Abstraction-Based Algorithm for 2QBF

Mikoláš Janota[2] and Joao Marques-Silva[1,2]

[1] University College Dublin, Ireland
[2] INESC-ID, Lisbon, Portugal

Abstract. Quantified Boolean Formulas (QBFs) enable standard representation of PSPACE problems. In particular, formulas with two quantifier levels (2QBFs) enable representing problems in the second level of the polynomial hierarchy (Π_2^P, Σ_2^P). This paper proposes an algorithm for solving 2QBF satisfiability by counterexample guided abstraction refinement (CEGAR). This represents an alternative approach to 2QBF satisfiability and, by extension, to solving decision problems in the second level of polynomial hierarchy. In addition, the paper presents a comparison of a prototype implementing the presented algorithm to state of the art QBF solvers, showing that a larger set of instances is solved.

1 Introduction

The *Quantified Boolean Formula* (QBF) decision problem represents the paradigmatic PSPACE-complete decision problem [21]. Restrictions of QBF have also been used to characterize the polynomial hierarchy [21]. The QBF problem is important not only from a theoretical perspective, but also from an applied one, with many applications being easily modeled as instances of QBF [14]. There has been renewed interest in QBF solving over the last decade, in part motivated by the practical success of SAT solvers [23]. Most modern QBF algorithms build on the success of SAT solvers, but implement dedicated techniques [14].

One of the most successful approaches for symbolic model checking is *counter-example-guided abstraction refinement* (CEGAR) [6,7], having been applied in BDD-based and SAT-based model checking [5,7,8]. The success of CEGAR motivated its use with more expressive logics [1,27,12]. Moreover, recent work has applied the CEGAR paradigm in handling quantification on a number of different settings with promising results, including propositional circumscription [20], quantified bit-vector formulas [40], and linear real arithmetic [26]. Although the previous approaches for handling quantification could be used for solving QBF, it is expected that dedicated solutions will result in more effective algorithms.

This paper develops a CEGAR approach for solving QBF with 2 levels of quantifiers. The proposed algorithm generalizes the algorithm from [20] to the 2QBF case. Although the two algorithms exhibit similar abstraction refinement loops, the actual implementation of the key steps of the algorithms differs substantially. These differences are detailed in this paper. Experimental results, obtained on a wide range of problem instances, shows that the new algorithm outperforms the best QBF solvers from the most recent QBF evaluation [28]. Moreover, the new algorithm also outperforms the encoding of 2QBF to propositional circumscription [20], thus confirming that reduction of 2QBF to other domains is unlikely to result in efficient algorithms.

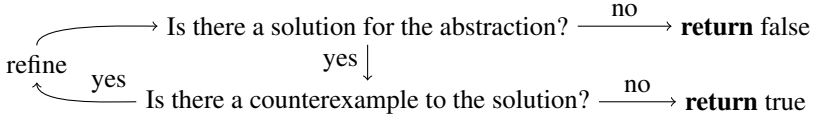

Fig. 1. CEGAR loop

2 Preliminaries

Lowercase letters from the end of the alphabet are used for Boolean variables (x_1, y, etc.); capital letters from the end of the alphabet (X, Y, etc.) are used to denote vectors of variables. Quantified Boolean formulas (QBF) are assumed to be in the *prenex* form $Q_1 z_1 \ldots Q_n z_n . \phi$ where $Q_i \in \{\forall, \exists\}$, z_i are distinct variables, and ϕ is a propositional formula using only the variables z_i and the constants 0 (false), 1 (true). The sequence of quantifiers in a QBF is called the *prefix* and the propositional formula the *matrix*. If a prefix contains a subsequence $\forall x_1 \ldots \forall x_n$, resp. $\exists x_1 \ldots \exists x_n$, we denote it by $\forall X$, resp. $\exists X$, for the variable vector $X = \{x_1, \ldots, x_n\}$.

The Greek letters ν and μ are used to denote vectors of the constants 0 and 1. For a Boolean formula ϕ and vectors $X = \{x_1, \ldots, x_n\}$, $\nu = \{a_1, \ldots, a_n\}$ we write $\phi[X/\nu]$ for the simultaneous substitution of occurrences of x_i by a_i. Further, $\phi[X/\nu]$ assumes that the formula has been partially evaluated ($x \vee 0 \equiv x$, etc.). If X are all the variables in ϕ, we treat the value vector ν as a variable valuation and $\phi[X/\nu]$ is the formula's value under that valuation. We write $\mathcal{B}$ for the set $\{0, 1\}$ and $\mathcal{B}^n$ for the set of vectors of the values 0,1 of length n.

A Boolean formula in *conjunctive normal form (CNF)* is a conjunction of *clauses*, where a clause is a disjunction of *literals*, and a literal is either a variable or a complement. Whenever convenient, a CNF formula is treated as a set of clauses and a clause is treated as a set of literals. For a literal l we write $\mathsf{var}(l)$ for the variable in l, i.e. $\mathsf{var}(\neg x) = \mathsf{var}(x) = x$.

Some of the heuristics proposed in the paper use a *partial MAX-SAT* problem formulation. The partial MAX-SAT problem is specified with two sets of clauses: a set of *hard* clauses, and a set of *soft* clauses. A solution to the problem is a variable valuation that satisfies all the hard clauses and maximizes the number of satisfied soft clauses [22].

2.1 Counterexample Guided Abstraction Refinement (CEGAR)

Counterexample Guided Abstraction Refinement (CEGAR) was designed for tackling problems whose implicit representation is infeasible to solve and thus an abstract representation is tackled instead. Here we present a informal description of the approach necessary for the understanding of the article. For more details see [7].

In CEGAR-based algorithm we talk about *concrete* and *abstract* representation of the problem. Then we talk about *abstract solutions*, which are solutions to the abstract version, and *concrete solutions* which are solutions to the actual problem. The goal of the CEGAR approach is to get to a concrete solution via abstract solutions. The relation between these two representations is characterized by the following properties:

1. If the abstraction does not have a solution, the concrete problem does not have a solution either.
2. If an abstract solution is not a concrete solution, a *counterexample* is produced to demonstrate this fact.
3. If there are no counterexamples to an abstract solution, it is also a concrete solution.

The search for a solution of the concrete problem is carried out in the following loop (see Figure 1). First, a solution for the abstraction is computed. If such does not exist, the search terminates unsuccessfully due to property 1. If the abstraction has a solution, it needs to be checked whether it is also a concrete solution. If there are no counterexamples to the abstract solution, it is also a concrete solution and the search terminates successfully due to property 3. Otherwise, the abstraction needs to be *refined* where the obtained counterexample is used to guide the refinement. Observe that an abstract solution is in fact a *candidate* for a solution to the given problem, the second step of the iteration then checks if it is really a solution.

We should note that conceptually, here we are looking for a solution to the problem. However, often, especially in verification, the goal of the CEGAR loop is to show *unsatisfiability* of the problem, i.e., lack of solutions. Finding a solution then corresponds to finding an error in the modeled system. Algorithmically, these goals are identical but the pertaining terminology in literature may differ.

3 Problems

This article focuses on the satisfiability of formulas with two levels on quantifiers. In particular, we focus on the following two problems.

Name: 2QBF PROBLEM
Given: $\exists X \forall Y.\phi$, where ϕ is a propositional formula
Question: Is there value a vector ν such that $\forall Y.\phi[X/\nu]$?

Name: 2QCNF PROBLEM
Given: $\exists X \forall Y.(\neg\phi')$, where ϕ' is a CNF
Question: Is there value a vector ν such that $\forall Y.\neg\phi'[X/\nu]$?

In both problems, a vector ν satisfying the condition is called a *solution* of $\exists X \forall Y.\phi$. While deciding the satisfiability of a QBF is PSPACE complete, the above problems are Σ_2^P-complete [24,21].

Here we make several notes on the specific form of the problems that we chose for this article. While 2QCNF PROBLEM is a special case of 2QBF PROBLEM, we single out 2QCNF PROBLEM as the uniformity of the format can be exploited for efficiency. The satisfiability of a formula of the form $\forall X \exists Y.\phi$ can be decided by negating to $\exists X \forall Y.\neg\phi$ and negating the response, which is why we consider only the latter form. Conceptually, $\exists X \forall Y.\neg\phi$ can be thought of as an attempt to *refute* $\forall X \exists Y.\phi$.

While for CNF ϕ' the satisfiability of $\forall X \exists Y.\phi'$ is in Π_2^P, $\exists X \forall Y.\phi'$ is an NP problem. Hence, we consider only the satisfiability of $\exists X \forall Y.(\neg\phi')$, which corresponds to refuting $\forall X \exists Y.\phi'$.

4 Algorithm for the 2QBF PROBLEM

This section presents a CEGAR-based algorithm for the 2QBF PROBLEM. The algorithm relies on a SAT oracle (a SAT solver) and hence we begin by observing the relation of 2QBF to Boolean satisfiability. First let us observe that the 2QBF PROBLEM can be expressed as a Boolean satisfiability problem if the universal quantifier is expanded into conjunctions.

4.1 Algorithm

Observation 1 *A value vector ν is a solution to $\exists X \forall Y.\phi$ iff ν is a satisfying assignment of the following formula:*

$$\bigwedge_{\mu \in \mathcal{B}^{|Y|}} \phi[Y/\mu] \tag{1}$$

Consequently, $\exists X \forall Y.\phi$ has a solution iff (1) is satisfiable.

Hence, a naïve approach to solving the 2QBF problem would be to perform the expansion outlined above and invoke a SAT solver. However, this is infeasible since the formula grows exponentially. We continue by observing that the question whether a certain value vector is a solution can be formulated as a Boolean satisfiability question[3].

Observation 2 *A value ν is a solution to $\exists X \forall Y.\phi$ iff the following formula is unsatisfiable:*

$$\neg\phi[X/\nu] \tag{2}$$

Example 1. Expanding the formula $Q = \exists x \forall y. x \to y$ yields $x \to 0 \wedge x \to 1$, which is equivalent to the formula $\neg x$. Hence, according to Observation 1, $\{x = 0\}$ is a solution of the formula Q. In contrast, Observation 2 tells us that the value vector $\{x = 1\}$ is not a solution since $1 \wedge \neg y$ is satisfiable.

The two observations above motivate the following abstraction-based approach. Instead of considering the full expansion of the given problem (see (1)), we consider only a partial expansion, which will serve as the abstraction in the approach.

Definition 1 (W-abstraction). *Let $W \subseteq \mathcal{B}^{|Y|}$, then the* W-abstraction *of $\exists X \forall Y.\phi$ is the following formula.*

$$\bigwedge_{\mu \in W} \phi[Y/\mu] \tag{3}$$

A satisfying valuation of a W-abstraction is not necessarily a solution for the given problem. Recall that a solution ν must satisfy $\forall Y.\phi[X/\nu]$. Hence, if ν is not a solution, then there must exist a valuation μ of the variables Y for which $\phi[X/\nu]$ does not hold, i.e. $\phi[X/\nu][Y/\mu] = 0$. This valuation μ is used as the counterexample in the approach.

[3] This observation is in fact a consequence of the fact that the problem is in Σ^2_{P}.

Algorithm 1: CEGAR loop for 2QBF

input : $\exists X \forall Y.\phi$
output: (true, ν) if there exists ν s.t. $\forall Y \phi[X/\nu]$,
$(\mathsf{false}, -)$ otherwise

```
1 ω ← 1
2 while true do
3     (outc_1, ν) ← SAT(ω)                    // find a candidate solution
4     if outc_1 = false then
5         return (false,–)                    // no candidate found
6     (outc_2, μ) ← SAT(¬φ[X/ν])              // find a counterexample
7     if outc_2 = false then
8         return (true, ν)                    // candidate is a solution
9     ω ← ω ∧ φ[Y/μ]                          // refine
```

Definition 2 (counterexample). *If ν and μ are value vectors and μ is a satisfying valuation of $\neg\phi[X/\nu]$ then μ is called a* counterexample *to ν.*

Proposition 1. *If a W-abstraction is unsatisfiable then the corresponding* 2QBF PROBLEM *has no solutions. If for a value vector ν there are no counterexamples, then ν is a solution to the* 2QBF PROBLEM.

Proof (sketch). For any $W \subseteq \mathcal{B}^{|Y|}$, the W-abstraction ω is weaker than (1), hence if ω is unsatisfiable, the given problem does not have a solution due to Observation 1. There are no counterexamples to ν iff $\neg\phi[X/\nu]$ is unsatisfiable, which is true only if ν is a solution (Observation 2).

Algorithm 1 shows a pseudo-code representation of the 2QBF CEGAR loop using the notion of abstraction and counterexample defined above. The pseudo-code assumes a satisfiability oracle `SAT` which for a Boolean formula returns whether it is satisfiable or not. If it is satisfiable, it also returns a satisfying valuation; this information is returned as a pair with the first element representing satisfiability and the second element the valuation (if applicable).

The algorithm maintains the W-abstraction in the variable ω and it starts with $W = \emptyset$, i.e. $\omega = 1$ (line 1). Each iteration of the loop begins by looking for a solution for the abstraction, this solution is called the *candidate*. If the abstraction is unsatisfiable—there are no candidates—the given problem is unsatisfiable and hence the loop terminates (line 5). If a candidate was found, the algorithm checks whether the candidate is indeed a concrete solution to the given problem or not. If the candidate is a solution, then the loop terminates successfully (line 8). If the candidate is not a concrete solution, the abstraction is refined according to the counterexample. The refinement consists in adding the counterexample μ to the set W, which corresponds to conjoining $\phi[Y/\mu]$ to ω (line 9). Observe that the set W monotonically increases from $\emptyset$ to $\mathcal{B}^{|Y|}$ with one iteration adding one element to it.

Example 2. Let $\phi = (x_1 \vee y_1) \wedge (x_2 \vee y_2)$ then $\neg\phi = (\neg x_1 \wedge \neg y_1) \vee (\neg x_2 \wedge \neg y_2)$ and the following is a possible run of Algorithm 1. Initial $\omega_1 = 1$, yields a candidate $\texttt{SAT}(\omega_1) = (\mathsf{true}, \nu_1)$ with $\nu_1 = \{x_1 = 0, x_2 = 0\}$. In turn we obtain a counterexample $\texttt{SAT}\,(\neg\phi[X/\nu_1]) = (\mathsf{true}, \mu_1)$ with $\mu_1 = \{y_1 = 1, y_2 = 0\}$. The counterexample yields the refinement $\omega_2 = \omega_1 \wedge x_2 = x_2$. The second iteration yields a candidate $\texttt{SAT}(\omega_2) = (\mathsf{true}, \nu_2)$ with $\nu_2 = \{x_1 = 0, x_2 = 1\}$ and a counterexample $\texttt{SAT}\,(\neg\phi[X/\nu_2]) = (\mathsf{true}, \mu_2)$ with $\mu_2 = \{y_1 = 0, y_0 = 1\}$. The corresponding refinement is $\omega_3 = \omega_2 \wedge x_1 = x_2 \wedge x_1$. The candidate in the third iteration is inevitably $\nu_3 = \{x_1 = 1, x_2 = 1\}$, which is a solution as there are no counterexamples to it ($\texttt{SAT}(\neg\phi[X/\nu_3]) = (\mathsf{false}, -)$).

4.2 Properties

Here we discuss the correctness and some other properties of the algorithm. A crucial observation is that no counterexample can appear in two distinct iterations of the CEGAR loop, which is stated by the following lemma.

Lemma 1. *Let μ_i and μ_k be counterexamples found in the i-th and k-th iterations of the loop, respectively, where $i < k$. Then $\mu_i \neq \mu_k$.*

Proof (sketch). For contradiction assume that $\mu = \mu_i = \mu_k$ and let ν be a candidate found in the k-th step. The candidate ν satisfies the current abstraction, which is of the form $\omega' \wedge \phi[Y/\mu]$ since the abstraction was refined with μ in the step i. Hence, μ is a model of $\phi[X/\nu]$, which is a contradiction since it is also a model of $\neg\phi[X/\nu]$.

The above lemma ensures that the CEGAR loop will have a finite number of iterations since there is only a finite number of possible counterexamples. However, the algorithm has even a stronger property. Once a counterexample μ is found, all candidates to which μ is a counterexample are eliminated from the space of possible candidates. This is stated formally in the following lemma.

Lemma 2. *Let μ_i be a counterexample found in the i-th iteration of the loop and ν_k be a candidate found in the k-th iteration of the loop, where $k > i$. Then μ_i is* not *a counterexample to ν_k. In particular, no candidate can appear more than once.*

Proof (sketch). In the i-th iteration of the loop the abstraction has been refined as $\omega = \omega' \wedge \phi[Y/\mu_i]$. Since ν_k must satisfy ω, and therefore also $\phi[Y/\mu_i]$. Consequently, μ_i and ν_k cannot together satisfy $\neg\phi$.

Lemma 1 and Lemma 2 tell us that neither candidates nor counterexamples can repeat in the iteration loop, which yields the following upper bound on the total number of iterations.

Proposition 2. *Let $k = min(|X|, |Y|)$, then Algorithm 1 performs at most 2^k iterations of the loop and requires $O(|\phi| * 2^k)$ space.*

Proof (sketch). Immediate consequence of Lemma 1, Lemma 2, and the fact that there are $2^{|Z|}$ different value assignments to a set of variables Z.

Algorithm 2: CEGAR loop for 2QCNF

input : $\exists X \forall Y.(\neg\phi')$
output: (true, ν) if there exists ν s.t. $\forall Y \neg\phi'[X/\nu]$,
$(\mathsf{false}, -)$ otherwise

```
1  ω ← 1
2  while true do
3      (outc_1, ν) ← SAT(ω)                          // find a candidate solution
4      if outc_1 = false then
5          return (false,–)                              // no candidate found
6      (outc_2, μ) ← SAT(φ'[X/ν])                    // find a counterexample
7      if outc_2 = false then
8          return (true, ν)                          // candidate is a solution
       // refine
9      C ← {c | c' ∈ φ' ∧ c = c'[Y/μ] ∧ c ≠ 1}                    // substitute
10     let z_c be a fresh variable for each c ∈ C
11     ω ← ω ∪ {¬z_c ∨ ¬l | c ∈ C ∧ l ∈ c}
12     ω ← ω ∪ {⋁_{c∈C} z_c}
```

While the theoretical upper bound given by Proposition 2 is rather crude, we can observe that Lemma 2 gives us some further insight. A refinement according to a counterexample μ prevents the algorithm from finding any candidates to which μ is also a counterexample. In other words, the space of possible candidates is diminished more if the counterexample is a counterexample to many possible candidates. This is illustrated by the following example.

Example 3. Let $\Phi = \exists xy \forall q.\, ((x \wedge q) \vee (x \wedge \neg q)) \wedge ((y \wedge q) \vee (y \wedge \neg q))$ and consider the following run of the algorithm. The first candidate $\nu_1 = \{x = 0, y = 0\}$ yielding the counterexample $\mu_1 = \{q = 1\}$. The corresponding refinement is $\omega_2 = x \wedge y$. Inevitably, the second candidate $\nu_1 = \{x = 1, y = 1\}$ is a solution to the problem. Observe that μ_1 is a counterexample to all candidates that are not solutions.

More generally, we hypothesize that the algorithm is likely to work well for problems where *one counterexample is a counterexample to many potential candidates*.

5 Algorithm for the 2QCNF PROBLEM

This section looks in more detail at the CNF formulation of the problem. Recall that in 2QCNF PROBLEM the input formula ϕ is of the form $\neg\phi'$ where ϕ' is in CNF. The structure of the algorithm remains the same but we make several observations that enable more efficient implementation. The pseudo-code is presented by Algorithm 2.

First, observe that $\neg\phi[X/\nu] = \phi'[X/\nu]$ since $\phi = \neg\phi'$. Hence, a search for a counterexample to the candidate ν is simplified to $\mathtt{SAT}(\phi'[X/\nu])$ (instead of $\mathtt{SAT}(\neg\phi[X/\nu])$). Second, the refinement $\omega \leftarrow \omega \wedge \phi[Y/\mu]$ now has the form $\omega \leftarrow \omega \wedge \neg\phi'[Y/\mu]$. To

maintain ω in CNF, we perform a variant of Plaisted-Greenbaum transformation [30][4]. For each clause $c \in \phi'[Y/\mu]$ introduce a fresh variable z_c and add to ω the clauses $\neg z_c \vee \neg l$ for each literal $l \in c$. Finally, add the clause $\bigvee_{c \in \phi'[Y/\mu]} z_c$. Intuitively, a variable z_c represents that the clause c is false and their disjunction represents that at least one of them is false, thus enforcing $\neg\phi'[Y/\mu]$.

The size of the refinement is trimmed by omitting those clauses that are immediately satisfied by the counterexample μ (this happens whenever μ satisfies at least one literal in the clause). Further, at the implementation level, the incremental interface of the SAT solver is used as ω is gradually strengthened; the variables z_c are reused if the clause c appears in multiple iterations of the CEGAR loop.

5.1 Heuristics

The CEGAR loop relies on two calls to a SAT solver and either of these two calls may yield different models for the same abstraction or candidate, respectively. While the correctness of the algorithm is not affected by which of these models is returned, the overall efficiency of the algorithm may be affected. Here we propose heuristics that determine which candidates and counterexamples are better.

Candidate heuristic The objective of the heuristic used in computing a candidate is to find such candidates that there likely to be solutions to the original problem $\exists X \forall Y.(\neg\phi')$. Since a solution must satisfy $\forall Y.\neg\phi'$, we propose a heuristic that maximizes the number of unsatisfied clauses in ϕ'. In particular, the satisfiability problem $\texttt{SAT}(\omega)$ is replaced by the following MAX-SAT problem:

$$\begin{array}{l} \{\textit{hard}\ c \mid c \in \omega\} \\ \{\textit{hard}\ \neg z_c \vee \neg l \mid z_c \text{ is a fresh variable } \wedge\, c \in \phi' \wedge l \in c \wedge \mathsf{var}(l) \notin Y\} \\ \{\textit{soft}\ z_c \mid c \in \phi'\} \end{array} \tag{4}$$

Counterexample heuristic In the refinement step we need to consider only those clauses that are not satisfied by the counterexample μ, i.e. $c \in \phi \wedge c[Y/\mu] \neq 1$. Hence, the clause $\bigvee z_c$, added in line 12, has less literals the more clauses are satisfied by μ. Since, in general, short clauses represent stronger constraints then long clauses, we propose the heuristic to look for those counterexamples that maximize the number of satisfied clauses in ϕ'. In particular, the satisfiability problem $\texttt{SAT}\,(\phi'[X/\nu])$ is replaced by the following MAX-SAT problem:

$$\begin{array}{l} \{\textit{hard}\ c \mid c' \in \phi' \wedge c = c[X/\nu]\} \\ \{\textit{soft}\ c \mid c' \in \phi' \wedge c = \{l \mid (l \in c') \wedge \mathsf{var}(l) \notin X\}\} \end{array} \tag{5}$$

Implementing heuristics In both of the aforementioned heuristics the corresponding SAT problem is transformed into a MAX-SAT problem. Solving these MAX-SAT problems in each iteration of the CEGAR loop is not feasible because typically a large number of iterations is required (up to hundreds of thousands) and MAX-SAT is significantly more time-consuming than SAT. Hence, in the implementation we compute

[4] As opposed to Tseitin transformation [38], implications in only one direction are introduced.

Table 1. Numbers of solved instances

	struqs	QuBE7.1	qbf2circ	AReQS	AReQS-H
2qbf '10 pre (114)	30	93	37	**101**	**101**
circ pre (117)	6	113	117	**117**	**117**
icore pre (140)	30	23	33	**62**	**62**
robots pre (999)	516	921	647	974	**975**
noprepro (232)	15	47	18	51	**55**
total (1602)	597	1197	852	1305	**1310**

an approximate solution to the MAX-SAT problems by skewing the default decision polarity and variable activity of a SAT solver. Hard clauses are given to the SAT solver as standard clauses without any change. Each soft clause c is represented by the clause $r_c \vee c$ where r_c is a fresh variable. The polarity of the variable r_c is set to 0 and the activity increased. This instructs the SAT solver to set r_c to 0 as soon as possible in the search for a satisfying valuation, which then enforces c to be satisfied. While this approach does not guarantee the optimum, it is commonly used in modern MAX-SAT and PB solvers and has been successfully applied to SAT solving with preference [35].

6 Experimental Results

A prototype implementing Algorithm 2 was developed using MiniSat2.2 as the underlying SAT solver [10]. In the following text we refer to the prototype as *AReQS* (**A**bstraction **Re**finement **Q**BF **S**olver). Two versions of AReQS were evaluated: one that does not use any heuristics (denoted *AReQS*) and the second that uses the heuristics described in Section 5.1 (denoted *AReQS-H*).

For comparison, two QBF solvers were chosen: *struqs* [37] and *QuBE7.1* [16], which are the official and unofficial[5] winner, respectively, of the 2QBF track of the 2010 QBF evaluation [31]. Besides comparing to these two solvers, AReQS was compared to our own tool *qbf2circ*. The tool utilizes a transformation from 2QBF to propositional circumscription [11], and invokes a dedicated *propositional circumscription solver* [20]. Since the dedicated solver is based on similar ideas presented in this paper, the purpose of this translation was to investigate whether a dedicated QBF solver pays off.

A variety of benchmarks was chosen for the empirical evaluation. The sources for the benchmarks were: QBF library [32], QBF evaluation [31], and two well-known Σ_2^P and Π_2^P complete problems. From the QBF library [32] we chose the Robots2D benchmarks, from QBF evaluation the set of problems used in 2010 2QBF track. Entailment in propositional circumscription is a well-known Π_2^P problem and instances from product configuration were used [19]; implicates core is the problem of deciding for a given clause c, a constant k, and a CNF ϕ whether there exists a clause $c' \subseteq c$, s.t. $|c'| < k$ and $\phi \rightarrow c'$, the problem is well known to be Σ_2^P-complete [39][6]. Only problems of the

[5] QuBE7 was disqualified because of discrepancies, which are already fixed in QuBE7.1.

[6] The problem is usually presented for an implicant rather than implicate, which is easily convertible to the implicate problem by negating the input formula.

form $\forall\exists$ were considered from the QBF library (this was true for all the problems in the 2QBF track of the QBF-Evaluation); the implicant core problem was directly generated in its negated form (the again producing the $\forall\exists$ form). All experimental results were obtained on an Intel Xeon 5160 3GHz, with 4GB of memory, and running Linux. The experiments were obtained with a 1000 seconds time limit and 2GB memory limit.

Our initial experiments showed that all the tested solvers perform extremely poorly when the input problem is not preprocessed. Hence, the preprocessor *sQueezeBF* [13] (part of QuBE7.1[7]) was first applied on the instances (discarding instances solved completely by the preprocessor). A random subset of the aforementioned problems were chosen for the evaluation without the processing (noprepro). Therefore, all the sets instances except for noprepro consist of instances already simplified by sQueezeBF.

Table 1 shows the number of solved instances for each set of benchmarks and solver. The new tool AReQS solves ca. 10% more instances than QuBE7.1 and more than double the instances solved by struqs. The gains achieved with AReQS are uniformly distributed among the classes of problem instances considered. The tool qbf2circ solves more instances than struqs, but less than QuBE7.1. Figure 2 shows a more detail overview of the runtimes with cactus and scatter plots. Both versions of AReQS consistently outperform all the other approaches; QuBE7.1 comes second; struqs performs slightly worse than qbf2circ, and both perform significantly worse than QuBE7.1.

The first scatter plot compares AReQS-H and QuBE7.1 on all the instances combined. This plot shows that AReQS-H not only solves more instances but the majority is solved faster. The last two scatter plots compare the heuristic approach to the non-heuristic approach. In the first of the to scatter plots the times are compared and in the second the number of iterations of the CEGAR loop. The heuristics yield an overall improvement, both in time and iteration count; in a number of instances the improvement is in orders of magnitude.

The experimental results suggest that CEGAR is a promising approach for developing dedicated algorithms for 2QBF. Although the AReQS tool is still a prototype, it consistently outperforms state of the art QBF solvers on several classes of problem instances. Nevertheless, the importance of preprocessing should be noted, and any approach needs preprocessing for achieving good overall performance.

7 Related Work

QBF is a well-known PSPACE-complete problem (e.g. [21]), with a wide range of practical applications [14]. Restrictions on the number of alternations have been used to characterize the polynomial hierarchy [21]. QBF algorithms have been the subject of significant improvements over the last decade [14,33,29]. Examples of recent work can be found in [28,15].

Counterexample guided abstraction refinement was successfully applied in model checking [6,7] and since then it has appeared in various forms. In satisfiability modulo theories (SMT) solvers, CEGAR has been used to abstract first order theories as propositional theories with the use of a SAT solver and decision procedures [1,27,12]. The refinement in these works consists in blocking the abstract solution just found.

[7] QuBE itself was then run with the no processing option.

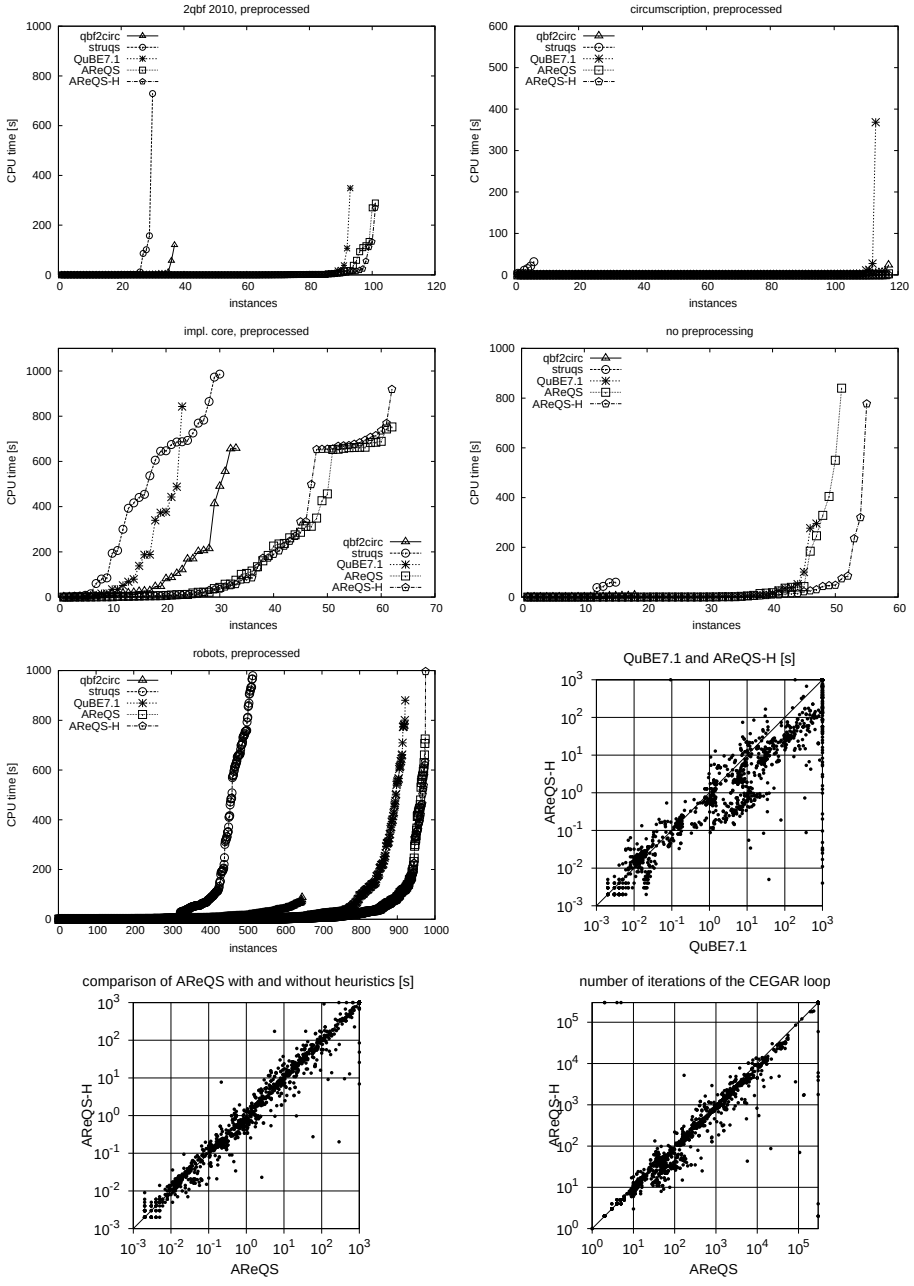

Fig. 2. Overview of the experimental results

Rintanen uses an idea similar to the W-abstraction in a technique called *inversion of quantifiers* in the context of search-based QBF solving [34]. This abstraction is populated randomly (whereas in our case it is determined by counterexamples); the abstraction does not drive the algorithm, instead, it is used as a simplification technique in a search-based algorithm. SAT-based algorithms were used for solving 2QBF formulas in the context of bounded model checking [9] and in planning [4]. However, these algorithms are highly specialized for the problems in question and it is not clear how they could be generalized for arbitrary 2QBF.

A variation of the CEGAR approach used in SMT [1,27,12] was applied to certain special forms of 2QBF. Mneimneh and Sakallah compute the *vertex eccentricity* of a transition system (also known as *the diameter*) [25]. Browning and Remshagen tackle the validity of *Q-ALL SAT* [3]. Besides the fact that the algorithms presented in these articles are specialized to subsets of 2QBF, there is also an important difference in the refinement they use. Once a candidate is found, it is simply blocked so that it is not found again. That means that the set of possible candidates is explored one by one. In contrast, in our approach multiple candidates are removed upon each iteration (see Lemma 2). The one-by-one iteration over candidates not only affects the theoretical upper bound for number of iterations (Proposition 2) but also is likely to lead to an unmanageable number of iterations, especially for unsatisfiable instances where *all* possible candidates need to be considered. Browning and Remshagen address this problem by a heuristic for decreasing the size of the blocking clause. This heuristic is computationally expensive since it requires additional calls to the solver, does not provide any theoretical guarantee, and it is unclear how it could be generalized for arbitrary QBF.

More recently, there has been increased interest in the CEGAR approach in the context of quantification [20,40,26]. Out of these works, our own work on *propositional circumscription entailment* is probably the most similar [20]. Although the algorithm based on propositional circumscription can be used to solve 2QBF, e.g. by using the well-known reduction from [11], the new dedicated 2QBF algorithm is shown to outperform this approach. The dedicated algorithm exploits the problem representation, and this provides a natural performance edge.

The work described in [26] is for quantified linear real arithmetic. Although this work could be used on QBF formulas, the key techniques do not aim Boolean formulas. Finally, the work reported in [40] is solving a computationally harder decision problem, namely quantified bit-vector formulas. This means that [40] can be used to solve arbitrary QBFs, but it is also unlikely to scale as well as a dedicated algorithm.

8 Conclusions

This paper develops a new algorithm for the 2QBF and 2QCNF problems. The algorithm exploits the counterexample-guided abstraction refinement paradigm [6,7], and is shown to outperform the best peforming QBF solvers from the most recent QBF Evaluation [28]. Although the work builds on recent work on using counterexample-guided abstraction for handling quantification [20,40,26], the algorithm exploits the natural properties of the problem formulation, and is shown to outperform approaches based on mapping QBF to another domain [20]. Refining the abstraction in some sense corre-

sponds to traversing the search space with the use of learned clauses [23,17]. However, there are some important differences. Learned clauses can be removed without affecting the correctness of the algorithm, which is not the case for the abstraction refinements. This has the adversary effect that the abstraction algorithm requires exponential space. One the other hand, the CEGAR-based search does not require traversal in any particular order, which enables us to focus on *likely solutions*. This advantage is demonstrated by the heuristics developed for the approach (Section 5.1). Further, we hypothesize that the approach will work well on certain types of problems (Section 4.2).

The promising experimental results motivate extending the work to arbitrary levels of the polynomial hierarchy and to general QBF. Nevertheless, many interesting applications lie in the second level of the polynomial hierarchy and this paper suggests that dedicated algorithm may in general represent the best approach for achieving the most efficient solutions.

Acknowledgement. This work is partially supported by SFI PI grant BEACON (09/IN.1/I2618), EC FP7 project MANCOOSI (214898), FCT grant ATTEST (CMU-PT/ELE/0009/2009), and INESC-ID multiannual PIDDAC program funds.

References

1. Barrett, C.W., Dill, D.L., Stump, A.: Checking satisfiability of first-order formulas by incremental translation to SAT. In: Computer Aided Verification. pp. 236–249 (2002)
2. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
3. Browning, B., Remshagen, A.: A SAT-based solver for Q-ALL SAT. In: Menezes, R. (ed.) ACM Southeast Regional Conference. pp. 30–33. ACM (2006)
4. Castellini, C., Giunchiglia, E., Tacchella, A.: Improvements to SAT-based conformant planning. In: European Conference on Planning (2001)
5. Chauhan, P., Clarke, E.M., Kukula, J.H., Sapra, S., Veith, H., Wang, D.: Automated abstraction refinement for model checking large state spaces using SAT based conflict analysis. In: Aagaard, M., O'Leary, J.W. (eds.) FMCAD. pp. 33–51. Springer (2002)
6. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Computer Aided Verification. pp. 154–169 (2000)
7. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM 50(5), 752–794 (2003)
8. Clarke, E.M., Gupta, A., Strichman, O.: SAT-based counterexample-guided abstraction refinement. IEEE Trans. on CAD of Integrated Circuits and Systems 23(7), 1113–1123 (2004)
9. Dershowitz, N., Hanna, Z., Katz, J.: Bounded model checking with QBF. In: Bacchus, F., Walsh, T. (eds.) SAT. pp. 408–414. Springer (2005)
10. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia and Tacchella [18]
11. Eiter, T., Gottlob, G.: Propositional circumscription and extended closed-world reasoning are Π_2^P-complete. Theor. Comput. Sci. 114(2), 231–245 (1993)
12. Flanagan, C., Joshi, R., Ou, X., Saxe, J.B.: Theorem proving using lazy proof explication. In: Jr., W.A.H., Somenzi, F. (eds.) CAV. vol. 2725, pp. 355–367. Springer (2003)
13. Giunchiglia, E., Marin, P., Narizzano, M.: An effective preprocessor for QBF pre-reasoning. In: 2nd International Workshop on Quantification in Constraint Programming (QiCP) (2008)

14. Giunchiglia, E., Marin, P., Narizzano, M.: Reasoning with quantified boolean formulas. In: Biere et al. [2], pp. 761–780
15. Giunchiglia, E., Marin, P., Narizzano, M.: sQueezeBF: An effective preprocessor for QBFs based on equivalence reasoning. In: Strichman and Szeider [36], pp. 85–98
16. Giunchiglia, E., Narizzano, M., Tacchella, A.: QuBE++: An Efficient QBF Solver. In: Hu, A.J., Martin, A.K. (eds.) FMCAD. pp. 201–213. Springer (2004)
17. Giunchiglia, E., Narizzano, M., Tacchella, A.: Clause/term resolution and learning in the evaluation of quantified boolean formulas. J. Artif. Intell. Res. (JAIR) 26, 371–416 (2006)
18. Giunchiglia, E., Tacchella, A. (eds.): Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003 (2004)
19. Janota, M., Botterweck, G., Grigore, R., Marques-Silva, J.: How to complete an interactive configuration process? In: Proceeding of 36th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM). Springer (2010)
20. Janota, M., Grigore, R., Marques-Silva, J.: Counterexample guided abstraction refinement algorithm for propositional circumscription. In: Proceeding of the 12th European Conference on Logics in Artificial Intelligence (JELIA). pp. 195–207 (Sep 2010)
21. Kleine-Büning, H., Bubeck, U.: Theory of quantified boolean formulas. In: Biere et al. [2]
22. Li, C.M., Manyà, F.: Maxsat, hard and soft constraints. In: Biere et al. [2], pp. 613–631
23. Marques-Silva, J., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Biere et al. [2], pp. 131–153
24. Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential space. In: Symposium Switching and Automata Theory (Oct 1972)
25. Mneimneh, M.N., Sakallah, K.A.: Computing vertex eccentricity in exponentially large graphs: QBF formulation and solution. In: Giunchiglia and Tacchella [18], pp. 411–425
26. Monniaux, D.: Quantifier elimination by lazy model enumeration. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV. vol. 6174, pp. 585–599. Springer (2010)
27. de Moura, L.M., Rueß, H., Sorea, M.: Lazy theorem proving for bounded model checking over infinite domains. In: Voronkov, A. (ed.) CADE. vol. 2392, pp. 438–455. Springer (2002)
28. Peschiera, C., Pulina, L., Tacchella, A., Bubeck, U., Kullmann, O., Lynce, I.: The seventh QBF solvers evaluation (QBFEVAL'10). In: Strichman and Szeider [36], pp. 237–250
29. Plaisted, D.A., Biere, A., Zhu, Y.: A satisfiability procedure for quantified boolean formulae. Discrete Applied Mathematics 130(2), 291–328 (2003)
30. Plaisted, D.A., Greenbaum, S.: A structure-preserving clause form translation. J. Symb. Comput. 2(3), 293–304 (1986)
31. QBF solver evaluation portal, http://www.qbflib.org/index_eval.php
32. The Quantified Boolean Formulas satisfiability library, http://www.qbflib.org/
33. Ranjan, D.P., Tang, D., Malik, S.: A comparative study of 2QBF algorithms. In: SAT (2004)
34. Rintanen, J.: Improvements to the evaluation of quantified Boolean formulae. In: Dean, T. (ed.) IJCAI. pp. 1192–1197. Morgan Kaufmann (1999)
35. Rosa, E.D., Giunchiglia, E., Maratea, M.: Solving satisfiability problems with preferences. Constraints 15(4), 485–515 (2010)
36. Strichman, O., Szeider, S. (eds.): Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings, vol. 6175. Springer (2010)
37. STRUQS: A Structural QBF Solver, www.qbflib.org/DESCRIPTIONS/struqs.pdf
38. Tseitin, G.S.: On the complexity of derivation in propositional calculus. Studies in constructive mathematics and mathematical logic 2(115-125), 10–13 (1968)
39. Umans, C.: The minimum equivalent DNF problem and shortest implicants. J. Comput. Syst. Sci. 63(4), 597–611 (2001)
40. Wintersteiger, C.M., Hamadi, Y., de Moura, L.: Efficiently solving quantified bit-vector formulas. In: Proceedings of Formal Methods in Computer Aided Design FMCAD (Oct 2010)