an assignment \( x_1 \) of \( X_1 \) such that the remaining formula \( Q_2 X_2 \ldots Q_n X_n \cdot \psi(x_1) \) 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 \( x_1 \). 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:

\[
C_{i,1} = \bigwedge_{l \in C_i \land \varphi(l)} \psi_{\geq 1} \setminus \{ t_i \}
\]

\[
C_{i,>1} = \bigwedge_{l \in C_i \land \varphi(l)} \psi_{\geq 1}(\{ l_i \}) \setminus \{ t_i \}
\]

The universal clausal abstraction for \( X_1 \) is then the conjunction \( \varphi_{\forall X_1} = \bigwedge_{l<k} C_{j,1} \). The universal clausal abstraction guarantees that for every assignment \( x_1 \) to \( X_1 \) there is a unique maximal assignment \( b_{\max}(x_1) \) 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 \( x_1 \) in \( C_1 \land \ldots \land C_k \) is the same as the matrix that results from fixing \( t = \{ t_i \mid b_i \notin b_{\max}, 1 \leq i \leq k \} \) in the remaining formula \( \psi_{\geq 1} = C_{1,>1} \land \ldots \land C_{k,>1} \).

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 \( \psi_{\geq 1} = Q_2 X_2 \ldots Q_n X_n \cdot \psi_{\geq 1} \) to denote the formula that remains when splitting the clausal abstraction \( \varphi_{Q_i X_i} \) from the QBF \( \Psi \). Expressions of the form \( c \cdot e_1 : e_2 \) denote abbreviated if-statements. If the conditional \( c \) evaluates to true we return \( e_1 \) and otherwise \( e_2 \).

The input to the algorithm \textbf{SOLVE} is a QBF \( Q X \). The output is either SAT or UNSAT.

1: \textbf{procedure} \textbf{SOLVE}(QX, \Psi)
2: \hspace{1em} if \( \Psi \) is propositional then
3: \hspace{2em} \return SAT(\Psi)
4: \hspace{1em} else
5: \hspace{2em} \alpha \leftarrow (Q = \exists) \cdot \varphi_{\exists X} : \varphi_{\forall X}
6: \hspace{1em} while true do
7: \hspace{2em} if result = UNSAT then
8: \hspace{3em} return \textbf{SAT}(Q = \exists) \cdot \Psi : SAT
9: \hspace{3em} result \leftarrow \textbf{SAT}(\alpha)
10: \hspace{3em} \textbf{return} \textbf{SAT}(Q = \exists) \cdot \Psi : SAT
11: \hspace{2em} if result = UNSAT then
12: \hspace{3em} \alpha \leftarrow \alpha \land \left( \bigvee_{l \in b} t_i \right)
13: \hspace{2em} else if result = UNSAT then
14: \hspace{3em} \alpha \leftarrow \alpha \land \left( \bigvee_{l \in b} l \right)
15: \hspace{2em} else
16: \hspace{3em} return \textbf{UNSAT}(Q = \exists) \cdot \Psi : UNSAT

The algorithm generates an assignment \( b \) for the variables \( B \) in the clausal abstraction, determines \( t = \{ t_i \mid b_i \notin b, 1 \leq i \leq k \} \), and then goes into recursion for the remaining formula \( Q_2 X_2 \ldots Q_n X_n \cdot \psi_{\geq 1}(t) \). Whenever a recursive call failed for an existential quantifier block, i.e., the remaining formula turned out to be false, we add the clause \( \bigvee_{l \in b} l \) to the universal 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 \( \bigvee_{l \in b} t_i \) 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.
Given a QBF with matrix $\psi$ we prepare a SAT solver for each existential quantifier block $\exists X$ with the clauses
\[
\varphi_X := \bigwedge_{C_i \in \psi} \left( \bigvee_{l \in C_i \land \text{var}(l) \in X} l \right) \lor t_i \lor b_i ,
\]
and for each universal quantifier block $\forall X$, we prepare a SAT solver with the clauses
\[
\varphi_X := \bigwedge_{C_i \in \psi} \left( \bigwedge_{l \in C_i \land \text{var}(l) \in X} (\bar{l} \lor t_i) \right) .
\]
In the construction of the clausal abstraction for the final level, i.e., $\varphi_X$, 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 $\Psi(t)$ we then simply assume the assignment $t$. 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 $\varphi_X$ 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 $\text{SOLVE}_{3}$ consists of a QBF $Q, X_t, \Psi$. Depending on the type of the outermost quantifier, it calls $\text{SOLVE}_{3}$ or $\text{SOLVE}_{\bar{v}}$ and fixes an initial assignment $t = \emptyset$ of $T$ that indicates that no clauses were satisfied so far.

1: procedure $\text{SOLVE}_{3}(\exists X, \Psi, t)$
2: while true do
3: \hspace{1em} result, $b$, failed $\leftarrow$ SAT($\varphi_X, t$)
4: \hspace{1em} if result = UNSAT then
5: \hspace{2em} return UNSAT, \_failed
6: \hspace{1em} else if $\Psi$ is propositional then
7: \hspace{2em} return SAT, \_\_\_
8: \hspace{1em} $t_b \leftarrow \{t_i | b_i \notin B, 1 \leq i \leq k\}$
9: \hspace{1em} result, $t', \text{failed}' \leftarrow \text{SOLVE}_{3}(\Psi, t \cup t_b)$
10: \hspace{1em} if result = UNSAT then
11: \hspace{2em} $\varphi_X \leftarrow \varphi_X \land \bigwedge_{i \in \text{failed}'} \neg b_i$
12: \hspace{1em} else
13: \hspace{2em} return SAT, $t', \_\_\_\_\_\_

Additionally return a set of failed assumptions, denoted by failed. The failed assumptions are a subset of the assumptions $t$ and suffice to make the formula unsatisfiable. In line 11, $\neg b_i$ denotes the variable $b_i \in B$ that corresponds to the same clause as variable $t_i$. The algorithm refines the clausal abstraction only by those variables $B$ that made the recursive call to $\text{SOLVE}_{3}$ unsatisfiable. This significantly strengthens the refinement compared to the basic algorithm.

Similar to the algorithm $\text{SOLVE}_{3}$, the algorithm $\text{SOLVE}_{\bar{v}}$ makes use of assumptions for SAT calls. Since, we joined the $T$ and $B$ variables, we only assume the positively assigned variables in $t$, denoted by the call SAT($\varphi_X, t^+$). The SAT call then produces an assignment $t'$ 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 $t''$ returned by the recursive call. The assignment $t''$ is a subset of $t'$ and therefore represents the information that the call to $\text{SOLVE}_{3}$ satisfied more clauses than required by the assignment $t'$.

**Theorem 1.** $\text{SOLVE}(\Phi)$ 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 $\text{SOLVE}_{3}$ by excluding a conjunction of clauses $C$, that are equivalent in the following sense: If some clause $C$ corresponds to a failed assumption, then all other clauses $C \setminus \{C\}$ 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 $\exists X_t$ be a quantifier block of a QBF with matrix $\psi$, let $\text{failed}'$ be the failed assumptions returned by the lower level (line 9), and let $t_k \in \text{failed}'$ be one of the failed assumptions. If $C_k$ cannot be satisfied by a quantifier block $Q, X_j$ with $j > i$, then any $C_l$ with $C_l \cap \bigcup_{j>i} X_j \subseteq C_k \cap \bigcup_{j>i} X_j$, $C_l \subseteq_i C_k$ for short, cannot be satisfied either. Hence, given the failed assumptions $\text{failed}'$, we refine
\[
\bigvee_{t_k \in \text{failed}'} \bigwedge_{C_l \in \psi, C_l \subseteq_i C_k} \neg b_l ,
\]
determine the quantification level. A clausal abstraction proof of assignments of clausal abstraction proof (CAP), that is a sequence 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 \( \forall \ SAT \)). 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 \( \exists \ SAT \)). 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 \( \forall \ SAT \)). We exhausted the assignments of \( x_2 \) and the proof hence ascends from the universal quantifier level (line \( \exists \ SAT \)). 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 \( \exists \ SAT \)) and the QBF is concluded to be true (\( \exists \ SAT \)).

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 \( \langle t, x, r \rangle \) in the tree is a tuple consisting of a \( T \) assignment \( t \), an \( X \) assignment \( x \), and the subtree result \( r \in \{ \text{SAT}, \text{UNSAT} \} \). 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 \( r = \text{SAT} \). 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 \textsc{ConstructFunction}, which takes three

\[
\exists x_1 \forall x_2 \exists x_3 : (x_1 \lor x_2 \lor \overline{x_3}) (x_1 \lor x_2 \lor \overline{x_3}) (x_1 \lor x_2 \lor x_3)
\]

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 that clause 3, corresponding to \( t_6 \), is satisfied by a higher level quantifier (line \( \exists \ SAT \)). 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 \( \forall \ SAT \)). We exhausted the assignments of \( x_2 \) and the proof hence ascends from the universal quantifier level (line \( \exists \ SAT \)). 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 \( \exists \ SAT \)) and the QBF is concluded to be true (\( \exists \ SAT \)).

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 \( \langle t, x, r \rangle \) in the tree is a tuple consisting of a \( T \) assignment \( t \), an \( X \) assignment \( x \), and the subtree result \( r \in \{ \text{SAT}, \text{UNSAT} \} \). 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 \( r = \text{SAT} \). 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 \textsc{ConstructFunction}, which takes three
arguments: the variable \( x \), the list of nodes \( N_x \), and the type of quantifier \( Q \in \{ \exists, \forall \} \).

1: procedure ConstructFunction\((x, N, Q)\)
2: \( f_x \leftarrow \text{false} \)
3: \( f_{\text{pre}} \leftarrow \text{true} \)
4: for \( \{t, x, r\} \in N \) do
5: \( \text{if } x \in x \text{ then} \)
6: \( f_x \leftarrow f_x \lor (f_{\text{pre}} \land \text{PRECONDITION}\(t, Q)\))
7: \( f_{\text{pre}} \leftarrow f_{\text{pre}} \land \neg\text{PRECONDITION}\(t, Q)\)
8: return \( f_x \)

In the algorithm above the formula \( f_x \) characterizes when the Skolem/Herbrand function will set \( x \) to true and \( f_{\text{pre}} \) characterizes the cases that are not yet covered by \( f_x \). In each iteration over the list of nodes we extend \( f_{\text{pre}} \) 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: \( \{t_3\}, \{x_3\}, \text{SAT} \) and \( \{t_1, t_2\}, \{x_3\}, \text{SAT} \), 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 \( f_{x_3} = (x_1 \lor x_2) \land (\overline{x_1} \lor x_2) \), 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.

A. Solved Instances per Family

Table I shows for each solver how many instances of the QBF Gallery benchmark set are solved within 10 minutes. We removed the preprocessing track of QBF Gallery and instead ran every solver with and without preprocessing using Bloqger [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 Bloqger improved the performance overall (in particular for CAQE using MiniSat), the analysis per family reveals that Bloqger decreased the performance of CAQE using PicoSAT for three benchmark families.
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>
<thead>
<tr>
<th>Family</th>
<th>total</th>
<th>picosat</th>
<th>picosat+bloq</th>
<th>CAQE</th>
<th>minisat</th>
<th>minisat+bloq</th>
<th>RAREQS</th>
<th>RAREQS+bloq</th>
<th>GHOSTQ</th>
<th>DEPQBF</th>
<th>DEPQBF+bloq</th>
</tr>
</thead>
<tbody>
<tr>
<td>eval2012r</td>
<td>276</td>
<td>75</td>
<td>112</td>
<td>55</td>
<td>98</td>
<td></td>
<td>81</td>
<td>129</td>
<td>14</td>
<td>38</td>
<td>128</td>
</tr>
<tr>
<td>bomb</td>
<td>132</td>
<td>91</td>
<td>74</td>
<td>75</td>
<td>59</td>
<td></td>
<td>84</td>
<td>82</td>
<td>75</td>
<td>67</td>
<td>80</td>
</tr>
<tr>
<td>complexity</td>
<td>104</td>
<td>50</td>
<td>67</td>
<td>60</td>
<td>67</td>
<td></td>
<td>75</td>
<td>91</td>
<td>26</td>
<td>49</td>
<td>57</td>
</tr>
<tr>
<td>dungeon</td>
<td>107</td>
<td>46</td>
<td>31</td>
<td>22</td>
<td>69</td>
<td></td>
<td>57</td>
<td>62</td>
<td>45</td>
<td>44</td>
<td>66</td>
</tr>
<tr>
<td>hardness</td>
<td>114</td>
<td>78</td>
<td>103</td>
<td>58</td>
<td>94</td>
<td></td>
<td>15</td>
<td>68</td>
<td>57</td>
<td>8</td>
<td>81</td>
</tr>
<tr>
<td>planning</td>
<td>147</td>
<td>84</td>
<td>79</td>
<td>50</td>
<td>55</td>
<td></td>
<td>146</td>
<td>135</td>
<td>31</td>
<td>57</td>
<td>47</td>
</tr>
<tr>
<td>testing</td>
<td>131</td>
<td>54</td>
<td>77</td>
<td>25</td>
<td>84</td>
<td></td>
<td>36</td>
<td>92</td>
<td>102</td>
<td>57</td>
<td>76</td>
</tr>
<tr>
<td>all</td>
<td>1011</td>
<td>478</td>
<td>543</td>
<td>345</td>
<td>526</td>
<td></td>
<td>494</td>
<td>659</td>
<td>460</td>
<td>370</td>
<td>535</td>
</tr>
</tbody>
</table>

### TABLE II

<table>
<thead>
<tr>
<th>Solver</th>
<th># solved</th>
<th># verified</th>
<th># unique</th>
<th>avg. size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAQE</td>
<td>428</td>
<td>340</td>
<td>146</td>
<td>3138</td>
</tr>
<tr>
<td>DepQBF</td>
<td>312</td>
<td>239</td>
<td>44</td>
<td>7447</td>
</tr>
<tr>
<td>virtual best</td>
<td>468</td>
<td>384</td>
<td>-</td>
<td>5357</td>
</tr>
</tbody>
</table>

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 $\varphi = Q_1 X_1 \ldots Q_n X_n \varphi$ with $n$ quantifier alternations, the algorithm picks an assignment $x_1$ to $X_1$ and recursively determines the truth of $\varphi'(x_1) = Q_2 X_2 \ldots Q_n X_n \varphi(x_1)$. If that call returns a counter-example, that is an assignment $x_2$ to $X_2$, then $\varphi$ is refined by the formula $\varphi'' = Q_1 X_1, Q_2 X_2, \ldots, Q_n X_n, \varphi(x_2)$, which has two quantifier alternations fewer than $\varphi$. Before checking $\varphi'(x_1)$ for other assignments $x_1$, it is first checked whether the assignment is already excluded by $x_1$. (That is, we first check $\varphi''(x'_1)$.) In this way, the assignment $x_1$ cannot occur a second time as a counter-example. A potential problem with this approach is that the formula $\varphi''$ is itself a QBF, and may itself be refined with other QBFs in later iterations. We may therefore have to check each new assignment $x_1$ 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
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