The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence
Beyond NP: Technical Report WS-16-05

Clauses Versus Gates in CEGAR-Based 2QBF Solving

Valeriy Balabanov,[1,2] **Jie-Hong R. Jiang,**[2]
Alan Mishchenko,[3] **and Christoph Scholl**[4]
[1]CSD, Mentor Graphics, Fremont, USA
[2]GIEE, National Taiwan University, Taipei, Taiwan
[3]EECS, University of California, Berkeley, USA
[4]IIF, University of Freiburg, Freiburg i. Br., Germany

Abstract

2QBF is a special case of general quantified Boolean formulae (QBF). It is limited to just two quantification levels, i.e., to a form $\forall X \exists Y.\phi$. Despite this limitation it applies to a wide range of applications, e.g., to artificial intelligence, graph theorysynthesis, etc.. Recent research showed that CEGAR-based methodsgive a performance boost to QBF solving (e.g, compared to QDPLL). Conjunctive normal form (CNF) is a commonly accepted representation for both SAT and QBF problems; however, it does not reflect the circuit structure that might be present in the problem. Existing attempts of extracting this structure from CNF and using it in 2QBF context do not show advantages over CNF based 2QBF solvers. In this work we introduce a new workflow for 2QBF, containing a new *semantic* circuit extraction algorithm and a CEGAR-based 2QBF solver that uses circuit structure and is improved by a so-called "cofactor sharing" heuristics. We evaluate the proposed methodology on a range of benchmarks and show the practicality of the new approach.

1 Introduction

Satisfiability solving (SAT) recently attracted a lot of attention due to its numerous applications in computer science (Silva and Sakallah 2000). Some problems (e.g., in the domains of artificial intelligence and games), however, are beyond the reach of SAT solving alone but are naturally expressible in terms of quantified Boolean formulas (QBFs) (Remshagen and Truemper 2005). 2QBF is a restriction of general QBF problems to only two quantification levels, i.e., to the form $\forall X \exists Y.\phi$ or $\exists Y \forall X.\phi$, where ϕ is a quantifier-free propositional formula. Despite this restriction many applications can be naturally expressed in 2QBF language (Remshagen and Truemper 2005; Mneimneh and Sakallah 2003; Mishchenko et al. 2015). 2QBF is a an old problem, and many methods were proposed for solving it. Recent research showed benefits of using counter-example guided abstraction refinement (CEGAR) approaches in QBF solving (Janota et al. 2012), and particularly for 2QBF (Janota and Marques-Silva 2015).

Conjunctive normal form (CNF) is a commonly accepted format for propositional satisfiability problems. It is as well extended to QCNF and used to represent QBF problems (QBFLIB 2005). It is not an uncommon case, however, that originally QBF is specified on a circuit, rather on a CNF. E.g., in (Mishchenko et al. 2015) FPGA synthesis benchmarks are formulated on And-Inverter graphs (AIGs), which is an efficient way to represent general Boolean networks. It is known that any Boolean circuit can be transformed into an equisatisfiable CNF formula, by the various *CNFization* procedures, e.g., by Tseitin transform (Tseitin 1970). The same procedure extends to the QBF context. As QCNF became a unifying platform for QBF problems from various domains, most of the time QBF solving engines do not have access to the original circuit structure of the problem. Previously there were attempts at extracting circuit information from CNF formulas, and use it in SAT and QBF solving (Ostrowski et al. 2002; Eén and Biere 2005; Pigorsch and Scholl 2009; Goultiaeva and Bacchus 2013). Most of them use the same "template matching" algorithms for gate definition detection. Extracted circuit information, however, previously did not show benefits over specialized CNF 2QBF solvers (Janota and Marques-Silva 2015).

In this work we introduce three contributions. First, we explicitly show that circuit information is crucial for robust CEGAR-based 2QBF solving (Section 3). Second, we show how the ideas from efficient CNF 2QBF solvers can be extended to the circuit context (Section 4). And last but not least, we propose a *semantic gate extraction* algorithm, and use it to convert 2QBF problems from QCNF to circuit format (Section 5). This step completes our new 2QBF solving workflow. In Section 6 we show that our 2QBF solver, based on all the three contributions, outperforms existing algorithms on benchmarks from QBF competitions as well as on the crafted benchmarks from a synthesis application.

2 Preliminaries

In this work we shall use commonly accepted notations from Boolean algebra and logic. A *Boolean variable* is interpreted over the binary domain $\{0, 1\}$. A *literal* is either a variable or its negation. A *clause* (resp. *cube*) is a disjunction (resp. conjunction) of literals (sometimes we might use set operations on clause/cube literals as well for convenience). A Boolean formula in conjunctive normal form (CNF) is a conjunction of clauses. A Boolean formula in disjunctive normal form (DNF) is a disjunction of cubes. Both CNF and DNF might

Cegar2Qbf

input: a QBF $\Phi = \forall X \exists Y.\phi$
output: True or False
begin
01 `synMan`$[X] := 1$; `verMan`$[X,Y] := \phi$;
02 **while** True
03 α_X := SatSolve(`synMan`);
04 **if** $\alpha_X = \emptyset$ **then return** True;
05 α_Y := SatSolve(`verMan`, α_X);
06 **if** $\alpha_Y = \emptyset$ **then return** False;
07 `negCof` := $\neg\phi|_{\alpha_Y}$;
08 `synMan` := `synMan` $\wedge$ `negCof`;
end

Figure 1: CEGAR algorithm for generic 2QBF solving.

be subject to set operations for convenience. As a notational convention, we may also sometimes omit the conjunction symbol ($\wedge$), denote disjunction ($\vee$) by the symbol "+," and represent negation ($\neg$) by an overline.

For a Boolean formula $\phi(x_1, .., x_i, .., x_n)$, we say its positive (resp. negative) cofactor with respect to variable x_i, denoted $\phi|_{x_i}$ (resp. $\phi|_{\overline{x}_i}$), is the formula $\phi(x_1, .., x_{i-1}, 1, x_{i+1}, .., x_n)$ (resp. $\phi(x_1, .., x_{i-1}, 0, x_{i+1}, .., x_n)$). The cofactor definition also extends to a cube of literals $\alpha = l_1 \wedge .. \wedge l_m$, with the following recursive definition $\phi|_\alpha = (\phi|_{\alpha \setminus l_m})|_{l_m}$, where $\phi|_{l_m}$ is a positive (resp. negative) cofactor with respect to variable $var(l_m)$ if l_m is a positive (resp. negative) literal. We say that formula ϕ is satisfiable if there is an assignment to its variables that evaluates ϕ to true. We say ϕ is unsatisfiable otherwise. We define an *unsatisfiable core* of an unsatisfiable CNF formula ϕ as an arbitrary unsatisfiable subset of clauses of ϕ.

A *two-quantified Boolean formula* (2QBF) Φ over universal variables $X = \{x_1, \ldots, x_m\}$ and existential variables $Y = \{y_1, \ldots, y_n\}$ in *prenex form* is of the form $\Phi = \forall x_1 \cdots x_m \exists y_1 \cdots y_n.\phi$, where the quantification part is called the *prefix*, denoted Φ_{pfx}, and ϕ, a quantifier-free formula in terms of variables X and Y, is called the *matrix*, denoted Φ_{mtx}. We say that 2QBF Φ is false if there exists an assignment α_X to a set X of variables (also referred to as the winning strategy or winning move of the universal player) such that $\Phi_{\text{mtx}}\ |_{\alpha_X}$ (which is a function of Y variables) is unsatisfiable. We say Φ is true otherwise (i.e., a winning strategy for the universal player does not exist). In this work we shall study efficient ways of determining the values of 2QBF formulas represented in prenex form.

3 Prior work on CEGAR-based 2QBF solving

In Figure 1 we outline the generic CEGAR-based algorithm `Cegar2Qbf` for solving an arbitrary 2QBF formula in prenex form, introduced in (Janota et al. 2012).

In line 1 of Figure 1 we initialize two SAT solving managers: *synthesis manager* `synMan` (i.e., trying to guess the winning strategy of the universal player) and *verification manager* `verMan` (i.e., trying to verify if the guess was correct or not). Initially `synMan` contains variables X and `verMan` contains variables X and Y. In line 3 we search for a candidate winning strategy. If all candidates have been blocked, the 2QBF is determined to be true in line 4. In line 5 we search for a counterexample to the candidate winning strategy, which renders ϕ true (i.e., it disproves the candidate winning move α_X). Note that if CNF is used as an underlying data structure for `verMan`, then α_X can be easily passed to the SAT solver via its assumptions interface (which is commonly available in modern SAT solvers, e.g., in MINISAT (Eén and Sörensson 2003)). If no counterexample is found, then we conclude in line 6 that 2QBF is false. Otherwise we compute a cofactor $\phi|_{\alpha_Y}$ in line 7, and block all the knowingly wrong candidates X', such that $\phi|_{\alpha_Y}(X') = 1$, in line 8. Please note that the major difference of this algorithm compared to QDPLL-based algorithms (Giunchiglia, Narizzano, and Tacchella 2006; Lonsing and Biere 2010) is that in QDPLL `negCof` is simply substituted with $\neg\alpha_X$, i.e., with intention to block the failed candidate within `synMan`. The strength of `Cegar2Qbf` is in that besides α_X it potentially blocks several other assignments.

Algorithm `Cegar2Qbf` may be applied to an arbitrary 2QBF in prenex form regardless of the representation of its matrix. Some QBF solvers use And-Inverter graphs (AIGs) as an underlying matrix structure (Pigorsch and Scholl 2009; Mishchenko et al. 2015). On the other hand, as CNF has proven to be an efficient data structure for SAT solving (e.g., due to an efficient representation of learnt information in the form of learnt clauses (Eén and Sörensson 2003)), (Q)CNF is also the most commonly accepted QBF matrix representation format (QBFLIB 2005). We therefore distinguish between *CNF* and *circuit* 2QBF solvers, depending on the matrix input format that they accept.

Up to the authors knowledge there is no much research done on the circuit CEGAR-based 2QBF solving besides that in (Mishchenko et al. 2015). On the other hand CNF CEGAR-based 2QBF solving was studied in (Janota et al. 2012) and (Janota and Marques-Silva 2015). Below we explain the efficient implementation idea of CNF CEGAR-based 2QBF solver, introduced in (Janota and Marques-Silva 2015) (we shall call it QESTO, by the name of the tool).

Please note that `verMan` is initialized to ϕ itself in line 1 of Figure 1 and is never changed, while `synMan` is constantly changed by conjunctions with `negCof` in line 8. If the matrix is already represented in CNF, then the main complication of the algorithm is the CNFization of `negCof` prior to conjunction with `synMan`. The approach of (Janota et al. 2012) suggested to use a Tseitin transform, i.e., to perform a syntactic negation at the cost of introducing fresh variables, independently for each cofactor. This approach, however, suffers from variable blow up within `synMan` after a large number of iterations. On the other hand the QESTO approach shall allow us to efficiently represent larger numbers of cofactors within `synMan` without suffering from the variables blow up. The idea is as follows: Consider a matrix

$\phi = C_1 \wedge C_2 \wedge .. \wedge C_n$, where each C_i is split into existential literals C_{ei} and universal literals C_{ui}. Notice that regardless of the specifics of the assignment α_Y, `negCof` always takes the form $\neg C_{uj_1} \vee \neg C_{uj_2} \vee .. \vee \neg C_{uj_k}$. This observation suggests to introduce definitions $d_i \equiv \neg C_{ui}$ for each $i \in [1..n]$, which allows to represent `negCof` now conveniently as a clause $(d_{j_1} + d_{j_2} + .. + d_{j_k})$. This means that we introduce at most n fresh variables, independently from the number of iterations of the while loop (line 2 in Figure 1). Under this scenario, the algorithm in Figure 1 is modified to initialize the synthesis manager by

$$\texttt{synMan}[X, D] := \bigwedge_{i \in [1..n]} (d_i \equiv \neg C_{ui}).^1$$

The QESTO approach was experimentally shown to be superior to others (Janota and Marques-Silva 2015). In the next section we will examine the advantages and disadvantages of CNF CEGAR-based 2QBF solving, and introduce a heuristics, inspired by QESTO, to improve existing circuit CEGAR-based 2QBF algorithms.

4 Circuit versus CNF 2QBF solving

Given a non CNF formula ϕ (e.g., represented as an AIG) one could *CNFize* it (i.e., transform it to an equisatisfiable CNF form, for example by using Tseitin transform (Tseitin 1970)) to get ϕ_{CNF}, and then run the QESTO algorithm introduced in Section 3. We, however, speculate that this is an inefficient approach. The following example shows that solving a 2QBF formula with its original matrix ϕ could be more effective compared to CNF-based solving.

Example 1. *Consider the following simple true 2QBF formula $\Phi = \forall x \exists ab.\phi$, where ϕ is given as a circuit*

$$(g_1 = \mathit{AND}(x, a))\ (g_2 = \mathit{AND}(\overline{x}, b))\ (f = \mathit{OR}(g_1, g_2)),$$

with a single output f and two internal gates g_1 and g_2. Assume now that the synthesis solver in algorithm `Cegar2Qbf` depicted in Figure 1 comes up with a candidate $\alpha_X = \overline{x}$ (i.e., assigns $x = 0$), and the verification solver returns a counterexample $\alpha_Y = ab$ (i.e., assigns $a = 1$ and $b = 1$). Note that in this case `negCof` $= 0$, i.e., it blocks all the universal candidate assignments, and we immediately conclude that Φ is true.

Now let us look at the same 2QBF, but CNFized by Tseitin transform prior to solving, i.e., at

$$\begin{aligned} \Phi' &= \forall x \exists ab g_1 g_2.\phi_{CNF}, \text{ where} \\ \phi_{CNF} &= (g_1 + \overline{x} + \overline{a})(\overline{g}_1 + x)(\overline{g}_1 + a) \\ &\wedge \ (g_2 + x + \overline{b})(\overline{g}_2 + \overline{x})(\overline{g}_2 + b)(g_1 + g_2). \end{aligned}$$

Similarly, assume that the synthesis solver in algorithm `Cegar2Qbf` guesses candidate $\alpha_X = \overline{x}$. The verification solver now replies with a counterexample $\alpha_Y = ab\overline{g}_1 g_2$. `negCof` in this case is computed to be `negCof` $= x$, i.e., it only blocks the candidate $x = 0$, and one more iteration is needed to conclude that Φ' is true.

The intuition behind Example 1 is as follows. Whenever the counterexample (line 5 in Figure 1) is computed on CNF level, it fixes a value assignment to the auxiliary variables (i.e., intermediate variables introduced during CNFization) as well. The negated cofactor (line 7 in Figure 1) in this case shall only block X assignments which respect those values. This phenomenon is summarized in Proposition 1.

Proposition 1. *Given a 2QBF $\Phi = \forall X \exists Y.\phi$, with ϕ represented as a circuit, and an assignment α_Y to Y variables. Further given two assignments α_{X_1} and α_{X_2} to X variables, such that*

$$\phi|_{\alpha_Y \alpha_{X_1}} \ \wedge \ \phi|_{\alpha_Y \alpha_{X_2}} \ \wedge \ \exists G \in \phi : G|_{\alpha_Y \alpha_{X_1}} \oplus G|_{\alpha_Y \alpha_{X_2}}$$

(i.e., the output of ϕ evaluates to the same value under two input assignments, but there is an internal gate disagreement). Then α_{X_1} and α_{X_2} shall be blocked within the same *iteration computing counterexample α_Y in line 5 of Figure 1, if algorithm `Cegar2Qbf` is applied to Φ. On the other hand α_{X_1} and α_{X_2}* shall not *be blocked within the same iteration, if `Cegar2Qbf` is applied to a (by Tseitin transform) CNFized version of Φ.*

Example 1 and Proposition 1 show that flattening the circuit structure into CNF affects the 2QBF solving process more than it does for propositional SAT. SAT solvers use CNF structure for efficiency reasons, e.g., for clause learning. In CEGAR based QBF solvers, however, the situation is different. As experiments shall confirm later, one may efficiently use CNF for underlying SAT queries, but cofactoring on circuit level instead of CNF decreases the number of iterations needed for completion of algorithm `Cegar2Qbf` a lot. We shall see that this decrease in the number of iterations even overcomes the benefits of efficient cofactor representation in QESTO algorithm applied after circuit CNFization.

Please note that, even in circuit 2QBF solvers, the SAT queries in line 3 of algorithm Cegar2Qqbf (Figure 1) are made to a CNF-based SAT solver. This choice is caused by a specific "incremental" nature of the underlying SAT calls: Please recall that synthesis manager `synMan` is updated by iterative conjunction with `negCof` in line 8 of Figure 1, i.e., in each iteration clauses from the CNFized `negCof` are added to manager `synMan`. Following the ideas of *structural hashing* of nodes in And-Inverter graphs (AIGs) we propose algorithm `AigShare2Qbf` depicted in Figure 2 for circuit 2QBF solving.

The core CEGAR procedure of algorithm `AigShare2Qbf` is the same as that of `Cegar2Qbf`. Furthermore `Cegar2Qbf` may use AIGs as an underlying structure of its verification manager and negated cofactors as well. The only difference is the *cofactor hashing* heuristics (lines 8 and 9 in Figure 2). More specifically in line 8 we extract a subset `newAnd` of AND gates from `negCof` that have not been hashed previously. Then in line 9 we hash them and add them to the AIG manager `aigMan`. In line 10 we add CNFized `newAnd` gates to

[1] $d_i \equiv \neg C_{ui}$ can be viewed as a conjunction of two implications. Since (apart from this definition) variables d_i only occur as positive literals in the formula `synMan`, $d_i \equiv \neg C_{ui}$ can be replaced by one of the two implications, namely by $d_i \Rightarrow \neg C_{ui}$. This modification corresponds to using the Plaisted/Greenbaum encoding (Plaisted and Greenbaum 1986) instead of Tseitin transformation (Tseitin 1970) and is also used in (Janota and Marques-Silva 2015).

AigShare2Qbf

input: a QBF $\Phi = \forall X \exists Y.\phi$
output: True or False
begin
01 `synMan`$[X]$:=1; `verMan`$[X,Y]$:=ϕ; `aigMan`:=$\emptyset$;
02 **while** True
03 α_X := SatSolve(`synMan`);
04 **if** $\alpha_X = \emptyset$ **then return** True;
05 α_Y := SatSolve(`verMan`, α_X);
06 **if** $\alpha_Y = \emptyset$ **then return** False;
07 `negCof` := AIG($\neg\phi|_{\alpha_Y}$);
08 `newAnd` := NotHashed(`negCof`,`aigMan`);
09 Hash(`aigMan`, `newAnd`);
10 `synMan` := `synMan` $\wedge$ CNF(`newAnd`);
end

Figure 2: CEGAR algorithm for AIG based 2QBF solving with cofactor sharing heuristics.

`synMan`. For AND gates in line 8 which have been hashed previously, `synMan` already contains clauses for the corresponding definitions. As shall be seen from experiments, cofactor sharing heuristic gives a significant improvement on the benchmarks where a lot of iterations are needed for algorithm `Cegar2Qbf` to converge.

5 Semantic gate extraction and circuit reconstruction

Given the extra-strength of circuit-based 2QBF solvers, in this section we introduce a procedure (referred to as *de-CNFization*) of converting a plain CNF formula into an equisatisfiable single-output circuit. *De-CNFization* was already partially used in SAT and QBF solving, by extracting gate definitions from CNF (Ostrowski et al. 2002; Eén and Biere 2005; Pigorsch and Scholl 2009; Goultiaeva and Bacchus 2013). Earlier approaches relied on a syntactic extraction of gate definitions, i.e., they looked for subsets of a CNF corresponding to gate definitions in a certain form. This approach has two basic disadvantages: First, the approach is only able to detect gate definitions for pre-defined gate types (typically "AND-like" and "XOR-like" gates), other gate definitions can not be found. Secondly, the CNF for the definition of a non-trivial gate does not need to be unique. In such a case, syntactic approaches looking for certain patterns will have difficulties. In this section we propose an efficient gate extraction algorithm relying on semantic rather than syntactic gate detection. Semantic gate extraction was also indirectly studied in (Grégoire et al. 2004; Lang and Marquis 2008; Lagniez and Marquis 2014), but never applied in the QBF context. Independent performance comparison of our de-CNFization approach to existing ones is out of the scope of this paper, and is left for future work.

An overview of our de-CNFization algorithm is given in Figure 3. We use standard "template-matching" ways to find AND-like gates (line 1 in Figure 3, function *FindAnd*) and

CnfToCircuit

input: a CNF $\phi_{CNF} = \{C_1, \ldots, C_n\}$
output: circuit ϕ
begin
01 `andGates` := FindAnd(ϕ_{CNF});
02 `restCls` := $\phi_{CNF} \setminus$ Cls(`andGates`) ;
03 `xorGates` := FindXor(`restCls`);
04 `restCls` := `restCls` $\setminus$ Cls(`xorGates`) ;
05 `semGates` := FindSem(`restCls`);
06 `restCls` := `restCls` $\setminus$ Cls(`semGates`) ;
07 `allGates` := `andGates` $\cup$ `xorGates` $\cup$ $\cup$ `semGates` ;
08 `PIs` := UndefinedVars(`allGates`);
09 **foreach** C_i in `restCls`
10 `allGates` := `allGates` $\cup$ $(g_i = C_i)$;
11 `allGates` := `allGates` $\cup$ $(\texttt{PO} = \bigwedge(g_i))$;
12 ϕ := BuildCircuit(`PIs`,`PO`,`allGates`);
end

Figure 3: De-CNFization procedure.

XOR-like gates (line 3 in Figure 3, function *FindXor*). After that we make use of the more general "semantic-based" definition detection (line 5 in Figure 3, function *FindSem*).

To see how the `CnfToCircuit` algorithm works consider the following CNF:

Example 2.

$$\begin{aligned}\phi_{CNF} &= (\overline{x}+\overline{a}+c)(x+\overline{c})(a+\overline{c})\\ &\wedge\ (b+d+x)(b+\overline{d}+\overline{x})(\overline{b}+d+\overline{y})(\overline{b}+\overline{d}+y)\\ &\wedge\ (\overline{f}+c+d)(f+\overline{c})(f+\overline{d})f\end{aligned}$$

Note that it encodes the satisfiability problem of the combinational circuit

$$\phi = [c = \mathit{AND}(a,x)] \wedge [d = \mathit{ITE}(b,y,\overline{x})] \wedge [f = \mathit{OR}(c,d)],$$

where a, b, x, y are the primary inputs of the circuit, c and d are internal gates, and f is its single primary output. Therefore the satisfiability of ϕ_{CNF} could be determined by the satisfiability of ϕ.

Note that *FindAnd* routine of algorithm `CnfToCircuit` shall detect gate definitions for variables c and f, but will fail to detect a gate definition for the variable d. The definition for variable d will be found using semantic extraction routine *FindSem*, as is explained below.

Intuition suggests the following informal condition for the presence of a gate definition: The *gate* definition for variable x is encoded in CNF, if under any assignment to other variables a *unique* assignment to x satisfies the formula. If we relax the *unique* condition to *at most one*, then we say that x is a *dependent* variable (the difference shall be seen later). In both cases x could be re-expressed as a function of other variables, without changing the satisfiability of the formula.

We first start with a general method for semantic gate extraction which is then followed by an efficient heuristics.

Let us consider a CNF ϕ_{CNF} which is partitioned into two disjoint sets ϕ_{def} and ϕ_{rem}. The subset ϕ_{def} contributes a gate definition for a variable x, if $\phi_{def}|_{\overline{x}} \oplus \phi_{def}|_x$ is valid (or equivalently if $\phi_{def}|_{\overline{x}} \equiv \phi_{def}|_x$ is unsatisfiable). Then $x = \overline{\phi_{def}}|_{\overline{x}}$ (or equivalently $x = \phi_{def}|_x$) is a suitable definition for variable x as the following theorem states.

Theorem 1. *Consider a CNF $\phi_1 = \phi_{def} \wedge \phi_{rem}$ and the (non CNF) formula $\phi_2 = (x \equiv \overline{\phi_{def}}|_{\overline{x}}) \wedge \phi_{rem}$, Then ϕ_1 and ϕ_2 are equivalent, if $\phi_{def}|_{\overline{x}} \equiv \phi_{def}|_x$ is unsatisfiable.*

Proof. It is easy to see that using the precondition of the theorem, ϕ_1 can be rewritten into ϕ_2. It holds

$$\begin{aligned}
\phi_1 &= \phi_{def} \wedge \phi_{rem} \\
&= (x + \phi_{def}|_{\overline{x}}) \wedge (\overline{x} + \phi_{def}|_x) \wedge \phi_{rem} \\
&= (x + \phi_{def}|_{\overline{x}}) \wedge (\overline{x} + \overline{\phi_{def}}|_{\overline{x}}) \wedge \phi_{rem} \\
&\quad \text{(since } \phi_{def}|_x \equiv \overline{\phi_{def}}|_{\overline{x}} \text{ due to precondition)} \\
&= (x \equiv \overline{\phi_{def}}|_{\overline{x}}) \wedge \phi_{rem} \;=\; \phi_2.
\end{aligned}$$

□

The semantic gate detection according to Theorem 1 has the disadvantage that the partition of the CNF ϕ_1 into ϕ_{def} and ϕ_{rem} has to be guessed. This problem can be solved in principle by introducing a set D of new de-activation variables d_i into the problem. From $\phi_1(Y)$ we compute a CNF $\phi_1'(D, Y)$ by replacing each clause C_i in ϕ_1 by $C_i + d_i$, i.e., the clause may be de-activated by setting $d_i = 1$. Now each assignment to the de-activation variables in some satisfying solution to the 2QBF $\exists D \forall Y \phi_1'|_{\overline{x}} \oplus \phi_1'|_x$ defines an appropriate subset ϕ_{def} of ϕ_1 according to Theorem 1.

To improve the performance of the introduced approach we propose to use two heuristics. First, we limit the search for the gate definitions of variable x only to clauses that contain x or $\overline{x}$. Note that for the commonly used Tseitin transform we can guarantee that all the definition clauses for variable x shall indeed contain either x or $\overline{x}$, therefore by using our heuristics we will not overlook any gate. On the other hand, in general it is possible that under this restriction we may miss some gate definitions.

As a further step, we relax our search from the "gate definitions" to the "dependent variables". To explain the approach we consider a general CNF

$\phi_1 = (A_1 + x) \dots (A_m + x)(B_1 + \overline{x}) \dots (B_n + \overline{x}) C_1 \dots C_k$,

where A_i and B_j ($i \in [1 \dots m]$ and $j \in [1 \dots n]$) are subclauses, and C_i ($i \in [1 \dots k]$) are clauses. Now, instead of the unsatisfiability check for the formula in Theorem 1, we propose to detect the unsatisfiability of the *CNF* $A_1 \dots A_m B_1 \dots B_n$. The consequences of this are summarized by Theorem 2. Please note that the notations introduced in this paragraph are re-used in Theorem 2.

Theorem 2. *Assume that $A_1 \dots A_m B_1 \dots B_n$ is unsatisfiable, and we are given its unsatisfiable core (not necessarily minimal) to be $A_{i_1} \dots A_{i_p} B_{j_1} \dots B_{j_q}$. Further let*

$$\begin{aligned}
\phi_{def} &= (A_{i_1} + x) \dots (A_{i_p} + x)(B_{j_1} + \overline{x}) \dots (B_{j_q} + \overline{x}) \\
\phi_{rem} &= C_1 \dots C_k \wedge (B_1 + \overline{x}) \dots (B_n + \overline{x}) \wedge \\
&\wedge \bigwedge\nolimits_{t \in [1 \dots m] \setminus \{i_1, \dots, i_p\}} (A_t + x) \\
\phi_2 &= (x \equiv \overline{\phi_{def}}|_{\overline{x}}) \wedge \phi_{rem}.
\end{aligned}$$

Then we say that $x = \overline{\phi_{def}}|_{\overline{x}}$ is a pseudo *definition for x, and the formulas ϕ_1 and ϕ_2 are equivalent.*

Note that in Theorem 2 ϕ_{def} and ϕ_{rem} are not disjoint, but both contain the clauses $(B_{j_1} + \overline{x}) \dots (B_{j_q} + \overline{x})$. The assumption of the theorem requires that $\phi_{def}|_{\overline{x}} \wedge \phi_{def}|_x$ is unsatisfiable instead of $\phi_{def}|_{\overline{x}} \equiv \phi_{def}|_x$ as in Theorem 1.

Proof. First observe that

$$(x \equiv \overline{\phi_{def}}|_{\overline{x}}) \leftrightarrow ((A_{i_1} + x) \dots (A_{i_p} + x)(\overline{x} \vee (\overline{A}_{i_1} \dots \overline{A}_{i_p})))$$

Therefore $\phi_2 = \phi_1 \wedge (\overline{x} \vee (\overline{A}_{i_1} \dots \overline{A}_{i_p}))$. Clearly, under any assignment to the variables $\phi_2 \rightarrow \phi_1$. Now assume that under a given assignment to its variables ϕ_1 is true. We prove that $(\overline{x} \vee (\overline{A}_{i_1} \dots \overline{A}_{i_p}))$ must be true by splitting on the assignment to x:

- If $x = 0$ then clearly $(\overline{x} \vee (\overline{A}_{i_1} \dots \overline{A}_{i_p}))$ is true.
- If $x = 1$ then $B_{j_1} \wedge \dots \wedge B_{j_q}$ must be true (in order to satisfy ϕ_1). Taking in account unsatisfiability of $A_{i_1} \dots A_{i_p} B_{j_1} \dots B_{j_q}$ we conclude that $A_{i_1} \wedge \dots \wedge A_{i_p}$ is false, and therefore $(\overline{x} \vee (\overline{A}_{i_1} \dots \overline{A}_{i_p}))$ is again true.

Now since $(\overline{x} \vee (\overline{A}_{i_1} \dots \overline{A}_{i_p}))$ is true and ϕ_1 is true, ϕ_2 must be true as well. □

Intuitively gate definitions and pseudo definitions are very similar. Both could be added into the structure of the de-CNFized circuit. The only difference is that after detection of a gate we remove all its defining clauses from the CNF, while for pseudo gates we only remove those defining clauses *containing positive literal x*. Please also note that if a CNF was obtained from a circuit through the Tseitin transformation, then there is a big chance that $B_{j_1} \wedge \dots \wedge B_{j_q} = \overline{A}_{i_1} \vee \dots \vee \overline{A}_{i_p}$, given that $A_{i_1} \dots A_{i_p} B_{j_1} \dots B_{j_q}$ is a *minimal* unsatisfiable core (MUS). In this special case pseudo definition becomes a gate definition, since $\phi_{def}|_{\overline{x}} \equiv \phi_{def}|_x$ is unsatisfiable, and the clauses $(B_{j_1} + \overline{x}) \dots (B_{j_q} + \overline{x})$ can be removed from ϕ_{rem}. For instance, this situation occurs for the definition of d in Example 2 as is shown below. After simplifying ϕ_{CNF} in Example 2 by unit propagation on f, and collecting all the clauses containing d, we obtain

$$\phi_d = (b + d + x)(b + \overline{d} + \overline{x})(\overline{b} + d + \overline{y})(\overline{b} + \overline{d} + y)(c + d).$$

Further, $\phi_d|_d \wedge \phi_d|_{\overline{d}} = (b + x)(b + \overline{x})(\overline{b} + \overline{y})(\overline{b} + y)(c)$ is unsatisfiable, with a minimal core $(b+x)(b+\overline{x})(\overline{b}+\overline{y})(\overline{b}+y)$. Following Theorem 2 we assign

$$\phi_{def} = (b + d + x)(b + \overline{d} + \overline{x})(\overline{b} + d + \overline{y})(\overline{b} + \overline{d} + y),$$

and notice that $\overline{\phi_{def}}|_{\overline{d}} \equiv \phi_{def}|_d$, therefore concluding that $d = (b + \overline{x})(\overline{b} + y)$ is a gate definition. Consequently, `CnfToCircuit`(ϕ_{CNF}) returns precisely the original circuit ϕ.

Note that the extraction of gate definitions as sketched in this section is not yet tailored towards QBF solving. If gate detection for a variable x is used for QBF solving, we need the additional restriction that x is an existential variable and does not depend on universal variables that follow x in the quantifier prefix (Pigorsch and Scholl 2009).

The last thing to mention is function *BuildCircuit* in line 12 of the algorithm in Figure 3. This function builds the final circuit from all the discovered gates. Careful cycle breaking is applied in the places where circular dependencies between variables occur (e.g., $a = b$ and $b = a$). It is also more desirable to use smaller unsatisfiable cores while searching for semantic definitions in order to reduce the support sets of the defined variables, which might decrease the chance of creating a circular dependency. Moreover, by extracting smaller (and more) definitions, we detect more structure in the CNF compared to extracting large definitions at once.

6 Experimental Results

We implemented the de-CNFization algorithm from Section 5 (with both heuristics enabled) into a tool CNF2BLIF (on top of the MINISAT SAT solver (Eén and Sörensson 2003)), and patched the 2QBF solver embedded in the synthesis tool ABC (Berkeley Logic Synthesis and Verification Group) to support cofactor sharing heuristics, proposed in Section 4[2]. We also used CEGAR QBF solvers RAREQS (Janota et al. 2012) and QESTO (Janota and Marques-Silva 2015) for comparison. All the experiments were performed on a Linux machine with Xeon 2.3 GHz CPU and 32Gb of RAM. All the tools were limited by 4Gb memory limit and 1200 seconds time limit.

For experiments we used two sets of benchmarks: 2QBF track of QBFEVAL'10 (QBFEVAL 2010) and FPGA mapping benchmarks from (Mishchenko et al. 2015).

2QBF track of QBFEVAL'10 (QBFEVAL 2010).

We used 200 CNF based QBFs in prenex form from 2QBF track of QBFEVAL'10 QBF solving competitions. Preprocessor BLOQQER was used to trim 135 benchmarks which can be solved by pure preprocessing. Both RAREQS and QESTO ran on the preprocessed benchmarks. On the other hand we used CNF2BLIF to extract circuits from unpreprocessed CNF formulas, and then ran 2QBF solver embedded into ABC under two settings: without cofactor sharing (further referred to as ABC-) and with cofactor sharing (further referred to as ABC+). A third version, called PREABC+ is similar to ABC+, with the only difference that the QBF preprocessor BLOQQER is used for preprocessing QBFs where CNF2BLIF did not find any circuit structure. *After preprocessing* the resulting CNFs are then just translated to products of sums circuits and then solved by ABC+. (In our experiments, CNF2BLIF did not find any circuit structure in 17 out of 65 benchmarks (*sortnetsort* family).)

Results of this part of experiments are shown in Table 1. A cactus plot of time performance is shown in Figure 4. Extraction time of CNF2BLIF was negligible compared to solving therefore we omitted it.

From Table 1 and Figure 4 we see that the circuit based 2QBF solver PREABC+ well outperforms existing CNF based solvers. More specific details are discussed below. On

[2]CNF2BLIF, ABC, and other tools can be found here: https://www.dropbox.com/s/c29kj1a3w2kv0d0/tools_benchmarks_results_aaai16.zip?dl=0

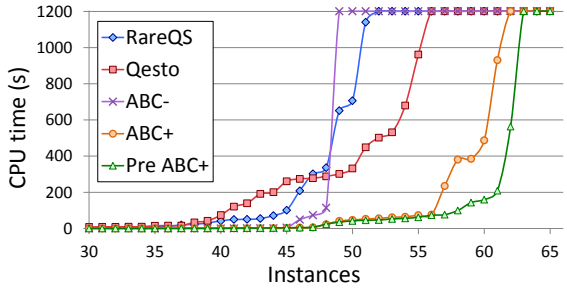

Figure 4: Cactus plot of solvers performance on 2QBF benchmark set from QBFEVAL'10.

Table 1: Statistics for 2QBF track from QBFEVAL'10.

	RAREQS	QESTO	ABC-	ABC+	PREABC+
Solved	50	55	48	61	62
Time, s	20658	17842	20677	7748	4124
Iterations	NA	7.26M	46.0K	368K	153K

17 benchmarks where no circuit structure was found PREABC+ was on average 3 times slower compared to QESTO. The remaining 48 benchmarks were found highly structural. On these ABC+ was on average 28 times faster than QESTO. ABC+ in comparison to ABC- was on average 30% faster, however if we only consider problems solved within more than 100 iterations (or alternatively solved in about more than 1 second) cofactor sharing gives about an order of magnitude speed up. This phenomenon is well explained by the fact that within a few first iterations cofactor sharing occurs rarely, while on the larger scale AIG nodes from the new cofactors are found to be previously hashed practically all the time.

FPGA mapping benchmarks from (Mishchenko et al. 2015).

In this part of experiments we used 100 (50 SAT and 50 UNSAT) 2QBF benchmarks from an FPGA mapping application (Mishchenko et al. 2015). This experiment was designed to test the reverse engineering ability of our tool CNF2BLIF in reconstructing the circuits from CNF. Original AIG circuits have 54 and 66 primary inputs, for UNSAT and SAT cases respectively, of which precisely 6 are existentially quantified (i.e., the remaining variables are outermost universal variables). Each circuit consists of about 250 AIG nodes. Circuits were transformed into QCNFs using ABC's internal engine. The resulting QCNFs on average have 93 variables and 240 clauses.

After applying CNF2BLIF only 2 extra (existentially quantified) primary inputs were introduced to the reconstructed circuits. This means that reconstruction was not identical with respect to the original circuit, but very close. This is due to the fact that our algorithm was not able to properly choose the right definition in several cases where 2 or more definitions were available (which is possible, e.g., in presence of XOR gates). Statistically, reconstructed AIG

Table 2: Statistics for FPGA mapping benchmark set.

	RAREQS	QESTO	ABC	ABC ORIGINAL
Solved	22	44	100	100
Time, s	96.9K	75.0K	100.2	64.3
Iterations	NA	6.46M	1403	1241

circuits have 56 and 68 primary inputs, for UNSAT and SAT cases respectively, and about 300 AIG nodes. Most of the found gates were found semantically (in contrast to QBFEVAL'10, where most of the found gates were either AND-like or XOR-like). Most of those gates are ITE-like gates which are hard to find by template matching. Our approach on the other hand had no difficulties in retrieving those gate definitions.

For the solving part we used ABC to solve original problems (denoted ABC ORIGINAL) and reconstructed circuits (denoted ABC).

The BLOQQER preprocessor was found to degrade the performance of CNF QBF solvers significantly on this benchmark set (due to its eager variable elimination and expansion settings), therefore we do not include it in this set of experiments. Solving statistics are shown in Table 2. As one can observe from Table 2, after reverse engineering the circuit solving time increased slightly, but did not suffer much. In contrast, CNF QBF solvers required several orders of magnitude more iterations, and significantly larger solving time.

Both experiments, and specifically the cumulated number of iterations shown in Tables 1 and 2 confirm our main conjecture: in CEGAR-based 2QBF solving CNF representation is good for carrying out SAT queries, but cofactoring should be done on circuit level instead.

7 Conclusions

In this work we presented several improvements to the state of the art algorithms for 2QBF solving. Experiments showed that the proposed methods outperform existing 2QBF solvers and therefore increase the value of QBF as a framework for various problems in theoretical computer science.

8 Acknowledgements

- Valeriy Balabanov in part was supported by IIS, Academia Sinica, Taipei, Taiwan, and in part by EECS, University of California, Berkeley, USA.
- Special thanks to Alexander Ivrii for valuable discussions and suggestions.

References

Berkeley Logic Synthesis and Verification Group. ABC: A system for sequential synthesis and verification. http://www.eecs.berkeley.edu/$\sim$alanmi/abc/.

Eén, N., and Biere, A. 2005. Effective Preprocessing in SAT Through Variable and Clause Elimination. In *Proc. of the 8th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2005)*, 61–75.

Eén, N., and Sörensson, N. 2003. An extensible SAT-solver. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 2919, 502–518. Springer.

Giunchiglia, E.; Narizzano, M.; and Tacchella, A. 2006. Clause/term resolution and learning in the evaluation of quantified Boolean formulas. *Journal on Artificial Intelligence Research (JAIR)* 26:371–416.

Goultiaeva, A., and Bacchus, F. 2013. Recovering and utilizing partial duality in QBF. In *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, 83–99.

Grégoire, É.; Ostrowski, R.; Mazure, B.; and Sais, L. 2004. Automatic extraction of functional dependencies. In *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*.

Janota, M., and Marques-Silva, J. 2015. Solving QBF by clause selection. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 325–331.

Janota, M.; Klieber, W.; Marques-Silva, J.; and Clarke, E. M. 2012. Solving QBF with counterexample guided refinement. In *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, 114–128.

Lagniez, J., and Marquis, P. 2014. Preprocessing for propositional model counting. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 2688–2694.

Lang, J., and Marquis, P. 2008. On propositional definability. *Artif. Intell.* 172(8-9):991–1017.

Lonsing, F., and Biere, A. 2010. DepQBF: A dependency-aware QBF solver (system description). *Journal on Satisfiability, Boolean Modeling and Computation* 7:71–76.

Mishchenko, A.; Brayton, R. K.; Feng, W.; and Greene, J. W. 2015. Technology mapping into general programmable cells. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, February 22-24, 2015*, 70–73.

Mneimneh, M. N., and Sakallah, K. A. 2003. Computing vertex eccentricity in exponentially large graphs: QBF formulation and solution. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, 411–425.

Ostrowski, R.; Grégoire, É.; Mazure, B.; and Sais, L. 2002. Recovering and Exploiting Structural Knowledge from CNF Formulas. In *Proc. of the 8th Int. Conf. on Principles and Practice of Constraint Programming (CP 2002)*, 185–199.

Pigorsch, F., and Scholl, C. 2009. Exploiting structure in an AIG based QBF solver. In *Design, Automation and Test in*

Europe, DATE 2009, Nice, France, April 20-24, 2009, 1596–1601.

Plaisted, D. A., and Greenbaum, S. 1986. A structure-preserving clause form translation. *J. Symb. Comput.* 2(3):293–304.

QBFEVAL. 2010. QBF solver evaluation portal. http://www.qbflib.org/qbfeval/.

QBFLIB. 2005. Qdimacs: standard QBF input format. http://www.qbflib.org/qdimacs.

Remshagen, A., and Truemper, K. 2005. An effective algorithm for the futile questioning problem. *J. Autom. Reasoning* 34(1):31–47.

Silva, J. P. M., and Sakallah, K. A. 2000. Boolean satisfiability in electronic design automation. In *DAC*, 675–680.

Tseitin, G. 1970. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*.