From DQBF to QBF by Dependency Elimination

Ralf Wimmer[1(✉)], Andreas Karrenbauer[2], Ruben Becker[2], Christoph Scholl[1], and Bernd Becker[1]

[1] Albert-Ludwigs-Universität Freiburg, Freiburg im Breisgau, Germany
{wimmer,scholl,becker}@informatik.uni-freiburg.de
[2] MPI for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
{karrenba,ruben}@mpi-inf.mpg.de

Abstract. In this paper, we propose the elimination of dependencies to convert a given dependency quantified Boolean formula (DQBF) to an equisatisfiable QBF. We show how to select a set of dependencies to eliminate such that we arrive at a smallest equisatisfiable QBF in terms of existential variables that is achievable using dependency elimination. This approach is improved by taking so-called don't-care dependencies into account, which result from the application of dependency schemes to the formula and can be added to or removed from the formula at no cost. We have implemented this new method in the state-of-the-art DQBF solver HQS. Experiments show that dependency elimination is clearly superior to the previous method using variable elimination.

1 Introduction

Dependency quantified Boolean formulas (DQBFs) have received considerable attention in research during the last years. They are a generalization of ordinary quantified Boolean formulas (QBFs). While the latter have the restriction that every existential variable depends on all universal variables in whose scope it is, DQBFs allow arbitrary dependencies, which are explicitly specified in the formula. This makes DQBFs more expensive to solve than QBFs – for DQBF the decision problem is NEXPTIME-complete, for QBF 'only' PSPACE-complete. However, there are practically relevant applications that require the higher expressiveness of DQBF for a natural and tremendously more compact modeling. Among them is the analysis of multi-player games with incomplete information [22], the synthesis of safe controllers [4] and of certain classes of LTL properties [8], and the verification of incomplete combinational and sequential circuits [12,27,35].

Driven by the needs of the applications mentioned above, research on DQBF solving has not only led to fundamental theoretical results on DQBF [1,3], but also to first solvers like iDQ and HQS [10,11,13,32].

This work was partly supported by the German Research Council (DFG) as part of the project "Solving Dependency Quantified Boolean Formulas" and by the Max Planck Center for Visual Computing and Communication (www.mpc-vcc.org). Ruben Becker is a member of the Saarbrücken Graduate School of Computer Science.

S. Gaspers and T. Walsh (Eds.): SAT 2017, LNCS 10491, pp. 326–343, 2017.
DOI: 10.1007/978-3-319-66263-3_21

While iDQ uses instantiation-based solving, i.e., it reduces deciding a DQBF to deciding a series of SAT problems which correspond to *partial* universal expansions, HQS [13] uses the elimination of universal variables to turn the DQBF at hand into an equisatisfiable QBF, which can be solved by an arbitrary QBF solver. The basic method is complemented by several preprocessing techniques for DQBF [9,32,33] and the application of dependency schemes [34] for manipulating the dependency sets of the DQBF formula without changing its truth value.

In this paper we improve on the state-of-the-art solver HQS by making the following contributions:

(1) We introduce a **novel technique called '*dependency elimination*'** for transforming a DQBF into an equisatisfiable QBF. While [13] uses a minimal number of universal expansions for turning a DQBF $\forall x_1 \ldots \forall x_n \exists y_1(D_{y_1}) \ldots \exists y_m(D_{y_m}) : \phi$ into an equisatisfiable QBF with linearly ordered dependency sets, dependency elimination is able not only to remove universal variables x_i *completely* from the formula, but also to remove universal variables x_i from *single* dependency sets D_{y_j}, i.e., it works with a finer granularity. Dependency elimination is used with the goal of producing fewer copies of existential variables in the final QBF.
(2) We provide a **method for selecting an optimal elimination set**. The main ingredients of this method are:

(a) Dependencies can be represented in a natural way by a bipartite tournament graph, also called the dependency graph. Determination of an optimal elimination set then corresponds to *breaking cycles* in this graph *by flipping a cost-minimal set of edges*. A (non-linear) cost function takes into account the number of existential variables after eliminating a set of dependencies.
(b) An exact and efficient solution for the optimization problem in (a) is presented. It is based on *integer linear programming with dynamically added constraints* similar to the so-called cutting plane approach [36].
(c) The efficiency of the optimal elimination set computation is significantly increased by integrating *symmetry reduction*. Symmetry reduction is based on the observation that in typical applications the number of different dependency sets is rather small. We prove that optimal solutions based on symmetry-reduced graphs are optimal solutions for the original graphs as well. Based on research on dependency schemes [34] we consider in our optimization also dependencies which can be removed 'free of charge' without dependency elimination, since their removal does not change the truth value of the DQBF.
(d) Furthermore, we prove that the problem of finding an optimal elimination set for DQBFs with 'don't-care dependencies' is NP-complete. Note that there are related problems in the literature like 'Minimal Feedback Arc Set' (FAS) for bipartite tournament graphs. FAS for bipartite tournament graphs has been shown to be NP-complete as well in [14], but it differs from our problem in two aspects: We are only allowed to flip a *subset* of all edges (only the edges representing dependencies of existential variables on universal ones) and our cost function does not simply count the number of flipped edges, but it is more complicated and non-linear.

(3) We perform an **extensive experimental evaluation**, proving that the computation time for selecting an optimal elimination set is typically negligible, the number of variable copies produced by the optimal dependency elimination is much smaller compared to full universal variable elimination in many cases, and – overall – that the performance of the solver HQS could be improved to a great extent by the novel approach.

The paper is structured as follows: In the next section we introduce the necessary foundations. Section 3 presents dependency elimination and our procedure which selects an appropriate set of dependencies to eliminate. In Sect. 4 we experimentally evaluate this novel method. Finally, in Sect. 5 we draw conclusions and point out future work.

2 Foundations

For a finite set V of Boolean variables, $\mathcal{A}(V)$ denotes the set of variable assignments of V, i.e., $\mathcal{A}(V) = \{\nu : V \to \mathbb{B}\}$ with $\mathbb{B} = \{0, 1\}$. Given quantifier-free Boolean formulas ϕ and κ over V and a Boolean variable $v \in V$, $\phi[\kappa/v]$ denotes the Boolean formula which results from ϕ by replacing all occurrences of v simultaneously by κ (simultaneous replacement is necessary when κ contains the replaced variable v).

Dependency quantified Boolean formulas are obtained by prefixing Boolean formulas with so-called Henkin quantifiers [15].

Definition 1 (Syntax of DQBF). *Let $V = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$ be a finite set of Boolean variables. A* dependency quantified Boolean formula (DQBF) *ψ over V has the form $\psi := \forall x_1 \ldots \forall x_n \exists y_1(D_{y_1}) \ldots \exists y_m(D_{y_m}) : \phi$, where $D_{y_i} \subseteq \{x_1, \ldots, x_n\}$ is the* dependency set *of y_i for $i = 1, \ldots, m$, and ϕ is a quantifier-free Boolean formula over V, called the* matrix *of ψ.*

We denote the existential variables of a DQBF ψ with $V_\psi^\exists = \{y_1, \ldots, y_m\}$ and its universal variables by $V_\psi^\forall = \{x_1, \ldots, x_n\}$. As the order of the variables in the quantifier prefix Q does not matter, we can regard it as a set: For instance, $Q \backslash \{v\}$ with a variable $v \in V$ is the prefix which results from Q by removing the variable v together with its quantifier (as well as its dependency set in case v is existential, and all its occurrences in dependency sets if it is universal).

The semantics of a DQBF is typically defined in terms of so-called Skolem functions.

Definition 2 (Semantics of DQBF). *Let ψ be a DQBF as above. It is* satisfied *if there are functions $s_y : \mathcal{A}(D_y) \to \mathbb{B}$ for $y \in V_\psi^\exists$ such that replacing each existential variable y by (a Boolean expression for) s_y turns ϕ into a tautology. The functions $(s_y)_{y \in V_\psi^\exists}$ are called* Skolem functions *for ψ.*

Deciding whether a given DQBF is satisfied is NEXPTIME-complete [22].

Definition 3 (Equisatisfiability of DQBFs). *Two DQBFs ψ and ψ' are* equisatisfiable *iff they are either both satisfied or both not satisfied.*

The elimination of universal variables in solvers like HQS [13] is done by *universal expansion* and leads to an equisatisfiable DQBF [1,5,6,12]:

Definition 4 (Universal Expansion). *For a DQBF* $\psi = \forall x_1 \ldots \forall x_n \exists y_1 (D_{y_1}) \ldots \exists y_m (D_{y_m}) : \varphi$ *with* $Z_{x_i} = \left\{ y_j \in V_\psi^\exists \,\middle|\, x_i \in D_{y_j} \right\}$, *the* universal expansion *of variable* $x_i \in V_\psi^\forall$ *is defined by*

$$\left(Q \backslash \{x_i\}\right) \cup \left\{ \exists y'_j (D_{y_j} \backslash \{x_i\}) \,\middle|\, y_j \in Z_{x_i} \right\} : \varphi[1/x_i] \wedge \varphi[0/x_i][y'_j/y_j \text{ for all } y_j \in Z_{x_i}].$$

An important special case of DQBFs is known as *quantified Boolean formulas*. They exhibit a linearly ordered quantifier prefix, where each existential variable y depends on all universal variables in whose scope it is:

Definition 5 (Syntax of QBF, Equivalent QBFs). *Let* $V = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$ *be a finite, non-empty set of Boolean variables,* $X_1, \ldots, X_k \subseteq \{x_1, \ldots, x_n\}$ *a partition of* $\{x_1, \ldots, x_n\}$ *such that* $X_i \neq \emptyset$ *for* $i = 2, \ldots, k$, *and* $Y_1, \ldots, Y_k \subseteq \{y_1, \ldots, y_m\}$ *a partition of* $\{y_1, \ldots, y_m\}$ *such that* $Y_i \neq \emptyset$ *for* $i = 1, \ldots, k-1$. *Additionally let* ϕ *be a quantifier-free Boolean formula over* V.

A quantified Boolean formula (QBF) Ψ *(in prenex form) is given by*

$$\Psi := \forall X_1 \exists Y_1 \forall X_2 \exists Y_2 \ldots \forall X_k \exists Y_k : \phi.$$

The QBF Ψ *is equivalent to the DQBF* $\psi := \forall x_1 \ldots \forall x_n \exists y_1 (D_{y_1}) \ldots \exists y_m (D_{y_m}) : \phi$, *if* $D_{y_i} = \bigcup_{\ell=1}^{L} X_\ell$ *such that* L *is the unique index with* $y_i \in Y_L$. *In this case we say that the DQBF* ψ *'can be written as a QBF* Ψ*' or the DQBF* ψ *'has an equivalent QBF prefix'.*

Lemma 1 ([13]). *A DQBF* ψ *has an equivalent QBF prefix if* $D_y \subseteq D_{y'}$ *or* $D_{y'} \subseteq D_y$ *holds for all* $y, y' \in V_\psi^\exists$.

QBFs can be solved more efficiently than general DQBFs. For QBF, the decision problem is "only" PSPACE-complete [21], and rather efficient solvers for QBF are available like DepQBF [19,20], AIGSolve [23,24], Qesto [17], RAReQS [16], to name just a few. Therefore the goal is to manipulate the DQBF at hand – preserving the truth value – in a way such that the resulting formula has an equivalent QBF prefix and can be solved by any available QBF solver.

3 Dependency Elimination

The DQBF solver HQS [13] uses universal expansion to turn the DQBF at hand into an equisatisfiable QBF. It determines a smallest possible set of universal variables whose elimination yields a QBF. This is done by solving a MAXSAT problem. Using universal expansion has the drawback that it copies *all* variables which depend on the eliminated universal variable.

Example 1. The DQBF $\forall x_1 \forall x_2 \exists y_1(x_1) \exists y_2(x_2) \exists y_3(x_1, x_2) \ldots \exists y_n(x_1, x_2) : \varphi$ does not have an equivalent QBF prefix. Therefore the expansion of either x_1 or x_2 is necessary. When x_1 is eliminated, $y_1, y_3, \ldots, y_n$ are doubled, creating $n-1$ additional existential variables. The elimination of x_2 creates copies of $y_2, y_3, \ldots, y_n$. However, only the dependencies of y_1 on x_1 and of y_2 on x_2 are responsible for the formula not being a QBF.

Therefore we propose an alternative operation that we can use to obtain an equisatisfiable DQBF with an equivalent QBF prefix, namely dependency elimination, which allows to remove single dependencies from a formula.

Theorem 1 (Dependency Elimination). *Assume ψ is a DQBF as in Definition 1 and, w.l.o.g., $x_1 \in D_{y_1}$. Then ψ is equisatisfiable to:*

$$\psi' := \forall x_1 \ldots \forall x_n \; \exists y_1^0\big(D_{y_1} \backslash \{x_1\}\big) \; \exists y_1^1\big(D_{y_1} \backslash \{x_1\}\big) \; \exists y_2\big(D_{y_2}\big) \ldots \exists y_m\big(D_{y_m}\big) : \phi\Big[\big((\neg x_1 \wedge y_1^0) \vee (x_1 \wedge y_1^1)\big)/y_1\Big].$$

Proof. Assume ψ is satisfiable with Skolem functions s_{y_i} for y_i $(1 \leq i \leq m)$. We have $s_{y_1} = (\neg x_1 \wedge s_{y_1\,|x_1=0}) \vee (x_1 \wedge s_{y_1\,|x_1=1})$ for the negative cofactor $s_{y_1\,|x_1=0}$ w.r.t. x_1 and the positive cofactor $s_{y_1\,|x_1=1}$ w.r.t. x_1. Then ψ' is satisfiable, too, with Skolem functions s_{y_i} for y_i $(2 \leq i \leq m)$, $s_{y_1\,|x_1=0}$ for y_1^0 and $s_{y_1\,|x_1=1}$ for y_1^1. Conversely, if ψ' is satisfiable with Skolem functions s_{y_i} for y_i $(2 \leq i \leq m)$, $s_{y_1^0}$ for y_1^0 and $s_{y_1^1}$ for y_1^1, then ψ is satisfiable with Skolem function $s_{y_1} = (x_1 \wedge s_{y_1^1}) \vee (\neg x_1 \wedge s_{y_1^0})$ for y_1 and s_{y_i} for y_i $(2 \leq i \leq m)$. □

Example 2. Consider again the formula from Example 1. If we eliminate the dependency of y_1 on x_1 we obtain the formula

$$\forall x_1 \forall x_2 \exists y_1^0(\emptyset) \exists y_1^1(\emptyset) \exists y_2(x_2) \exists y_3(x_1, x_2) \ldots \exists y_n(x_1, x_2) : \varphi[(\neg x_1 \wedge y_1^0) \vee (x_1 \wedge y_1^1)/y_1].$$

This formula can be written as the QBF

$$\exists y_1^0 \exists y_1^1 \forall x_2 \exists y_2 \forall x_1 \exists y_3 \ldots \exists y_n : \varphi[(\neg x_1 \wedge y_1^0) \vee (x_1 \wedge y_1^1)/y_1].$$

Instead of creating $n-1$ additional existential variables as in Example 1, we only had to double y_1 in order to obtain an equisatisfiable QBF.

The main question that we have to answer is which dependencies should be eliminated in order to obtain an equisatisfiable QBF. If we eliminate n dependencies of an existential variable y, we have to create $2^n - 1$ new copies of y. Therefore it is typically not feasible to eliminate all dependencies, but we have to take care to find a set of dependencies which requires the fewest variable copies and still turns the formula into a QBF.

3.1 Selecting Dependencies to Eliminate

In order to facilitate the selection of dependencies to eliminate we make use of the following dependency graph:

Definition 6 (Dependency Graph). *Let ψ be a DQBF as above. The* dependency graph $G_\psi = \big(V_\psi, E_\psi\big)$ *is a directed graph with the set $V_\psi = V$ of variables as nodes and edges*

$$E_\psi = \big\{(x,y) \in V_\psi^\forall \times V_\psi^\exists \,\big|\, x \in D_y\big\} \dot\cup \big\{(y,x) \in V_\psi^\exists \times V_\psi^\forall \,\big|\, x \notin D_y\big\}.$$

G_ψ is a so-called *bipartite tournament graph* [2,7,14]: The nodes can be partitioned into two disjoint sets according to their quantifier and there are only edges that connect variables with different quantifiers – this is the bipartiteness property. We also write $G_\psi = (V_\psi^\forall, V_\psi^\exists, E_\psi)$ to make the two disjoint node sets apparent. Additionally, for each pair $(x,y) \in V_\psi^\forall \times V_\psi^\exists$ there is either an edge from x to y or vice-versa – this property is referred to by the term 'tournament'.

Theorem 2. *Let ψ be a DQBF and G_ψ its dependency graph. The graph G_ψ is acyclic iff ψ has an equivalent QBF prefix.*

Proof. Assume that ψ has an equivalent QBF prefix. The left-to-right order of this QBF prefix defines a total order $\prec$ on V with $D_y = \{x \in V_\psi^\forall \,|\, x \prec y\}$ for all $y \in V_\psi^\exists$. Then for all edges $(x,y) \in E_\psi$ we have $x \prec y$, and $y \prec x$ holds for all edges $(y,x) \in E_\psi$. That means all edges point to larger elements w.r.t. $\prec$. Therefore G_ψ is acyclic.

Now assume that G_ψ is acyclic. Then we can find a topological order for G_ψ, i. e., there exists a total order $\prec$ on the nodes of G_ψ such that we have: If $(v_1, v_2) \in E_\psi$ then $v_1 \prec v_2$. Now choose the QBF prefix from left to right according to the total order $\prec$. If $x \in D_y$, then $(x,y) \in E_\psi$, $x \prec y$ and x is to the left of y; if $x \notin D_y$, then $(y,x) \in E_\psi$, $y \prec x$ and x is to the right of y. Thus we have found an equivalent QBF prefix. □

Eliminating a dependency essentially corresponds to flipping the direction of an edge $(x,y) \in E_\psi \cap (V_\psi^\forall \times V_\psi^\exists)$ from a universal to an existential variable. The cost of copying existential variables will be taken into account by choosing an appropriate cost for flipping sets of edges. This cost will count the number of existential variables after eliminating a set of dependencies. Our goal is to find a cost-minimal set of edges such that flipping those edges makes the dependency graph acyclic.

In the following, we will first determine a set $R \subseteq E_\psi \cap (V_\psi^\forall \times V_\psi^\exists)$ of edges whose *deletion* makes $G_{\text{del}}^R := (V_\psi^\forall, V_\psi^\exists, E_\psi \backslash R)$ acyclic. However, if R is such a set of edges, then we can turn it into a set $R' \subseteq R$ such that *flipping* the edges in R' yields an acyclic graph: Let $\prec$ be a topological order of G_{del}^R's nodes. Then we set $R' := \big\{(x,y) \in R \,\big|\, x \not\prec y\big\}$, i.e., we flip only those edges of R which point backward according to $\prec$. Then $\prec$ is also a topological order

of $G^{R'}_{\text{flip}} := (V^\forall_\psi, V^\exists_\psi, (E_\psi \backslash R') \cup \{(y,x) \,|\, (x,y) \in R'\})$. Of course, $\prec$ is a topological order of $G^{R'}_{\text{del}} := (V^\forall_\psi, V^\exists_\psi, E_\psi \backslash R')$ as well and $G^{R'}_{\text{del}}$ is acyclic. Thus, if we choose a *minimal* set $R^{\min} \subseteq E_\psi \cap (V^\forall_\psi \times V^\exists_\psi)$ such that $G^{R^{\min}}_{\text{del}}$ is acyclic, then $G^{R^{\min}}_{\text{flip}}$ is acyclic as well. So we can restrict our attention to *removing* edges from $E_\psi \cap (V^\forall_\psi \times V^\exists_\psi)$ for turning G_ψ into an acyclic graph, as long as we remove a *minimal* set of edges – although in our application (turning a DQBF into a QBF by elimination of dependencies) we are only able to *flip* edges from $V^\forall_\psi$ to $V^\exists_\psi$.

We define an elimination set as follows.

Definition 7 (Elimination set). *A set $R \subseteq E_\psi \cap (V^\forall_\psi \times V^\exists_\psi)$ is an* elimination set *for the bipartite tournament graph $G_\psi = (V^\forall_\psi, V^\exists_\psi, E_\psi)$ if $G^R_{del} = (V^\forall_\psi, V^\exists_\psi, E_\psi \backslash R)$ is acyclic. An elimination set R is* minimal *if $R \backslash \{e\}$ is not an elimination set for every $e \in R$.*

Let R be a minimal elimination set. For $y \in V^\exists_\psi$, we set $R_y = \{x \in V^\forall_\psi \,|\, (x,y) \in R\}$. The cost of R is then given by $\text{cost}(R) := \sum_{y \in V^\exists_\psi} 2^{|R_y|}$. The cost of R corresponds to the number of existential variables in the formula after the dependencies in R have been eliminated. Hence, our goal is to determine an elimination set of minimal cost.

3.2 Symmetry Reduction

Before we look into the optimization problem of computing an elimination set of minimal cost, we consider reducing the size of the dependency graph by exploiting symmetries: The existential and universal variables are partitioned according to the dependency sets. We define an equivalence relation $\sim$ by:

$$\begin{aligned} y_i \sim y_j \quad &\Leftrightarrow \quad D_{y_i} = D_{y_j} \\ x_i \sim x_j \quad &\Leftrightarrow \quad \{y_\ell \in V^\exists_\psi \,|\, x_i \in D_{y_\ell}\} = \{y_\ell \in V^\exists_\psi \,|\, x_j \in D_{y_\ell}\}. \end{aligned}$$

The dependency graph modulo $\sim$ is based on the equivalence classes $[v]_\sim$ of $\sim$ and is defined by $G^\sim_\psi = (V^\sim_\psi, E^\sim_\psi)$ where

$$\begin{aligned} V^\sim_\psi &= \big\{ [v]_\sim \,\big|\, v \in V \big\} \text{ and} \\ E^\sim_\psi &= \big\{ ([x_i]_\sim, [y_j]_\sim) \,\big|\, x_i \in D_{y_j} \big\} \dot\cup \big\{ ([y_j]_\sim, [x_i]_\sim) \,\big|\, x_i \notin D_{y_j} \big\}. \end{aligned}$$

By definition, the resulting graph $G^\sim_\psi$ is a bipartite tournament graph again. It is well defined: If $([x]_\sim, [y]_\sim) \in E^\sim_\psi$ holds for some $x \in V^\forall_\psi$ and $y \in V^\exists_\psi$, then $x' \in D_{y'}$ holds for all $x' \in [x]_\sim$ and all $y' \in [y]_\sim$. If $([y]_\sim, [x]_\sim) \in E^\sim_\psi$ holds, then $x' \notin D_{y'}$ for all $x' \in [x]_\sim$ and all $y' \in [y]_\sim$.

A further reduction can be obtained by the following observation: Incident edges of sources (i.e., nodes without incoming edges) and sinks (i.e., nodes without outgoing edges) never need to be flipped, since they cannot occur on a cycle. They can be removed from the graph, together with their incident edges. This can be repeated until the graph does not change anymore. The result is again a bipartite tournament graph.

If $R^\sim$ is the set of edges to be eliminated, transferring the cost function to the reduced graph yields:

$$\text{cost}^\sim(R^\sim) = \sum_{[y]_\sim \in V_\psi^\sim} \left|[y]_\sim\right| \cdot 2^{\sum_{([x]_\sim,[y]_\sim)\in R^\sim} |[x]_\sim|}.$$

The intuition behind the definition of $\text{cost}^\sim(R^\sim)$ is that eliminating $([x]_\sim, [y]_\sim)$ in $G_\psi^\sim$ 'means' eliminating all edges (x', y') in G_ψ with $x' \in [x]_\sim$ and $y' \in [y]_\sim$.

The following Theorem 3 justifies the application of symmetry reduction to the dependency graph: A cost-optimal elimination set for the reduced graph induces a cost-optimal elimination set for the original graph.

Theorem 3. *(a) If $R^\sim$ is a minimal elimination set for $G_\psi^\sim$, then $R = \left\{(x,y) \in (V_\psi^\forall \times V_\psi^\exists) \cap E_\psi \,\middle|\, ([x]_\sim, [y]_\sim) \in R^\sim\right\}$ is a minimal elimination set for G_ψ such that $\text{cost}(R) = \text{cost}^\sim(R^\sim)$.*

(b) If R is a minimal elimination set for G_ψ, then the set $R^\sim := \left\{([x]_\sim, [y]_\sim) \,\middle|\, (x,y) \in R\right\}$ is a minimal elimination set for $G_\psi^\sim$ such that $\text{cost}(R) = \text{cost}^\sim(R^\sim)$.

Before we prove Theorem 3, we show the following Lemma 2.

Lemma 2. *Let R be a minimal elimination set for G_ψ. Then for all $(x,y) \in V_\psi^\forall \times V_\psi^\exists$, we have $(x,y) \in R$ iff $[x]_\sim \times [y]_\sim \subseteq R$.*

Proof (Lemma 2). Let $x_1 \to y_1 \to x_2 \to \cdots \to y_k \to x_1$ be a cycle of G_ψ. Assume that for all $1 \le i \le k$ $[x_i]_\sim \times [y_i]_\sim \subseteq R$ does *not* hold. Then for all $1 \le i \le k$ there are (x_i', y_i') with $x_i' \sim x_i$ and $y_i' \sim y_i$ such that $(x_i', y_i') \notin R$. According to the definition of the relation $\sim$, G_ψ contains the cycle $x_1' \to y_1' \to x_2' \to \cdots \to y_k' \to x_1'$ which is not broken by R. This contradicts our assumption that R is an elimination set.

We conclude that for each cycle $x_1 \to y_1 \to x_2 \to \cdots \to y_k \to x_1$ we have $[x_i]_\sim \times [y_i]_\sim \subseteq R$ for some $1 \le i \le k$. All other (x_j, y_j) with $[x_j]_\sim \times [y_j]_\sim \not\subseteq R$ are not needed to break cycles in G_ψ and are thus not included in R due to minimality of R. □

Proof (Theorem 3). Proof of part (a): Let $x_1 \to y_1 \to x_2 \to \cdots \to y_k \to x_1$ be an arbitrary cycle in G_ψ (if there is no cycle in G_ψ, then it trivially follows that R is an elimination set). By definition of $G_\psi^\sim$, $[x_1]_\sim \to [y_1]_\sim \to [x_2]_\sim \to \cdots \to [y_k]_\sim \to [x_1]_\sim$ is a cycle in $G_\psi^\sim$. Since $R^\sim$ is an elimination set for $G_\psi^\sim$, $\left([x_i]_\sim, [y_i]_\sim\right) \in R^\sim$ for some $i \in \{1, \ldots, k\}$ and by definition of R we have

$(x_i, y_i) \in R$. Therefore R is an elimination set. That the values of the cost functions coincide is easy to see.

We still have to show that R is minimal. Assume the contrary, i.e., there is $(x_1, y_1) \in R$ such that $R \backslash \{(x_1, y_1)\}$ is still an elimination set. By construction, $([x_1]_\sim, [y_1]_\sim) \in R^\sim$. Due to minimality of $R^\sim$, there has to be a cycle in $G_\psi^\sim$ containing $([x_1]_\sim, [y_1]_\sim)$. Let $[x_1]_\sim \to [y_1]_\sim \to [x_2]_\sim \to \cdots \to [y_k]_\sim \to [x_1]_\sim$ be such a cycle without node repetitions (i.e., a simple cycle). By definition of $G_\psi^\sim$, $x_1 \to y_1 \to x_2 \to \cdots \to y_k \to x_1$ is a cycle in G_ψ and $(x_i, y_i) \in R$ for some $i \in \{2, \ldots, k\}$, since $R \backslash \{(x_1, y_1)\}$ is an elimination set. By definition of R, $([x_i]_\sim, [y_i]_\sim) \in R^\sim$ and $([x_i]_\sim, [y_i]_\sim) \neq ([x_1]_\sim, [y_1]_\sim)$, since the cycle is simple. Therefore, all simple cycles broken by $([x_1]_\sim, [y_1]_\sim)$ are also broken by another edge from $R^\sim$. That means, $R^\sim$ is not minimal, contradicting our assumption.

The proof of part (b) immediately follows from Lemma 2. □

3.3 An Optimization Approach

The Underlying Optimization Problem. Our goal in the above described problem is to determine an elimination set $R^\sim$ of minimal cost $\text{cost}^\sim(R^\sim)$. We can determine such an elimination set by selecting one edge from each simple cycle in the dependency graph. A cycle is called simple if it does not contain any subcycle. This can be formulated as an optimization problem in a given arbitrary bipartite tournament graph G with disjoint node sets X, Y and node weights $\omega : X \cup Y \to \mathbb{R}_{>0}$. (Remember that in our application the nodes $X \cup Y$ represent equivalence classes $[v]_\sim$. Their weights correspond to the cardinality of $[v]_\sim$.) We introduce a decision variable $d_{(x,y)} \in \{0, 1\}$ for each edge $(x, y) \in E_{XY}$, where $E_{XY} = E \cap (X \times Y)$ with the interpretation that $d_{(x,y)} = 1$ indicates that (x, y) belongs to the elimination set. Let C denote the set of all simple cycles $c = x_1 \to y_1 \to x_2 \to \cdots \to y_k \to x_1$ such that $x_i \in X$, $y_i \in Y$ for all $i = 1, \ldots, k$ and $x_i \neq x_j$, $y_i \neq y_j$ for $i \neq j$. Moreover, for a simple cycle $c \in C$, let us denote with $F(c) = E(c) \cap (X \times Y)$, the set of all arcs in c that are directed from a node in X to a node in Y. The optimization problem can then be formulated as

$$\text{minimize} \quad \sum_{y \in Y} \omega(y) \cdot 2^{\sum_{x \in \text{pre}(y)} \omega(x) \cdot d_{(x,y)}} \tag{1a}$$

$$\text{such that} \quad \sum_{(x,y) \in F(c)} d_{(x,y)} \geq 1 \qquad \forall c \in C \tag{1b}$$

$$d_{(x,y)} \in \{0, 1\} \qquad \forall (x, y) \in E_{XY} \tag{1c}$$

Two challenges make solving this optimization problem difficult: First, the objective function is non-linear as it is the sum of exponential functions. Second, the number of cycles in the dependency graph may be prohibitively large.

Solving the Optimization Problem. In order to bring the optimization problem into a more convenient form, we first rewrite it by introducing variables $f_y \in \mathbb{Z}$ and $z_y \in \mathbb{Z}$ for every $y \in Y$:

$$\text{minimize} \quad \sum_{y \in Y} \omega(y) \cdot z_y \tag{2a}$$

$$\text{such that} \quad \sum_{(x,y) \in F(c)} d_{(x,y)} \geq 1 \qquad \forall c \in C \tag{2b}$$

$$\sum_{x \in \text{pre}(y)} \omega(x) \cdot d_{(x,y)} = f_y \qquad \forall y \in Y \tag{2c}$$

$$z_y \geq 2^{f_y} \qquad \forall y \in Y \tag{2d}$$

$$d_{(x,y)} \in \{0,1\} \qquad \forall (x,y) \in E_{XY} \tag{2e}$$

After this transformation, (2d) is the only non-linear constraint and the objective function is linear. We will handle these non-linear constraints dynamically as follows. We first solve the optimization problem consisting only of (2b), (2c), and (2e). These constraints form an integer linear program (ILP). Solving this ILP yields a solution $\bar{d}, \bar{f}, \bar{z}$ that are not necessarily feasible for the complete optimization problem (2). Thus, we check whether, for some $y \in Y$, the constraint $\bar{z}_y \geq 2^{\bar{f}_y}$ is violated. In this case, we add a linear inequality (lazy constraint approach) that cuts off this infeasible solution, but none of the feasible points. Such an inequality is, e.g., the constraint that z_y lies on or above the tangent to the function 2^{f_y} in the current value $\bar{f}_y$ of f_y. This tangent is described by $t(f_y) = 2^{\bar{f}_y} \cdot (1 + (f_y - \bar{f}_y) \cdot \ln 2)$. Thus, we could add the inequality $z_y \geq 2^{\bar{f}_y} \cdot \big(1 + (f_y - \bar{f}_y) \cdot \ln 2\big)$. However, such inequalities are not rational and their closure yields non-integral extreme points. Instead, we can take the secants through two adjacent extreme points of the convex hull of the integer points satisfying (2d) (feasible solutions are integer). The secants through $\bar{f}_y$ and $\bar{f}_y + 1$ and through $\bar{f}_y - 1$ and $\bar{f}_y$ yield the constraints

$$z_y \geq 2^{\bar{f}_y}(1 - \bar{f}_y + f_y) \quad \text{and} \quad z_y \geq 2^{\bar{f}_y - 1}(2 - \bar{f}_y + f_y).$$

Taken together, the two secant constraints are tighter than the tangent constraint and moreover their description contains only integer coefficients. This is why the secant constraints are preferable (see Fig. 1).

To further increase efficiency, we also relax the cycle constraints (2b) by only adding constraints for C_4, the set of all 4-cycles, first.[1] The longer cycle constraints are handled dynamically as well: If we obtain a solution, we check by depth-first search whether the induced graph is acyclic. If it is not, we add (2b) for the found cycle.[2] The described approach leads to Algorithm 1. Adding

[1] Note that each cyclic bipartite tournament graph has a cycle of length 4.

[2] The approach of dynamically or lazily adding constraints is similar to the cutting plane approach [36] and is used as one of the main ingredients for efficiently solving many NP-hard problems for which only a description with exponentially many constraints is at hand, as for example the traveling salesman problem.

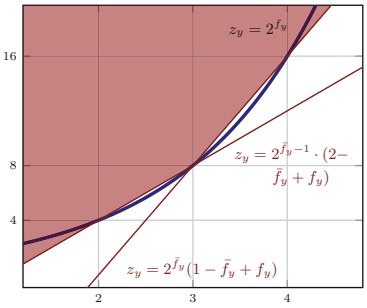

Fig. 1. The two secants on the function $z_y = 2^{f_y}$ at $\bar{f}_y = 2$. The shaded area denotes the feasible region defined by the two inequalities corresponding to the two secants.

separation constraints dynamically is typically supported by ILP solvers like Gurobi using call-back functions.

3.4 Don't-Care Dependencies

During preprocessing, often dependencies can be identified, which are only pseudo-dependencies (also called don't-care dependencies), i.e., an existential variable y contains a universal variable x in its dependency set, but it can be shown that removing the dependency does not change the satisfiability of the formula. Don't-care dependencies correspond to edges in the dependency graph which can be flipped without any costs.

Definition 8 (DQBF with Don't-Care Dependencies). *Let* $\psi = \forall x_1 \ldots \forall x_n \exists y_1(D_{y_1}) \ldots \exists y_m(D_{y_m}) : \phi$ *be a DQBF as before and* $S_{y_i} \subseteq D_{y_i}$ *for* $i = 1, \ldots, m$. *The sets* S_{y_i} *are called* don't-care sets *of* ψ *if* ψ *is equisatisfiable to*

$$\forall x_1 \ldots \forall x_n \exists y_1(D_{y_1} \setminus S_{y_1}) \ldots \exists y_m(D_{y_m} \setminus S_{y_m}) : \phi.$$

Detecting Don't-Care Dependencies. Using the same proof idea as described in [26] for QBF, we can show that deciding whether a dependency is a don't-care dependency has the same complexity as deciding DQBF itself. Therefore one usually resorts to efficient approximations for computing don't-care dependencies.

Lemma 3. *Let* $\psi = \forall x_1 \ldots \forall x_n \exists y_1(D_{y_1}) \ldots \exists y_m(D_{y_m}) : \phi$ *be a DQBF. Deciding whether* $x \in D_y$ *for* $x \in V_\psi^\forall$ *and* $y \in V_\psi^\exists$ *is a don't-care dependency is NEXPTIME-complete.*

Don't-care dependencies can be detected using so-called dependency schemes, which provide over-approximations of the actually dependent variables, see [25,28–30] for QBF and [32,34] for DQBF. Dependency schemes are based on efficient syntactic criteria, and by over-approximating the dependent variables they under-approximate the sets of don't-care dependencies.

Algorithm 1. SolveExact$(G = (X, Y, E), \omega)$

$C \leftarrow C_4$, $P \leftarrow \emptyset$
while True **do**
  Determine $(\bar{d}, \bar{f}, \bar{z})$ as the optimal solution of

$$\begin{aligned}
\text{minimize} \quad & \sum_{y \in Y} \omega(y) \cdot z_y \\
\text{such that} \quad & \sum_{(x,y) \in F(c)} d_{(x,y)} \geq 1 && \forall c \in C \\
& \sum_{x \in \mathrm{pre}(y)} \omega(x) \cdot d_{(x,y)} = f_y && \forall y \in Y \\
& z_y \geq 2^{\bar{f}_y}(1 - \bar{f}_y + f_y) \\
& z_y \geq 2^{\bar{f}_y - 1}(2 - \bar{f}_y + f_y) && \forall (y, \bar{f}_y) \in P \\
& d_{(x,y)} \in \{0, 1\} && \forall (x, y) \in E_{XY}
\end{aligned}$$

  constr_added $\leftarrow$ **false**
  for $y \in Y$ **do**
    if $\bar{z}_y < 2^{\bar{f}_y}$ **then**
      $P \leftarrow P \cup \{(y, \bar{f}_y)\}$, constr_added $\leftarrow$ **true**
  if *cycle c exists in* $G' := (X, Y, E \backslash \{(x, y) : d_{(x,y)} = 1\})$ **then**
    $C \leftarrow C \cup c$, constr_added $\leftarrow$ **true**
  if ***not*** constr_added **then**
    return $(\bar{d}, \bar{f}, \bar{z})$

Exploiting Don't-Care Dependencies. To exploit don't-care dependencies, we first have to refine the symmetry reduction to take not only the dependency sets, but also the don't-care dependencies into account. This yields the following refined equivalence relation $\approx \subseteq V \times V$:

$$\begin{aligned}
y_i \approx y_j \quad &\Leftrightarrow \quad D_{y_i} = D_{y_j} \wedge S_{y_i} = S_{y_j}, \\
x_i \approx x_j \quad &\Leftrightarrow \quad \{y \in V_\psi^\exists \mid x_i \in D_y\} = \{y \in V_\psi^\exists \mid x_j \in D_y\} \\
&\qquad \wedge \{y \in V_\psi^\exists \mid x_i \in S_y\} = \{y \in V_\psi^\exists \mid x_j \in S_y\}.
\end{aligned}$$

The graph $G_\psi^{\approx}$, resulting from G_ψ by merging equivalent nodes, is again a bipartite tournament graph.

Let $G = (X, Y, E)$ be the bipartite graph corresponding to a DQBF ψ with don't-care sets. We define the weight function $\omega : X \cup Y \to \mathbb{R}$ as follows: $\omega(v) = 1$ for all $v \in X \cup Y$ if the graph was not reduced, otherwise $\omega(v) = \left|[v]_\approx\right|$ if we have applied symmetry reduction using $\approx$. Let $DC = \{(x, y) \mid x \in S_y\}$ be the set

of all don't-care dependencies and let $R \subseteq E_{XY}$ be an elimination set. Then we just do not count the cost for eliminating don't-care dependencies, i.e.,

$$\text{cost}(R) = \sum_{y \in Y} \omega(y) \cdot 2^{\sum_{(x,y) \in (R \setminus DC)} \omega(x)}.$$

The don't-care sets can easily be taken into account in Algorithm 1: Before applying Algorithm 1 we delete all edges in DC (i.e., all edges corresponding to don't-care dependencies) from the dependency graph, since those eliminations are free of charge, and apply Algorithm 1 without any other change. This means that we implicitly start with $R = DC$ and then add to the elimination set all $(x, y) \in V_\psi^\forall \times V_\psi^\exists$ with $\bar{d}_{(x,y)} = 1$ in the solution returned by Algorithm 1. The resulting elimination set R is not necessarily minimal and thus eliminating all those dependencies (which corresponds to flipping all edges in R) does not necessarily make the dependency graph acyclic. However, we can find an appropriate subset $R' \subseteq R$ such that flipping all edges from R' makes the dependency graph acyclic using the method from Sect. 3.1. (Another option would be to remove elements of DC from R one after the other as long as the resulting set R remains an elimination set, finally arriving at a minimal elimination set.)

Finally, we provide a complexity result for computing cost-minimal elimination sets in the presence of don't-care dependencies.

Lemma 4. *Given a bipartite tournament graph $G_\psi = \big((X, Y), E_\psi\big)$, a set of don't-care dependencies $S_\psi \subseteq E_\psi$, and an integer $c \geq 0$, deciding whether an elimination set R with $\text{cost}(R) \leq c$ exists is NP-complete.*

This lemma can be proven by a reduction from vertex cover [18]. It is easy to see that Lemma 4 holds for symmetry-reduced graphs as well.

Since there is a one-to-one correspondence between dependency graphs and DQBF prefixes, we can conclude:

Theorem 4. *Given a DQBF with don't-care dependencies and an integer $c \geq 0$, deciding whether there is an elimination set R with $\text{cost}(R) \leq c$ is NP-complete.*

4 Experimental Evaluation

We have extended the DQBF solver HQS [13] to support dependency elimination on its internal formula representation as an And-Inverter Graph (AIG). To determine an optimal elimination set, we use a Python script which is called by HQS and which in turn calls the MILP solver Gurobi 7.0.2 to solve the optimization problem as in Algorithm 1. We use our preprocessor HQSpre [33] to simplify the instances before the actual solution process starts. Since the benchmarks used for evaluation were generated from incomplete circuits and controller synthesis problems, we run HQSpre in its gate-preserving mode. Additionally, we apply the reflexive quadrangle resolution path dependency scheme [34] to identify don't-care dependencies. As the last step of preprocessing, we use syntactic

gate detection to reconstruct the underlying circuit structure; this removes variables which have been introduced artificially to obtain a formula in conjunctive normal form and leads to more compact AIGs.

All experiments were run on one core of an Intel Xeon CPU E5-2450 (8 cores) running at 2.10 GHz clock frequency and having 32 GB of main memory. Ubuntu 16.04 in 64 bit mode was used as the operating system. We aborted each experiment which took more than 3600 s of CPU time or more than 10 GB of memory. We used the same 4811 benchmark instances as in [13,31,32,34]. They mainly encompass partial equivalence checking problems [12,27] for combinational circuits, controller synthesis problems for sequential circuits and safety properties [4].

For the comparison of variable and dependency elimination, we switched off the UNSAT filtering procedure, which is based on QBF abstractions [9]; it affects both solution procedures in exactly the same way. Additionally we skipped those instances which were solved by the preprocessor or which the preprocessor could turn into QBFs. This led to a benchmark set of 3618 instances.

We solved all instances with the original version of HQS [13] using variable elimination and by dependency elimination for a cost-minimal elimination set as described in Sect. 3. In the latter case, if the elimination set contains, for some universal variable $x \in V_\psi^\forall$, all dependencies x is involved in, then we call universal expansion for x instead as it has the same effect as first eliminating the dependencies and then expanding x, but is slightly faster. Otherwise we eliminate the selected dependencies according to Theorem 1. We distinguished instances for which only universal expansion needed to be applied from those which also required the elimination of dependencies.

For 3233 out of 3618 instances, the optimal elimination set removed universal variables from all dependency sets they were involved in, i.e., only universal expansion was used. Those instances do not profit from dependency elimination in terms of copied existential variables. From those instances, variable elimination as in [13] was able to solve 2429 instances, and the novel dependency elimination 2411 instances. The reason for the small difference of 18 instances is that both methods do not necessarily expand the same variables. Since the selection procedures only consider the formula's prefix and not the structure of the matrix, there is no guarantee that an elimination set is found which leads to small formulas during subsequent solution of the resulting QBF. This can also be observed in Fig. 2 where we compare dependency and variable elimination. The left plot compares the computation times, the right one the number of existential variables in the resulting QBF.

For the remaining 385 instances, dependency elimination has an advantage over variable elimination. It can yield an equisatisfiable QBF with fewer existential variables. Accordingly, using dependency elimination, we could solve 325 instances, while variable elimination succeeded only for 177. All instances which could be solved using variable elimination were also solved using dependency elimination. A more detailed comparison of the two methods on these 385 instances is shown in Fig. 3. We can distinguish two subsets of instances: There are some for which the difference between variable and dependency elimination

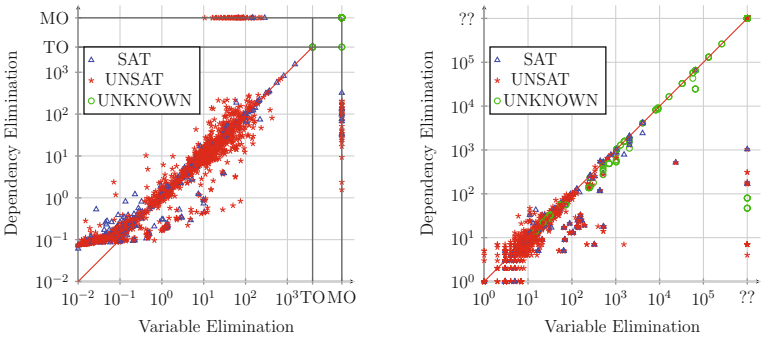

Fig. 2. Comparing the total computation times (in seconds) (left) and the number of existential variables in the resulting QBF (right) after variable and dependency elimination on the instances for which variable elimination *is* optimal.

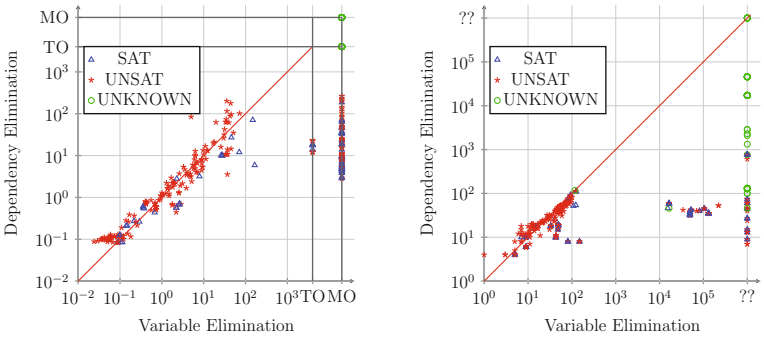

Fig. 3. Comparing the total computation times (in seconds) (left) and the number of existential variables in the resulting QBF (right) after variable and dependency elimination on the instances for which variable elimination is *not* optimal.

is small. Here the computation times are similar. Note that the number of existential variables in the QBF can even be slightly smaller for variable elimination because during expansion often unit and pure variables are detected which are immediately replaced by appropriate constants [13]. In contrast, dependency elimination replaces an existential variable by the representation of a multiplexer. This is an efficient local operation on AIGs, but typically does not allow to detect unit and pure variables. They are found later during the QBF solution process.

For the other subset of instances, dependency elimination is by orders of magnitude superior compared to variable elimination. While the latter runs exceed the memory limit for most of the instances, dependency elimination was able to solve them in little time and with much less memory consumption.

The computation times for selecting an optimal elimination set are negligible in most cases and are always below 10 s for our benchmark set. One reason for the

small computation times is that most benchmarks contain only a small number of *different* dependency sets and thus our symmetry reduction from Sect. 3.2 works nicely; typically the reduced graphs consisted of 10–50 nodes only; the largest one had 54 nodes.

In our current (preliminary) implementation the effect of don't-care dependencies is negligible. The reason for this lies (a) in a restricted preprocessing that is tailored to our backend QBF solver AIGSolve and (b) in the structure of most of our benchmarks. The dependency schemes find a considerable amount of don't-care dependencies on our set of benchmarks [34], but only after intensive preprocessing including operations like blocked clause elimination. Since AIGSolve profits from structure extraction from a CNF, we omit such preprocessing steps that destroy the structure. If we used a different backend QBF solver which does not rely on structure extraction, then we could use the full power of preprocessing, obtain many more don't-care dependencies, and profit much more from don't-care dependencies than in the current scenario.

In summary, for instances where variable elimination is already optimal, dependency elimination yields similar results. However, dependency elimination can yield equisatisfiable QBF that are smaller by orders of magnitude and allows to solve more instances in less time when variable elimination is not optimal. Therefore, dependency elimination is clearly superior to variable elimination.

5 Conclusion

We have presented a novel method to turn a DQBF into an equisatisfiable QBF. This is done by eliminating an appropriate set of dependencies from the formula, which requires to create copies of the involved existential variables. To determine an optimal elimination set that requires the fewest variable copies, we formulate this problem as a constraint system with non-linear objective function. This is solved using an MILP solver by handling the non-linearities by separation. Experiments show that dependency elimination allows to solve more instances with less memory consumption compared to variable elimination. Future research will try to integrate the structure of the formula into the selection process (which is currently only based on the quantifier prefix).

References

1. Balabanov, V., Chiang, H.K., Jiang, J.R.: Henkin quantifiers and Boolean formulae: a certification perspective of DQBF. Theor. Comput. Sci. **523**, 86–100 (2014)
2. Beineke, L.W., Little, C.H.C.: Cycles in bipartite tournaments. J. Comb. Theor. Ser. B **32**(2), 140–145 (1982)
3. Beyersdorff, O., Chew, L., Schmidt, R.A., Suda, M.: Lifting QBF resolution calculi to DQBF. In: Creignou, N., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 490–499. Springer, Cham (2016). doi:10.1007/978-3-319-40970-2_30
4. Bloem, R., Könighofer, R., Seidl, M.: SAT-based synthesis methods for safety specs. In: McMillan, K.L., Rival, X. (eds.) VMCAI 2014. LNCS, vol. 8318, pp. 1–20. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54013-4_1

5. Bubeck, U.: Model-based transformations for quantified Boolean formulas. Ph.D. thesis, University of Paderborn (2010)
6. Bubeck, U., Büning, H.K.: Dependency quantified horn formulas: models and complexity. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 198–211. Springer, Heidelberg (2006). doi:10.1007/11814948_21
7. Cai, M., Deng, X., Zang, W.: A min-max theorem on feedback vertex sets. Math. Oper. Res. **27**(2), 361–371 (2002)
8. Chatterjee, K., Henzinger, T.A., Otop, J., Pavlogiannis, A.: Distributed synthesis for LTL fragments. In: FMCAD 2013, pp. 18–25. IEEE, October 2013
9. Finkbeiner, B., Tentrup, L.: Fast DQBF refutation. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 243–251. Springer, Cham (2014). doi:10.1007/978-3-319-09284-3_19
10. Fröhlich, A., Kovásznai, G., Biere, A.: A DPLL algorithm for solving DQBF. In: International Workshop on Pragmatics of SAT (POS), Trento, Italy (2012)
11. Fröhlich, A., Kovásznai, G., Biere, A., Veith, H.: iDQ: instantiation-based DQBF solving. In: Le Berre, D. (ed.) International Workshop on Pragmatics of SAT (POS 2014), Vienna, Austria. EPiC Series, vol. 27, pp. 103–116. EasyChair, July 2014
12. Gitina, K., Reimer, S., Sauer, M., Wimmer, R., Scholl, C., Becker, B.: Equivalence checking of partial designs using dependency quantified Boolean formulae. In: ICCD 2013, Asheville, NC, USA, pp. 396–403. IEEE CS, October 2013
13. Gitina, K., Wimmer, R., Reimer, S., Sauer, M., Scholl, C., Becker, B.: Solving DQBF through quantifier elimination. In: DATE 2015, Grenoble, France. IEEE, March 2015
14. Guo, J., Hüffner, F., Moser, H.: Feedback arc set in bipartite tournaments is NP-complete. Inf. Process. Lett. **102**(2–3), 62–65 (2007)
15. Henkin, L.: Some remarks on infinitely long formulas. In: Infinitistic Methods: Proceedings of the 1959 Symposium on Foundations of Mathematics, Warsaw, pp. 167–183. Panstwowe Wydawnictwo Naukowe, Panstwowe, September 1961
16. Janota, M., Klieber, W., Marques-Silva, J., Clarke, E.: Solving QBF with counterexample guided refinement. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 114–128. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31612-8_10
17. Janota, M., Marques-Silva, J.: Solving QBF by clause selection. In: Yang, Q., Wooldridge, M. (eds.) IJCAI 2015, Buenos Aires, Argentina, pp. 325–331. AAAI Press (2015). http://ijcai.org/Abstract/15/052
18. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Proceedings of the Symposium on the Complexity of Computer Computations. The IBM Research Symposia Series, pp. 85–103. Plenum Press, New York, IBM Thomas J. Watson Research Center, Yorktown Heights (1972)
19. Lonsing, F., Biere, A.: DepQBF: a dependency-aware QBF solver. J. Satisf. Boolean Model. Comput. **7**(2–3), 71–76 (2010)
20. Lonsing, F., Egly, U.: Incremental QBF solving by DepQBF. In: Hong, H., Yap, C. (eds.) ICMS 2014. LNCS, vol. 8592, pp. 307–314. Springer, Heidelberg (2014). doi:10.1007/978-3-662-44199-2_48
21. Meyer, A.R., Stockmeyer, L.J.: Word problems requiring exponential time: preliminary report. In: STOC, pp. 1–9. ACM Press (1973)
22. Peterson, G., Reif, J., Azhar, S.: Lower bounds for multiplayer non-cooperative games of incomplete information. Comput. Math. Appl. **41**(7–8), 957–992 (2001)
23. Pigorsch, F., Scholl, C.: Exploiting structure in an AIG based QBF solver. In: DATE 2009, Nice, France, pp. 1596–1601. IEEE, April 2009

24. Pigorsch, F., Scholl, C.: An AIG-based QBF-solver using SAT for preprocessing. In: Sapatnekar, S.S. (ed.) DAC 2010, Anaheim, CA, USA, pp. 170–175. ACM Press, July 2010
25. Samer, M.: Variable dependencies of quantified CSPs. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS, vol. 5330, pp. 512–527. Springer, Heidelberg (2008). doi:10.1007/978-3-540-89439-1_49
26. Samer, M., Szeider, S.: Backdoor sets of quantified Boolean formulas. J. Autom. Reason. **42**(1), 77–97 (2009)
27. Scholl, C., Becker, B.: Checking equivalence for partial implementations. In: DAC 2001, Las Vegas, NV, USA, pp. 238–243. ACM Press, June 2001
28. Slivovsky, F., Szeider, S.: Computing resolution-path dependencies in linear time. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 58–71. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31612-8_6
29. Slivovsky, F., Szeider, S.: Quantifier reordering for QBF. J. Autom. Reason. **56**, 459–477 (2015)
30. Gelder, A.: Variable independence and resolution paths for quantified Boolean formulas. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 789–803. Springer, Heidelberg (2011). doi:10.1007/978-3-642-23786-7_59
31. Wimmer, K., Wimmer, R., Scholl, C., Becker, B.: Skolem functions for DQBF. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 395–411. Springer, Cham (2016). doi:10.1007/978-3-319-46520-3_25
32. Wimmer, R., Gitina, K., Nist, J., Scholl, C., Becker, B.: Preprocessing for DQBF. In: Heule, M., Weaver, S. (eds.) SAT 2015. LNCS, vol. 9340, pp. 173–190. Springer, Cham (2015). doi:10.1007/978-3-319-24318-4_13
33. Wimmer, R., Reimer, S., Marin, P., Becker, B.: HQSpre – an effective preprocessor for QBF and DQBF. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 373–390. Springer, Heidelberg (2017). doi:10.1007/978-3-662-54577-5_21
34. Wimmer, R., Scholl, C., Wimmer, K., Becker, B.: Dependency schemes for DQBF. In: Creignou, N., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 473–489. Springer, Cham (2016). doi:10.1007/978-3-319-40970-2_29
35. Wimmer, R., Wimmer, K., Scholl, C., Becker, B.: Analysis of incomplete circuits using dependency quantified Boolean formulas. In: International Workshop on Logic and Synthesis (IWLS) (2016)
36. Wolsey, L.A.: Integer Programming. Wiley-Interscience, New York (1998)