Preprint from 21th International Conference on Theory and Applications of Satisfiability Testing (SAT), July 2018, Oxford, UK

Dependency Quantified Boolean Formulas: An Overview of Solution Methods and Applications*

– Extended Abstract –

Christoph Scholl and Ralf Wimmer

Albert-Ludwigs-Universität Freiburg, Freiburg im Breisgau, Germany
{scholl | wimmer}@informatik.uni-freiburg.de

Abstract. Dependency quantified Boolean formulas (DQBFs) as a generalization of quantified Boolean formulas (QBFs) have received considerable attention in research during the last years. Here we give an overview of the solution methods developed for DQBF so far. The exposition is complemented with the discussion of various applications that can be handled with DQBF solving.

1 Introduction

Dependency quantified Boolean formulas (DQBFs) [30] 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 [29], for QBF 'only' PSPACE-complete [28]. 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 [29], the synthesis of safe controllers [8] and of certain classes of LTL properties [12], and the verification of incomplete combinational and sequential circuits [33,17,39].

Driven by the needs of the applications mentioned above, research on DQBF solving has not only led to fundamental theoretical results on DQBF analyzing which proof calculi for QBF are still sound and/or complete for DQBF [3,5], but also to first solvers like iDQ and HQS [15,16,18,36].

In this work, we give an overview of the solution methods for DQBF developed during the last years as well as of various applications that can be handled with DQBF solving.

* This work was partly supported by the German Research Council (DFG) as part of the project "Solving Dependency Quantified Boolean Formulas"

2 Notions and Problem Definition

In this section, we provide preliminaries and, in particular, we define dependency quantified Boolean formulas as a generalization of quantified Boolean formulas.

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 $v \in V$, $\varphi[\kappa/v]$ 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).

Quantified Boolean formulas are obtained by prefixing Boolean formulas with a 'linearly ordered' quantifier prefix.

Definition 1 (Syntax of QBF). *Let $V = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$ be a finite set of Boolean variables. A* quantified Boolean formula (QBF) *ψ over V (in prenex form) is given by $\psi := \forall X_1 \exists Y_1 \ldots \forall X_k \exists Y_k : \varphi$, where $k \geq 1$, $X_1, \ldots, X_k$ is a partition of the universal variables $\{x_1, \ldots, x_n\}$, $Y_1, \ldots, Y_k$ is a partition of the existential variables $\{y_1, \ldots, y_m\}$, $X_i \neq \emptyset$ for $i = 2, \ldots, k$, and $Y_j \neq \emptyset$ for $j = 1, \ldots, k-1$, and φ is a quantifier-free Boolean formula over V, called the* matrix *of ψ.*

Dependency quantified Boolean formulas are obtained by prefixing Boolean formulas with so-called Henkin quantifiers [21].

Definition 2 (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 (in prenex form) has the form $\psi := \forall x_1 \ldots \forall x_n \exists y_1(D_{y_1}) \ldots \exists y_m(D_{y_m}) : \varphi$, 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.*

A QBF can be seen as a DQBF where the dependency sets are linearly ordered. A QBF $\psi := \forall X_1 \exists Y_1 \ldots \forall X_k \exists Y_k : \varphi$ is equivalent to the DQBF $\psi := \forall x_1 \ldots \forall x_n \exists y_1(D_{y_1}) \ldots \exists y_m(D_{y_m}) : \varphi$ with $D_{y_i} = \cup_{j=1}^{\ell} X_j$ iff Y_ℓ is the unique set with $y_i \in Y_\ell$, $1 \leq \ell \leq k$, $1 \leq i \leq m$.

We denote the existential variables of a DQBF ψ with $V_\psi^\exists = \{y_1, \ldots, y_m\}$ and its universal variables with $V_\psi^\forall = \{x_1, \ldots, x_n\}$. We often write $\psi = Q : \varphi$ with the quantifier prefix Q and the matrix φ. As the order of the variables in a DQBF quantifier prefix Q does not matter, we can regard it as a set: For instance, $Q \setminus \{v\}$ 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 3 (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.*

Definition 4 (Equisatisfiability of DQBFs). *Two DQBFs ψ and ψ' are* equisatisfiable *iff they are either both satisfied or both not satisfied.*

Deciding whether a given DQBF is satisfied is NEXPTIME-complete [29], whereas deciding whether a given QBF is satisfied is 'only' PSPACE-complete [28].

3 Overview of Solution Methods

In this section, we give an overview of different solution methods to the DQBF problem. After briefly discussing incomplete solution methods, we present various sound and complete methods. Whereas search-based solvers using the conflict-driven clause learning (CDCL) paradigm [34] seem to outperform other sound and complete solution paradigms for the SAT problem, the situation is not that clear for QBF solving and neither is it for DQBF solving.

3.1 Incomplete Approximations

An obvious approximation approach is an approximation of a DQBF by a QBF formulation whose (implicitly given) dependency sets are supersets of the original dependency sets in the DQBF. If there is no solution to the relaxed problem, then there is also no solution to the original problem (see [33], e. g.). The QBF approximation can even be approximated further by replacing (some or all) universal variables by existential variables which may assume values from the ternary domain $\{0, 1, X\}$ [22,33]. Here X represents an unknown value and is propagated according to standard rules for unknowns [1].

The work of Finkbeiner and Tentrup [14] was the first one to increase the exactness of the obvious QBF approximations by a series of more and more complex QBF formulations.

Balabanov et al. [3] show that resolution together with universal reduction, which is sound and complete for QBF, is no longer complete (but still sound) for DQBF, leading to another incomplete method for DQBF.

3.2 Search-based Solution

A natural idea for DQBF solving is to generalize successful search-based QBF solvers like DepQBF [25,26]. The problem with QBF solvers applied to DQBF is that the solver assignments to existential variables depend on all universal variables assigned before. That means that an unmodified search-based QBF solver can only respect linearly ordered dependency sets. In [15], for a given DQBF linearly ordered dependency sets are computed that over-approximate the original dependency sets. A search-based QBF solver respecting those linearly ordered dependency sets now has to consider additional constraints: In different paths of the search tree that lead to the same existential variable y_i, but do not differ in the assignments to the variables in D_{y_i}, y_i has to be assigned the same

value. In order to enforce those constraints, [15] extends the search-based QDPLL algorithm by learning additional clauses, called Skolem clauses, after assignments to existential variables y_i. The Skolem clauses just record an implication between the current assignments to the variables in D_{y_i} and the current assignment to y_i for transporting this information into other paths of the search tree. Backtracking in case of a conflict has to take these Skolem clauses into account and removes them once they become invalid (which is in contrast to conventional learned clauses that can remain after a conflict, since they are implied).

3.3 Abstraction-based Solution

In the QBF context there are rather strong solvers like RAReQS [23] and Qesto [24] which work according to the CEGAR paradigm. Fröhlich et al. [16] use a similar idea by solving a series of SAT instantiations. Their solver is based on computing partial universal expansions, which yield over-approximations. If a SAT solver determines that the over-approximation is unsatisfiable, they can conclude the unsatisfiability of the DQBF. In case of satisfiability, they check if the satisfying assignment is consistent with the dependencies of the DQBF; if this is the case, the original DQBF is satisfied. Otherwise, the instantiation is refined using the inconsistent assignment, and the check is repeated. It can be shown that this process finally terminates.

3.4 Fork Resolution

In [32] information fork splitting is proposed as a basic concept for DQBF solving. Information forks are clauses C which can be split into two parts C_1 and C_2 that depend on incomparable dependency sets D_1 and D_2. After splitting C into C_1 and C_2, a fresh existential variable y_j depending on $D_1 \cap D_2$ is introduced, y_j is added to C_1, $\neg y_j$ to C_2. [32] proves that information fork splitting together with resolution and universal reduction forms a sound and complete proof calculus. The proof idea for the completeness is based on the observation that existential variables with incomparable dependency sets occurring in a single clause impede variable elimination by resolution. So information fork splitting is done before. To the best of our knowledge no practical implementation of a DQBF solver is available so far which uses the information fork splitting idea.

3.5 Basic Approach Using Universal Expansion

Universal Expansion for QBF Already in the QBF context universal expansion has been used as a basic operation to remove universal quantifiers [2,6]. Universal expansion for QBF is defined as follows:

Definition 5 (Universal Expansion for QBF). *For a QBF* $\psi = \forall X_1 \exists Y_1 \ldots \forall X_k \exists Y_k : \varphi$, universal expansion *of variable* $x_i \in X_\ell$ *(*$1 \leq \ell \leq k$*) is defined by*

$$\forall X_1 \exists Y_1 \ldots \forall X_{\ell-1} \exists Y_{\ell-1} \forall X_\ell \setminus \{x_i\} \exists Y'_\ell \forall X_{\ell+1} \exists Y'_{\ell+1} \ldots \forall X_k \exists Y'_k :$$
$$\varphi[0/x_i][y^0_j/y_j \text{ for all } y_j \in Y_h,\ h \geq \ell] \wedge \varphi[1/x_i][y^1_j/y_j \text{ for all } y_j \in Y_h,\ h \geq \ell]\,.$$

with $Y'_h = \{y_j^0 \,|\, y_j \in Y_h\} \cup \{y_j^1 \,|\, y_j \in Y_h\}$.

Existential variables to the right of x_i (i. e., depending on x_i) result in two copies, all other existential variables are not copied.

Universal Expansion for DQBF Universal expansion can be easily generalized to DQBF which has been observed, e. g., in [10,9,3,17].

Definition 6 (Universal Expansion for DQBF). *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* $Z_{x_i} = \left\{y_j \in V_\psi^\exists \,\middle|\, x_i \in D_{y_j}\right\}$, universal expansion *of variable* $x_i \in V_\psi^\forall$ *is defined by*

$$\left(Q \setminus \left(\{x_i\} \cup \bigcup_{y_j \in Z_{x_i}} \{y_j\}\right)\right) \cup \left\{\exists y_j^b(D_{y_j} \setminus \{x_i\}) \,\middle|\, y_j \in Z_{x_i}, b \in \{0,1\}\right\} : \\ \varphi[0/x_i][y_j^0/y_j \text{ for all } y_j \in Z_{x_i}] \wedge \varphi[1/x_i][y_j^1/y_j \text{ for all } y_j \in Z_{x_i}] \,.$$

As for QBFs universal expansion leads to an equisatisfiable formula.

By universally expanding all universal variables both QBFs and DQBFs can be transformed into a SAT problem with a potential exponential increase in variables. Thus, complete universal expansion followed by SAT solving has a double exponential upper bound for the run-time. The upper bound is suboptimal for QBF which is just PSPACE-complete (and can be solved by a simple search-based algorithm in exponential time), whereas for DQBF (which is NEXPTIME-complete) it is unknown whether there is an algorithm with a better worst-case complexity.

3.6 Transformation into QBF

Due to the high cost of complete universal expansion, the solver HQS [18] eliminates universal variables only until a QBF is obtained, which can be solved by an arbitrary QBF solver then.

Transformation into QBF by Universal Expansion The basic observation underlying the transformation of a DQBF into a QBF is the fact that a DQBF $\psi = \forall x_1 \ldots \forall x_n \exists y_1(D_{y_1}) \ldots \exists y_m(D_{y_m}) : \varphi$ can be written as a QBF if and only if the dependency sets are all comparable, i. e., iff for all $i, j \in \{1, \ldots m\}$ $D_{y_i} \subseteq D_{y_j}$ or $D_{y_j} \subseteq D_{y_i}$ $(\star)$. If this condition $(\star)$ holds, a linear order of the dependency sets w. r. t. $\subseteq$ can be computed. Such a linear order immediately results in the needed QBF prefix.

For each pair of existential variables y_i and y_j with incomparable dependency sets, either the universal variables in $D_{y_i} \setminus D_{y_j}$ or in $D_{y_j} \setminus D_{y_i}$ have to be eliminated. In [18] finding a minimum set $U \subseteq V^\forall$ of elimination variables leading to a QBF prefix is reduced to a MAXSAT problem. (For each universal variable x a variable m_x is created in the MAXSAT solver such that $m_x = 1$ means that x needs to be eliminated. Soft clauses are used to get an assignment

with a minimum number of variables assigned to 1. Hard clauses enforce that for all $y_i, y_j \in V^\exists$, $y_i \neq y_j$, either all variables in $D_{y_i} \setminus D_{y_j}$ or in $D_{y_j} \setminus D_{y_i}$ are eliminated.)

Transformation into QBF by Dependency Elimination In [37] the method of universal expansions turning a DQBF into a QBF is refined by so-called *dependency elimination*. 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.

The basic transformation removing a universal variable x_i from a single dependency set D_{y_j} is based on the following theorem:

Theorem 1 (Dependency Elimination). *Assume ψ is a DQBF as in Definition 2 and, w. l. o. g., $x_1 \in D_{y_1}$. Then ψ is equisatisfiable to:*

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

The following example shows that dependency elimination is able to transform a DQBF into an equisatisfiable QBF with much fewer copies of existential variables than needed for universal expansion:

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 universal expansion of x_1 is performed, $y_1, y_3, \ldots, y_n$ are doubled, creating $n - 1$ additional existential variables. The universal expansion 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. If we eliminate the dependency of y_1 on x_1, e. g., 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, we only have to double y_1 in order to obtain an equisatisfiable QBF.

In order to facilitate the selection of dependencies to eliminate, [37] makes use of the following dependency graph:

Definition 7 (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* [4,11,20]: 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. 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 ([37]). *Let ψ be a DQBF and G_ψ its dependency graph. The graph G_ψ is acyclic iff ψ has 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. Our goal is to find a cost-minimal set of edges such that flipping those edges makes the dependency graph acyclic.

The cost of a flipping set $R \subseteq E_\psi \cap (V_\psi^\forall \times V_\psi^\exists)$ corresponds to the number of existential variables in the formula after the dependencies in R have been eliminated. It is given by $\text{cost}(R) := \sum_{y \in V_\psi^\exists} 2^{|R_y|}$. where for $y \in V_\psi^\exists$ $R_y = \{x \in V_\psi^\forall \mid (x, y) \in R\}$. In spite of this non-linear cost function, the computation of a cost-minimal flipping set can be reduced to *integer linear programming with dynamically added constraints* similar to the so-called cutting plane approach [40]. 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.

Don't-Care Dependencies Moreover, based on research on dependency schemes [38], 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.

3.7 The Role of Preprocessing

Part of the success of SAT and QBF solving is due to efficient preprocessing of the formula under consideration. The goal of preprocessing is to simplify the formula by reducing/modifying the number of variables, clauses and quantifier alternations, such that it can be solved more efficiently afterwards. For SAT and QBF, efficient and effective preprocessing tools are available like SatELite [13], Coprocessor [27] for SAT and squeezeBF [19], bloqqer [7] for QBF. In [36] we demonstrated that preprocessing is an essential step for DQBF solving as well. Standard preprocessing techniques were generalized and adapted to work with a DQBF solver core. Those techniques include

- using backbones, monotonic variables, and equivalent literals;
- reducing dependency sets based on dependency schemes [38];
- universal reduction, variable elimination by resolution, universal expansion as preprocessing steps;
- blocked clause elimination with hidden and covered literal addition; and
- structure extraction that leads to circuit representations of the matrix instead of a CNF representation.

An important observation made in [36] is that different preprocessing strategies are advisable depending on the DQBF solver core used (e. g., CNF-based vs. circuit-based solvers).

3.8 Computing Skolem Functions

Computing Skolem functions is important both for proof checking of satisfied DQBFs and for various applications such as the ones mentioned in the next section. In [35] we demonstrated how Skolem functions can be obtained from our DQBF solver HQS. The approach computes Skolem functions for the original formula, even taking all preprocessing steps from [36] into account. The computation of Skolem functions can be done with very little overhead compared to the mere solution of the formula.

4 Applications

Here we give three applications that can be formulated as DQBF problems in a natural way. For the first and the third one we can even prove that they are *equivalent* to DQBF which means that each DQBF problem can be translated into the corresponding problem class in polynomial time. Moreover, this means for those applications that they are NEXPTIME-complete as well [17,39].

In all applications mentioned below the translation into DQBF is based on an observation that has been summarized by Rabe [32] as 'DQBF can encode existential quantification over functions'.

4.1 Partial Equivalence Checking for Combinational Circuits

As a first application we look into partial equivalence checking for combinational circuits [33]. As a specification we consider a complete combinational circuit C_{spec}. As an implementation we consider an *incomplete* combinational circuit C_{impl} containing missing parts, so-called 'Black Boxes'. Missing parts may result from abstraction or they are not yet implemented so far. For each Black Box only the interface of the Black Boxes, i. e., their input and output signals, are known, their functionality is completely unknown. The Partial Equivalence Checking (PEC) problem answers the following question:

Definition 8 (*Partial Equivalence Checking Problem (PEC)*). *Given an incomplete circuit C_{impl} and a (complete) specification C_{spec}, are there implementations of the Black Boxes in C_{impl} such that C_{impl} and C_{spec} become equivalent (i. e., they implement the same Boolean function)?*

Assume that the specification C_{spec} implements a Boolean function $\boldsymbol{f}_{\text{spec}}(\boldsymbol{x})$ with primary input variables $\boldsymbol{x}$. For each Black Box BB_i the input signals are denoted by $\boldsymbol{I}_i$, its output signals by $\boldsymbol{O}_i$. Let us further assume that the Black Boxes can be sorted topologically (otherwise there are replacements leading to

cycles in the combinational circuit), w. l. o. g. in the order $\mathrm{BB}_1, \ldots, \mathrm{BB}_n$. Then the input cone computing the input signals $\boldsymbol{I}_i$ of BB_i represents a vector of Boolean functions $\boldsymbol{F}_i(\boldsymbol{x}, \boldsymbol{O}_1, \ldots, \boldsymbol{O}_{i-1})$. The incomplete implementation C_{impl} implements a Boolean function $\boldsymbol{f}_{\mathrm{impl}}(\boldsymbol{x}, \boldsymbol{O}_1, \ldots, \boldsymbol{O}_n)$ depending on the primary inputs and the Black Box outputs.

The following DQBF is satisfied iff there is an appropriate implementation of the Black Boxes:

$$\forall \boldsymbol{x} \forall \boldsymbol{I}_1 \ldots \forall \boldsymbol{I}_n \exists \boldsymbol{O}_1(\boldsymbol{I}_1) \ldots \exists \boldsymbol{O}_m(\boldsymbol{I}_m) \, : \left(\bigwedge_{i=1}^{n} \boldsymbol{I}_i \equiv \boldsymbol{F}_i \right) \Rightarrow (\boldsymbol{f}_{\mathrm{spec}} \equiv \boldsymbol{f}_{\mathrm{impl}}) \, .$$

We have to ask that for all valuations of the primary inputs $\boldsymbol{x}$ and all input signals of the Black Boxes $\boldsymbol{I}_1, \ldots, \boldsymbol{I}_n$ of the Black Boxes there is a choice for the output signals $\boldsymbol{O}_1, \ldots, \boldsymbol{O}_n$ of the Black Boxes such that specification and implementation are equivalent, i. e., $\boldsymbol{f}_{\mathrm{spec}}(\boldsymbol{x}) \equiv \boldsymbol{f}_{\mathrm{impl}}(\boldsymbol{x}, \boldsymbol{O}_1, \ldots, \boldsymbol{O}_n)$. However, this is only required for all valuations to the signals that are *consistent with the given circuit*, i. e., only if $\bigwedge_{i=1}^{n} \boldsymbol{I}_i \equiv \boldsymbol{F}_i(\boldsymbol{x}, \boldsymbol{O}_1, \ldots, \boldsymbol{O}_{i-1})$ holds. The requirement that the Black Box output signals are only allowed to depend on the Black Box input signals is simply expressed by the dependency sets $\boldsymbol{I}_i$ of the corresponding output signals $\boldsymbol{O}_i$.

4.2 Controller Synthesis

In [8] SAT- and QBF-based techniques for controller synthesis of safety specifications are considered. A footnote in [8] gives a hint how a simple and elegant DQBF formulation can be used for that purpose as well.

In the controller synthesis problem a sequential circuit with a vector of present state bits $\boldsymbol{s}$ and a vector of next state bits $\boldsymbol{s}'$ is considered. The next state is computed by a transition function $\boldsymbol{\Lambda}(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{c})$. Here $\boldsymbol{x}$ are uncontrollable primary inputs and $\boldsymbol{c}$ are controllable inputs which are computed by a controller on the basis of the present state bits and the uncontrollable primary inputs. We consider invariant properties $\mathrm{inv}(\boldsymbol{s}, \boldsymbol{x})$ which are required to hold at any time. The controller synthesis problem asks whether there is an implementation of the controller such that the resulting sequential circuit satisfies the invariant $\mathrm{inv}(\boldsymbol{s}, \boldsymbol{x})$ in all states that are reachable from the circuit's initial state(s), given as a predicate init.

The DQBF formulation of controller synthesis is based on the notion of a 'winning set'.

Definition 9. *Let S be the state set of the sequential circuit. A subset $W \subseteq S$ is a* winning set *if all states in W satisfy the invariant and, for all values of the primary inputs, the controller can ensure (by computing appropriate values for the controlled inputs) that the successor state is again in W.*

An appropriate controller can be found iff there is a winning set that includes the initial states of the sequential circuit. This can be formulated as a DQBF. To encode the winning sets, we introduce two existential variables w and w'; w

depends on the current state and is supposed to be true for a state $\boldsymbol{s}$ if $\boldsymbol{s}$ is in the winning set. The variable w' depends on the next state variables $\boldsymbol{s}'$ and has the same Skolem function as w (but defined over $\boldsymbol{s}'$ instead of $\boldsymbol{s}$). To ensure that w and w' have the same semantics the condition $\big(\boldsymbol{s} \equiv \boldsymbol{s}' \Rightarrow w \equiv w'\big)$ is used. Using those two encodings of the winning set the controller synthesis problem is reduced to the following DQBF [8]:

$$\begin{aligned}\forall \boldsymbol{s}\, \forall \boldsymbol{s}' \forall \boldsymbol{x}\, \exists w(\boldsymbol{s})\ \exists w'(\boldsymbol{s}') \exists \boldsymbol{c}(\boldsymbol{s}, \boldsymbol{x}) :\\ \big(\mathrm{init}(\boldsymbol{s}) \Rightarrow w\big) \wedge \big(w \Rightarrow \mathrm{inv}(\boldsymbol{s}, \boldsymbol{x})\big) \wedge \big(\boldsymbol{s} \equiv \boldsymbol{s}' \Rightarrow w \equiv w'\big) \wedge \\ \Big(\big(w \wedge (\boldsymbol{s}' \equiv \boldsymbol{\Lambda}(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{c}))\big) \Rightarrow w'\Big). \quad (1)\end{aligned}$$

The controlled input variables $\boldsymbol{c}$ are allowed to depend on the current state variables $\boldsymbol{s}$ and uncontrolled inputs $\boldsymbol{x}$ only. If the DQBF is satisfied, then the Skolem functions for $\boldsymbol{c}$ provide a suitable controller implementation. (Note that the solver HQS can compute Skolem functions with very little overhead compared to the mere solution of the formula [35].)

4.3 Realizability Checking for Sequential Circuits

The controller synthesis problem can be seen as a special sequential problem with the controller as a single Black Boxes having access to all state bits and all primary circuit inputs. Here we look into a generalization where sequential circuits may contain an arbitrary number of Black Boxes and the exact interface of the Black Boxes, i. e., the signals entering and leaving the Black Boxes, is strictly taken into account [39]. That means that Black Boxes are *not* necessarily able to read all primary inputs and state bits. We confine ourselves to combinational Black Boxes or Black Boxes with bounded memory. The even more general problem considering distributed architectures containing several Black Boxes with *unbounded* memory is undecidable [31].

Black Boxes with *bounded* memory can be reduced to combinational Black Boxes, simply by extracting the memory elements out of the Black Box into the known part of the circuit, such that the incoming and outgoing signals of these memory elements are written and read only by the Black Boxes. Thus, we assume w. l. o. g. sequential circuits with arbitrary combinational Black Boxes in the circuit implementing their transition function.

As in Sect. 4.1, we assume n Black Boxes $\mathrm{BB}_1, \ldots, \mathrm{BB}_n$ with input signals $\boldsymbol{I}_i$ and output signals $\boldsymbol{O}_i$, respectively. Again, the input cone computing the input signals $\boldsymbol{I}_i$ of BB_i represents a vector of Boolean functions $\boldsymbol{F}_i(\boldsymbol{x}, \boldsymbol{O}_1, \ldots, \boldsymbol{O}_{i-1})$. The transition function depending on the current state variables $\boldsymbol{s}$, the primary inputs $\boldsymbol{x}$ and the Black Box outputs $\boldsymbol{O}_1, \ldots, \boldsymbol{O}_n$ is given by $\boldsymbol{\Lambda}(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{O}_1, \ldots, \boldsymbol{O}_n)$. As before, the transition function computes new valuations to the next state variables $\boldsymbol{s}'$.

We investigate the following problem:

Definition 10. *The* realizability problem for incomplete sequential circuits (RISC) *is defined as follows: Given an incomplete sequential circuit with multiple*

combinational (or bounded-memory) Black Boxes and an invariant property, are there implementations of the Black Boxes such that in the complete circuit the invariant holds at all times?

In order to formulate the realizability problem as a DQBF problem, we slightly modify Definition 9 into:

Definition 11. *A subset $W \subseteq S$ is a* winning set *if all states in W satisfy the invariant and, for all values of the primary inputs, the Black Boxes can ensure (by computing appropriate values) that the successor state is again in W.*

Similarly to the controller synthesis problem, a given RISC is realizable iff there is a winning set that includes the initial states of the circuit. This leads us to the following theorem (using the same encoding of the winning set by existential variables w and w' depending on the current state variables $\boldsymbol{s}$ and next state variables $\boldsymbol{s}'$, respectively):

Theorem 3. *Given a RISC as defined above, the following DQBF is satisfied if and only if the RISC is realizable:*

$$\forall \boldsymbol{s}\, \forall \boldsymbol{s}' \forall \boldsymbol{x}\, \forall \boldsymbol{I}_1 \ldots \forall \boldsymbol{I}_n\ \exists w(\boldsymbol{s})\ \exists w'(\boldsymbol{s}')\ \exists \boldsymbol{O}_1(\boldsymbol{I}_1) \ldots \exists \boldsymbol{O}_n(\boldsymbol{I}_n)\ :$$
$$\big(\mathrm{init}(\boldsymbol{s}) \Rightarrow w\big) \wedge \big(w \Rightarrow \mathrm{inv}(\boldsymbol{s}, \boldsymbol{x})\big) \wedge \big(\boldsymbol{s} \equiv \boldsymbol{s}' \Rightarrow w \equiv w'\big) \wedge$$
$$\Bigg(\Bigg(w \wedge \Bigg[\bigwedge_{i=1}^{n} \boldsymbol{I}_i \equiv \boldsymbol{F}_i(\boldsymbol{x}, \boldsymbol{O}_1, \ldots, \boldsymbol{O}_{i-1})\Bigg] \wedge (\boldsymbol{s}' \equiv \boldsymbol{\Lambda}(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{O}_1, \ldots, \boldsymbol{O}_n))\Bigg) \Rightarrow w'\Bigg).$$

The main difference to (1) consists in the following fact: The requirement that the successor state of a winning state is again a winning state obtains an additional precondition (similar to the DQBF for PEC in Sect. 4.1). The requirement is only needed for signal assignments that are completely consistent with the circuit functionality, i. e., only if the Black Box inputs are assigned consistently with the values computed by their input cones and, of course, the next state variables $\boldsymbol{s}'$ are assigned in accordance with the transition function $\boldsymbol{\Lambda}$.

The Black Box outputs $\boldsymbol{O}_i$ of Black Box BB_i are only allowed to depend on the Black Box inputs $\boldsymbol{I}_i$ and, if the DQBF is satisfied, the Skolem functions for $\boldsymbol{O}_i$ provide an appropriate implementation for BB_i.

5 Conclusion and Future Challenges

Dependency quantified Boolean formulas are a powerful formalism for a natural and compact description of various problems. In this paper, we provided an overview of several solution methods for DQBFs.

In the future, the scalability of the solvers has to be further improved and they might be tuned towards specific applications. Further optimizing the single solution methods as well as combining advantages of different solution strategies seems to be an interesting and rewarding task. This should be combined with more powerful preprocessing techniques as well. Moreover, it will be interesting

in the future to look into sound but incomplete approximations both disproving *and* proving the satisfiability of DQBFs.

We hope that with the availability of solvers more applications of these techniques will become feasible or will be newly discovered, thereby inspiring further improvements of the solvers – just as it is/was the case for propositional SAT solving and for QBF solving.

Acknowlegment

We are grateful to Bernd Becker, Ruben Becker, Andreas Karrenbauer, Jennifer Nist, Sven Reimer, Matthias Sauer, and Karina Wimmer for heavily contributing to the contents summarized in this paper.

References

1. Abramovici, M., Breuer, M.A., Friedman, A.D.: Digital systems testing and testable design. Computer Science Press (1990)
2. Ayari, A., Basin, D.A.: QUBOS: deciding quantified Boolean logic using propositional satisfiability solvers. In: FMCAD. LNCS, vol. 2517, pp. 187–201. Springer (Nov 2002). https://doi.org/10.1007/3-540-36126-x_12
3. Balabanov, V., Chiang, H.K., Jiang, J.R.: Henkin quantifiers and Boolean formulae: A certification perspective of DQBF. Theoretical Computer Science **523**, 86–100 (2014). https://doi.org/10.1016/j.tcs.2013.12.020
4. Beineke, L.W., Little, C.H.C.: Cycles in bipartite tournaments. J. Comb. Theory, Ser. B **32**(2), 140–145 (1982). https://doi.org/10.1016/0095-8956(82)90029-6
5. 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, Bordeaux, France (Jul 2016). https://doi.org/10.1007/978-3-319-40970-2_30
6. Biere, A.: Resolve and expand. In: Hoos, H.H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 59–70. Springer, Vancouver, BC, Canada (May 2004). https://doi.org/10.1007/11527695_5
7. Biere, A., Lonsing, F., Seidl, M.: Blocked clause elimination for QBF. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS, vol. 6803, pp. 101–115. Springer (2011). https://doi.org/10.1007/978-3-642-22438-6_10
8. 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, San Diego, CA, USA (Jan 2014). https://doi.org/10.1007/978-3-642-54013-4_1
9. Bubeck, U.: Model-based transformations for quantified Boolean formulas. Ph.D. thesis, University of Paderborn, Germany (2010)
10. Bubeck, U., Kleine Büning, H.: Dependency quantified Horn formulas: Models and complexity. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 198–211. Springer, Seattle, WA, USA (Aug 2006). https://doi.org/10.1007/11814948_21
11. Cai, M., Deng, X., Zang, W.: A min-max theorem on feedback vertex sets. Mathematics of Operations Research **27**(2), 361–371 (2002). https://doi.org/10.1287/moor.27.2.361.328

12. Chatterjee, K., Henzinger, T.A., Otop, J., Pavlogiannis, A.: Distributed synthesis for LTL fragments. In: FMCAD 2013. pp. 18–25. IEEE (Oct 2013). https://doi.org/10.1109/FMCAD.2013.6679386
13. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 61–75. Springer, St. Andrews, UK (Jun 2005). https://doi.org/10.1007/11499107_5
14. Finkbeiner, B., Tentrup, L.: Fast DQBF refutation. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 243–251. Springer, Vienna, Austria (Jul 2014). https://doi.org/10.1007/978-3-319-09284-3_19
15. Fröhlich, A., Kovásznai, G., Biere, A.: A DPLL algorithm for solving DQBF. In: Int'l Workshop on Pragmatics of SAT (POS) 2012. Trento, Italy (2012)
16. Fröhlich, A., Kovásznai, G., Biere, A., Veith, H.: iDQ: Instantiation-based DQBF solving. In: Le Berre, D. (ed.) Int'l Workshop on Pragmatics of SAT (POS) 2014. EPiC Series, vol. 27, pp. 103–116. EasyChair, Vienna, Austria (Jul 2014)
17. 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. pp. 396–403. IEEE CS, Asheville, NC, USA (Oct 2013). https://doi.org/10.1109/ICCD.2013.6657071
18. Gitina, K., Wimmer, R., Reimer, S., Sauer, M., Scholl, C., Becker, B.: Solving DQBF through quantifier elimination. In: DATE 2015. IEEE, Grenoble, France (Mar 2015). https://doi.org/10.7873/date.2015.0098
19. Giunchiglia, E., Marin, P., Narizzano, M.: sQueezeBF: An effective preprocessor for QBFs based on equivalence reasoning. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 85–98. Springer, Edinburgh, UK (Jul 2010). https://doi.org/10.1007/978-3-642-14186-7_9
20. Guo, J., Hüffner, F., Moser, H.: Feedback arc set in bipartite tournaments is NP-complete. Information Processing Letters **102**(2-3), 62–65 (2007). https://doi.org/10.1016/j.ipl.2006.11.016
21. Henkin, L.: Some remarks on infinitely long formulas. In: Infinitistic Methods: Proc. of the 1959 Symp. on Foundations of Mathematics. pp. 167–183. Pergamon Press, Warsaw, Panstwowe (Sep 1961)
22. Jain, A., Boppana, V., Mukherjee, R., Jain, J., Fujita, M., Hsiao, M.S.: Testing, verification, and diagnosis in the presence of unknowns. In: IEEE VLSI Test Symposium (VTS) 2000. pp. 263–270. IEEE Computer Society, Montreal, Canada (2000). https://doi.org/10.1109/VTEST.2000.843854
23. Janota, M., Klieber, W., Marques-Silva, J., Clarke, E.M.: Solving QBF with counterexample guided refinement. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 114–128. Springer, Trento, Italy (Jun 2012). https://doi.org/10.1007/978-3-642-31612-8_10
24. Janota, M., Marques-Silva, J.: Solving QBF by clause selection. In: Yang, Q., Wooldridge, M. (eds.) IJCAI 2015. pp. 325–331. AAAI Press, Buenos Aires, Argentina (2015), `http://ijcai.org/Abstract/15/052`
25. Lonsing, F., Biere, A.: DepQBF: A dependency-aware QBF solver. Journal on Satisfiability, Boolean Modelling and Computation **7**(2-3), 71–76 (2010)
26. Lonsing, F., Egly, U.: Incremental QBF solving by DepQBF. In: Hong, H., Yap, C. (eds.) Int'l Congress on Mathematical Software (ICMS) 2014. LNCS, vol. 8592, pp. 307–314. Springer, Seoul, South Korea (Aug 2014). https://doi.org/10.1007/978-3-662-44199-2_48
27. Manthey, N.: Coprocessor 2.0 – A flexible CNF simplifier – (tool presentation). In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 436–441. Springer, Trento, Italy (Jun 2012). https://doi.org/10.1007/978-3-642-31612-8_34

28. Meyer, A.R., Stockmeyer, L.J.: Word problems requiring exponential time: Preliminary report. In: STOC 1973. pp. 1–9. ACM Press (1973). https://doi.org/10.1145/800125.804029
29. Peterson, G., Reif, J., Azhar, S.: Lower bounds for multiplayer non-cooperative games of incomplete information. Computers & Mathematics with Applications **41**(7–8), 957–992 (Apr 2001). https://doi.org/10.1016/S0898-1221(00)00333-3
30. Peterson, G.L., Reif, J.H.: Multiple-person alternation. In: Annual Symp. on Foundations of Computer Science (FOCS). pp. 348–363. IEEE Computer Society, San Juan, Puerto Rico (Oct 1979). https://doi.org/10.1109/SFCS.1979.25
31. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Annual Symp. on Foundations of Computer Science 1990. pp. 746–757. IEEE Computer Society, St. Louis, Missouri, USA (Oct 1990). https://doi.org/10.1109/FSCS.1990.89597
32. Rabe, M.N.: A resolution-style proof system for DQBF. In: Gaspers, S., Walsh, T. (eds.) SAT. LNCS, vol. 10491, pp. 314–325. Springer, Melbourne, VIC, Australia (2017). https://doi.org/10.1007/978-3-319-66263-3_20
33. Scholl, C., Becker, B.: Checking equivalence for partial implementations. In: DAC 2001. pp. 238–243. ACM Press, Las Vegas, NV, USA (Jun 2001). https://doi.org/10.1145/378239.378471
34. Silva, J.P.M., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Transactions on Computers **48**(5), 506–521 (1999). https://doi.org/10.1109/12.769433
35. 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, Chiba, Japan (Oct 2016). https://doi.org/10.1007/978-3-319-46520-3_25
36. 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, Austin, TX, USA (Sep 2015). https://doi.org/10.1007/978-3-319-24318-4_13
37. Wimmer, R., Karrenbauer, A., Becker, R., Scholl, C., Becker, B.: From DQBF to QBF by dependency elimination. In: SAT 2017. LNCS, vol. 10491, pp. 326–343. Springer, Melbourne, VIC, Australia (Aug 2017). https://doi.org/10.1007/978-3-319-66263-3_21
38. 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, Bordeaux, France (Jul 2016). https://doi.org/10.1007/978-3-319-40970-2_29
39. Wimmer, R., Wimmer, K., Scholl, C., Becker, B.: Analysis of incomplete circuits using dependency quantified Boolean formulas. In: Reis, A.I., Drechsler, R. (eds.) Advanced Logic Synthesis, pp. 151–168. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-67295-3_7
40. Wolsey, L.A.: Integer programming. Wiley-Interscience, New York, NY, USA (1998)