Preprint from Proceedings of Formal Methods in Computer Aided Design, October 2019, San Jose, USA
(Best paper award)

Localizing Quantifiers for DQBF*

Aile Ge-Ernst[1], Christoph Scholl[1], Ralf Wimmer[1,2]
[1]Institute of Computer Science, Albert-Ludwigs-Universität Freiburg, Germany
[2]Concept Engineering GmbH, Freiburg im Breisgau, Germany
{geernsta, scholl, wimmer}@informatik.uni-freiburg.de

***Abstract*—Dependency quantified Boolean formulas (DQBFs) are a powerful formalism, which subsumes quantified Boolean formulas (QBFs) and allows an explicit specification of dependencies of existential variables on universal variables. Driven by the needs of various applications that can be encoded by DQBFs in a natural, compact, and elegant way, research on DQBF solving has emerged in the past few years. However, most works focus on closed DQBFs in prenex form (where all quantifiers are placed in front of a propositional formula), and non-prenex DQBFs have almost not been studied in the literature. In this paper we provide a formal definition for syntax and semantics of non-closed non-prenex DQBFs and prove useful properties enabling quantifier localization. Moreover, we make use of our theory by integrating quantifier localization into a state-of-the-art DQBF solver. Experiments with prenex DQBF benchmarks, including those from the QBFEVAL'18 competition, clearly show that quantifier localization pays off in this context.**

I. Introduction

During the last two decades enormous progress in the solution of quantifier-free Boolean formulas (SAT) has been observed. Nowadays, SAT solving is successfully used in many applications, e. g., in planning [1], automatic test pattern generation [2], [3], and formal verification of hard- and software systems [4], [5], [6]. Motivated by the success of SAT solvers, efforts have been made, e. g., [7], [8], [9], [10], to consider the more general formalism of quantified Boolean formulas (QBFs).

Although QBFs are capable of encoding decision problems in the PSPACE complexity class, they are not powerful enough to succinctly encode many natural and practical problems that involve decisions under partial information. For example, the analysis of games with incomplete information [11], topologically constrained synthesis of logic circuits [12], synthesis of safe controllers [13], synthesis of fragments of linear-time temporal logic (LTL) [14], and verification of partial designs [15], [16] fall into this category and require an even more general formalism, which is known as *dependency quantified Boolean formulas (DQBFs)* [11].

Unlike QBFs, where an existential variable implicitly depends on all the universal variables preceding its quantification level, DQBFs admit that the dependency sets are explicitly specified. Essentially, existential quantifiers that are specified in this way correspond to Henkin quantifiers [17]. The semantics of a DQBF can be interpreted from a game-theoretic viewpoint as a game played by one universal player and multiple non-cooperative existential players with incomplete information, each partially observing the moves of the universal player as specified by his/her own dependency set. A DQBF is true if and only if the existential players have winning strategies. This flexibility in modeling dependencies allows DQBF encodings to be exponentially more compact than their equivalent QBF counterparts. In contrast to the PSPACE-completeness of QBF, the decision problem of DQBF is NEXPTIME-complete [11].

Driven by the needs of the applications mentioned above, research on DQBF solving has emerged in the past few years, leading to solvers such as iDQ [18], dCAQE [19], and HQS [20], [21], [22].

As an example for a DQBF, consider the formula $\forall x_1 \forall x_2 \exists y_1(x_1) \exists y_2(x_2) : (x_1 \wedge x_2) \equiv (y_1 \equiv y_2)$. This DQBF asks whether there are choices for y_1 only depending on the value of x_1, denoted $\exists y_1(x_1)$, and for y_2 only depending on x_2, denoted $\exists y_2(x_2)$, such that the Boolean formula after the quantifier prefix evaluates to true for all assignments to x_1 and x_2. The Boolean formula in turn states that the existential variables y_1 and y_2 have to be equal iff x_1 and x_2 are true. Since y_1 can only 'see' x_1 and y_2 only x_2, y_1 and y_2 'cannot coordinate' to satisfy the constraint. Thus, the formula is false. Note that the dependencies mentioned above cannot be expressed by a QBF formula, since existential variables in a QBF must depend on *all* universal variables left to it in the prefix. An example for a DQBF in the context of PEC can be found in [16].

So far, syntax and semantics of DQBFs have been defined only for closed prenex forms (see for instance [12]), i. e., for DQBFs where all quantifiers are placed in front of the matrix and all variables occurring in the matrix are either universally or existentially quantified. In this paper, we consider quantifier localization for DQBF, which transforms prenex DQBFs into non-prenex DQBFs for more efficient DQBF solving.

Quantifier localization for *QBF* has been used with great success for image and preimage computations in the context of sequential equivalence checking and symbolic model checking where it has been called "early quantification". Here existential quantifiers were moved over *AND* operations [23], [24], [25], [26]. Benedetti [27] considers quantifier localization for QBFs where the matrix is restricted to conjunctive normal form (CNF). He moves universal and existential quantifiers over *AND* operations and proposes a method to construct a tree-shaped quantifier structure from a QBF instance with linear quantifier prefix. Moreover, Benedetti shows how to benefit from this structure in the QBF solving phase. This work has been used and generalized in [28] for a QBF solver based on symbolic quantifier elimination.

To the best of our knowledge, quantifier localization has not been considered for DQBF so far, apart from the seminal theoretical work on DQBF by Balabanov et al. [12], which considers – as a side remark – quantifier localization for DQBF, transforming prenex DQBFs into non-prenex DQBFs. For

*This work was partly supported by the German Research Council (DFG) as part of the project "Solving Dependency Quantified Boolean Formulas" (278046454).

quantifier localization they gave two propositions. However, a formal definition of the semantics of non-prenex DQBFs was missing in that work and, in addition, the two propositions are not sound, as we will show in our paper.

In this paper, we provide a formal definition of syntax and semantics of non-prenex non-closed DQBFs. The semantics is based on Skolem functions and is a natural generalization of the semantics for closed prenex DQBFs known from the literature. We introduce an alternative constructive definition of the semantics and show that both semantics are equivalent. Then we define rules for transforming DQBFs into equivalent or equisatisfiable DQBFs, which enable the translation of prenex DQBFs into non-prenex DQBFs. The rules are similar to their QBF counterparts, but it turns out that some of them need additional conditions for being sound for DQBF as well. Moreover, the proof techniques are completely different from those for their corresponding QBF counterparts. We provide proofs for all the rules.[1] Finally, we show a method that transforms a prenex DQBF into a non-prenex DQBF based on those rules. It is inspired by the method constructing a tree-shaped quantifier structure from [27] and works for DQBFs with an arbitrary formula (circuit) structure for the matrix. The approach tries to push quantifiers "as deep into the formula" as possible. Whenever a sub-formula fulfills conditions that we will specify in Sect. III, it is processed by symbolic quantifier elimination. When traversing the structure back, quantifiers which could not be eliminated are pulled back into the direction of the root. At the end, a prenex DQBF solver is used for the simplified formula. Experimental results demonstrate the benefits of our method applied to a set of 4811 DQBF benchmarks (including QBFEVAL'18 competition benchmarks).

The paper is structured as follows: In Sect. II we provide preliminaries needed to understand the paper, including existing transformation rules for QBFs. Sect. III contains the main conceptual results of the paper whereas Sect. IV shows how to make use of them algorithmically. Sect. V presents experimental results and Sect. VI concludes the paper.

II. Preliminaries

Let φ, κ be quantifier-free Boolean formulas over the set V of variables and $v \in V$. We denote by $\varphi[{}^{\kappa}/_{v}]$ the Boolean formula which results from φ by replacing all occurrences of v (simultaneously) by κ. For a set $V' \subseteq V$ we denote by $\mathbf{A}(V')$ the set of *Boolean assignments* for V', i. e., $\mathbf{A}(V') = \{\mu \,|\, \mu : V' \to \{0,1\}\}$. As usual, for a Boolean assignment $\mu \in \mathbf{A}(V')$ and $V'' \subseteq V'$ we denote the restriction of μ to V'' by $\mu_{|V''}$. For each formula φ over V, a variable assignment $\mu \in \mathbf{A}(V)$ induces a truth value 0 or 1 of φ, which we call $\mu(\varphi)$. If $\mu(\varphi) = 1$ for all $\mu \in \mathbf{A}(V)$, then φ is a *tautology*. In this case we write $\models \varphi$.

A *Boolean function* with the set of input variables V is a mapping $f : \mathbf{A}(V) \to \{0,1\}$. The set of Boolean functions over V is denoted by $\mathbb{F}_V$. The *support* of a function $f \in \mathbb{F}_V$ is denoted by $\mathrm{supp}(f) \subseteq V$. The constant zero and constant one function are $\mathbf{0}$ and $\mathbf{1}$, resp. A quantifier-free Boolean formula φ over V defines a Boolean function $f_\varphi : \mathbf{A}(V) \to \{0,1\}$ by $f_\varphi(\mu) := \mu(\varphi)$. When clear from the context, we do not differentiate between quantifier-free Boolean formulas and the corresponding Boolean functions, e. g., if φ is a Boolean formula representing f_φ, we write $\varphi[{}^{v'}/_{v}]$ for the Boolean function where the input variable v is replaced by a (new) input variable v'.

Now we consider Boolean formulas with quantifiers. The usual definition for a *closed prenex* DQBF is given as follows:

Definition 1 (Closed prenex DQBF)**.** *Let* $V = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$ *be a set of Boolean variables. A* dependency quantified Boolean formula *(DQBF)* ψ *over* V *has the form* $\psi := \forall x_1 \forall x_2 \ldots \forall x_n \exists y_1(D_{y_1}) \exists y_2(D_{y_2}) \ldots \exists y_m(D_{y_m}) : \varphi$ *where* $D_{y_i} \subseteq \{x_1, \ldots, x_n\}$ *for* $i = 1, \ldots, m$ *is the* dependency set *of* y_i*, and* φ *is a quantifier-free Boolean formula over* V*, called the* matrix *of* ψ*.*

We denote the set of universal variables of ψ by $V_\psi^\forall = \{x_1, \ldots, x_n\}$ and its set of existential variables by $V_\psi^\exists = \{y_1, \ldots, y_m\}$. The former part of ψ, $\forall x_1 \forall x_2 \ldots \forall x_n \exists y_1(D_{y_1}) \exists y_2(D_{y_2}) \ldots \exists y_m(D_{y_m})$, is called its prefix. Sometimes we abbreviate this prefix as Q such that $\psi = Q : \varphi$.

The semantics of closed prenex DQBFs is given as follows:

Definition 2 (Semantics of closed prenex DQBF)**.** *Let* ψ *be a DQBF with matrix* φ *as above.* ψ *is* satisfiable *iff there are functions* $s_{y_i} : \mathbf{A}(D_{y_i}) \to \{0,1\}$ *for* $1 \le i \le m$ *such that replacing each* y_i *by (a Boolean formula for)* s_{y_i} *turns* φ *into a tautology. Then the functions* $(s_{y_i})_{i=1,\ldots,m}$ *are called* Skolem functions *for* ψ*.*

A DQBF is a QBF if its dependency sets satisfy certain conditions:

Definition 3 (Closed prenex QBF)**.** *Let* $V = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$ *be a set of Boolean variables. A* quantified Boolean formula (QBF) *(more precisely, a closed QBF in prenex normal form)* ψ *over* V *is given by* $\psi := \forall X_1 \exists Y_1 \ldots \forall X_k \exists Y_k : \varphi$*, where* $k \ge 1$*,* $X_1, \ldots, X_k$ *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*.*

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 $D_{y_i} = \bigcup_{j=1}^{\ell} X_j$ where Y_ℓ is the unique set with $y_i \in Y_\ell$, $1 \le \ell \le k$, $1 \le i \le m$.

Quantifier localization for QBF is based on the following theorem (see, e. g., [27]), which can be used to transform prenex QBFs into equisatisfiable non-prenex QBFs (where the quantifiers are not necessarily placed before the matrix). Two QBFs ψ_1 and ψ_2 are equisatisfiable ($\psi_1 \approx \psi_2$), when ψ_1 is satisfiable iff ψ_2 is satisfiable.

Theorem 1. *Let* $\diamond \in \{\wedge, \vee\}$*, let* $Q \in \{\exists, \forall\}$*,* $\overline{Q} = \exists$*, if* $Q = \forall$ *and* $\overline{Q} = \forall$ *otherwise. Let* V_ψ^{free} *be the set of all variables occurring in* ψ *which are not bound by a quantifier. The*

[1] Some of the more technical proofs are available as a technical report [29].

following holds for all QBFs:

$$\neg(Qx:\psi) \approx \overline{Q}x:(\neg\psi) \tag{1a}$$
$$Qx:\psi \approx \psi, \quad \text{if } x \notin V_\psi^{\text{free}} \tag{1b}$$
$$\forall x:(\psi_1 \wedge \psi_2) \approx (\forall x:\psi_1) \wedge (\forall x:\psi_2) \tag{1c}$$
$$\exists x:(\psi_1 \vee \psi_2) \approx (\exists x:\psi_1) \vee (\exists x:\psi_2) \tag{1d}$$
$$Qx:(\psi_1 \diamond \psi_2) \approx (\psi_1 \diamond (Qx:\psi_2)), \quad \text{if } x \notin V_{\psi_1}^{\text{free}} \tag{1e}$$
$$Qx_1\,Qx_2:\psi \approx Qx_2\,Qx_1:\psi \tag{1f}$$

III. Non-Closed Non-Prenex DQBFs

A. Syntax and Semantics

In this section, we define syntax and semantics of non-prenex DQBFs. Since the syntax definition is recursive, we need non-closed DQBFs as well.

Definition 4 (Syntax)**.** *Let V be a finite set of Boolean variables. Let φ^{-v} result from φ by removing v from the dependency sets of all existential variables in φ.*

The set Φ^{ncnp} of non-closed non-prenex DQBFs in negation normal form *(NNF) over V as well as their existential, universal, and free variables are defined by the following rules. As usual, Φ^{ncnp} is defined to be the smallest set satisfying those rules.*

1) *If $v \in V$, then $v \in \Phi^{\text{ncnp}}$, $V_v^\exists = V_v^\forall = \emptyset$, $V_v^{\text{free}} = \{v\}$.*
2) *If $v \in V$, then $\neg v \in \Phi^{\text{ncnp}}$, $V_{\neg v}^\exists = V_{\neg v}^\forall = \emptyset$, $V_{\neg v}^{\text{free}} = \{v\}$.*
3) *If $\varphi_1 \in \Phi^{\text{ncnp}}$, $\varphi_2 \in \Phi^{\text{ncnp}}$, and $(\star)$, then $\psi = (\varphi_1 \wedge \varphi_2) \in \Phi^{\text{ncnp}}$, $V_\psi^\exists = V_{\varphi_1}^\exists \dot\cup V_{\varphi_2}^\exists$, $V_\psi^\forall = V_{\varphi_1}^\forall \dot\cup V_{\varphi_2}^\forall$, $V_\psi^{\text{free}} = V_{\varphi_1}^{\text{free}} \cup V_{\varphi_2}^{\text{free}}$.*
4) *If $\varphi_1 \in \Phi^{\text{ncnp}}$, $\varphi_2 \in \Phi^{\text{ncnp}}$, and $(\star)$, then $\psi = (\varphi_1 \vee \varphi_2) \in \Phi^{\text{ncnp}}$, $V_\psi^\exists = V_{\varphi_1}^\exists \dot\cup V_{\varphi_2}^\exists$, $V_\psi^\forall = V_{\varphi_1}^\forall \dot\cup V_{\varphi_2}^\forall$, $V_\psi^{\text{free}} = V_{\varphi_1}^{\text{free}} \cup V_{\varphi_2}^{\text{free}}$.*
5) *If $\varphi \in \Phi^{\text{ncnp}}$, $v \in V_\varphi^{\text{free}}$, $D_v \subseteq V \setminus (V_\varphi^\exists \dot\cup V_\varphi^\forall \dot\cup \{v\})$, then $\psi = \exists v(D_v) : \varphi^{-v} \in \Phi^{\text{ncnp}}$, $V_\psi^\exists = V_\varphi^\exists \dot\cup \{v\}$, $V_\psi^\forall = V_\varphi^\forall$, $V_\psi^{\text{free}} = V_\varphi^{\text{free}} \setminus \{v\}$.*
6) *If $\varphi \in \Phi^{\text{ncnp}}$, $v \in V_\varphi^{\text{free}}$, then $\psi = \forall v : \varphi \in \Phi^{\text{ncnp}}$, $V_\psi^\exists = V_\varphi^\exists$, $V_\psi^\forall = V_\varphi^\forall \dot\cup \{v\}$, $V_\psi^{\text{free}} = V_\varphi^{\text{free}} \setminus \{v\}$.*

Here the condition $(\star)$ means $(V_{\varphi_1}^\exists \dot\cup V_{\varphi_1}^\forall) \cap (V_{\varphi_2}^\exists \dot\cup V_{\varphi_2}^\forall) = \emptyset \quad \wedge \quad V_{\varphi_1}^{\text{free}} \cap (V_{\varphi_2}^\exists \dot\cup V_{\varphi_2}^\forall) = \emptyset \quad \wedge \quad V_{\varphi_2}^{\text{free}} \cap (V_{\varphi_1}^\exists \dot\cup V_{\varphi_1}^\forall) = \emptyset$. We set $V_\psi = V_\psi^\exists \dot\cup V_\psi^\forall \dot\cup V_\psi^{\text{free}}$ for $\psi \in \Phi^{\text{ncnp}}$.

Remark 1. *For the sake of simplicity, we assume in Def. 4 that variables are either free or bound by some quantifier, but not both, and that no variable is quantified more than once. Every formula that violates this assumption can easily be brought into the required form by renaming variables. We restrict ourselves to NNF, since prenex DQBFs are not syntactically closed under negation [12]. For closed prenex DQBFs the (quantifier-free) matrix can be simply transformed into NNF by applying De Morgan's rules and omitting double negations (exploiting that $x \equiv \neg\neg x$) at the cost of a linear blow-up of the formula.*

Definition 5 (Skolem Function Candidates)**.** *For a DQBF ψ over variables V_ψ in NNF, we define a* Skolem function candidate *as a mapping from existential and free variables to functions over universal variables $s : V_\psi^{\text{free}} \dot\cup V_\psi^\exists \to \mathbb{F}_{V_\psi^\forall}$ with*

1) *$\text{supp}(s(v)) = \emptyset$ for all $v \in V_\psi^{\text{free}}$, i.e., $s(v) \in \{\mathbf{0}, \mathbf{1}\}$, and*
2) *$\text{supp}(s(v)) \subseteq (D_v \cap V_\psi^\forall)$ for all $v \in V_\psi^\exists$.*

$\mathbb{S}_\psi$ is the set of all such Skolem function candidates.

That means, $\mathbb{S}_\psi$ is the set of all Skolem function candidates satisfying the constraints imposed by the dependency sets of the existential and free variables.

Notation 1. *Given $s \in \mathbb{S}_\psi$ for a DQBF $\psi \in \Phi^{\text{ncnp}}$, we write $s(\psi)$ for the formula that results from ψ by replacing each variable v for which s is defined by $s(v)$ and omitting all quantifiers from ψ, i.e., $s(\psi)$ is a quantifier-free Boolean formula, containing only variables from $V_\psi^\forall$.*

Definition 6 (Semantics of DQBFs in NNF)**.** *Let $\psi \in \Phi^{\text{ncnp}}$. We define the semantics $[\![\psi]\!]$ of ψ as follows: $[\![\psi]\!] := \{s \in \mathbb{S}_\psi \mid \vDash s(\psi)\} = \{s \in \mathbb{S}_\psi \mid \forall \mu \in \mathbf{A}(V_\psi^\forall) : \mu(s(\psi)) = 1\}$. ψ is* satisfiable *if $[\![\psi]\!] \neq \emptyset$; otherwise we call it* unsatisfiable*. The elements of $[\![\psi]\!]$ are called* Skolem functions *for ψ.*

The semantics $[\![\psi]\!]$ of ψ is the subset of $\mathbb{S}_\psi$ such that for all $s \in [\![\psi]\!]$ we have: Replacing each free or existential variable $v \in V_\psi^{\text{free}} \dot\cup V_\psi^\exists$ with a Boolean expression for $s(v)$ turns ψ into a tautology.

Example 1. *Consider the DQBF*

$$\psi := \forall x_1 \forall x_2 : (x_1 \equiv x_2) \vee (\exists y_1(x_2) : (x_1 \not\equiv y_1)).$$

y_1 with dependency set $\{x_2\}$ is the only existential variable in ψ and there are no free variables. Thus $\mathbb{S}_\psi = \{y_1 \mapsto \mathbf{0}, y_1 \mapsto \mathbf{1}, y_1 \mapsto x_2, y_1 \mapsto \neg x_2\}$. It is easy to see that $s = y_1 \mapsto x_2$ is a Skolem function for ψ, since $\vDash s(\psi) = ((x_1 \equiv x_2) \vee (x_1 \not\equiv x_2))$, and that the other Skolem function candidates do not define Skolem functions.

Remark 2. *For closed prenex DQBFs the semantics defined here obviously coincides with the usual semantics as specified in Def. 2 if we transform the (quantifier-free) matrix into NNF first.*

Remark 3. *A (non-prenex) DQBF ψ is a (non-prenex) QBF if every existential variable depends on all universal variables in whose scope it is (and possibly on free variables as well).*

The following theorem provides a constructive characterization of the semantics of a DQBF ψ.

Theorem 2. *The set $[\![\psi]\!]$ for a DQBF ψ over variables V_ψ in NNF can be characterized recursively as follows:*

$$[\![v]\!] = \{s \in \mathbb{S}_v \mid s(v) = \mathbf{1}\} \text{ for } v \in V_\psi, \tag{2a}$$
$$[\![\neg v]\!] = \{s \in \mathbb{S}_{\neg v} \mid s(v) = \mathbf{0}\} \text{ for } v \in V_\psi, \tag{2b}$$
$$[\![\varphi_1 \wedge \varphi_2]\!] = \{s \in \mathbb{S}_\psi \mid s_{|V_{\varphi_1}^{\text{free}} \dot\cup V_{\varphi_1}^\exists} \in [\![\varphi_1]\!] \wedge s_{|V_{\varphi_2}^{\text{free}} \dot\cup V_{\varphi_2}^\exists} \in [\![\varphi_2]\!]\}, \tag{2c}$$
$$[\![\varphi_1 \vee \varphi_2]\!] = \{s \in \mathbb{S}_\psi \mid s_{|V_{\varphi_1}^{\text{free}} \dot\cup V_{\varphi_1}^\exists} \in [\![\varphi_1]\!] \vee s_{|V_{\varphi_2}^{\text{free}} \dot\cup V_{\varphi_2}^\exists} \in [\![\varphi_2]\!]\}, \tag{2d}$$
$$[\![\exists v(D_v) : \varphi^{-v}]\!] = [\![\varphi^{-v}]\!], \tag{2e}$$
$$[\![\forall v : \varphi]\!] = \Big\{t \in \mathbb{S}_\psi \mid \exists s_0, s_1 \in [\![\varphi]\!] : s_0(v) = \mathbf{0} \wedge s_1(v) = \mathbf{1} \wedge \forall w \in V_\psi^{\text{free}} : t(w) = s_0(w) = s_1(w) \wedge \forall w \in V_\psi^\exists, v \notin D_w : t(w) = s_0(w) = s_1(w) \wedge \forall w \in V_\psi^\exists, v \in D_w : t(w) = \text{ITE}(v,\ s_1(w),\ s_0(w))\Big\} \tag{2f}$$

For the proof as well as for the following example, we denote the semantics defined in Def. 6 by $[\![\psi]\!]_D$ (i.e., $[\![\psi]\!]_D = \{s \in \mathbb{S}_\psi \mid \vDash s(\psi)\}$) and the set that is characterized by Thrm. 2 by $[\![\psi]\!]_T$.

Proof: $[\![\psi]\!]_D = [\![\psi]\!]_T$ is shown by induction on the structure of ψ, for details see [29]. ∎

The following example illustrates the recursive characterization of Thrm. 2 (and again the recursive Def. 4).

Example 2. *Let us consider the DQBF ψ from Ex. 1 again. We compute $[\![\psi]\!]_T$ recursively. As an abbreviation for $(\neg x_1 \wedge y_1) \vee (x_1 \wedge \neg y_1)$, $(x_1 \not\equiv y_1)$ is a DQBF based on rules 1–4 of Def. 4 with $V^{\exists}_{x_1 \not\equiv y_1} = V^{\forall}_{x_1 \not\equiv y_1} = \emptyset$, $V^{\text{free}}_{x_1 \not\equiv y_1} = \{x_1, y_1\}$. With Thrm. 2, (2a)–(2d) we get $[\![x_1 \not\equiv y_1]\!]_T = \{s : \{y_1, x_1\} \to \mathbb{F}_\emptyset \mid s(y_1) \neq s(x_1)\}$. For $\psi' = (\exists y_1(x_2) : (x_1 \not\equiv y_1))$, we obtain by rule 5: $V^{\forall}_{\psi'} = \emptyset$, $V^{\exists}_{\psi'} = \{y_1\}$, $V^{\text{free}}_{\psi'} = \{x_1\}$. According to Thrm. 2, (2e) we have $[\![\exists y_1(x_2) : (x_1 \not\equiv y_1)]\!]_T = [\![x_1 \not\equiv y_1]\!]_T$. Similarly we obtain $[\![x_1 \equiv x_2]\!]_T = \{s : \{x_1, x_2\} \to \mathbb{F}_\emptyset \mid s(x_1) = s(x_2)\}$. Then, for $\psi'' = (x_1 \equiv x_2) \vee (\exists y_1(x_2) : (x_1 \not\equiv y_1))$ we have $V^{\forall}_{\psi''} = \emptyset$, $V^{\exists}_{\psi''} = \{y_1\}$, $V^{\text{free}}_{\psi''} = \{x_1, x_2\}$, and by Thrm. 2, (2d) $[\![\psi'']\!]_T = \{s : \{x_1, x_2, y_1\} \to \mathbb{F}_\emptyset \mid \big(s(x_1), s(x_2), s(y_1)\big) \in \{(\mathbf{0},\mathbf{0},\mathbf{0}), (\mathbf{0},\mathbf{0},\mathbf{1}), (\mathbf{0},\mathbf{1},\mathbf{1}), (\mathbf{1},\mathbf{0},\mathbf{0}), (\mathbf{1},\mathbf{1},\mathbf{0}), (\mathbf{1},\mathbf{1},\mathbf{1})\}\}$.*

Now we consider $\forall x_2 : \psi''$. $V^{\forall}_{\forall x_2 : \psi''} = \{x_2\}$. $V^{\exists}_{\forall x_2 : \psi''} = \{y_1\}$, $V^{\text{free}}_{\forall x_2 : \psi''} = \{x_1\}$. We use (2f) to construct $[\![\forall x_2 : \psi'']\!]_T$. In principle, there are 3 possible choices $s_0 \in [\![\psi'']\!]_T$ with $s_0(x_2) = \mathbf{0}$ and 3 possible choices $s_1 \in [\![\psi'']\!]_T$ with $s_1(x_2) = \mathbf{1}$. Due to the constraint $s_0(x_1) = s_1(x_1)$ in the third line of (2f), there remain only 4 possible combinations $s_0^{(1)}, s_1^{(1)}, \ldots, s_0^{(4)}, s_1^{(4)}$:

- $\big(s_0^{(1)}(x_1), s_0^{(1)}(x_2), s_0^{(1)}(y_1)\big) = (\mathbf{0},\mathbf{0},\mathbf{0})$, $\big(s_1^{(1)}(x_1), s_1^{(1)}(x_2), s_1^{(1)}(y_1)\big) = (\mathbf{0},\mathbf{1},\mathbf{1})$, *leading to* $t^{(1)}(x_1) = \mathbf{0}$, $t^{(1)}(y_1) = \text{ITE}(x_2, s_1^{(1)}(y_1), s_0^{(1)}(y_1)) = \text{ITE}(x_2, \mathbf{1}, \mathbf{0}) = x_2$,
- $\big(s_0^{(2)}(x_1), s_0^{(2)}(x_2), s_0^{(2)}(y_1)\big) = (\mathbf{0},\mathbf{0},\mathbf{1})$, $\big(s_1^{(2)}(x_1), s_1^{(2)}(x_2), s_1^{(2)}(y_1)\big) = (\mathbf{0},\mathbf{1},\mathbf{1})$, *leading to* $t^{(2)}(x_1) = \mathbf{0}$, $t^{(2)}(y_1) = \text{ITE}(x_2, s_1^{(2)}(y_1), s_0^{(2)}(y_1)) = \text{ITE}(x_2, \mathbf{1}, \mathbf{1}) = \mathbf{1}$,
- $\big(s_0^{(3)}(x_1), s_0^{(3)}(x_2), s_0^{(3)}(y_1)\big) = (\mathbf{1},\mathbf{0},\mathbf{0})$, $\big(s_1^{(3)}(x_1), s_1^{(3)}(x_2), s_1^{(3)}(y_1)\big) = (\mathbf{1},\mathbf{1},\mathbf{0})$, *leading to* $t^{(3)}(x_1) = \mathbf{1}$, $t^{(3)}(y_1) = \text{ITE}(x_2, s_1^{(3)}(y_1), s_0^{(3)}(y_1)) = \text{ITE}(x_2, \mathbf{0}, \mathbf{0}) = \mathbf{0}$,
- $\big(s_0^{(4)}(x_1), s_0^{(4)}(x_2), s_0^{(4)}(y_1)\big) = (\mathbf{1},\mathbf{0},\mathbf{0})$, $\big(s_1^{(4)}(x_1), s_1^{(4)}(x_2), s_1^{(4)}(y_1)\big) = (\mathbf{1},\mathbf{1},\mathbf{1})$, *leading to* $t^{(4)}(x_1) = \mathbf{1}$, $t^{(4)}(y_1) = \text{ITE}(x_2, s_1^{(4)}(y_1), s_0^{(4)}(y_1)) = \text{ITE}(x_2, \mathbf{1}, \mathbf{0}) = x_2$.

Altogether, $[\![\forall x_2 : \psi'']\!]_T = \{t^{(1)}, t^{(2)}, t^{(3)}, t^{(4)}\}$.

Finally, for $\psi = \forall x_1 \forall x_2 : \psi''$ we have $V^{\forall}_{\psi} = \{x_1, x_2\}$, $V^{\exists}_{\psi} = \{y_1\}$, $V^{\text{free}}_{\psi} = \emptyset$. For the choice of s_0 and s_1 in (2f) we need $s_0(x_1) = \mathbf{0}$, $s_1(x_1) = \mathbf{1}$ and, due to $x_1 \notin D_{y_1}$, $s_0(y_1) = s_1(y_1)$ (see fourth line of (2f)). Thus, the only possible choice is $s_0 = t^{(1)}$ and $s_1 = t^{(4)}$ and $t(y_1) = x_2$ is the only possible Skolem function for ψ. This result agrees with the Skolem function computed using Def. 6 in Ex. 1.

B. Equivalent and Equisatisfiable DQBFs

Now we define rules for replacing DQBFs by equivalent and equisatisfiable ones.

Definition 7 (Equivalence and Equisatisfiability)**.** *Let $\psi_1, \psi_2 \in \Phi^{\text{ncnp}}$. We call them* equivalent *(written $\psi_1 \equiv \psi_2$) if $[\![\psi_1]\!] = [\![\psi_2]\!]$, and* equisatisfiable *(written $\psi_1 \approx \psi_1$) if $[\![\psi_1]\!] = \emptyset \Leftrightarrow [\![\psi_2]\!] = \emptyset$.*

Now we prove Thrm. 3, which is the DQBF counterpart to Thrm. 1 for QBF.

Theorem 3. *Let $\diamond \in \{\wedge, \vee\}$ and $Q \in \{\exists, \forall\}$, $\varphi, \varphi_1, \varphi_2 \in \Phi^{\text{ncnp}}$. We assume that x' and y' are fresh variables, which do not occur in φ, φ_1, and φ_2. The following equivalences and equisatifiabilities hold for all DQBFs in NNF.*

$$\forall x : \varphi \;\approx\; \varphi \quad \text{if } x \notin V_\varphi \tag{3a}$$

$$\forall x : \varphi \;\equiv\; \varphi[^0/_x] \wedge \varphi_2[^1/_x] \quad \text{if } V^{\forall}_\varphi = V^{\exists}_\varphi = \emptyset \tag{3b}$$

$$\forall x : (\varphi_1 \wedge \varphi_2) \;\approx\; (\forall x : \varphi_1) \wedge (\forall x' : \varphi_2[^{x'}/_x])^2 \tag{3c}$$

$$\exists y(D_y) : (\varphi_1 \vee \varphi_2) \;\approx\; (\exists y(D_y) : \varphi_1) \vee (\exists y'(D_y) : \varphi_2[^{y'}/_y]) \tag{3d}$$

$$\forall x : (\varphi_1 \diamond \varphi_2) \;\equiv\; (\varphi_1 \diamond (\forall x : \varphi_2)) \quad \text{if } x \notin V_{\varphi_1} \text{ and } x \notin D_y \text{ for all } y \in V^{\exists}_{\varphi_1} \tag{3e}$$

$$\exists y(D_y) : (\varphi_1 \diamond \varphi_2) \;\equiv\; (\varphi_1 \diamond (\exists y(D_y) : \varphi_2)) \text{ if } y \notin V_{\varphi_1} \tag{3f}$$

$$\exists y_1(D_{y_1}) \exists y_2(D_{y_2}) : \varphi \;\equiv\; \exists y_2(D_{y_2}) \exists y_1(D_{y_1}) : \varphi \tag{3g}$$

$$\forall x_1 \forall x_2 : \varphi \;\equiv\; \forall x_2 \forall x_1 : \varphi \tag{3h}$$

$$\forall x \exists y(D_y) : \varphi \;\equiv\; \exists y(D_y) \forall x : \varphi \quad \text{if } x \notin D_y. \tag{3i}$$

Note that the duality of $\exists$ and $\forall$ under negation as in QBF ($\exists\varphi \equiv \neg\forall\neg\varphi$) does not hold for DQBF as DQBFs are not syntactically closed under negation [12]. Moreover, the existential counterpart of (3a) does not make much sense, since by Thrm. 2 we have $[\![\exists v(D_v) : \varphi^{-v}]\!] = [\![\varphi^{-v}]\!]$.

Example 3. *We give an example that shows that – in contrast to (1e) of Thrm. 1 for QBF – the condition $x \notin D_y$ for all $y \in V^{\exists}_{\varphi_1}$ is really needed in (3e) if $\diamond = \vee$. We consider the satisfiable DQBF $\psi = \forall x_1 \forall x_2 : (x_1 \equiv x_2) \vee (\exists y_1(x_2) : (x_1 \not\equiv y_1))$ from Ex. 1 again. First of all, neglecting the above condition, we could transform ψ into $\psi' = \forall x_1 : \big((\forall x_2 : (x_1 \equiv x_2)) \vee (\exists y_1(x_2) : (x_1 \not\equiv y_1))\big)$ which is not well-formed according to Def. 4. However, by renaming x_2 into x_2' in the dependency set of y_1 we would arrive at a well-formed DQBF $\psi'' = \forall x_1 : \big((\forall x_2 : (x_1 \equiv x_2)) \vee (\exists y_1(x_2') : (x_1 \not\equiv y_1))\big)$. According to Def. 5 the only possible Skolem function candidates for y_1 in ψ'' are $\mathbf{0}$ and $\mathbf{1}$. It is easy to see that neither inserting $\mathbf{0}$ nor $\mathbf{1}$ for y_1 turns ψ'' into a tautology, thus ψ'' is unsatisfiable and therefore not equisatisfiable with ψ.*

Whereas the proof of Thrm. 1 for QBF is rather easy using the equisatisfiabilities $\exists y : \varphi \approx \varphi[^0/_y] \vee \varphi[^1/_y]$ and $\forall x : \varphi \approx \varphi[^0/_x] \wedge \varphi[^1/_x]$, the proof of Thrm. 3 is more involved:

Proof: (3a): Since $x \notin V_\varphi$, $V^{\text{free}}_{\forall x:\varphi} = V^{\text{free}}_\varphi$ and $V^{\exists}_{\forall x:\varphi} = V^{\exists}_\varphi$ and $V^{\forall}_{\forall x:\varphi} = V^{\forall}_\varphi \cup \{x\}$. If $[\![\varphi]\!] \neq \emptyset$, then for each $s \in [\![\varphi]\!]$ with $\vDash s(\varphi)$ we also have $\vDash s(\forall x : \varphi)$. Now assume $[\![\forall x : \varphi]\!] \neq \emptyset$ and $s \in [\![\forall x : \varphi]\!]$. Since $\vDash s(\forall x : \varphi)$, we have $\vDash s(\varphi)$ and $\vDash s(\varphi)[^c/_x]$ for an arbitrary constant $c \in \{0, 1\}$. Since $x \notin V_\varphi$, $s(\varphi)[^c/_x]$ results from $s(\varphi)$ by replacing x in the Skolem functions for existential variables in φ by the constant c. Altogether we have found Skolem functions s' for φ where $s'(v)$ does not depend on x for $v \in V^{\exists}_\varphi$, i. e., $s' \in [\![\varphi]\!]$.

(3b): The statement easily follows from Thrm. 2, (2f) considering that φ contains only free variables. Let $\psi_1 = \forall x : \varphi$, $\psi_2 = \varphi[^0/_x] \wedge \varphi_2[^1/_x]$. We have $\mathbb{S}_{\psi_1} = \mathbb{S}_{\psi_2}$ and $[\![\psi_1]\!] = \big\{t \in \mathbb{S}_{\psi_1} \,\big|\, \exists s_0, s_1 \in [\![\varphi]\!] : s_0(x) = \mathbf{0} \wedge s_1(x) = \mathbf{1} \wedge \forall w \in V^{\text{free}}_{\psi_1} : t(w) := s_0(w) = s_1(w)\big\} = [\![\psi_2]\!]$.

(3c): We set $\psi_1 := \forall x : (\varphi_1 \wedge \varphi_2)$ and $\psi_2 := (\forall x : \varphi_1) \wedge (\forall x' : \varphi_2[^{x'}/_x])$. The proof follows from the fact that, for a Skolem function t, $t(\varphi_1) \wedge t(\varphi_2)$ can only be a tautology if $t(\varphi_1)$ and $t(\varphi_2)$ are tautologies. A detailed proof can be found in [29].

(3d): We set $\psi_1 := \exists y(D_y) : (\varphi_1 \vee \varphi_2)$ and $\psi_2 := (\exists y(D_y) : \varphi_1) \vee (\exists y'(D_y) : \varphi_2[^{y'}/_y])$. This case is analogous to the previous

[2] By $\varphi_2[^{x'}/_x]$ we mean that all occurrences of x are replaced by x', including the occurrences in dependency sets.

case and needs the additional argument that $t(\varphi_1) \vee t(\varphi_2)$ can only be tautology if $t(\varphi_1)$ or $t(\varphi_2)$ is a tautology, because the variables occurring in $t(\varphi_1)$ and $t(\varphi_2)$ are disjoint.

(3e): Let $\psi_1 := \forall x : (\varphi_1 \diamond \varphi_2)$ and $\psi_2 := (\varphi_1 \diamond (\forall x : \varphi_2))$ and assume that $x \notin V_{\varphi_1}$ and $x \notin D_y$ for any $y \in V^{\exists}_{\varphi_1}$. From $x \notin D_y$ for any $y \in V^{\exists}_{\varphi_1}$ we conclude that $\mathbb{S}_{\psi_1} = \mathbb{S}_{\psi_2}$. Then we have: $[\![\psi_1]\!] = \{s \in \mathbb{S}_{\psi_1} \mid \vDash s(\forall x : (\varphi_1 \diamond \varphi_2))\} = \{s \in \mathbb{S}_{\psi_1} \mid \vDash s(\varphi_1 \diamond \varphi_2)\} = \{s \in \mathbb{S}_{\psi_1} \mid \vDash s(\varphi_1 \diamond (\forall x : \varphi_2))\} = \{s \in \mathbb{S}_{\psi_2} \mid \vDash s(\varphi_1 \diamond (\forall x : \varphi_2))\}$, since $\mathbb{S}_{\psi_1} = \mathbb{S}_{\psi_2}$, and finally $[\![\psi_1]\!] = [\![\psi_2]\!]$.

(3f): Let $\psi_1 := \exists y(D_y) : (\varphi_1 \diamond \varphi_2)$ and $\psi_2 := \varphi_1 \diamond (\exists y(D_y) : \varphi_2)$. Note that we need $y \notin V_{\varphi_1}$, since otherwise ψ_2 would not be well-formed according to Def. 4. Then we have:

$[\![\psi_1]\!] = \{s \in \mathbb{S}_{\psi_1} \mid \vDash s(\exists y(D_y) : (\varphi_1 \diamond \varphi_2))\} = \{s \in \mathbb{S}_{\psi_1} \mid \vDash s(\varphi_1 \diamond \varphi_2)\} = \{s \in \mathbb{S}_{\psi_1} \mid \vDash s(\varphi_1 \diamond (\exists y(D_y) : \varphi_2))\} = \{s \in \mathbb{S}_{\psi_2} \mid \vDash s(\varphi_1 \diamond (\exists y(D_y) : \varphi_2))\}$, since $\mathbb{S}_{\psi_1} = \mathbb{S}_{\psi_2}$, and finally $[\![\psi_1]\!] = [\![\psi_2]\!]$.

(3g): By applying Thrm. 2, Eqn. (2e) multiple times, we get: $[\![\exists y_1(D_{y_1}) \exists y_2(D_{y_2}) : \varphi]\!] = [\![\exists y_2(D_{y_2}) : \varphi]\!] = [\![\varphi]\!] = [\![\exists y_1(D_{y_1}) : \varphi]\!] = [\![\exists y_2(D_{y_2}) \exists y_1(D_{y_1}) : \varphi]\!]$.

(3h): We set $\psi_1 := \forall x_1 \forall x_2 : \varphi$ and $\psi_2 := \forall x_2 \forall x_1 : \varphi$. Then we have: $[\![\psi_1]\!] = [\![\forall x_1 \forall x_2 : \varphi]\!] = \{s \in \mathbb{S}_{\psi_1} \mid \vDash s(\forall x_1 \forall x_2 : \varphi)\} = \{s \in \mathbb{S}_{\psi_1} \mid \vDash s(\varphi)\} = \{s \in \mathbb{S}_{\psi_2} \mid \vDash s(\varphi)\}$, since $\mathbb{S}_{\psi_1} = \mathbb{S}_{\psi_2}$, and then $[\![\psi_1]\!] = \{s \in \mathbb{S}_{\psi_2} \mid \vDash s(\forall x_2 \forall x_1 : \varphi)\} = [\![\psi_2]\!]$.

(3i): We set $\psi_1 := \forall x \exists y(D_y) : \varphi$ and $\psi_2 := \exists y(D_y) \forall x : \varphi$.

First note that $\exists y(D_y) \forall x : \varphi$ is not well-formed according to Def. 4 if $x \in D_y$, because x is universal in $\forall x : \varphi$. With $x \notin D_y$ we show that $[\![\psi_1]\!] = [\![\psi_2]\!]$. We have: $[\![\psi_1]\!] = \{s \in \mathbb{S}_{\psi_1} \mid \vDash s(\forall x \exists y(D_y) : \varphi)\} = \{s \in \mathbb{S}_{\psi_1} \mid \vDash s(\varphi)\}$. Because $x \notin D_y$, the Skolem function candidates for y in ψ_1 are restricted to constant functions. The same holds for y in ψ_2. Therefore $\mathbb{S}_{\psi_1} = \mathbb{S}_{\psi_2}$ is true. So we can write: $[\![\psi_1]\!] = \{s \in \mathbb{S}_{\psi_2} \mid \vDash s(\varphi)\} = \{s \in \mathbb{S}_{\psi_2} \mid \vDash s(\exists y(D_y) \forall x : \varphi)\} = [\![\psi_2]\!]$. ∎

Remark 4. *Note that rules (3c) and (3d) would actually establish equivalence instead of equisatisfiability if we would not have decided to forbid in the formal definition (for sake of simplicity) that variables are quantified more than once.*

The next theorem shows that (3e) can be strengthened, if we confine ourselves to $\wedge$ and consider equisatisfiability only:

Theorem 4. *Let $\varphi \in \Phi^{\text{ncnp}}$ be a DQBF and let $\forall x : (\varphi_1 \wedge \varphi_2)$ be a subformula of φ with $x \notin V_{\varphi_1}$. Then $\varphi \approx \varphi'$ where φ' results from φ by replacing the subformula $\forall x : (\varphi_1 \wedge \varphi_2)$ by $(\varphi_1^{-x} \wedge (\forall x : \varphi_2))$.*

Proof: (Sketch) Let $\psi = \forall x : (\varphi_1 \wedge \varphi_2)$ and $\psi' = (\varphi_1^{-x} \wedge (\forall x : \varphi_2))$. Note that we need $x \notin V_{\varphi_1}$, since otherwise ψ' would not be well-formed according to Def. 4. We have to prove equisatisfiability of φ and φ'. It is easy to see that for each Skolem function s' of φ', $\vDash s'(\varphi')$ implies $\vDash s'(\varphi)$. Now assume a Skolem function s of φ, i. e., $\vDash s(\varphi)$ or $\mu(s(\varphi)) = 1$ $\forall \mu \in \mathbf{A}(V^{\forall}_{\varphi})$. We can show by contradiction that s can be transformed into another Skolem function s' by replacing $s(v)$ for all existential variables $v \in V^{\exists}_{\varphi_1}$ with $s'(v) = s(v)[^c/_x]$ with an arbitrary constant $c \in \{0, 1\}$ inserted for x. The proof by contradiction uses the facts that φ is in NNF, i. e., $s(\varphi)$ ($s'(\varphi)$) is a tree of *or* and *and* operations with the inputs replaced by negated or non-negated Skolem functions according to s (s') and thus shows certain monotonicity properties, φ_1 and φ_2 are connected by an $\wedge$-operation (not by $\vee$), x is only contained in the subformula ψ / ψ' of φ, and $\mu(s(\varphi)) = 1$ *for all* $\mu \in \mathbf{A}(V^{\forall}_{\varphi})$. It is easy to see that s' is then a Skolem function for φ'. The detailed proof can be found in [29]. ∎

Finally, we prove a theorem which is needed for our algorithm taking advantage of quantifier localization. It shows that, under certain conditions, we can do symbolic quantifier elimination for non-prenex DQBFs as it is known from QBFs:

Theorem 5. *Let $\varphi_1 \in \Phi^{\text{ncnp}}$ be a DQBF and let $\exists y(D_y) : \varphi_2$ be a subformula of φ_1 such that φ_2 does not include any quantification and includes only variables from $D_y \cup V^{\text{free}}_{\varphi_1} \cup \{v \in V^{\exists}_{\varphi_1} \mid D_v \subseteq D_y\}$. Then $\varphi_1 \approx \varphi_1'$ where φ_1' results from φ_1 by replacing the subformula $\exists y(D_y) : \varphi_2$ by $\varphi_2[^0/_y] \vee \varphi_2[^1/_y]$.*

Proof: (Sketch) We show equisatisfiability by proving that $[\![\varphi_1']\!] \neq \emptyset$ implies $[\![\varphi_1]\!] \neq \emptyset$ and vice versa.

First assume that there is a Skolem function $s' \in [\![\varphi_1']\!]$ with $\vDash s'(\varphi_1')$. We define $s \in \mathbb{S}_{\varphi_1}$ by $s(v) = s'(v)$ for all $v \in V^{\exists}_{\varphi_1'} \cup V^{\text{free}}_{\varphi_1'}$ and $s(y) = s'(\varphi_2[^1/_y])$. The fact that $s \in \mathbb{S}_{\varphi_1}$ follows from the restriction that φ_2 contains only variables from $D_y \cup V^{\text{free}}_{\varphi_1} \cup \{v \in V^{\exists}_{\varphi_1} \mid D_v \subseteq D_y\}$, i. e., $\text{supp}(s(y)) = \text{supp}(s'(\varphi_2[^1/_y])) \subseteq D_y$. $\vDash s(\varphi_1)$ follows by some rewriting from a result in [30] proving that quantifier elimination can be done by composition, i. e., $\varphi_2[^{\varphi_2[^1/_y]}/_y]$ is equivalent to $\varphi_2[^0/_y] \vee \varphi_2[^1/_y]$.

Now assume $s \in [\![\varphi_1]\!]$ with $\vDash s(\varphi_1)$ and define $\tilde{s}$ just by removing y from the domain of s. In a first step we change s into s'' by replacing $s(y)$ with $s''(y) = \tilde{s}(\varphi_2)[^1/_y]$. We conclude $\vDash s''(\varphi_1)$ from [30] and monotonicity properties of φ_1 in negation normal form. In a second step we use [30] again to show that $s''(\varphi_1)$ is equivalent to $\tilde{s}(\varphi_1')$. Thus finally $\vDash \tilde{s}(\varphi_1')$. Again, the detailed proof can be found in [29]. ∎

C. Refuting Propositions 4 and 5 from [12]

A first paper looking into quantifier localization for DQBF was [12]. To this end, they proposed Propositions 4 and 5 which are unfortunately unsound. We literally repeat Proposition 4:

Proposition 4 ([12])**.** *The DQBF $\forall \vec{x} \exists y_1(S_1) \ldots \exists y_m(S_m) : (\phi_A \vee \phi_B)$ where $\forall \vec{x}$ denotes $\forall x_1 \ldots \forall x_n$, sub-formula ϕ_A (respectively ϕ_B) refers to variables $X_A \subseteq X$ and $Y_A \subseteq Y$ (respectively $X_B \subseteq X$ and $Y_B \subseteq Y$), is logically equivalent to*

$$\forall \vec{x}_c((\forall \vec{x}_a \exists y_{a_1}(S_{a_1} \cap X_A) \ldots \exists y_{a_p}(S_{a_p} \cap X_A) : \phi_A) \vee (\forall \vec{x}_b \exists y_{b_1}(S_{b_1} \cap X_B) \ldots \exists y_{b_q}(S_{b_q} \cap X_B) : \phi_B)),$$

where variables $\vec{x}_c$ are in $X_A \cap X_B$, variables $\vec{x}_a$ are in $X_A \setminus X_B$, variables $\vec{x}_b$ are in $X_B \setminus X_A$, $y_{a_i} \in Y_A$, and $y_{b_j} \in Y_B$.

Lemma 1. *Proposition 4 is unsound.*

Proof: Consider the following DQBF

$$\psi^1 := \forall x_1 \forall x_2 \exists y_1(x_2) : \big(\underbrace{(x_1 \equiv x_2)}_{\phi_A} \vee \underbrace{(x_1 \not\equiv y_1)}_{\phi_B}\big)$$

By (3f), ψ^1 is equisatisfiable with ψ from Ex. 1 and thus satisfiable. According to Proposition 4 we can identify the sets $X_A = \{x_1, x_2\}$, $X_B = \{x_1\}$, $Y_A = \emptyset$, and $Y_B = \{y_1\}$. and rewrite the formula to $\psi^2 := \forall x_1 : \big((\forall x_2 : (x_1 \equiv x_2)) \vee (\exists y_1(\emptyset) : (x_1 \not\equiv y_1))\big)$. This formula, in contrast to ψ^1, is unsatisfiable because the only Skolem functions candidates for y_1 are $\mathbf{0}$ and $\mathbf{1}$. Both Skolem function candidates do not turn ψ^2 into a tautology. ∎

In the example from the proof, the "main mistake" was to replace $D_{y_1} = \{x_2\}$ by $\emptyset$. If this *were* correct, then the remainder would follow from (3f) and (3e).

Remark 5. *Proposition 4 of [12] is already unsound when we consider the commonly accepted semantics of closed prenex DQBFs as stated in Def. 2. The proposition claims that ψ^1 is equisatisfiable with ψ^2 . Additionally, it claims that $\psi^3 := \forall x_1 \forall x_2 \exists y_1(\emptyset) : \big((x_1 \equiv x_2) \vee (x_1 \not\equiv y_1)\big)$ is equisatisfiable with ψ^2. Due to transitivity of equisatisfiability, Proposition 5 claims that ψ^1 is equisatisfiable with ψ^3. However, according to the semantics in Def. 2, ψ^1 is satisfiable and ψ^3 unsatisfiable. Also note that ψ^1 and ψ^3 are actually QBFs; so Proposition 4 is also unsound when restricted to QBFs.*

Next we literally repeat Proposition 5 from [12]:

Proposition 5 ([12])**.** *The DQBF $\forall \vec{x} \exists y_1(S_1) \dots \exists y_k(S_k)(\phi_A \wedge \phi_B)$ where $\forall \vec{x}$ denotes $\forall x_1 \dots \forall x_n$, sub-formula ϕ_A (respectively ϕ_B) refers to variables $X_A \subseteq X$ and $Y_A \subseteq Y$ (respectively $X_B \subseteq X$ and $Y_B \subseteq Y$), is logically equivalent to*

$$\forall \vec{x} \exists y_2(S_2) \dots \exists y_k(S_k)((\exists y_1(S_1 \cap X_A)(\phi_A)) \wedge \phi_B)$$

for $y_1 \notin Y_B$.

Lemma 2. *Proposition 5 is unsound.*

Proof: For a counterexample, consider the formula

$$\psi^4 := \forall x_1 \forall x_2 \exists y_1(x_1, x_2) \exists y_2(x_1, x_2) : \underbrace{(y_1 \equiv \neg y_2)}_{\phi_A} \wedge \underbrace{(y_2 \equiv (x_1 \wedge x_2))}_{\phi_B}.$$

with the corresponding variable sets $X_A = \emptyset$, $X_B = \{x_1, x_2\}$, $Y_A = \{y_1, y_2\}$, and $Y_B = \{y_2\}$. We have $y_1 \notin Y_B$ and $\{x_1, x_2\} \cap X_A = \emptyset$. Proposition 5 says that ψ^4 is equisatisfiable with:

$$\psi^5 := \forall x_1 \forall x_2 \exists y_2(x_1, x_2) : \big(\exists y_1(\emptyset) : (y_1 \equiv \neg y_2)\big) \wedge (y_2 \equiv (x_1 \wedge x_2)).$$

The formula ψ^4 is satisfiable; the Skolem function s with $s(y_1) = \neg(x_1 \wedge x_2)$ and $s(y_2) = (x_1 \wedge x_2)$ is in $[\![\psi]\!]$.

The formula ψ^5, however, is unsatisfiable: Since $D^{\psi^5}_{y_1} = \emptyset$, there are only two Skolem function candidates for y_1, either $s(y_1) = \mathbf{0}$ or $s(y_1) = \mathbf{1}$. In the first case, we need to find a function for y_2 such that $(\mathbf{0} \equiv \neg y_2) \wedge (y_2 \equiv (x_1 \wedge x_2))$ becomes a tautology. In order to satisfy the first part, $\mathbf{0} \equiv \neg y_2$, we need to set $s(y_2) = \mathbf{1}$. Then the formula can be simplified to $(x_1 \wedge x_2)$, which is not a tautology. In the second case, $s(y_1) = \mathbf{1}$, we get the expression $(\mathbf{1} \equiv \neg y_2) \wedge (y_2 \equiv (x_1 \wedge x_2))$. This requires to set $s(y_2) = \mathbf{0}$ in order to satisfy the first part, turning the formula into $\mathbf{0} \equiv (x_1 \wedge x_2)$, or more concisely, $\neg(x_1 \wedge x_2)$, which is neither a tautology. Therefore we can conclude that ψ^5 is unsatisfiable and, accordingly, Proposition 5 of [12] is unsound. ■

For Proposition 5 we make a similar observation as for Proposition 4:

Remark 6. *Also Proposition 5 of [12] is already unsound when we consider the commonly accepted semantics of closed prenex DQBFs as stated in Def. 2. The proposition claims that ψ^4 is equisatisfiable with ψ^5. Additionally, it claims that $\psi^6 := \forall x_1 \forall x_2 \exists y_1(\emptyset) \exists y_2(x_1, x_2) : (y_1 \equiv \neg y_2) \wedge (y_2 \equiv (x_1 \wedge x_2))$ is equisatisfiable with ψ^5. Due to transitivity of equisatisfiability, Proposition 5 claims that $\psi^4 \approx \psi^6$ holds. However, according to the semantics in Def. 2, ψ^4 is satisfiable and ψ^6 unsatisfiable. Again, ψ^4 and ψ^6 are actually QBFs; so Proposition 5 is also unsound when restricted to QBFs.*

Algorithm 1: DQBFQuantLoc

input : DQBF $\psi := Q : \varphi$
$Q := \forall x_1 \dots \forall x_n \exists y_1(D_{y_1}) \dots \exists y_m(D_{y_m})$
φ has an arbitrary structure given as AIG
output : DQBF ψ'

1 ψ^{np} := `NormalizeToNNF`(ψ);
2 ψ^{np} := `BuildMacroGates`(ψ^{np});
3 ψ^{np} := `Localize`(ψ^{np});
4 ψ' := `Eliminate`(ψ^{np});
5 **return** ψ'

IV. Taking Advantage of Quantifier Localization

In this section, we explain the implementation of the algorithm that exploits the properties of non-prenex DQBFs to simplify a given formula. First, we define necessary concepts and give a coarse sketch of the algorithm. Then, step by step, we dive into the details.

Benedetti introduced in [27] *quantifier trees* for pushing quantifiers into a CNF. In a similar way we construct a *quantifier graph*, which is an And-Inverter-Graph-like structure to perform quantifier localization according to Thrms. 3 and 4.

Definition 8 (Quantifier Graph)**.** *For a non-prenex DQBF ψ^{np}, a* quantifier graph *is a directed acyclic graph $G = (N, E)$. Each node from N is either an inner node with exactly two children (i. e., successors) or a terminal node without any children. Each inner node $n \in N$ is labeled with an operation $\diamond \in \{\wedge, \vee\}$ from ψ^{np}. Each terminal node $n \in N$ is labeled with a variable $v \in V_{\psi^{\text{np}}}$. Each edge from E is pointing from a node to one of its children and is possibly augmented with quantified variables and / or negations.*

The input to the basic algorithm for quantifier localization (*DQBFQuantLoc*) shown in Alg. 1 is a closed prenex DQBF ψ. The matrix φ of ψ is represented as an And-Inverter-Graph (AIG) [31] and the prefix Q is a set of quantifiers as stated in Def. 1. (If the matrix is initially given in CNF, we preprocess it by circuit extraction (see for instance [28], [21]) and the resulting circuit is then represented by an AIG.) The output of *DQBFQuantLoc* is a DQBF in closed prenex form again. In intermediate steps, we convert ψ into a non-prenex DQBF ψ^{np}, represented as a quantifier graph, by pushing quantifiers of the prefix into the matrix. After pushing the quantified variables as deep as possible into the formula, we eliminate quantifiers wherever it is possible. If a quantifier cannot be eliminated, it is pulled out of the formula again. In this manner we finally obtain a modified and possibly simplified prenex DQBF ψ'.

In Line 1 of Alg. 1, we first translate the matrix φ of the DQBF ψ into negation normal form (NNF) by pushing the negations in the circuit to the primary inputs (using De Morgan's law). The resulting matrix in NNF is represented as a quantifier graph as in Def. 8, where we only have negations at those edges which point to terminals. Fig. 1 shows a quantifier graph as returned by *NormalizeToNNF*. We will use it as a running example to illustrate our algorithm.

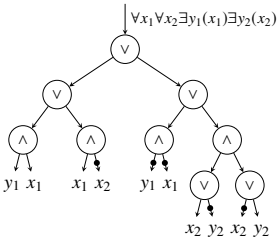


Fig. 1: Quantifier graph in NNF.

Algorithm 2: Localize

```
   input : Quantifier graph for DQBF ψ^np
1  for each macrogate g in topological order of the quantifier graph do
2      V_com := CollectCommonVariables(g);
3      if g is a disjunction then
4          for each existential variable y in V_com do
5              push y to macrochildren;
6              delete y from V_com;
7          end
8          while V_com ≠ ∅ do
9              v := FindBestVariableDis(V_com);
10             try to push v to macrochildren;
11             delete v from V_com;
12         end
13     else
14         while V_com ∩ V_ψ^∃ ≠ ∅ do
15             v := FindBestVariableCon(V_com);
16             try to push v to macrochildren;
17             delete v from V_com;
18         end
19         for each universal variable x in V_com do
20             try to push x to macrochildren;
21             delete x from V_com;
22         end
23     end
24 end
```

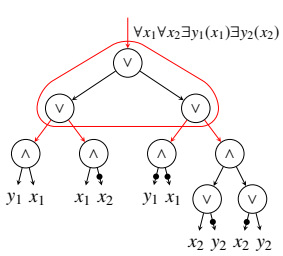
Then, in Line 2 of Alg. 1, we combine subcircuits into *AND / OR macrogates*. The combination into macrogates is essential to increase the degrees of freedom given by different decompositions of *AND*s / *OR*s that enable different applications of the transformation rules according to Thrms. 3 and 4. A *macrogate* is a multi-input gate, which we construct by collecting consecutive nodes representing the same logic operation. Except for the topmost node within a macrogate no other node may have more than one incoming edge, i. e., macrogates are subtrees of fanout-free cones. During the collection of nodes, we stop the search along a path when we visit a node with multiple parents. From this node we later start a new search. The nodes which are the target of an edge leaving a macrogate are the *macrochildren* of the macrogate and the parents of its root are called the *macroparents*. It is clear that a macrogate consisting of only one node has exactly two children like a standard node. For such nodes we use the terms macrogate and node interchangeably. In Fig. 2a we show a macrogate found in the running example.

After calling *NormalizeToNNF* and *BuildMacroGates* the only edge that carries quantified variables is the root edge. By shifting quantified variables to edges below the root node we push them into the formula. Sometimes we say that we push a quantified variable to a child by which we mean that we write the variable to the edge pointing to this child.

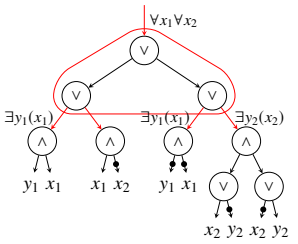
On the new DQBF ψ^{np} we perform the localization of quantifiers according to Thrm. 3 and 4 with the function *Localize* in Line 3. Alg. 2 presents the details.

The quantifier graph is traversed topologically from the root to the terminals. For each macrogate g we determine the set of quantified variables V_{com} that occur on *all* incoming edges of g by *CollectCommonVariables*. These are the only ones which we can push further into the graph. If an existential variable y cannot be pushed because it does not appear on all incoming edges, then all the universal variables in y's dependency set are also ruled out for pushing (see also (3i)).

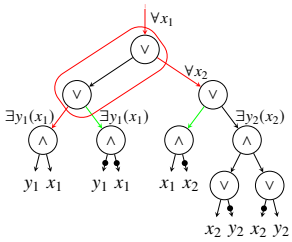
In the following we can push existential variables always before the universal variables due to (3i) ($x \notin D_y$ holds for



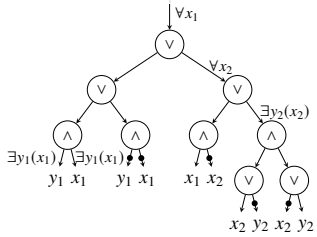
(a) A macrogate, marked in red.



(b) ψ^{np} after distributing y_1, y_2 according to (3d) and (3f).



(c) ψ^{np} after collecting macrochildren to enable pushing x_2.



(d) ψ^{np} after processing all macrogates.

Fig. 2: *BuildMacroGates* and *LocalizeVariables*.

$\exists y(D_y)\, \forall x : \varphi$ by construction). Universal variables x can only be pushed if all existential variables y on the same edge do not contain x in their dependency set (see (3i)). In Lines 3 and 13 of Alg. 2 we distinguish between a disjunction and a conjunction. In case g is a disjunction, at first we simply distribute each existential variable y from V_{com} to all macrochildren with y in their supports according to (3d) or (3f) (see Fig. 2b). To push a universal variable x from V_{com} we can apply (3e). If there is only one macrochild with x in its support and additionally all other macrochildren have no existential variable in their support which depends on x, then we can write x to the single macrochild without further efforts. This child then can be regarded as φ_2 from (3e). However, if there are several macrochildren with x or existential variables depending on x in their support, (and at least one other macrochild), then the macrogate g has to be restructured and split to enable the pushing as shown in Fig. 2c). We merge all children from the first set mentioned above and treat them as φ_2 from (3e), i. e., we decompose the *OR* macrogate g into one *OR* macrogate g' combining the children in the first set, and another macrogate g'' (replacing g) whose children are the remaining children of g as well as the new g'. Pushing x, we write $\forall x$ on the incoming edge of g'. The function *FindBestVariableDis* in Line 9 determines the order of pushing universal variables (see also (3h)). It greedily chooses those universal variables x first whose number of macrochildren that neither have x nor existential variables depending on x in their support is maximal.

If g is a conjunction, a decomposition of the macrogate g can take place when we push existential quantifiers according to (3f) (which we do first). Here we apply *FindBestVariableCon* (Line 15) to determine the order of pushing (see also (3g)) with a similar criterion as for the disjunction. Subsequently, only universal variables are left for pushing. This is done by (3c), (3e) or Thrm. 4. As mentioned above a universal variable x cannot be pushed, however, if there is some existential variable

y with $x \in D_y$ left on the incoming edge of g, because it could not be pushed.

The complete procedure is illustrated in Fig. 2.

Finally, in Line 4 of Alg. 1, we try to eliminate those variables that can be symbolically quantified after quantifier localization. The conditions are given by Thrm. 5 and (3b). We proceed from the terminals to the root and check each edge with at least one quantified variable. If a variable could not be eliminated, we pull it back to the incoming edges of this edges' source node. If a variable has been duplicated according to (3c) or (3d) and some duplications are brought back to one edge, then we merge them into a single variable again.

As Fig. 2d shows, we can eliminate both occurrences of variable y_1 since there are no other variables in the support of the target nodes. The same holds for y_2 because x_2 is the only variable different from y_2 in the support of the target node and x_2 is in the dependency set of y_2. Subsequently, x_2 and x_1 can be eliminated such that we obtain a constant function.

In general, having all remaining variables pulled back to the root edge, we return to a closed prenex DQBF with potentially fewer variables, fewer dependencies and a modified matrix, which we can pass back to a solver for prenex DQBFs.

V. Experimental Results

We embedded our algorithm into the DQBF solver HQS, which was the winner of the DQBF track of the QBFEVAL'18 and '19 competitions [32]. HQS includes the powerful DQBF-preprocessor HQSpre [33]. After preprocessing has finished, we call the algorithm *DQBFQuantLocalization* to simplify the formula. HQS augmented with the localization of quantifiers is denoted as *HQS$_{np}$*.[3]

The experiments were run on one core of an Intel Xeon CPU E5-2650v2 with 2.6 GHz. The runtime per benchmark was limited to 30 min and the memory consumption to 4 GB. We tested our theory with the same 4811 instances as in [22] [34] [21] [20]. They encompass equivalence checking problems for incomplete circuits [15], [16], [35], [18], controller synthesis problems [13] and instances from [36] where a DQBF has been obtained from a SAT problem.

Out of 4811 DQBF instances we focus here on those 974 which actually reach our algorithm. The remaining ones are solved by the preprocessor HQSpre or already exceed the time / memory limit either during preprocessing or while translating the formula into an AIG, i. e., in those cases the results for HQS and HQS$_{np}$ do not differ.

When we reach the function *DQBFQuantLocalization* from Alg. 1, for 963 out of 974 instances we can perform the localization of quantifiers. Quantifier localization enables the elimination of variables in subformulas in 840 instances. For 66918 times local quantifier elimination takes place and reduces the number of variables in 591 benchmarks. Note that if a variable has been doubled according to (3c) / (3d) and not all of the duplicates are eliminated, this variable cannot be deleted from the formula as some duplicates will be dragged back to the root. The size of the AIG after *DQBFQuantLocalization* has been decreased in 551 cases and has grown only in 286 cases, although in general it is not unusual that symbolic quantifier elimination increases the size of an AIG.

[3] A recent binary of HQS$_{np}$ and all DQBF benchmarks we used are provided at https://abs.informatik.uni-freiburg.de/src/projects_view.php?projectID=21

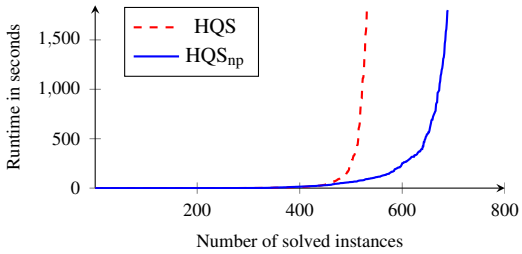


Fig. 3: HQS vs. HQS$_{np}$ – solved instances

Altogether 689 instances out of 974 were solved by HQS$_{np}$ in the end, whereas HQS could only solve 531. This increases the number of solved instances by 29.8% (for a cactus plot comparing HQS with HQS$_{np}$ see Fig. 3). The largest impact of quantifier localization has been observed on equivalence checking benchmarks for incomplete circuits from [35].

Fig. 4 shows the runtime for single benchmarks needed for HQS resp. HQS$_{np}$. The figure reveals that quantifier localization, in its current implementation, does not lead to a better result in every case. 12 benchmarks have not been solved by HQS$_{np}$, but by HQS. In all of these 12 instances the AIG sizes have grown during local quantifier elimination and processing larger AIGs resulted in larger run times. However, Fig. 4 also shows that in most cases the run times of HQS$_{np}$ are faster than those of HQS. Moreover, 170 benchmarks have been solved by HQS$_{np}$, but not by HQS.

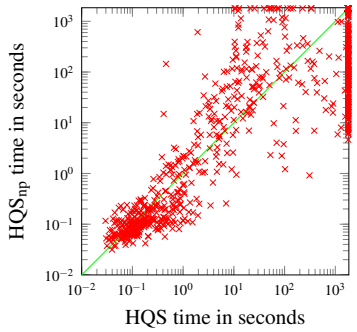


Fig. 4: HQS vs. HQS$_{np}$ – time comparison

For the benchmarks from QBFEVAL'18 the situation is pretty similar. 68 out of 334 benchmarks reach our algorithm and in all instances variables are pushed into the formula. On 20 benchmarks variables are eliminated locally and this makes it possible to solve 8 more instances. Here, all instances solved by HQS have also been solved by HQS$_{np}$ and, altogether, HQS solves 22 out of those 68 benchmarks whereas HQS$_{np}$ could solve 30 (36.4% more).

VI. Conclusions

In this paper, we presented syntax and semantics of non-prenex DQBFs and proved rules to transform prenex DQBFs into non-prenex DQBFs. We could demonstrate that we can achieve significant improvements by extending the DQBF solver HQS based on this theory. Simplifications of DQBFs were due to symbolic quantifier eliminations that were enabled by pushing quantifiers into the formula based on our rules for non-prenex DQBFs.

In the future, we aim at improving the results of quantifier localization, e. g., by introducing estimates on costs and benefits of quantifier localization operations as well as local quantifier elimination and by using limits on the growth of AIG sizes caused by local quantifier elimination.

References

[1] J. Rintanen, K. Heljanko, and I. Niemelä, "Planning as satisfiability: parallel plans and algorithms for plan search," *Artificial Intelligence*, vol. 170, no. 12–13, pp. 1031–1080, 2006.

[2] S. Eggersglüß and R. Drechsler, "A highly fault-efficient SAT-based ATPG flow," *IEEE Design & Test of Computers*, vol. 29, no. 4, pp. 63–70, 2012.

[3] A. Czutro, I. Polian, M. D. T. Lewis, P. Engelke, S. M. Reddy, and B. Becker, "Thread-parallel integrated test pattern generator utilizing satisfiability analysis," *Int'l Journal of Parallel Programming*, vol. 38, no. 3–4, pp. 185–202, 2010.

[4] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," *Advances in Computers*, vol. 58, pp. 117–148, 2003.

[5] E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving," *Formal Methods in System Design*, vol. 19, no. 1, pp. 7–34, 2001.

[6] F. Ivancic, Z. Yang, M. K. Ganai, A. Gupta, and P. Ashar, "Efficient SAT-based bounded model checking for software verification," *Theoretical Computer Science*, vol. 404, no. 3, pp. 256–274, 2008.

[7] F. Lonsing and A. Biere, "DepQBF: A dependency-aware QBF solver," *Journal on Satisfiability, Boolean Modelling and Computation*, vol. 7, no. 2–3, pp. 71–76, 2010.

[8] M. Janota, W. Klieber, J. Marques-Silva, and E. M. Clarke, "Solving QBF with counterexample guided refinement," in *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, ser. LNCS, A. Cimatti and R. Sebastiani, Eds., vol. 7317. Trento, Italy: Springer, Jun. 2012, pp. 114–128.

[9] M. Janota and J. Marques-Silva, "Solving QBF by clause selection," in *Int'l Joint Conf. on Artificial Intelligence (IJCAI)*, Q. Yang and M. Wooldridge, Eds. Buenos Aires, Argentina: AAAI Press, 2015, pp. 325–331.

[10] L. Tentrup and M. N. Rabe, "CAQE: A certifying QBF solver," in *Int'l Conf. on Formal Methods in Computer Aided Design (FMCAD)*. Austin, TX, USA: IEEE, Sep. 2015, pp. 136–143.

[11] G. Peterson, J. Reif, and S. Azhar, "Lower bounds for multiplayer non-cooperative games of incomplete information," *Computers & Mathematics with Applications*, vol. 41, no. 7–8, pp. 957–992, Apr. 2001.

[12] V. Balabanov, H. K. Chiang, and J. R. Jiang, "Henkin quantifiers and Boolean formulae: A certification perspective of DQBF," *Theoretical Computer Science*, vol. 523, pp. 86–100, 2014.

[13] R. Bloem, R. Könighofer, and M. Seidl, "SAT-based synthesis methods for safety specs," in *Int'l Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, ser. LNCS, K. L. McMillan and X. Rival, Eds., vol. 8318. San Diego, CA, USA: Springer, Jan. 2014, pp. 1–20.

[14] K. Chatterjee, T. A. Henzinger, J. Otop, and A. Pavlogiannis, "Distributed synthesis for LTL fragments," in *Int'l Conf. on Formal Methods in Computer Aided Design (FMCAD)*. IEEE, 2013, pp. 18–25.

[15] C. Scholl and B. Becker, "Checking equivalence for partial implementations," in *ACM/IEEE Design Automation Conference (DAC)*. Las Vegas, NV, USA: ACM Press, Jun. 2001, pp. 238–243.

[16] K. Gitina, S. Reimer, M. Sauer, R. Wimmer, C. Scholl, and B. Becker, "Equivalence checking of partial designs using dependency quantified Boolean formulae," in *IEEE Int'l Conf. on Computer Design (ICCD)*. Asheville, NC, USA: IEEE Computer Society, Oct. 2013, pp. 396–403.

[17] L. Henkin, "Some remarks on infinitely long formulas," in *Infinitistic Methods: Proc. of the 1959 Symp. on Foundations of Mathematics*. Warsaw, Panstwowe: Pergamon Press, Sep. 1961, pp. 167–183.

[18] A. Fröhlich, G. Kovásznai, A. Biere, and H. Veith, "iDQ: Instantiation-based DQBF solving," in *Int'l Workshop on Pragmatics of SAT (POS)*, ser. EPiC Series, D. L. Berre, Ed., vol. 27. Vienna, Austria: EasyChair, Jul. 2014, pp. 103–116.

[19] L. Tentrup and M. N. Rabe, "Clausal abstraction for DQBF," in *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, ser. LNCS. Lisboa, Portugal: Springer, Jul. 2019.

[20] K. Gitina, R. Wimmer, S. Reimer, M. Sauer, C. Scholl, and B. Becker, "Solving DQBF through quantifier elimination," in *Int'l Conf. on Design, Automation & Test in Europe (DATE)*. Grenoble, France: IEEE, Mar. 2015.

[21] R. Wimmer, K. Gitina, J. Nist, C. Scholl, and B. Becker, "Preprocessing for DQBF," in *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, ser. LNCS, M. Heule and S. Weaver, Eds., vol. 9340. Austin, TX, USA: Springer, Sep. 2015, pp. 173–190.

[22] R. Wimmer, A. Karrenbauer, R. Becker, C. Scholl, and B. Becker, "From DQBF to QBF by dependency elimination," in *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, ser. LNCS, vol. 10491. Melbourne, VIC, Australia: Springer, Aug. 2017, pp. 326–343.

[23] D. Geist and I. Beer, "Efficient model checking by automated ordering of transition relation partitions," in *Int'l Conf. on Computer-Aided Verification (CAV)*, ser. LNCS, vol. 818. Springer, 1994, pp. 299–310.

[24] R. Hojati, S. C. Krishnan, and R. K. Brayton, "Early quantification and partitioned transition relations," in *IEEE Int'l Conf. on Computer Design (ICCD)*. IEEE Computer Society, 1996, pp. 12–19.

[25] G. Cabodi, P. Camurati, L. Lavagno, and S. Quer, "Disjunctive partitioning and partial iterative squaring: An effective approach for symbolic traversal of large circuits," in *ACM/IEEE Design Automation Conference (DAC)*. ACM Press, 1997, pp. 728–733.

[26] I. Moon, J. H. Kukula, K. Ravi, and F. Somenzi, "To split or to conjoin: the question in image computation," in *ACM/IEEE Design Automation Conference (DAC)*. ACM, 2000, pp. 23–28.

[27] M. Benedetti, "Quantifier trees for QBFs," in *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, ser. LNCS, F. Bacchus and T. Walsh, Eds., vol. 3569. St. Andrews, UK: Springer, Jun. 2005, pp. 378–385.

[28] F. Pigorsch and C. Scholl, "Exploiting structure in an AIG based QBF solver," in *Int'l Conf. on Design, Automation & Test in Europe (DATE)*. Nice, France: IEEE, Apr. 2009, pp. 1596–1601.

[29] A. Ge-Ernst, C. Scholl, and R. Wimmer, "Quantifier localization for DQBF," arXiv, Tech. Rep. 1905.04755, 2019. [Online]. Available: http://arxiv.org/abs/1905.04755

[30] J. R. Jiang, "Quantifier elimination via functional composition," in *Int'l Conf. on Computer-Aided Verification (CAV)*, ser. LNCS, A. Bouajjani and O. Maler, Eds., vol. 5643. Grenoble, France: Springer, Jun. 2009, pp. 383–397.

[31] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust Boolean reasoning for equivalence checking and functional property verification," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 21, no. 12, pp. 1377–1394, 2002.

[32] L. Pulina and M. Seidl, "QBFEval'18 – Competitive evaluation of QBF solvers," http://www.qbflib.org/qbfeval18.php, 2018.

[33] R. Wimmer, S. Reimer, P. Marin, and B. Becker, "HQSpre – An effective preprocessor for QBF and DQBF," in *Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, ser. LNCS, A. Legay and T. Margaria, Eds., vol. 10205. Uppsala, Sweden: Springer, Mar. 2017, pp. 373–390.

[34] K. Gitina, R. Wimmer, C. Scholl, and B. Becker, "Skolem functions for DQBF," in *submitted for publication*, 2016.

[35] B. Finkbeiner and L. Tentrup, "Fast DQBF refutation," in *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, ser. LNCS, C. Sinz and U. Egly, Eds., vol. 8561. Vienna, Austria: Springer, Jul. 2014, pp. 243–251.

[36] V. Balabanov and J.-H. R. Jiang, "Reducing satisfiability and reachability to DQBF," Austin, TX, USA, Sep. 2015, talk at the Int'l Workshop on Quantified Boolean Formulas (QBF).