Journal on Satisfiability, Boolean Modeling, and Computation 11 (2019) 3-52

The (D)QBF Preprocessor HQSpre – Underlying Theory and Its Implementation*

Ralf Wimmer wimmer@informatik.uni-freiburg.de
Concept Engineering GmbH and Albert-Ludwigs-Universität Freiburg
Freiburg im Breisgau, Germany

Christoph Scholl scholl@informatik.uni-freiburg.de
Bernd Becker becker@informatik.uni-freiburg.de
Albert-Ludwigs-Universität Freiburg
Freiburg im Breisgau, Germany

Abstract

Preprocessing turned out to be an essential step for SAT, QBF, and DQBF solvers to reduce/modify the number of variables and clauses of the formula, before the formula is passed to the actual solving algorithm. These preprocessing techniques often reduce the computation time of the solver by orders of magnitude. In this paper, we present the preprocessor HQSpre that was developed for Dependency Quantified Boolean Formulas (DQBFs) and that generalizes different preprocessing techniques for SAT and QBF problems to DQBF. We give a presentation of the underlying theory together with detailed proofs as well as implementation details contributing to the efficiency of the preprocessor. HQSpre has been used with obvious success by the winners of the DQBF track, and, even more interestingly, the QBF tracks of QBFEVAL'18.

KEYWORDS: *Preprocessing, DQBF, Solver technology, Henkin quantifiers*

Submitted November 2018; revised March 2019; published September 2019

1. Introduction

Many problems, practically relevant and at the same time hard from a complexity-theoretic point of view, can be reduced to solving quantifier-free (SAT) or quantified (QBF) Boolean formulas. Such applications range, among many others, from automatic test pattern generation [16, 18] and formal verification of hard- and software systems [4, 14, 37] to planning [62], product configuration [69], and cryptanalysis [52]. During the last three decades, the development of very efficient algorithms to solve such formulas has paved the way from academic interest to industrial application of solver techniques. SAT-formulas with hundred thousands of variables and millions of clauses can be solved nowadays, with QBF about two orders of magnitude behind.

*This work was partly supported by the German Research Council (DFG) as part of the project "Solving Dependency Quantified Boolean Formulas". It extends the conference publications [77] and [79] by additional theoretical results on (D)QBF preprocessing techniques, more (and more recent) details on the implementation of HQSpre, complete proofs for all statements, and new experimental results on benchmarks from QBFEVAL'18.

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 verification of partial designs [65, 27], topologically constrained synthesis of logic circuits [1], synthesis of safe controllers [7], synthesis of fragments of linear-time temporal logic (LTL) [13], and the analysis of games with incomplete information [55] fall into this category and require an even more general formalism known as *dependency quantified Boolean formulas (DQBFs)* [55]. "Standard" quantified Boolean formulas have the restriction that each existential variable depends on all universal variables in whose scope it is. This restriction is relaxed for DQBF, which allows arbitrary dependencies, leading to so-called Henkin quantifiers [30] with explicitly specified dependency sets. 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 specificity of dependencies allows DQBF encodings to be exponentially more compact than their equivalent QBF counterparts. On the other hand, the increased expressiveness of DQBFs comes at the cost of a higher complexity for the decision problem – for SAT it is NP-complete [15], for QBF PSPACE-complete [51], and for DQBF it is NEXPTIME-complete [55].

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 [21], iProver [44], dCAQE [61], and HQS [28, 77, 78, 22]. iDQ [21] reduces the solution of a DQBF to the solution of a series of SAT instantiations. iProver [44] makes use of a translation of DQBFs into Effectively Propositional Logic (EPR) formulas [47]. dCAQE generalizes the CEGAR-based QBF solver CAQE [74]. HQS [28] applies quantifier elimination to solve the formula and works on circuit structures (And-Inverter Graphs) instead of CNFs.

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. However, there is typically a trade-off between the number of variables and the number of clauses; for instance, eliminating variables by resolution can increase the number of clauses significantly, which in turn increases memory consumption and the cost of subsequent operations on the formula. Removing redundant clauses is also not always beneficial: search-based SAT and QBF solvers add implied clauses to the formula to drive the search away from unsatisfiable parts of the search space [2, 68], which often reduces computation times considerably.

For SAT and QBF, efficient and effective preprocessing tools are available like SatELite [17], Coprocessor [50] for SAT and squeezeBF [29], Bloqqer [6] for QBF. Due to the success of preprocessing in SAT and QBF, one can expect that preprocessing is beneficial for DQBF, too – even more because the actual solving process is more costly than for QBF. This raises the question which techniques can be generalized from SAT and QBF to DQBF. Which adaptations need to be made to make them correct for the more general formalism? After suitable adaptations have been found, the correctness proofs have to be re-done for DQBF carefully because for QBF they often exploit the fact that dependencies in QBF follow a linear order. But also techniques like the detection of backbone literals (literals that are true

in all satisfying assignments of the matrix) [42, 39], which work for DQBF in the same way as for SAT and QBF, have to be re-thought: in SAT only incomplete, but cheap syntactic tests for the special case of unit literals are useful – determining backbone literals completely is as expensive as solving the SAT problem itself. For DQBF the situation is different as the decision problem is much harder. Even solving QBF approximations [27, 19] of the formula at hand as an incomplete decision procedure can be beneficial. Additionally the higher flexibility regarding the dependency sets in DQBF makes some techniques more powerful compared to QBF and enables new techniques.

Taken together, this paper lays the foundations for DQBF preprocessing. We provide several extensions and adaptations and provide techniques demonstrating that preprocessing for DQBF also conceptually goes beyond standard SAT/QBF techniques. We generalize successful preprocessing techniques for QBF to DQBF like blocked clause elimination (BCE) [45, 41, 6], equivalence reasoning [29], structure extraction [57], and variable elimination by resolution [3]. For some QBF preprocessing techniques that were given without proof so far we provide proofs for the DQBF generalizations (which include proofs for the original QBF version). For another existing QBF preprocessing technique (given without proof before), which allows to eliminate existential variables by resolution under too relaxed conditions, we provide a counterexample disproving the correctness of the corresponding theorem. It is interesting to note that sometimes the generalization of preprocessing techniques from QBF to DQBF even leads to a stronger *QBF* preprocessing technique when back-translated from the DQBF to the QBF case. The correctness proofs for all described techniques are available in this paper.

All techniques have been implemented in our (D)QBF preprocessor HQSpre.[1] Frequently, the efficiency of good solvers and preprocessors does not only rely on sound theoretical foundations, but also on sophisticated implementation details and optimizations. Therefore we provide a detailed description of the implementation as well, including remarks on data structures, on the order of applied preprocessing techniques, on the frequency of their application (depending on cost) etc. Moreover, we demonstrate that the applied techniques have to be chosen with care depending on the solving techniques applied in the solver core. For example, BCE prevents an effective undoing of Tseitin transformation [75], which is used to transform a formula into conjunctive normal form (CNF). Therefore, it is better to disable BCE if the underlying solver core does not rely on a formula in CNF, and to use BCE if undoing Tseitin transformation is not possible because the solver core requires a formula in CNF.

HQSpre has been successfully used by the winners of the DQBF track, and, even more interestingly, the QBF tracks of QBFEVAL'18. In our experiments we thoroughly evaluate the effect of HQSpre on different QBF and DQBF solvers from QBFEVAL'18. The results show that HQSpre significantly increases the number of solved instances for all considered solvers and in that way our preprocessor makes a considerable contribution to the success of state-of-the-art QBF and DQBF solvers.

Structure of this paper The next section introduces the necessary foundations of DQBF. Section 3 reviews incomplete, but cheap decision procedures for DQBF, Section 4 presents

[1] HQSpre is available as an open source tool. The most recent version can be downloaded from https://projects.informatik.uni-freiburg.de/projects/dqbf/files

the theoretical foundations of the preprocessing techniques for (D)QBF that we apply in our tool to simplify the DQBF at hand, as well as details on the implementation of the tool. Section 6 gives an experimental evaluation, and Section 7 concludes the paper.

2. Preliminaries

In this section, we briefly review the necessary foundations regarding dependency quantified Boolean formulas.

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 $\mathcal{A}(V')$ the set of Boolean assignments for V', i. e., $\mathcal{A}(V') = \{\nu \mid \nu : V' \to \{0,1\}\}$. As usual, for a Boolean assignment $\nu : V' \to \{0,1\}$ and $V'' \subseteq V'$ we denote the restriction of ν to V'' by $\nu_{|V''}$. A Boolean function with the set of input variables V' is a mapping $f : \mathcal{A}(V') \to \{0,1\}$. For each formula φ over V, a variable assignment ν to the variables in V induces a truth value 0 or 1 of φ, which we call $\nu(\varphi)$.

Definition 1 (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*

$$\psi := \forall x_1 \forall x_2 \ldots \forall x_n \exists y_1(D^\psi_{y_1}) \exists y_2(D^\psi_{y_2}) \ldots \exists y_m(D^\psi_{y_m}) : \varphi \tag{1}$$

where $D^\psi_{y_i} \subseteq \{x_1, \ldots, x_n\}$ for $i = 1, \ldots, m$ is the dependency set *of y_i, and φ is a quantifier-free Boolean formula over V, the* matrix *of ψ.*

To simplify the notation, we often write $\psi = Q : \varphi$ with the quantifier prefix Q and the matrix φ. We denote its set of universal variables by $V^\psi_\forall = \{x_1, \ldots, x_n\}$ and its set of existential variables by $V^\psi_\exists = \{y_1, \ldots, y_m\}$. If we do not need to distinguish between existential and universal variables, we write $v \in V$. $Q \setminus \{v\}$ denotes the prefix that results from removing a variable $v \in V$ from Q together with its quantifier. If v is existential, then its dependency set is removed as well; if v is universal, then all occurrences of v in the dependency sets of existential variables are removed. Similarly we use $Q \cup \{\exists y(D^\psi_y)\}$ to add existential variables to the prefix. The order in which the variables appear in the prefix is irrelevant. We introduce the dependency function $\mathrm{dep}_\psi : V \to 2^V$ by $\mathrm{dep}_\psi(v) = D^\psi_v$ if $v \in V^\psi_\exists$, and $\mathrm{dep}_\psi(v) = \{v\}$ for $v \in V^\psi_\forall$.

Definition 2 (Semantics of DQBF) *Let ψ be a DQBF with matrix φ as above. ψ is* satisfiable *iff there are functions $s_{y_i} : \mathcal{A}(D^\psi_{y_i}) \to \{0,1\}$ for $1 \le i \le m$ such that replacing each y_i by (a Boolean expression 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.*

Definition 3 (Equisatisfiability and Equivalence of DQBFs) *Two DQBFs ψ and ψ' are called* equisatisfiable *(written $\psi \approx \psi'$) if they are either both unsatisfiable or both satisfiable. Two DQBFs ψ and ψ' are called* equivalent *(written $\psi \equiv \psi'$) if they are either both unsatisfiable or both satisfiable with exactly the same sets of Skolem functions.*

Definition 4 (QBF) *A* quantified Boolean formula *(QBF)*[2] *is a DQBF ψ such that $D_y^\psi \subseteq D_{y'}^\psi$ or $D_{y'}^\psi \subseteq D_y^\psi$ holds for any pair $y, y' \in V_\exists^\psi$ of existential variables.*

Usually, the quantifier prefix of a *QBF* is represented by a series of universal and existential quantifications where for each existential variable y the dependency set D_y^ψ is exactly the set of universal variables written to the left of y.

In the following we assume, unless explicitly stated differently, that the matrix φ is given in *conjunctive normal form* (CNF). A formula is in CNF if it is a conjunction of *clauses*; a clause is a disjunction of *literals*, and a literal is either a variable v or its negation $\neg v$. We identify a formula in CNF with its set of clauses and a clause with its set of literals, e. g., we write $\big\{\{x_1, \neg x_2\}, \{x_2, \neg x_3\}\big\}$ for the formula $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3)$. A clause C *subsumes* a clause C' iff $C \subseteq C'$. For a literal ℓ, $\mathrm{var}(\ell)$ denotes the corresponding variable, i. e., $\mathrm{var}(v) = \mathrm{var}(\neg v) = v$ and $\mathrm{dep}_\psi(\ell) = \mathrm{dep}_\psi\big(\mathrm{var}(\ell)\big)$. For a clause C, $\mathrm{var}(C) = \bigcup_{\ell \in C} \{\mathrm{var}(\ell)\}$ and $\mathrm{dep}_\psi(C) = \bigcup_{\ell \in C} \mathrm{dep}_\psi(\ell)$. We ignore double negations, i. e., $v = \neg\neg v$. Moreover, we define the "sign" of a literal as $\mathrm{sgn}(v) = 1$ and $\mathrm{sgn}(\neg v) = 0$.

Each DQBF can be transformed such that the matrix is in CNF. While transforming the matrix directly into CNF can cause an exponential blow-up in size, Tseitin transformation [75] can do this with only a linear increase in size at the cost of additional existential variables. The idea is to introduce auxiliary existential variables that store the truth value of sub-expressions. Since the values of these variables are uniquely determined by the sub-expression, they can simply depend on all universal variables.

We assume that none of the clauses of the CNF φ under consideration is tautological, i. e., there is no variable v such that $\{v, \neg v\} \subseteq C$ for all $C \in \varphi$. The preprocessing operations we present check the modified or added clauses whether they are tautologies and, if this is the case, remove or ignore them.

Propositional resolution is a central operation on formulas in CNF:

Definition 5 (Resolution) *Let φ be a formula in CNF, ℓ a literal, and $C, C' \in \varphi$ clauses such that $\ell \in C$ and $\neg\ell \in C'$. The* resolvent *of C and C' w. r. t. to the* pivot literal *ℓ is given by $C \otimes_\ell C' := \big(C \setminus \{\ell\}\big) \cup \big(C' \setminus \{\neg\ell\}\big)$.*

Resolvents are implied by the formula, i. e., if R is a resolvent of two clauses in φ, then φ and $\varphi \cup \{R\}$ are equivalent [5, Section 3.2.1].

A few solvers for DQBF have been proposed in the literature: An extension of the DPLL algorithm, typically applied for solving SAT and QBF formulas, has been described in [20]. However, no implementation thereof is available. iDQ [21] uses instantiation-based solving, i. e., it reduces deciding a DQBF to deciding a series of SAT problems. iDQ can be seen as a specialization of the solver iProver [44] to the purely propositional domain. Nevertheless, iProver itself, which is able to solve Effectively Propositional Logic (EPR) formulas [47], can be used for solving DQBFs as well by translating each DQBF into an EPR formula in a simple manner (using an extension of a translation from QBF to EPR [67]). dCAQE is another DQBF solver based on the QBF solver CAQE [74]. Finally, there is the solver HQS [28], which applies quantifier elimination on And-Inverter Graphs (AIGs) to solve the formula. An AIG is essentially a circuit which consists of AND and inverter gates only.

[2] We only consider closed QBFs in prenex form here, i. e., QBFs in which all variables are bound by a quantifier and in which the quantifiers precede the matrix.

Although HQS reads a CNF-based input format as other solvers, too, its back-end can handle Boolean formulas of arbitrary structure.

3. Incomplete, but Cheap Decision Procedures

Before we present our preprocessing techniques for QBF and DQBF, we review incomplete, but cheap decision procedures for such formulas. Using such techniques as a preprocessing step can help decide formulas without actually calling a (D)QBF solver. We call such techniques "filters". Since the complexity of solving the formulas involved in these filters is lower than of solving the actual problem, running filters before calling the (D)QBF solver is often beneficial.

We start with SAT-based filters which are suitable for both QBF and DQBF and proceed with QBF-based filters suitable for DQBF.

3.1 SAT-based Filters for QBF and DQBF

SAT checks over the matrix are used in order to find *trivially (un)satisfiable formulas* [12].

A (D)QBF is trivially unsatisfiable if the matrix φ is already unsatisfiable for an arbitrary but fixed assignment of the universal variables. In HQSpre we use for this check an assignment of the universal variables which satisfies the fewest clauses, i. e., we assign x to 1 if x occurs in fewer clauses than $\neg x$.

A (D)QBF is trivially satisfiable if, after removing each occurrence of a universal literal within the matrix φ, the resulting matrix φ' is satisfiable. The correctness of this statement immediately follows from the fact that removing each occurrence of universal literals in the matrix corresponds to removing all universal variables from the dependency sets of existential variables followed by universal reduction, see Section 4.4.1. That means, we consider a restriction of the Skolem functions to constants. If we find Skolem functions in this way, then they work for the original formula as well.

Of course, if a (D)QBF formula does not contain any universal variables at all, we immediately employ a SAT solver for deciding the formula. This situation rarely occurs in the very beginning, but rather after universal expansions, see Section 4.2.4.

3.2 QBF-based Filters for DQBF

For DQBF we check whether we can prove satisfiability or unsatisfiability based on QBF solving, which usually needs much less resources than solving the DQBF. Our approach is as follows: First we apply preprocessing for DQBF, which is helpful for both the filter technique and the actual solver core. Then we run a filter technique to prove unsatisfiability, and only if it finishes with an inconclusive result, we apply the solver core.

The filter is based on QBF approximations: By using an appropriate quantifier prefix and the same matrix, a DQBF ψ can be over-approximated by a QBF $\Psi^{\uparrow}$ such that the unsatisfiability of $\Psi^{\uparrow}$ implies the unsatisfiability of ψ [27]. Similarly one can construct an under-approximation $\Psi^{\downarrow}$ such that the satisfiability of $\Psi^{\downarrow}$ implies the satisfiability of ψ. As the under-approximation was inconclusive for all instances in our experiments, we focus on over-approximations, which allow to show unsatisfiability of DQBFs. The theory for under-approximations is analogous as for over-approximations.

Definition 6 (QBF over-approximation) *A QBF $\Psi^\uparrow = Q' : \varphi$ is an* over-approximation *of ψ (written $\psi \sqsubseteq \Psi^\uparrow$) if $D_y^{\Psi^\uparrow} \supseteq D_y^\psi$ holds for all existential variables $y \in V_\exists^\psi$.*

Lemma 1 *Let ψ be a DQBF and $\psi \sqsubseteq \Psi^\uparrow$. If $\Psi^\uparrow$ is unsatisfiable, so is ψ.*

This lemma directly follows from the fact that Skolem functions for ψ are Skolem functions for $\Psi^\uparrow$, too.

Typically, there are several QBF over-approximations of a DQBF. They can be more or less precise. Let $\Psi_1^\uparrow = Q_1 : \varphi$ and $\Psi_2^\uparrow = Q_2 : \varphi$ be two QBF over-approximations of the same DQBF $\psi = Q : \varphi$. We call $\Psi_1^\uparrow$ *stronger* than $\Psi_2^\uparrow$ (written $\Psi_1^\uparrow \sqsubseteq \Psi_2^\uparrow$) if for all $y \in V_\exists^\psi$ we have $D_y^{\Psi_1^\uparrow} \subseteq D_y^{\Psi_2^\uparrow}$. If $\Psi_1^\uparrow \sqsubseteq \Psi_2^\uparrow$ and $\Psi_2^\uparrow$ is unsatisfiable, so is $\Psi_1^\uparrow$. A QBF over-approximation is a *strongest* QBF over-approximation if there is no different QBF over-approximation that is stronger. Strongest over-approximations are as close to the original DQBF w. r. t. the dependency sets as possible. Therefore it is desirable to solve a strongest over-approximation as an incomplete decision procedure for DQBF.

QBF approximations as defined above choose an appropriate QBF prefix for the DQBF at hand, but leave the matrix and the set of variables unchanged. More powerful QBF filters can be obtained by allowing modifications of the matrix and the variables. Finkbeiner and Tentrup [19] propose a series of more and more precise QBF formulas, starting with a strongest QBF over-approximation. Their construction uses $k \geq 1$ copies of the matrix and its variables. For $k = 1$ it reduces to a strongest QBF over-approximation as described above. For $k > 1$ it is required that the existential variables are assigned consistently over all copies and that all copies of the matrix are satisfied. Consistent means that if the universal variables in the dependency set of an existential variable are assigned the same values in two copies, then the existential variables have to carry the same value. This is expressed with the following formula:

$$\mathrm{Cons}(Y,k) := \bigwedge_{y \in Y} \bigwedge_{i=1}^{k} \bigwedge_{j=i+1}^{k} \Big((y^i \equiv y^j) \vee \bigvee_{x \in D_y^\psi} (x^i \not\equiv x^j) \Big) .$$

Let Q be the prefix of a strongest QBF approximation $\Psi^\uparrow$ of the DQBF ψ and Q^i be created from Q by replacing all variables v by their i-th copy v^i. We define the QBF $\Psi(k)$ for a parameter $k \geq 1$ by[3.]:

$$\Psi(k) := Q^1 Q^2 \ldots Q^k : \mathrm{Cons}\big(V_\exists^\psi, k\big) \wedge \bigwedge_{i=1}^{k} \varphi^i .$$

Theorem 1 ([19]) *The DQBF ψ is unsatisfiable iff $\Psi(k)$ is unsatisfiable for some $k \geq 1$.*

Experiments show that this technique can identify many unsatisfiable instances with fairly small values of k (with the majority of unsatisfiable instances identified already by $k = 1$ when $\Psi(k)$ is equal to a strongest QBF over-approximation). Since the sizes of the QBF instances grow considerably with increasing values of k, in most cases only values $k \leq 3$ seem beneficial. For more details we refer the reader to [19].

[3.] For consistency reasons we have negated the formula compared to [19].

4. Theoretical Background of Preprocessing Techniques for DQBF

In this section, we describe techniques which can be applied to preprocess a DQBF.

Before we look into (D)QBF simplifications by (1) variable elimination, (2) clause elimination, (3) clause strengthening, and (4) minimizing dependency sets, we consider gate detection and structure extraction, which are especially successful when the matrix originally results from a circuit or a Boolean expression that was transformed into CNF.

4.1 Detecting Gate Definitions and Structure Extraction

The (D)QBF's matrix in CNF is often created from a circuit or a Boolean expression by Tseitin transformation [75], where a new existential variable v_e is created for each sub-expression (or gate output) e. Clauses encoding the relationship $v_e \equiv e$ are added and the sub-expression e is replaced by the variable v_e. For example, a k-input AND gate $y \equiv \text{AND}(\ell_1, \ldots, \ell_k)$ has a Tseitin encoding consisting of $(k+1)$ clauses $\{\neg y, \ell_1\}, \ldots, \{\neg y, \ell_k\}, \{y, \neg\ell_1, \ldots, \neg\ell_k\}$. In such a functional definition $y \equiv f(\ell_1, \ldots, \ell_k)$, y is called the *defined variable*, f is the *definition* of y, and the clauses corresponding to the relationship $y \equiv f(\ell_1, \ldots, \ell_k)$ are the *defining clauses*.

For solvers that do not rely on a matrix in CNF (like the DQBF solver HQS [28] or the QBF solver AIGsolve [57]) transformation steps introducing gate definitions can be undone. Here all artificially introduced variables are removed and circuit structure is extracted from the CNF: If a relationship $y \equiv f(\ell_1, \ldots, \ell_k)$ is detected which does not lead to cyclic dependencies, y can be removed from the prefix and the defining clauses can be removed from the matrix. Additionally, a data structure is used which assigns to each defined variable its definition. To create an AIG representation that can be passed to a non-CNF-based solver core like HQS or AIGsolve, the remaining clauses are converted into an AIG and then the defined variables are substituted by their definitions. Those structure extraction steps can only be performed, if a certain condition on the dependency sets of the defined variable and the variables occurring in the defining clauses are fulfilled:

Theorem 2 *Let $\psi = Q : \varphi$ be a DQBF and $\varphi^f \subseteq \varphi$ the defining clauses for the relationship $y \equiv f(\ell_1, \ldots, \ell_k)$. Then ψ is equisatisfiable with*

$$Q \setminus \{y\} : (\varphi \setminus \varphi^f)[f(\ell_1, \ldots, \ell_k)/y]$$

if the following conditions are satisfied:

1. $y \in V_\exists^\psi$,
2. *for $i = 1, \ldots, k$ we have* $\text{dep}_\psi(\ell_i) \subseteq \text{dep}_\psi(y)$.

Proof. We set $\psi' := Q \setminus \{y\} : (\varphi \setminus \varphi^f)[f(\ell_1, \ldots, \ell_k)/y]$ and show that ψ and ψ' are equisatisfiable.

First assume that ψ is unsatisfiable. Then there is no set of Skolem functions which turns φ into a tautology. In particular sets of Skolem functions for which $s_y = f(\ell_1, \ldots, \ell_k)[s_{y'}/y'$ for $y' \in \{\text{var}(\ell_1), \ldots, \text{var}(\ell_k)\} \cap V_\exists^\psi]$ holds do not turn φ into a tautology. (Note that s_y defined in that way is an admissible Skolem function due to condition 2. in the theorem.) Hence, $Q \setminus \{y\} : \varphi[f(\ell_1, \ldots, \ell_k)/y]$ is unsatisfiable. Since φ^f is equivalent to $y \equiv f(\ell_1, \ldots, \ell_k)$, $\varphi^f[f(\ell_1, \ldots, \ell_k)/y]$ is

a tautology. Therefore $\varphi[f(\ell_1,\ldots,\ell_k)/y]$ and $(\varphi \setminus \varphi^f)[f(\ell_1,\ldots,\ell_k)/y]$ are equivalent. This shows that ψ' is unsatisfiable.

Now assume that ψ is satisfiable. This implies that there are Skolem functions $s_{y_1},\ldots,s_{y_m}$ for $y_1,\ldots,y_m$. Because of the defining clauses that encode $y \equiv f(\ell_1,\ldots,\ell_k)$, the Skolem function s_y for y satisfies the relationship $s_y = f(\ell_1,\ldots,\ell_k)[s_{y'}/y' \text{ for } y' \in V_\exists^\psi \setminus \{y\}]$. Thus $\{s_{y_1},\ldots,s_{y_m}\} \setminus \{s_y\}$ is a set of Skolem functions for $Q \setminus \{y\} : \varphi[f(\ell_1,\ldots,\ell_k)/y]$. This means that $Q \setminus \{y\} : \varphi[f(\ell_1,\ldots,\ell_k)/y]$ is satisfiable. Satisfiability of $Q \setminus \{y\} : \varphi[f(\ell_1,\ldots,\ell_k)/y]$ implies satisfiability of $Q \setminus \{y\} : (\varphi \setminus \varphi^f)[f(\ell_1,\ldots,\ell_k)/y]$. □

Even if the solver back-end requires a matrix in CNF, detecting gate definitions may be useful for eliminating the defined variable (see Section 4.2.3), for detecting equivalent variables (see Section 4.2.2), and for manipulating dependency sets of existential variables (see Section 4.5).

4.2 Variable Elimination Routines

First of all, we describe techniques that reduce the number of variables in the formula.

4.2.1 Unit and Pure Literals, Backbones, and Monotonic Variables

Unit and pure variables are well-known concepts from SAT and QBF solving. In the context of DQBF solving, pure literals have first been discussed in [20]. Both unit and pure literals can be replaced by constant values without influencing the formula's truth value. Typically a literal is defined as unit if the matrix contains a clause consisting only of this literal. A variable is pure if it occurs in the whole matrix either only positive or only negative:

Definition 7 (Unit and pure literals) *A literal ℓ is a* unit literal *if $\{\ell\} \in \varphi$; ℓ is a* pure literal *if $\neg\ell$ does not appear in any clause of φ.*

These are syntactic criteria that can be checked efficiently. This is necessary because in particular the detection of unit literals is one of the main operations of search-based SAT and QBF solvers as a part of *Boolean constraint propagation (BCP)* [5]: Given a set L of unit literals, BCP finds all literals L' that become unit because of the assignment enforced by the literals in L (written $\mathrm{BCP}_\varphi(L) = L'$). The same process also detects when a clause becomes unsatisfied because of the unit literals in L. The latter is called a conflict and denoted by $\mathrm{BCP}_\varphi(L) = \bot$.

For (D)QBF preprocessing, it is possible to use more expensive checks to determine variables which may be replaced by constants. Therefore we give a more general semantic definition:

Definition 8 (Backbones and monotonic variables) *A variable $v \in V$ is a positive (negative)* backbone *if $\varphi[0/v]$ ($\varphi[1/v]$, resp.) is unsatisfiable. A literal ℓ is a backbone, if $\ell = v$ and v a positive backbone, or if $\ell = \neg v$ and v a negative backbone.*

A variable $v \in V$ is positive (negative) monotonic *if $\varphi[0/v] \wedge \neg\varphi[1/v]$ ($\varphi[1/v] \wedge \neg\varphi[0/v]$, resp.) is unsatisfiable.*

The following lemma says that backbones generalize unit literals and monotonic variables generalize pure literals:

Lemma 2 *Unit literals are backbones, and pure literals monotonic.*

Proof. Let ℓ be a literal such that $\{\ell\} \in \varphi$, i. e., $\varphi = \varphi' \cup \big\{\{\ell\}\big\}$. W. l. o. g. we assume $\operatorname{sgn}(\ell) = 1$. Replacing ℓ by 0 yields $\varphi[0/\ell] = \varphi'[0/\ell] \cup \big\{\{0\}\big\}$, which is unsatisfiable as the clause $\{0\}$ is false. Therefore ℓ is a backbone.

Now consider the case that φ does not contain $\neg\ell$. Then we can partition φ into the clauses φ^ℓ which contain ℓ and the clauses $\varphi^\emptyset$ not containing ℓ, i. e., $\varphi = \varphi^\ell \wedge \varphi^\emptyset$. We can write φ as $\varphi = (\ell \vee \tilde{\varphi}^\ell) \wedge \varphi^\emptyset$ with an appropriate formula $\tilde{\varphi}^\ell$ that does not contain ℓ. We have

$$\begin{aligned}\varphi[0/\ell] \wedge \neg\varphi[1/\ell] &= (0 \vee \tilde{\varphi}^\ell) \wedge \varphi^\emptyset \wedge \neg\big((1 \vee \tilde{\varphi}^\ell) \wedge \varphi^\emptyset\big) \\ &= \tilde{\varphi}^\ell \wedge \varphi^\emptyset \wedge \neg\varphi^\emptyset \\ &= 0.\end{aligned}$$

Hence, ℓ is monotonic. □

The following theorem states how we can exploit backbones and monotonic variables to reduce the size of the formula:

Theorem 3 *Let $\psi = Q : \varphi$ be a DQBF and $v \in V$ a backbone or a monotonic variable.*

If v is a positive or negative backbone and universal, ψ is unsatisfiable. Otherwise ψ is equisatisfiable with ψ' where

- *$\psi' = Q \setminus \{v\} : \varphi[1/v]$ if v is existential and either a positive backbone or positive monotonic, or v is universal and negative monotonic;*
- *$\psi' = Q \setminus \{v\} : \varphi[0/v]$ if v is existential and either a negative backbone or negative monotonic, or v is universal and positive monotonic.*

Proof. Let $Q : \varphi$ be the DQBF

$$\psi = \forall x_1 \forall x_2 \ldots \forall x_n \exists y_1(D^\psi_{y_1}) \exists y_2(D^\psi_{y_2}) \ldots \exists y_m(D^\psi_{y_m}) : \varphi.$$

- First assume, v is *existentially quantified and a positive backbone*. W. l. o. g. we assume $v = y_1$. ψ is satisfiable iff there are Skolem functions $s_{y_1}, \ldots, s_{y_m}$ such that

  $$\varphi(x_1, \ldots, x_n, s_{y_1}(D_{y_1}), \ldots, s_{y_m}(D_{y_m}))$$

  is a tautology. We prove that $s_{y_1} = 1$ whenever ψ is satisfiable. This implies the equisatisfiability of $Q \setminus \{v\} : \varphi[1/v]$ and ψ . We prove this statement by contradiction and assume that ψ is satisfiable with Skolem functions $s_{y_1}, \ldots, s_{y_m}$, but s_{y_1} is *not* the constant 1 function. Then there exists an assignment ν of the universal variables such that $s_{y_1}(\nu(D_{y_1})) = 0$ holds. We have:

  $$\begin{aligned}&\varphi(\nu(x_1), \ldots, \nu(x_n), s_{y_1}(\nu(D_{y_1})), \ldots, s_{y_m}(\nu(D_{y_m}))) \\ \equiv\ &\varphi(\nu(x_1), \ldots, \nu(x_n), 0, s_{y_2}(\nu(D_{y_2})), \ldots, s_{y_m}(\nu(D_{y_m}))) \\ \equiv\ &\varphi[0/y_1](\nu(x_1), \ldots, \nu(x_n), s_{y_2}(\nu(D_{y_2})), \ldots, s_{y_m}(\nu(D_{y_m}))) \\ \equiv\ &0\ .\end{aligned}$$

  The last step holds since $v = y_1$ is a positive backbone. $\varphi(\nu(x_1), \ldots, \nu(x_n), s_{y_1}(\nu(D_{y_1})), \ldots, s_{y_m}(\nu(D_{y_m}))) = 0$ contradicts the assumption that $\varphi(x_1, \ldots, x_n, s_{y_1}(D_{y_1}), \ldots, s_{y_m}(D_{y_m}))$ is a tautology and thus s_{y_1} has to be the constant 1 function. The proof in case that v is existentially quantified and a negative backbone can be executed analogously.

- Now let $v = x_1$ be *universally quantified and a positive backbone*. $Q : \varphi$ is satisfiable iff there are Skolem functions $s_{y_1}, \ldots, s_{y_m}$ such that $\varphi(x_1, \ldots, x_n, s_{y_1}(D_{y_1}), \ldots, s_{y_m}(D_{y_m}))$ is a tautology. Now choose an arbitrary assignment ν of the universal variables with $\nu(x_1) = 0$. We have

$$\begin{aligned} & \varphi(\nu(x_1), \ldots, \nu(x_n), s_{y_1}(\nu(D_{y_1})), \ldots, s_{y_m}(\nu(D_{y_m}))) \\ \equiv\ & \varphi(0, \ldots, \nu(x_n), s_{y_1}(\nu(D_{y_1})), \ldots, s_{y_m}(\nu(D_{y_m}))) \\ \equiv\ & \varphi[0/x_1](\nu(x_2), \ldots, \nu(x_n), s_{y_1}(\nu(D_{y_1})), \ldots, s_{y_m}(\nu(D_{y_m}))) \\ \equiv\ & 0\ . \end{aligned}$$

  Again, the last step holds due to the assumption that $v = x_1$ is a positive backbone. Since $\varphi(x_1, \ldots, x_n, s_{y_1}(D_{y_1}), \ldots, s_{y_m}(D_{y_m}))$ must be 1 for each assignment of $x_1, \ldots x_n$, $Q : \varphi$ is not satisfiable. The case that v is universally quantified and a negative backbone is analogous.

- Let $v = x_1$ be *universally quantified and positive monotonic*, i. e., $\varphi[0/x_1] \wedge \neg\varphi[1/x_1]$ is unsatisfiable. This expression is equivalent to $\varphi[0/x_1] \Rightarrow \varphi[1/x_1]$ being a tautology and to every satisfying assignment of $\varphi[0/x_1]$ being also a satisfying assignment of $\varphi[1/x_1]$.

  If $Q \setminus \{x_1\} : \varphi[0/x_1]$ is unsatisfiable, then also $Q : \varphi$. So let $Q \setminus \{x_1\} : \varphi[0/x_1]$ be satisfiable. Then there are Skolem functions $s_{y_1}, \ldots, s_{y_m}$ for the existential variables $y_1, \ldots, y_m$ such that $\varphi[0/x_1](x_2, \ldots, x_n, s_{y_1}(D'_{y_1}), \ldots, s_{y_m}(D'_{y_m}))$ is a tautology (with $D'_{y_i} = D_{y_i} \setminus \{x_1\}$). By assumption, $\varphi[1/x_1]\big(x_2, \ldots, x_n, s_{y_1}(D'_{y_1}), \ldots, s_{y_m}(D'_{y_m})\big)$ is a tautology, too, and therefore

$$\begin{aligned} \big(x_1 \wedge \varphi[1/x_1](x_2, \ldots, x_n, s_{y_1}(D'_{y_1}), \ldots, s_{y_m}(D'_{y_m}))\big) \vee & \\ \big(\neg x_1 \wedge \varphi[0/x_1](x_2, \ldots, x_n, s_{y_1}(D'_{y_1}), \ldots, s_{y_m}(D'_{y_m}))\big) & \\ \equiv\quad \varphi(x_1, \ldots, x_n, s_{y_1}(D'_{y_1}), \ldots, s_{y_m}(D'_{y_m})) & \end{aligned}$$

  is a tautology as well. Therefore, if $Q \setminus \{x_1\} : \varphi[0/x_1]$ is satisfiable, then also $Q : \varphi$. The proof for negative monotonic universal variables can be carried out analogously.

- Finally let $v = y_1$ be *existentially quantified and positive monotonic*. That means, as for universal positive monotonic variables that every satisfying assignment of $\varphi[0/y_1]$ is also a satisfying assignment of $\varphi[1/y_1]$.

  Assume that $Q \setminus \{\exists y_1\} : \varphi[1/y_1]$ is satisfiable. So there are Skolem functions $s_{y_2}(D_{y_2}), \ldots, s_{y_m}(D_{y_m})$ such that $\varphi[1/y_1](x_1, \ldots, x_n, s_{y_2}(D_{y_2}), \ldots, s_{y_m}(D_{y_m}))$ is a tautology. This is equivalent to $\varphi(x_1, \ldots, x_n, 1, s_{y_2}(D_{y_2}), \ldots, s_{y_m}(D_{y_m}))$ being a tautology. Therefore $s_{y_1}(D_{y_1}) = 1$ is a Skolem function for y_1 in ψ.

  Now let $\psi := Q : \varphi$ be satisfiable. We have to show that $s_{y_1}(D_{y_1}) = 1$ is a Skolem function for y_1 in ψ. If ψ is satisfiable, there are Skolem functions $s_{y_1}(D_{y_1}), \ldots, s_{y_m}(D_{y_m})$ with $\varphi(x_1, \ldots, x_n, s_{y_1}(D_{y_1}), \ldots, s_{y_m}(D_{y_m}))$ being a tautology. Let $\nu : V^{\psi}_{\forall} \to \{0, 1\}$ be an arbitrary assignment of the universal variables. Then we have

$$\varphi(\nu(x_1), \ldots, \nu(x_n), s_{y_1}(\nu(D_{y_1})), \ldots, s_{y_m}(\nu(D_{y_m}))) = 1.$$

  If $s_{y_1}(\nu(D_{y_1})) = 0$ holds, then ν is a satisfying assignment of $\varphi[0/y_1]\big(x_1, \ldots, x_n, s_{y_2}(D_{y_2}), \ldots, s_{y_m}(D_{y_m})\big)$ and therefore by assumption also of $\varphi[1/y_1](x_1, \ldots, x_n, s_{y_2}(D_{y_2}), \ldots, s_{y_m}(D_{y_m}))$. If $s_{y_1}(\nu(D_1)) = 1$ holds, then ν is a satisfying assignment of $\varphi[1/y_1](x_1, \ldots, x_n, s_{y_2}(D_{y_2}), \ldots, s_{y_m}(D_{y_m}))$.

  Hence every assignment ν satisfies $\varphi[1/y_1](x_1, \ldots, x_n, s_{y_2}(D_{y_2}), \ldots, s_{y_m}(D_{y_m}))$, and $s_{y_2}(D_{y_2})$, $\ldots, s_{y_m}(D_{y_m})$ are Skolem functions of $Q \setminus \{y_1\} : \varphi[1/y_1]$.

  The proof for existential negative monotonic variables is similar. □

The truth value of backbone and monotonic variables v can be fixed and propagated through φ using Boolean constraint propagation.

Checks whether a variable is a backbone or monotonic can be done using a SAT solver. As already mentioned, in the SAT and QBF context typically efficient (sound but not complete) syntactic criteria are applied to detect backbones and monotonic variables.

Another cheap criterion to identify backbones uses the binary implication graph of a formula (which is later also used to identify equivalent literals):

Definition 9 *Let* $\varphi^2 = \{C \in \varphi \mid |C| = 2\}$ *be the set of binary clauses. The* binary implication graph *of* ψ *is the directed graph* $\mathrm{BIP}(\psi) = (L, E)$ *with the set* $L = \{v, \neg v \mid v \in V\}$ *of literals as its set of nodes and* $E = \{(\neg\ell, k), (\neg k, \ell) \mid \{\ell, k\} \in \varphi^2\}$ *the set of edges.*

Paths in the binary implication graph have useful properties:

Lemma 3 *Let* $\mathrm{BIP}(\psi) = (L, E)$ *be the binary implication graph of* $\psi = Q : \varphi$ *and* $\ell_1, \ell_2 \in L$ *two literals. If there is a path from* ℓ_1 *to* ℓ_2 *in* $\mathrm{BIP}(\psi)$*, then* φ *implies the clause* $\{\neg\ell_1, \ell_2\}$*.*

Proof. Let $\ell_1 = k_1 \to k_2 \to \cdots \to k_p = \ell_2$ be a path in $\mathrm{BIP}(\psi)$ from ℓ_1 to ℓ_2. This means that φ contains the clauses $C_i := \{\neg k_i, k_{i+1}\}$ for $i = 1, \ldots, p-1$. We set $r_1 = \{\neg k_1, k_2\}$ and $r_i = r_{i-1} \otimes_{k_i} C_i = \{\neg k_1, k_{i+1}\}$ for $i = 2, \ldots, p-1$. So we can derive $r_{k-1} = \{\neg k_1, k_p\} = \{\neg\ell_1, \ell_2\}$ from $C_1, \ldots, C_{p-1}$ by resolution. As resolvents are implied by the formula, this shows the claim. □

This directly implies the following lemma:

Lemma 4 *A literal* ℓ *is a backbone if there is a path in* $\mathrm{BIP}(\psi)$ *from* $\neg\ell$ *to* ℓ*.*

Proof. Assume that there is a path in $\mathrm{BIP}(\psi)$ from $\neg\ell$ to ℓ. Then we can derive the clause $\{\neg\neg\ell, \ell\} = \{\ell\}$ by resolution according to Lemma 3. Therefore $Q : \varphi$ is equivalent to $Q : \varphi \wedge \{\ell\}$. By Definition 7 we have that ℓ is unit in $\varphi \wedge \{\ell\}$. Hence ℓ is a backbone in $\varphi \wedge \{\ell\}$ and also in φ. □

Unit and pure literals according to Definition 7 and backbones according to Lemma 4 can be determined efficiently by traversing the matrix or, respectively, the binary implication graph. Since solving a DQBF is much harder than solving a SAT (or even QBF) problem and the gain by eliminating one variable is larger, it often pays off to additionally use semantic checks (cf. Definition 8) for backbones and monotonic variables, which are based on solving a sequence of (incremental) SAT problems. For backbones in the QBF context this observation has already been made in [58].

4.2.2 Equivalent Variables

Another well-known variable elimination technique is the detection of *equivalences*, i. e., determining whether a literal ℓ is logically equivalent to another literal k. In this case, one of the variables can be eliminated by replacing all occurrences with the other one. In the (D)QBF case one has to take into account the quantifiers and the dependencies of the affected variables.

Definition 10 (Equivalent literals) *The literals* ℓ *and* k *are* equivalent *w. r. t. a propositional formula* φ *iff* φ *is equivalent to* $\varphi \wedge (\ell \equiv k)$*.*

Theorem 4 *Let ℓ and k be equivalent literals. We assume w. l. o. g. that* $\operatorname{sgn}(\ell) = 1$.

- *If* $\operatorname{var}(\ell), \operatorname{var}(k) \in V_\forall^\psi$, *then ψ is unsatisfiable.*

Otherwise, we assume w. l. o. g. that $\operatorname{var}(\ell) \in V_\exists^\psi$.

- *If* $\operatorname{var}(k) \in V_\forall^\psi$ *and* $\operatorname{var}(k) \notin D^\psi_{\operatorname{var}(\ell)}$, *then ψ is unsatisfiable.*
- *If* $\operatorname{var}(k) \in V_\forall^\psi$ *and* $\operatorname{var}(k) \in D^\psi_{\operatorname{var}(\ell)}$, *then ψ is equisatisfiable with* $Q \setminus \{\operatorname{var}(\ell)\} : \varphi[k/\ell]$.
- *If* $\operatorname{var}(\ell), \operatorname{var}(k) \in V_\exists^\psi$, *then ψ is equisatisfiable with*

$$\psi' := \big(Q \setminus \{\operatorname{var}(k), \operatorname{var}(\ell)\}\big) \cup \big\{\exists \operatorname{var}(k)(D^\psi_{\operatorname{var}(k)} \cap D^\psi_{\operatorname{var}(\ell)})\big\} : \varphi[k/\ell]\,.$$

Note that in the last case, the intersection of $\operatorname{var}(\ell)$ and $\operatorname{var}(k)$'s dependencies has to be taken. When ℓ and k are equivalent, the same Skolem function (modulo negation) must be used for both variables. Since the Skolem function for $\operatorname{var}(\ell)$ must only depend on $\operatorname{var}(\ell)$'s dependencies and the Skolem function for $\operatorname{var}(k)$ only on $\operatorname{var}(k)$'s dependencies, their joint Skolem function may depend only on the dependencies $\operatorname{var}(\ell)$ and $\operatorname{var}(k)$ have in common.

Theorem 4 can be proved as follows:

Proof. Assume that ℓ and k are equivalent, i. e., φ is equivalent to $\varphi \wedge (\neg\ell \vee k) \wedge (\ell \vee \neg k)$, which is the same as $\varphi \wedge (\ell \equiv k)$.

- We first consider the case where $\operatorname{var}(\ell), \operatorname{var}(k) \in V_\forall^\psi$.
  Since $(\ell \equiv k)$ does not contain an existential variable and universal quantifiers distribute over $\wedge$, ψ is equivalent to
  $$\big(Q : \varphi\big) \wedge \big(\forall \operatorname{var}(\ell) \forall \operatorname{var}(k) : (\ell \equiv k)\big).$$
  Obviously, $(\ell \equiv k)$ is not a tautology, and therefore ψ is unsatisfiable.

- Next, let $\operatorname{var}(\ell) \in V_\exists^\psi$, $\operatorname{var}(k) \in V_\forall^\psi$, and $\operatorname{var}(k) \notin D^\psi_{\operatorname{var}(\ell)}$.
  Assume ψ were satisfiable. Then there were Skolem functions that turn $\varphi \wedge (\ell \equiv k)$ into a tautology. The only Skolem function for $\operatorname{var}(\ell)$ which is able to turn $(\ell \equiv k)$ into a tautology is $s_{\operatorname{var}(\ell)} = k$ (note that we assume $\operatorname{sgn}(\ell) = 1$). However, this Skolem function is not admissible as $\operatorname{var}(k) \notin D^\psi_{\operatorname{var}(\ell)}$. Therefore ψ is unsatisfiable.

- Now consider the case that $\operatorname{var}(\ell) \in V_\exists^\psi$, $\operatorname{var}(k) \in V_\forall^\psi$, and $\operatorname{var}(k) \in D^\psi_{\operatorname{var}(\ell)}$.
  With a similar argumentation as in the previous case, we can derive that if ψ is satisfiable, the only Skolem function for $\operatorname{var}(\ell)$ is $s_{\operatorname{var}(\ell)} = k$, which is admissible. Therefore, replacing ℓ by k again yields a satisfiable formula. On the other hand, if ψ is unsatisfiable, replacing the existential variables by any admissible function does not turn φ into a tautology. Therefore replacing ℓ by k yields an unsatisfiable formula again.

- Finally, we consider $\operatorname{var}(\ell), \operatorname{var}(k) \in V_\exists^\psi$.
  First assume that ψ is unsatisfiable, i. e., there is no set of Skolem functions for the existential variables which turns the matrix into a tautology. In particular, this also holds for all sets of Skolem functions in which $s_{\operatorname{var}(\ell)}$ and $s_{\operatorname{var}(k)}$ are identical (modulo negation). Therefore replacing ℓ by k and restricting the dependency set accordingly yields an unsatisfiable formula again.

  Now assume that ψ is satisfiable. To simplify the notations assume w. l. o. g. that not only $\operatorname{sgn}(\ell) = 1$, but also $\operatorname{sgn}(k) = 1$. (The proof for $\operatorname{sgn}(k) = 0$ is analogous.) Because the Skolem functions for $\operatorname{var}(\ell)$ and $\operatorname{var}(k)$ have to turn $(\ell \equiv k)$ into a tautology, they have to be equal because of $(\ell \equiv k)$. Since the Skolem function must be admissible for $\operatorname{var}(\ell)$, it must not depend on $D^\psi_{\operatorname{var}(k)} \setminus D^\psi_{\operatorname{var}(\ell)}$. Similarly, because it has to be admissible for $\operatorname{var}(k)$, it must not

depend on $D^{\psi}_{\mathrm{var}(\ell)} \setminus D^{\psi}_{\mathrm{var}(k)}$. Therefore it may only depend on $D^{\psi}_{\mathrm{var}(\ell)} \cap D^{\psi}_{\mathrm{var}(k)}$. So we may replace ℓ by k if we restrict $\mathrm{var}(k)$'s dependency set to $D^{\psi}_{\mathrm{var}(\ell)} \cap D^{\psi}_{\mathrm{var}(k)}$.

□

To detect equivalent literals, we exploit the following lemma:

Lemma 5 *Two literals ℓ, k are equivalent if there is a path in $\mathrm{BIP}(\psi)$ from ℓ to k and vice versa.*

Proof. The path from ℓ to k allows us, according to Lemma 3, to derive the clause $\{\neg\ell, k\}$, the path from k to ℓ the corresponding clause $\{\neg k, \ell\}$. Both clauses may be added to φ as they are resolvents. This implies according to Definition 10 that ℓ and k are equivalent. □

We decompose $\mathrm{BIP}(\psi)$ into strongly connected components (SCCs) using Tarjan's SCC algorithm [73]. SCCs have the property that there is a path between each pair of nodes in an SCC. Therefore all literals within one SCC are equivalent. They are replaced by one representative by applying Theorem 4. This procedure was described, e. g., in [8, 23, 26, 33, 50] for SAT preprocessing and also for QBF preprocessing.

Another easy way to detect equivalent variables is based on gate definitions. If the matrix contains clauses encoding the two gate definitions $y_1 \equiv f(\ell_1, \ldots, \ell_k)$ and $y_2 \equiv f(\ell_1, \ldots, \ell_k)$, y_1 is apparently equivalent to y_2 and Theorem 4 can be applied to remove one of the two variables.

Of course, even SAT checks based on Definition 10 may be beneficial in the (D)QBF context for detecting equivalent literals.

4.2.3 Eliminating Defined Variables

Gate definitions $y \equiv f(\ell_1, \ldots, \ell_k)$ can be used for variable elimination as well. If the solver core does not rely on a matrix in CNF, the defined variable can be removed by replacing it by its definition as described in Section 4.1. If the solver core only works on CNFs, the same thing can be done (including removal of the defining clauses), but afterwards the matrix has to be transformed into CNF again. Substitution and transformation into CNF can be simulated by a series of resolutions [17]; the exploitation of gate definitions usually yields fewer resolvents than a "standard" elimination of y by resolution, see Section 4.2.4. Nevertheless it can produce very large formulas in some cases, and hence it is only performed, if the formula does not grow above a user-given bound, for details see Section 5.

If the elimination of gate definitions fails due to size limits, *gate rewriting* can be used instead. Gate rewriting has been introduced (without proof) for QBF in [29]. The generalization to DQBF is straightforward. It adds a new existential variable y' with the same dependency set as y. For the implication direction $f(\ell_1, \ldots, \ell_k) \Rightarrow y$ of the Tseitin encoding of the gate, y is replaced by y'. The negative occurrences of y in the part of the matrix without the defining clauses are replaced by $\neg y'$. (The transformation can be interpreted as replacing a Tseitin transformation by a double *Plaisted-Greenbaum encoding* [59].) Gate rewriting is based on the following theorem:

Theorem 5 (Gate Rewriting) *Let $\psi = Q : \varphi$ be a DQBF,*

$$\varphi = \varphi^{y \Rightarrow f} \dot\cup \varphi^{f \Rightarrow y} \dot\cup \varphi^{y} \dot\cup \varphi^{\neg y} \dot\cup \varphi^{\emptyset},$$

where $\varphi^{y\Rightarrow f}$ *are the clauses representing the relationship* $y \Rightarrow f(\ell_1,\ldots,\ell_k)$, $\varphi^{f\Rightarrow y}$ *the clauses representing the relationship* $f(\ell_1,\ldots,\ell_k) \Rightarrow y$, $\varphi^{y} = \{C \in \varphi \setminus (\varphi^{y\Rightarrow f} \cup \varphi^{f\Rightarrow y}) \,|\, y \in C\}$, $\varphi^{\neg y} = \{C \in \varphi \setminus (\varphi^{y\Rightarrow f} \cup \varphi^{f\Rightarrow y}) \,|\, \neg y \in C\}$, *and* $\varphi^{\emptyset} = \varphi \setminus (\varphi^{y\Rightarrow f} \cup \varphi^{f\Rightarrow y} \cup \varphi^{y} \cup \varphi^{\neg y})$. *Let* $y \in V^{\psi}_{\exists}$ *and* $\mathrm{dep}_{\psi}(\ell_i) \subseteq \mathrm{dep}_{\psi}(y)$ *for* $i = 1,\ldots,k$.

Then ψ *is equisatisfiable with*

$$\psi' = Q \cup \left\{\exists y'(D^{\psi}_{y})\right\} : \varphi^{y\Rightarrow f} \cup \varphi^{f\Rightarrow y}[y'/y] \cup \varphi^{y} \cup \varphi^{\neg y}[y'/y] \cup \varphi^{\emptyset}.$$

Proof. In a first step we prove that ψ' is equisatisfiable with

$$\psi'' = Q \cup \left\{\exists y'(D^{\psi}_{y})\right\} : \varphi^{y\Rightarrow f} \cup \varphi^{f\Rightarrow y} \cup \varphi^{y\Rightarrow f}[y'/y] \cup \varphi^{f\Rightarrow y}[y'/y] \cup \varphi^{y} \cup \varphi^{\neg y}[y'/y] \cup \varphi^{\emptyset}.$$

Since ψ'' results from ψ' by adding clauses, the satisfiability of ψ'' implies the satisfiability of ψ'. Now let us assume that ψ' is satisfiable, i. e., there are Skolem functions s_{y_i} for all existential variables y_i $(i = 1,\ldots,m)$ such that replacing y_i by s_{y_i} turns the matrix φ' of ψ' into a tautology. From the Skolem functions s_{y_i} of ψ' we derive Skolem functions s'_{y_i} of ψ'' by the following definitions:

$s'_z = s_z$, if $z \neq y$ and $z \neq y'$. For an arbitrary assignment $\nu : V^{\psi}_{\forall} \to \{0,1\}$ of the universal variables we define[4.]

$$s'_{y}(\nu_{|\mathrm{dep}_{\psi'}(y)}) = \begin{cases} 1, & \text{if } \nu(f(\ell_1,\ldots,\ell_k)[s_{\mathrm{var}(\ell_i)}/\,\mathrm{var}(\ell_i) \text{ for all } i = 1,\ldots,k]) = 1, \\ s_y(\nu_{|\mathrm{dep}_{\psi}(y)}), & \text{if } \nu(f(\ell_1,\ldots,\ell_k)[s_{\mathrm{var}(\ell_i)}/\,\mathrm{var}(\ell_i) \text{ for all } i = 1,\ldots,k]) = 0, \end{cases}$$

and

$$s'_{y'}(\nu_{|\mathrm{dep}_{\psi'}(y)}) = \begin{cases} 0, & \text{if } \nu(f(\ell_1,\ldots,\ell_k)[s_{\mathrm{var}(\ell_i)}/\,\mathrm{var}(\ell_i) \text{ for all } i = 1,\ldots,k]) = 0, \\ s_{y'}(\nu_{|\mathrm{dep}_{\psi'}(y)}), & \text{if } \nu(f(\ell_1,\ldots,\ell_k)[s_{\mathrm{var}(\ell_i)}/\,\mathrm{var}(\ell_i) \text{ for all } i = 1,\ldots,k]) = 1. \end{cases}$$

Note that s'_y and $s'_{y'}$ are well-defined and admissible for ψ'', since $\mathrm{dep}_{\psi'}(\ell_i) \subseteq \mathrm{dep}_{\psi'}(y) = \mathrm{dep}_{\psi'}(y')$ for $i = 1,\ldots,k$.

Now we have to prove that the functions defined in that way are Skolem functions for ψ''. Consider an arbitrary assignment $\nu : V^{\psi}_{\forall} \to \{0,1\}$ of the universal variables.

- First consider the clauses from $\varphi^{y\Rightarrow f}$. They are in included both in φ', the matrix of ψ', and in φ'', the matrix of ψ''. Since φ' with the existential variables replaced by Skolem functions of ψ' is a tautology, we have $\nu(\varphi^{y\Rightarrow f}[s_{y_i}/y_i \text{ for } i = 1,\ldots,m]) = 1$. Since $\varphi^{y\Rightarrow f}$ is equivalent to $y \Rightarrow f(\ell_1,\ldots,\ell_k)$ and $s'_y(\nu_{|\mathrm{dep}_{\psi'}(y)}) \neq s_y(\nu_{|\mathrm{dep}_{\psi'}(y)})$ only if $\nu(f(\ell_1,\ldots,\ell_k)[s_{\mathrm{var}(\ell_i)}/\,\mathrm{var}(\ell_i) \text{ for all } i = 1,\ldots,k]) = \nu(f(\ell_1,\ldots,\ell_k)[s'_{\mathrm{var}(\ell_i)}/\,\mathrm{var}(\ell_i) \text{ for all } i = 1,\ldots,k]) = 1$, we can conclude that $\nu(\varphi^{y\Rightarrow f}[s'_{y_i}/y_i \text{ for } i = 1,\ldots,m]) = 1$ as well.
- $\varphi^{f\Rightarrow y}$ is included in φ'', but not in φ'. $\varphi^{f\Rightarrow y}$ is equivalent to $f(\ell_1,\ldots,\ell_k) \Rightarrow y$. By construction of s'_y we have $s'_y(\nu_{|\mathrm{dep}_{\psi'}(y)}) = 1$, if $\nu(f(\ell_1,\ldots,\ell_k)[s_{\mathrm{var}(\ell_i)}/\,\mathrm{var}(\ell_i) \text{ for all } i = 1,\ldots,k]) = \nu(f(\ell_1,\ldots,\ell_k)[s'_{\mathrm{var}(\ell_i)}/\,\mathrm{var}(\ell_i) \text{ for all } i = 1,\ldots,k]) = 1$, and thus

$$\nu\big((f(\ell_1,\ldots,\ell_k) \Rightarrow y)\,[s'_{\mathrm{var}(\ell_i)}/\,\mathrm{var}(\ell_i) \text{ for all } i = 1,\ldots,k][s'_y/y]\big) = \nu\big(\varphi^{f\Rightarrow y}[s'_{\mathrm{var}(\ell_i)}/\,\mathrm{var}(\ell_i) \text{ for all } i = 1,\ldots,k][s'_y/y]\big) = 1.$$

- $\varphi^{y\Rightarrow f}[y'/y]$ is included in φ'', but not in φ'. The proof for $\varphi^{y\Rightarrow f}[y'/y]$ is similar to the previous case: $\varphi^{y\Rightarrow f}[y'/y]$ is equivalent to $y' \Rightarrow f(\ell_1,\ldots,\ell_k)$. By construction of $s'_{y'}$ we have

[4.] To simplify notations in this proof we define for universal variables $\mathrm{var}(\ell_i) \in V^{\psi}_{\forall}$: $s_{\mathrm{var}(\ell_i)} := \mathrm{var}(\ell_i)$.

$s'_{y'}(\nu_{|\mathrm{dep}_{\psi'}(y)}) = 0$, if

$$\nu(f(\ell_1,\ldots,\ell_k)[s_{\mathrm{var}(\ell_i)}/\mathrm{var}(\ell_i) \text{ for all } i=1,\ldots,k]) =$$
$$\nu(f(\ell_1,\ldots,\ell_k)[s'_{\mathrm{var}(\ell_i)}/\mathrm{var}(\ell_i) \text{ for all } i=1,\ldots,k]) = 0,$$

and thus

$$\nu\big((y' \Rightarrow f(\ell_1,\ldots,\ell_k))\,[s'_{\mathrm{var}(\ell_i)}/\mathrm{var}(\ell_i) \text{ for all } i=1,\ldots,k][s'_{y'}/y']\big) =$$
$$\nu\big(\varphi^{y\Rightarrow f}[y'/y][s'_{\mathrm{var}(\ell_i)}/\mathrm{var}(\ell_i) \text{ for all } i=1,\ldots,k][s'_y/y']\big) = 1.$$

- The clauses in $\varphi^{f\Rightarrow y}[y'/y]$ are included both in φ' and in φ''. Using our precondition, we have $\nu(\varphi^{f\Rightarrow y}[y'/y][s_{y_i}/y_i \text{ for } i=1,\ldots,m]) = 1$. Since $\varphi^{f\Rightarrow y}[y'/y]$ is equivalent to $f(\ell_1,\ldots,\ell_k) \Rightarrow y'$ and $s'_{y'}(\nu_{|\mathrm{dep}_{\psi'}(y)}) \neq s_{y'}(\nu_{|\mathrm{dep}_{\psi'}(y)})$ only if $\nu(f(\ell_1,\ldots,\ell_k)[s_{\mathrm{var}(\ell_i)}/\mathrm{var}(\ell_i) \text{ for all } i=1,\ldots,k]) = \nu(f(\ell_1,\ldots,\ell_k)[s'_{\mathrm{var}(\ell_i)}/\mathrm{var}(\ell_i) \text{ for all } i=1,\ldots,k]) = 0$, we can conclude $\nu(\varphi^{f\Rightarrow y}[y'/y][s'_{y_i}/y_i \text{ for } i=1,\ldots,m]) = 1$ as well.
- The clauses in φ^y contain only positive occurrences of y and no occurrences of y'. Since $s'_y \geq s_y$ and $s'_z = s_z$ for $z \in V_\exists^{\psi'} \setminus \{y, y'\}$, $\nu(\varphi^y[s_{y_i}/y_i \text{ for } i=1,\ldots,m]) = 1$ implies $\nu(\varphi^y[s'_{y_i}/y_i \text{ for } i=1,\ldots,m]) = 1$.
- The clauses in $\varphi^{\neg y}[y'/y]$ contain only negative occurrences of y' and no occurrences of y. Since $s'_{y'} \leq s_{y'}$ and $s'_z = s_z$ for $z \in V_\exists^{\psi'} \setminus \{y, y'\}$, $\nu(\varphi^y[s_{y_i}/y_i \text{ for } i=1,\ldots,m]) = 1$ implies $\nu(\varphi^y[s'_{y_i}/y_i \text{ for } i=1,\ldots,m]) = 1$.
- The clauses in $\varphi^\emptyset$ do not have any occurrences of y or y'. Since $s'_z = s_z$ for $z \in V_\exists^{\psi'} \setminus \{y, y'\}$, we have $\nu(\varphi^y[s'_{y_i}/y_i \text{ for } i=1,\ldots,m]) = \nu(\varphi^y[s_{y_i}/y_i \text{ for } i=1,\ldots,m])$.

Now it is easy to see that ψ'' is equivalent to

$$\psi''' = Q \cup \{\exists y'(D_y^\psi)\} : (y \equiv f(\ell_1,\ldots,\ell_k)) \wedge (y' \equiv f(\ell_1,\ldots,\ell_k)) \wedge \varphi^y \wedge \varphi^{\neg y}[y'/y] \wedge \varphi^\emptyset.$$

$(y \equiv f(\ell_1,\ldots,\ell_k)) \wedge (y' \equiv f(\ell_1,\ldots,\ell_k))$ implies $y \equiv y'$ and by Theorem 4 we can conclude that replacing y' by y in ψ''' results in an equisatisfiable DQBF (remember that $\mathrm{dep}_{\psi'}(y) = \mathrm{dep}_{\psi'}(y')$). The resulting DQBF is equivalent to ψ, which finishes the proof. □

In fact, introducing a copy y' of y does the contrary to eliminating variables, but the hope is that the transformation pays off by triggering simplifications later on during the work of the preprocessor and the solver core. The purpose of the transformation is to favor the detection of pure literals when the clauses including y or $\neg y$ (the clauses including y' or $\neg y'$) evaluate to true and to increase the chance that clauses are blocked [6] during the solver run.

4.2.4 Resolution and Universal Expansion

Finally, there are techniques for the elimination of existential and universal variables by applying resolution and universal expansion, respectively. For both, the QBF version has been described in [5]. Generally speaking, both methods eliminate a variable at the cost of expanding the formula.

For QBF, resolution together with universal reduction [43] is able to derive the empty clause iff the formula is unsatisfiable. This does not hold for DQBF [1]. While *adding* resolvents is sound without any further restrictions for DQBF as well, *eliminating* existential

variables by resolution [17] has to be handled with care. Here we give a set of sufficient conditions which allow variable elimination by resolution for DQBF. In particular when the formula is created by Tseitin transformation [75], variable elimination by resolution is applicable to a large subset of the formula's existential variables.

Theorem 6 (Variable elimination by resolution) *Let $y \in V_\exists^\psi$ be an existential variable of ψ. We partition φ into the sets $\varphi^y = \{C \in \varphi \,|\, y \in C\}$, $\varphi^{\neg y} = \{C \in \varphi \,|\, \neg y \in C\}$, and $\varphi^\emptyset = \varphi \setminus (\varphi^y \cup \varphi^{\neg y})$.*
If one of the following conditions is satisfied:

1. *for all $C \in \varphi^y$ and all $k \in C$ we have $\mathrm{dep}_\psi(k) \subseteq \mathrm{dep}_\psi(y)$,*
2. *for all $C' \in \varphi^{\neg y}$ and all $k \in C'$ we have $\mathrm{dep}_\psi(k) \subseteq \mathrm{dep}_\psi(y)$, or*
3. *y is the defined variable of a functional definition, i. e., there are clauses encoding the relationship $y \equiv f(V')$ for some function f and arguments $V' \subseteq V \setminus \{y\}$, $\mathrm{dep}_\psi(v) \subseteq \mathrm{dep}_\psi(y)$ for all $v \in V'$ (cf. Section 4.1),*

then ψ is equisatisfiable with

$$\psi' := Q \setminus \{y\} : \varphi^\emptyset \wedge \bigwedge_{C \in \varphi^y} \bigwedge_{C' \in \varphi^{\neg y}} C \otimes_y C' \,.$$

Proof. Since the variable y does not occur in $\varphi^\emptyset \wedge \bigwedge_{C \in \varphi^y} \bigwedge_{C' \in \varphi^{\neg y}} C \otimes_y C'$, it is clear that

$$Q \setminus \{y\} : \varphi^\emptyset \wedge \bigwedge_{C \in \varphi^y} \bigwedge_{C' \in \varphi^{\neg y}} C \otimes_y C'$$

and

$$Q : \underbrace{\varphi^\emptyset \wedge \bigwedge_{C \in \varphi^y} \bigwedge_{C' \in \varphi^{\neg y}} C \otimes_y C'}_{\varphi'} \tag{2}$$

are equisatisfiable. So we have to prove the equisatisfiability of $Q : \varphi$ and $Q : \varphi' = Q : \varphi^\emptyset \wedge \bigwedge_{C \in \varphi^y} \bigwedge_{C' \in \varphi^{\neg y}} C \otimes_y C'$.

First let φ be satisfiable. Since adding resolvents of φ to φ does not change φ, we have

$$Q : \varphi \quad \equiv \quad Q : \varphi \wedge \bigwedge_{C \in \varphi^y} \bigwedge_{C' \in \varphi^{\neg y}} C \otimes_y C'.$$

Since deleting clauses from a satisfiable formula yields a satisfiable formula again, we can conclude that (2) is satisfiable.

Now let (2) be satisfiable. We show that this implies the satisfiability of $Q : \varphi$ if one of the conditions in the theorem is satisfied.

1. First assume that $\mathrm{dep}_\psi(k) \subseteq \mathrm{dep}_\psi(y)$ holds for all $C \in \varphi^y$ and all $k \in C$. If (2) is satisfiable, there are Skolem functions s'_{y_i} for all existential variables y_i $(i = 1, \ldots, m)$ such that replacing y_i by s'_{y_i} turns φ' into a tautology. We define the following set of Skolem functions for φ:

$$s_z = \begin{cases} s'_z, & \text{if } z \neq y, \\ \neg\varphi^y[0/y][s'_{y_i}/y_i \text{ for all } i = 1, \ldots, m \text{ with } y_i \neq y], & \text{if } z = y. \end{cases}$$

That means: for all existential variables except y we use the Skolem functions from (2). The Skolem function for y is chosen such that it assigns 1 to y iff there is a clause in φ^y which is not already satisfied by (the Skolem functions of) the other literals. s_y defined in the given way depends on a subset of variables from $\text{dep}_\psi(y)$, since we have $\text{dep}_\psi(k) \subseteq \text{dep}_\psi(y)$ for all $C \in \varphi^y$ and all $k \in C$.

Clearly, $s_{y_1}, \ldots, s_{y_m}$ satisfy the clauses in $\varphi^\emptyset$. We distinguish two cases for proving that $s_{y_1}, \ldots, s_{y_m}$ satisfy all clauses in φ^y and $\varphi^{\neg y}$ as well (for all variable assignments to the universal variables). Consider an arbitrary assignment ν of values to the universal variables.

- Case 1:
  Each clause $C \in \varphi^y$ contains a literal $k \neq y$ such that $s_{\text{var}(k)}(\nu_{|\text{dep}_\psi(k)}) = \text{sgn}(k)$, if $k \in V_\exists^\psi$, and $\nu(\text{var}(k)) = \text{sgn}(k)$, if $k \in V_\forall^\psi$, i. e., the formula φ^y evaluates to 1, if we replace existential variables by Skolem functions and evaluate w. r. t. ν. This is independent from the value for y. According to the definition of $s_y = \neg\varphi^y[0/y][s'_{y_i}/y_i$ for all $i = 1, \ldots, m$ with $y_i \neq y]$ we then have $s_y(\nu_{|\text{dep}_\psi(y)}) = 0$. Thus, all clauses in $\varphi^{\neg y}$ evaluate to 1, if we replace existential variables by Skolem functions and evaluate w. r. t. ν, since all those clauses contain $\neg y$.
- Case 2:
  There is a clause $C \in \varphi^y$ such that for all literals $k \in C$ with $k \neq y$ it holds $s_{\text{var}(k)}(\nu_{|\text{dep}_\psi(k)}) = \neg\,\text{sgn}(k)$, if $k \in V_\exists^\psi$, and $\nu(\text{var}(k)) = \neg\,\text{sgn}(k)$, if $k \in V_\forall^\psi$, i. e., the formula φ^y would evaluate to 0, if we would replace y by 0. According to the definition of $s_y = \neg\varphi^y[0/y][s'_{y_i}/y_i$ for all $i = 1, \ldots, m$ with $y_i \neq y]$ we then have $s_y(\nu_{|\text{dep}_\psi(y)}) = 1$ and thus all clauses in φ^y evaluate to 1 for ν after replacing existential variables by Skolem functions. We have to show that in this case all clauses in $\varphi^{\neg y}$ evaluate to 1 as well. Assume the opposite, i. e., there is a clause $C' \in \varphi^{\neg y}$ that does not evaluate to 1. Then consider the resolvent $C \otimes_y C'$. As for all literals k in $C \setminus \{y\}$ it holds $s_{\text{var}(k)}(\nu_{|\text{dep}_\psi(k)}) = \neg\,\text{sgn}(k)$, if $k \in V_\exists^\psi$, $\nu(\text{var}(k)) = \neg\,\text{sgn}(k)$, if $k \in V_\forall^\psi$, and for all literals ℓ in $C' \setminus \{\neg y\}$ it holds $s_{\text{var}(k)}(\nu_{|\text{dep}_\psi(\ell)}) = \neg\,\text{sgn}(\ell)$, if $\ell \in V_\exists^\psi$, $\nu(\text{var}(\ell)) = \neg\,\text{sgn}(\ell)$, if $\ell \in V_\forall^\psi$, the resolvent $(C \otimes_y C')[s_{y_i}/y_i$ for all $i = 1, \ldots, m]$ as well evaluates to 0 for assignment ν. This contradicts the assumption that (2) is satisfiable with the Skolem functions s'_{y_i} ($C \otimes_y C'$ does not contain y and the Skolem functions s_{y_i} are equal to s'_{y_i} for all $y_i \neq y$). Thus $\varphi^{\neg y}[s_{y_i}/y_i$ for all $i = 1, \ldots, m]$ evaluates to 1 for assignment ν.

2. The second condition is dual to the first one. As Skolem functions for φ we choose $s_{y_i} = s'_{y_i}$ for all $i = 1, \ldots, m$ with $y_i \neq y$ and

$$s_y = \neg\varphi^{\neg y}[1/y][s'_{y_i}/y_i \text{ for all } i = 1, \ldots, m \text{ with } y_i \neq y].$$

3. The third condition requires that there are clauses which define the relationship $y \equiv f(V')$ and $\text{dep}_\psi(v) \subseteq \text{dep}_\psi(y)$ for all $v \in V'$. This implies that given Skolem functions for the remaining existential variables in (2), there is only one Skolem function for y, which is given by the gate definition. Therefore we can increase $\text{dep}_\psi(y)$ to $V_\forall^\psi$ without changing the truth value of the DQBF, i. e., we set the dependency set $\text{dep}_\psi(y) := V_\forall^\psi$. Then both conditions 1. and 2. are satisfied which completes the proof. □

Theorem 6 does not provide a decision algorithm for arbitrary DQBFs, since it is possible that the conditions do not hold for any existential variable. Moreover, eliminating all existential variables fulfilling the conditions of Theorem 6 is in general not feasible because the number of clauses can grow considerably during elimination. More details on addressing this problem will be found in Section 5.

The following remark shows how the number of variables that can be eliminated by resolution can be increased using dependency schemes (see Section 4.5.2):

Remark 1 *Dependency schemes (see Section 4.5.2) allow to modify the dependencies of an existential variable while preserving satisfiability by identifying so-called pseudo-dependencies. Pseudo-dependencies are dependencies between universal and existential variables which can be added to or removed from the formula without changing its satisfiability. They can be used to increase the number of existential variables that satisfy the prerequisites of Theorem 6: Before checking whether a variable y satisfies condition 1 or 2, the dependency sets of the variables that occur together with y or $\neg y$ in a clause are reduced by deleting all their identified pseudo-dependencies, and y's dependency set is increased by adding all missing pseudo-dependencies that were identified by the dependency scheme.*

Theorem 6 is formulated for the general case of DQBFs. It is interesting to note that its specialization to QBF even generalizes the QBF version known from the literature [5]. There variable elimination by resolution is only allowed if *both* conditions 1 and 2 are fulfilled, i. e., if there is no clause containing y together with variables to the right of y in the quantifier prefix. In fact, we could prove that it is sufficient to require that either condition 1 *or* condition 2 holds.

Other attempts from the literature to weaken the needed conditions failed however:

Remark 2 *In Section 3.3 of paper [29] on the QBF preprocessor SqueezeBF, a generalization of [5] has been given for QBF: The authors claim that an existential variable y may be eliminated by resolution, if none of the created resolvents contains both v and $\neg v$ for a variable v that occurs to the right of y in the QBF quantifier prefix (i. e., which has a strictly larger dependency set than y). The claim is unsound for both QBF and DQBF. Here we give a counterexample that refutes this claim.*

At first, we rephrase the claim as follows:

Claim 1 ([29]) *Let $\psi = Q : \varphi$ be a* **QBF**. *Given an existential variable y and two clauses $C_1 = \{y \vee \ell_1 \vee \cdots \vee \ell_n\}$ and $C_2 = \{\neg y \vee \ell'_1 \vee \cdots \vee \ell'_m\}$ such that $\neg \ell_i \neq \ell'_j$ when $\mathrm{dep}_\psi(y) \subsetneq \mathrm{dep}_\psi(\ell_i)$ or $\mathrm{dep}_\psi(y) \subsetneq \mathrm{dep}_\psi(\ell'_j)$, let the clause $C = \{\ell_1 \vee \cdots \vee \ell_m \vee \ell'_1 \vee \cdots \vee \ell'_m\}$ be called the* resolvent *of C_1 and C_2 (on the variable y), denoted by $C_1 \otimes_y C_2$. If φ^y (resp. $\varphi^{\neg y}$) is the set of clauses in which y (resp. $\neg y$) occurs, let Q-resolution between φ^y and $\varphi^{\neg y}$ be defined as the set of clauses*

$$\varphi^y \otimes_y \varphi^{\neg y} := \{C_y \otimes_y C_{\neg y} \mid C_y \in \varphi^y \wedge C_{\neg y} \in \varphi^{\neg y}\}.$$

Assuming we can perform the resolution of each clause in φ^y with each clause in $\varphi^{\neg y}$, we can replace the clauses in $\varphi^y \cup \varphi^{\neg y}$ with the clauses in $\varphi^y \otimes_y \varphi^{\neg y}$ and delete y and its quantifier from the prefix, resulting in an equisatisfiable problem.

Lemma 6 *Claim 1 is unsound.*

Proof. A counterexample for Claim 1 is given in Table 1.[5] The centered clauses are common to both formulas. The red clauses $(\neg y_3, y_5, y_7)$ and $(y_2, y_3, \neg y_4, y_5, y_6)$ in the left column are the only

[5] The counterexample was the result of our effort to integrate Claim 1 into HQSpre. This extension led to incorrect results for some benchmarks. In order to find the implementation bug, we used HQSpre (without

Table 1. Counterexample for out-of-order resolution (Claim 1).

	Unsatisfiable		Satisfiable	
	$\exists y_1 \exists y_2 \exists y_3 \exists y_4\ \forall x_1 \forall x_2\ \exists y_5 \exists y_6 \exists y_7 :$		$\exists y_1 \exists y_2 \exists y_4\ \forall x_1 \forall x_2\ \exists y_5 \exists y_6 \exists y_7 :$	
C_1	$(x_1, \neg x_2, \neg y_7)$			
C_2	$(\neg x_1, y_5, \neg y_6)$			
C_3	$(\neg y_2, x_1, \neg y_5)$			
C_4	$(\neg x_2, \neg y_5)$			
C_5	(y_5, y_6, y_7)			
C_6	(y_1, x_2, y_5)			
C_7	$(y_1, y_4, \neg x_1, y_5)$			
C_8	$(\neg y_1, \neg x_1, \neg y_5)$			
C_9	$(\neg x_1, x_2, y_5, \neg y_7)$			
C_{10}	$(\neg y_3, y_5, y_7)$		$(y_2, \neg y_4, y_5, y_6, y_7)$	C'_{11}
C_{11}	$(y_2, y_3, \neg y_4, y_5, y_6)$			

ones that contain y_3 or $\neg y_3$. Resolving them w. r. t. y_3 yields the red clause $(y_2, \neg y_4, y_5, y_6, y_7)$ in the right column, which is not a tautology, i. e., eliminating y_3 by resolution is allowed according to Claim 1. That means, the formula in the right column is created from the left one by eliminating y_3, and according to Claim 1 both formulas are equisatisfiable. However, the formula in the left column is unsatisfiable, the one in the right column is satisfiable.

We can see that the formula in the right column is satisfiable by using the following Skolem functions:

Variable	y_1	y_2	y_4	y_5	y_6	y_7
Skolem	0	0	1	$\neg x_2$	$\neg x_1$	x_1

It is easy to check that replacing the existential variables with their Skolem functions turns each clause into a tautology.

The formula in the left column is unsatisfiable. A Q-resolution proof that derives the empty clause from this QBF is shown in Figure 1. For the derived clauses, universal reduction [43] has been applied in each step after resolution. □

Another technique which has been proposed to be used in combination with resolution is *fork extension* [60]. Fork extension splits clauses consisting of two parts C_1 and C_2 with incomparable dependency sets, i. e., $\text{dep}_\psi(C_1) \not\subseteq \text{dep}_\psi(C_2)$ and $\text{dep}_\psi(C_2) \not\subseteq \text{dep}_\psi(C_1)$, based on the following lemma [60]:

Lemma 7 (Splitting, [60]) *Let $\psi = Q : \phi \wedge (C_1 \cup C_2)$ be a DQBF containing a clause $C_1 \cup C_2$ and let y be a new variable not occurring in ψ. Then ψ and $Q \cup \{\exists y(\text{dep}_\psi(C_1) \cap \text{dep}_\psi(C_2))\} : \phi \wedge (C_1 \cup \{y\}) \wedge (C_2 \cup \{\neg y\})$ are equisatisfiable.*

Remark 3 *Theorem 1 in [60] claims that resolution, the generalization of universal reduction [43] to DQBF (see also Section 4.4.1), and fork extension form a sound and complete proof*

the extension) to reduce the counterexample until we could check it by hand. It turned out, however, that the error was not on the implementation side.

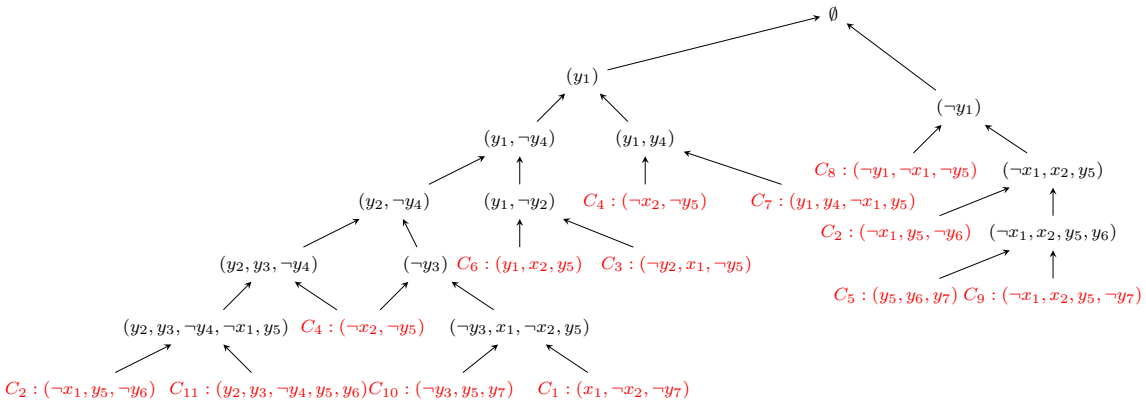

Figure 1. Q-resolution proof of unsatisfiability for the formula in Table 1 (left). The original clauses are marked in red.

calculus for DQBF. Unfortunately, the completeness part is incorrect. Nevertheless, fork extension is useful for deciding certain non-trivial subclasses of DQBF [66].

Universal expansion [10, 9, 1, 27] is the corresponding method for eliminating universal variables. Universal expansion of a universal variable x allows to remove x by introducing a copy y' for every existential variable y depending on x, which has to depend on the same variables as y. Therefore every clause in which y occurs has to be copied, too, such that y is replaced by y' in the copy. Every occurrence of x in the original part of the formula is now replaced by 1, and every occurrence in the copied part is replaced by 0 (or vice versa) resulting in an equisatisfiable formula. Universal expansion is the main operation that is used by the solver HQS [28] to transform the DQBF at hand into an equisatisfiable QBF. This QBF can be solved by an arbitrary QBF solver.

Theorem 7 (Universal expansion) *Let* $x_i \in V_\forall^\psi$, *and* $E_{x_i}^\psi = \left\{ y_j \in V_\exists^\psi \,\middle|\, x_i \in D_{y_j}^\psi \right\}$. *Then* ψ *is equisatisfiable with*

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

Proof. We write $\vDash_{\text{sat}} \psi$ here to indicate that ψ is satisfiable. To simplify notation, w. l. o. g. assume $i = 1$, i. e., we eliminate x_1. Then we have:

$$\begin{aligned}
\vDash_{\text{sat}} \psi \quad &\Leftrightarrow \quad \exists s_{y_1}(D_1), \ldots, s_{y_m}(D_m) \text{ with } \vDash_{\text{sat}} \forall x_1 \ldots \forall x_n : \varphi[s_{y_j}(D_j)/y_j \ \forall y_j \in V_\exists^\psi] \\
&\Leftrightarrow \quad \exists s_{y_1}(D_1), \ldots, s_{y_m}(D_m) \text{ with} \\
&\qquad \vDash_{\text{sat}} \forall x_2 \ldots \forall x_n : \varphi[s_{y_j}(D_j)/y_j \ \forall y_j \in V_\exists^\psi][0/x_1] \\
&\qquad\qquad \wedge \varphi[s_{y_j}(D_j)/y_j \ \forall y_j \in V_\exists^\psi][1/x_1] \\
&\Leftrightarrow \quad \exists s_{y_1}(D_1), \ldots, s_{y_m}(D_m) \text{ with} \\
&\qquad \vDash_{\text{sat}} \forall x_2 \ldots \forall x_n : \\
&\qquad \varphi[0/x_1][s_{y_k}(D_k)/y_k \ \forall y_k \in V_\exists^\psi \setminus E_{x_1}][s_{y_j}(D_j)|_{x_1=0}/y_j \ \forall y_j \in E_{x_1}] \\
&\qquad \wedge \varphi[1/x_1][s_{y_k}(D_k)/y_k \ \forall y_k \in V_\exists^\psi \setminus E_{x_1}][s_{y_j}(D_j)|_{x_1=1}/y_j \ \forall y_j \in E_{x_1}]
\end{aligned}$$

$$
\begin{aligned}
\Leftrightarrow \quad & \exists s_{y_1}(D_1), \ldots, s_{y_m}(D_m) \text{ with} \\
& \vDash_{\text{sat}} \forall x_2 \ldots \forall x_n : \Big(\varphi[0/x_1] \wedge \varphi[1/x_1][y'_j/y_j \ \forall y_j \in E_{x_1}]\Big) \\
& [s_{y_k}(D_k)/y_k \ \forall y_k \in V_\exists^\psi \setminus E_{x_1}] \\
& [s_{y_j}(D_j)|_{x_1=0}/y_j \ \forall y_j \in E_{x_1}][s_{y_j}(D_j)|_{x_1=1}/y'_j \ \forall y_j \in E_{x_1}] \\
\Leftrightarrow \quad & \vDash_{\text{sat}} \forall x_2 \ldots \forall x_n \ \underbrace{\exists y_k(D_k)}_{\text{for all } y_k \notin E_{x_1}} \ \underbrace{\exists y_j(D_j \setminus \{x_1\}) \exists y'_j(D_j \setminus \{x_1\})}_{\text{for all } y_j \in E_{x_1}} : \\
& \varphi[0/x_1] \wedge \varphi[1/x_1][y'_j/y_j \ \forall y_j \in E_{x_1}] \\
\Leftrightarrow \quad & \vDash_{\text{sat}} \forall x_2 \ldots \forall x_n \\
& \exists y_1(D_1 \setminus \{x_1\}) \ldots \exists y_m(D_m \setminus \{x_1\}) \underbrace{\exists y'_j(D_j \setminus \{x_1\})}_{\text{for all } y_j \in E_{x_1}} : \\
& \varphi[0/x_1] \wedge \varphi[1/x_1][y'_j/y_j \ \forall y_j \in E_{x_1}].
\end{aligned}
$$

□

In order to avoid unnecessary variable copies, we try to reduce the dependency sets as much as possible. Reducing dependency sets without changing the satisfiability of the DQBF can be achieved using so-called dependency schemes, see Section 4.5.

4.3 Clause Elimination Routines

Under clause elimination routines [34] we understand techniques which eliminate a clause $C \in \varphi$ such that deleting C yields an equisatisfiable formula.

4.3.1 Tautology Elimination

The simplest form of clause elimination is *tautology elimination (TE)*: A clause $C \in \varphi$ is a tautology iff $\{v, \neg v\} \subseteq C$ for some variable v. Tautological clauses can be eliminated from φ. This condition is independent from the quantifier and hence can be applied for QBF and DQBF without any restrictions.

4.3.2 Subsumption Elimination

Another well-known technique is *subsumption elimination (SE)* [5]. A clause $C \in \varphi$ is subsumed if there exists another clause $C' \in \varphi$ such that the set of occurring literals in C' is a subset of those in C, i. e., if $\exists C' \in \varphi : C' \subseteq C$. In this case, $C \wedge C'$ is logically equivalent to C', and C can be removed from φ.

Whenever we add a new clause C to the formula or strengthen a clause C, we perform a restricted version of subsumption checking called backward subsumption, which is rather inexpensive, see Section 5. We only check whether there is a clause C' already in the formula with $C \subseteq C'$. If yes, then we replace C' by C.

Subsumption can be applied without any restrictions in the same manner for QBF as for DQBF as it yields a logically equivalent matrix.

4.3.3 Blocked Clause Elimination

The concept of blocked clauses has been introduced by Kullmann [45] and used by Järvisalo et al. [41] for simplifying SAT instances. Later is has been generalized to QBF by Biere et al. in [6]. Blocked clauses can be removed from a formula without changing its truth value. Before checking whether a clause is blocked, it can be extended by so-called hidden and covered literals [31, 32, 6]. This does not change the truth value of the formula, but increases the chance that the clause is blocked.

In this section, we first generalize the notion of blocked clauses to DQBF such that blocked clauses satisfy the same properties as in SAT and QBF. In the following section, we investigate how to generalize hidden and covered literals to DQBF.

For a *QBF* $Q : \varphi \wedge C$, a clause C containing an existential literal $\ell \in C$ can be omitted (resulting in an equisatisfiable formula), if "ℓ is blocking for C", which means that for all $C' \in \varphi$ with $\neg\ell \in C'$ there is a variable k such that $\{k, \neg k\} \subseteq C \otimes_\ell C'$ and k precedes ℓ in the quantifier prefix (which means in DQBF notions: $\mathrm{dep}_\psi(k) \subseteq \mathrm{dep}_\psi(\ell)$). In the QBF context the intuitive background of blocked clause elimination is simple: Consider a solving approach to QBF which always removes the innermost existential quantifiers (which depend on all universal ones) by resolution[6.] and the innermost universal quantifiers (upon which no existential variable depends) by universal reduction until all quantifiers have been removed [3]. If ℓ is blocking for C, all resolvents resulting from C contain $\{k, \neg k\}$, i. e., are tautological, and their addition makes no contribution. The condition "k precedes ℓ in the quantifier prefix" ensures that $\mathrm{var}(k)$ has not been removed before ℓ in the process sketched above, i. e., the reason $\{k, \neg k\}$ for the resolvents being tautological has not been removed. This implies that we can alternatively remove C from $\varphi \wedge C$ in the very beginning without changing the result of the solving process.

Fortunately, we can show that the notion of blocked clauses has a natural generalization to DQBF. However, the proof idea of blocked clause elimination sketched above does not work anymore, since in DQBF there is no linear order for the quantifiers such that "removing quantifiers starting with the innermost" does not have a counterpart in DQBF; the correctness proof has to be re-done for DQBF carefully taking into account that arbitrary dependencies may be defined in a DQBF. We first give the generalized definition of blocked clauses:

Definition 11 (Blocked clauses) *Let $Q : \varphi \wedge C$ be a DQBF, C a clause, and $\ell \in C$. Literal ℓ is a* blocking literal *for C if ℓ is existential, and for all $C' \in \varphi$ with $\neg\ell \in C'$ there is a variable k such that $\{k, \neg k\} \subseteq C \otimes_\ell C'$ and $\mathrm{dep}_\psi(k) \subseteq \mathrm{dep}_\psi(\ell)$. A clause is* blocked *if it contains a blocking literal.*

Now we can prove results that are analogous to QBF and SAT.

Theorem 8 (Blocked clause elimination, BCE) *Let $Q : \varphi \wedge C$ be a DQBF with a blocked clause C. Then $Q : \varphi \wedge C$ and $Q : \varphi$ are equisatisfiable.*

Proof. If $Q : \varphi \wedge C$ is satisfiable, then $Q : \varphi$ is satisfiable as well, because φ results from $\varphi \wedge C$ by removing a conjunctive constraint. Thus, the hard part is to prove that satisfiability of $Q : \varphi$

[6.] Adding all possible resolvents with pivot variable v and then removing all clauses containing v or $\neg v$ corresponds to existential quantification of v.

implies satisfiability of $Q : \varphi \wedge C$. This holds, if it is always possible to construct Skolem functions for $Q : \varphi \wedge C$ from Skolem functions of $Q : \varphi$. Exactly this is shown in the following Lemma 8. □

Similar to the QBF case, which is described in [35], Lemma 8 is based on the notion of an "outer formula", given by the following definition:

Definition 12 (Outer clause, outer formula) *Let $\psi = Q : \varphi$ be a DQBF, $C \in \varphi$ a clause, and $\ell \in C$ a literal of C. The* outer clause *of C on ℓ is given by*

$$\mathcal{OC}(\psi, C, \ell) = \big\{k \in C \,\big|\, k \neq \ell \wedge \mathrm{dep}_\psi(k) \subseteq \mathrm{dep}_\psi(\ell)\big\}.$$

Let ℓ be a literal in a DQBF ψ. The outer formula *of ψ on ℓ is given by*

$$\mathcal{OF}(\psi, \ell) = \big\{\mathcal{OC}(\psi, D, \neg\ell) \,\big|\, D \in \varphi \wedge \neg\ell \in D\big\}.$$

Lemma 8 (Blocked clause elimination) *Let $\psi = Q : \varphi$ be a DQBF with $\varphi = \varphi' \wedge C$ and let the clause C be blocked w. r. t. the existential literal $\ell \in C$. Let $\psi' = Q : \varphi'$. If $(s'_y)_{y \in V_\exists^\psi}$ are Skolem functions for ψ', then $(s_y)_{y \in V_\exists^\psi}$ are Skolem functions for ψ where*

$$s_y = \begin{cases} s'_y, & \text{if } y \neq \mathrm{var}(\ell), \\ \mathrm{ITE}\big(\mathcal{OF}(\psi, \ell)[s'_z/z \text{ for } z \in V_\exists^\psi],\ \mathrm{sgn}(\ell),\ s'_y\big), & \text{if } y = \mathrm{var}(\ell). \end{cases}$$

Proof. Remember that ℓ is existential. Let $y^* = \mathrm{var}(\ell)$. First we have to show that s_{y^*} is an admissible Skolem function, i. e., that it depends only on variables in y^*'s dependency set. By definition, $\mathcal{OF}(\psi, \ell)$ contains only variables $v \in V$ with $\mathrm{dep}_\psi(v) \subseteq \mathrm{dep}_\psi(y^*)$. These are either universal variables upon which y^* depends or existential variables whose dependency set is a subset of D_{y^*}. Therefore the support of $\mathcal{OF}(\psi, \ell)[s'_z/z \text{ for } z \in V_\exists^\psi]$ is a subset of D_{y^*}. Furthermore, $\mathrm{sgn}(\ell)$ is a constant, and s'_{y^*} depends on D_{y^*} by assumption. Therefore s^*_y is admissible.

Now we show that $(s_y)_{y \in V_\exists^\psi}$ is a set of Skolem functions for ψ. Let $\tilde{\nu} : V_\forall^\psi \to \mathbb{B}$ be an arbitrary variable assignment of the universal variables. For ψ' we extend $\tilde{\nu}$ to a complete assignment of all variables in $V_\forall^\psi \cup V_\exists^\psi$ by setting $\nu'(x) = \tilde{\nu}(x)$ for $x \in V_\forall^\psi$ and $\nu'(y) = s'_y(\tilde{\nu}_{|\mathrm{dep}_\psi(y)})$ for $y \in V_\exists^\psi$. In the same way we extend $\tilde{\nu}$ for ψ to a complete assignment for $V_\forall^\psi \cup V_\exists^\psi$ by $\nu(x) = \tilde{\nu}(x)$ for $x \in V_\forall^\psi$ and $\nu(y) = s_y(\tilde{\nu}_{|\mathrm{dep}_\psi(y)})$ for $y \in V_\exists^\psi$. Since $(s'_y)_{y \in V_\exists^\psi}$ are Skolem functions for ψ', we know that $\nu'(\varphi') = 1$ and have to show that $\nu(\varphi) = 1$. We partition the set φ into three subsets of clauses:

$$\varphi^\ell = \{D \in \varphi \,|\, \ell \in D\},$$
$$\varphi^{\neg\ell} = \{D \in \varphi \,|\, \neg\ell \in D\},$$
$$\varphi^\emptyset = \varphi \setminus (\varphi^\ell \cup \varphi^{\neg\ell}).$$

Now we distinguish two cases:

- **Case 1:** $\nu\big(\mathcal{OF}(\psi, \ell)\big) = 1$:
  In this case we have $\nu(y^*) = s_{y^*}(\nu) = \mathrm{sgn}(\ell)$. All clauses in $\varphi^\emptyset \cup \varphi^{\neg\ell}$ are satisfied – those in $\varphi^\emptyset$ because they also appear in φ', they are independent of y^*, and ν' satisfies all clauses in φ'. The clauses in $\varphi^{\neg\ell}$ are satisfied because $\nu(\mathcal{OF}(\psi, \ell) = 1$ implies $\nu(\mathcal{OC}(\psi, D, \neg\ell)) = 1$ for all $D \in \mathcal{OF}(\psi, \ell)$ and this in turn $\nu(D) = 1$ as $\mathcal{OC}(\psi, D, \neg\ell) \subseteq D$.
  Setting $\nu(y^*) = s_{y^*}(\nu) = \mathrm{sgn}(\ell)$ satisfies all remaining clauses in φ^ℓ, which contain ℓ.

- **Case 2:** $\nu\big(\mathcal{OF}(\psi,\ell)\big) = 0$:
  Here we have $\nu(y^*) = s_{y^*}(\nu) = s'_{y^*}(\nu) = \nu'(y^*)$. By assumption, all clauses $D \in \varphi \setminus \{C\}$ are satisfied, as $\nu(z) = \nu'(z)$ for all $z \in V_\exists^\psi$ and $\varphi \setminus \{C\} = \varphi'$. We have to show that $\nu(C) = 1$. Since $\nu\big(\mathcal{OF}(\psi,\ell)\big) = 0$, there is some $D \in \varphi^{\neg l}$ with $\nu\big(\mathcal{OC}(\psi, D, \neg\ell)\big) = 0$. Since C is blocked, we can conclude that $\mathcal{OC}(\psi, C, \ell) \cup \mathcal{OC}(\psi, D, \neg\ell)$ is a tautology. That means there is a literal k such that $k \in \mathcal{OC}(\psi, C, \ell)$ and $\neg k \in \mathcal{OC}(\psi, D, \neg\ell)$. Since $\nu\big(\mathcal{OC}(\psi, D, \neg\ell)\big) = 0$, we have $\nu(\neg k) = 0$ or equivalently $\nu(k) = 1$. Since $k \in C$, $\nu(C) = 1$ holds.

Therefore $(s_z)_{z \in V_\exists^\psi}$ are Skolem functions for ψ. □

Lemma 9 *BCE for DQBF has a unique fixed point.*

Proof. The proof is similar as for BCE on SAT problems [41]: If $\psi := Q : \varphi \wedge C$ is a DQBF and C a blocked clause w. r. t. ψ, then any clause $C' \in \varphi$ which is blocked w. r. t. ψ is also blocked w. r. t. $Q : \varphi$. Therefore the result of BCE is independent of the order in which the clauses are removed, and hence BCE has a unique fixed point. □

That means that the result of elimination does not depend on the order in which the clauses are considered.

4.3.4 Hidden and Covered Literals

The purpose of the following techniques is to extend clauses by redundant literals. This increases the chance that the clause is tautological, subsumed, or blocked, and hence can be deleted. If the clause extension does not result in clause deletion, the additional literals are removed again. Exploiting hidden literals was proposed in the context of SAT in [31], covered literals in [32]. For an overview of clause elimination techniques for SAT and QBF, we refer the reader to [34].

Definition 13 (Hidden literals) *Let $Q : \varphi \wedge C$ be a DQBF. A literal $\ell \notin C$ is a* hidden literal *for C if there is a clause $\{\ell_1, \ldots, \ell_n, \neg\ell\} \in \varphi$ such that $\{\ell_1, \ldots, \ell_n\} \subseteq C$.*

Theorem 9 (Hidden literal addition, HLA) *Let $Q : \varphi \wedge C$ be a DQBF and ℓ a hidden literal for C. Then $Q : \varphi \wedge C$ and $Q : \varphi \wedge \big(C \cup \{\ell\}\big)$ are equivalent.*

The idea of hidden literal addition is based on *self-subsuming resolution* [17]. The resolvent $(C \cup \{\ell\}) \otimes_\ell \{\ell_1, \ldots, \ell_n, \neg\ell\}$ is equal to C and subsumes $C \cup \{\ell\}$. Thus after adding the resolvent C, $C \cup \{\ell\}$ can be removed, leading to an equivalent formula. Note that the argument for hidden literal addition is based on a consideration of the matrix only, thus in this case the argumentation is exactly the same as for SAT and QBF. More formally, Theorem 9 is proved as follows:

Proof. Assume that ℓ is a hidden literal for C according to Definition 13. We set $C' := C \cup \{\ell\}$. ℓ being a hidden literal for C means that there is a clause $D := \{\ell_1, \ldots, \ell_n, \neg\ell\} \in \varphi$ with $\{\ell_1, \ldots, \ell_n\} \subseteq C \subseteq C'$. For the resolvent of C' and D w. r. t. ℓ we have $C' \otimes_\ell D = C$. Adding a resolvent to a CNF yields an equivalent CNF, i. e.,

$$\varphi \wedge C' \quad \equiv \quad \varphi \wedge C' \wedge C.$$

C subsumes $C' = C \cup \{\ell\}$; therefore C' can be removed from the formula. Hence, $\varphi \wedge C$ and $\varphi \wedge C'$ are logically equivalent. Replacing the matrix of a DQBF with an equivalent formula does not change the truth value of a DQBF. □

The situation for hidden literal addition is in contrast to "covered literal addition" described in the following. For covered literals we need a careful generalization of the QBF definition together with a non-trivial proof of the generalization to DQBF.

Definition 14 (Covered literals) *Let $\psi = Q : \varphi \wedge C$ be a DQBF and let ℓ be an existential literal with $\ell \in C$. The set of* resolution candidates *for C w. r. t. ℓ is the set*

$$R_\psi(C,\ell) = \left\{C' \in \varphi \,\middle|\, \neg\ell \in C' \wedge \nexists v \in V : (\{v, \neg v\} \subseteq C \otimes_\ell C' \ \wedge\ \mathrm{dep}_\psi(v) \subseteq \mathrm{dep}_\psi(\ell))\right\}.$$

A literal k is a covered literal *for C w. r. t. ℓ if $\mathrm{dep}_\psi(k) \subseteq \mathrm{dep}_\psi(\ell)$ and $k \in \bigcap R_\psi(C,\ell) \setminus \{\neg\ell\}$.*

Theorem 10 (Covered literal addition, CLA) *Let $Q : \varphi \wedge C$ be DQBF and k a covered literal for C. Then $Q : \varphi \wedge C$ and $Q : \varphi \wedge \big(C \cup \{k\}\big)$ are equisatisfiable.*

A rough basic intuition for covered literal addition is as follows: "If a literal k is already contained in all non-tautological resolvents of a clause C with pivot literal ℓ, then k may be added to C resulting in an equisatisfiable formula." In addition to this basic idea, we need the condition $\mathrm{dep}_\psi(k) \subseteq \mathrm{dep}_\psi(\ell)$ and a bigger set of resolution candidates $R_\psi(C,\ell) = \left\{C' \in \varphi \,\middle|\, \neg\ell \in C' \wedge \nexists v \in V : (\{v, \neg v\} \subseteq C \otimes_\ell C' \ \wedge\ \mathrm{dep}_\psi(v) \subseteq \mathrm{dep}_\psi(\ell))\right\}$ instead of $R_\psi(C,\ell) = \left\{C' \in \varphi \,\middle|\, \neg\ell \in C' \wedge \nexists v \in V : \{v, \neg v\} \subseteq C \otimes_\ell C'\right\}$ in order to be able to perform the (rather involved) proof of Theorem 10.

Proof. If $Q : \varphi \wedge C$ is satisfiable, then $Q : \varphi \wedge \big(C \cup \{k\}\big)$ is satisfiable as well, because $Q : \varphi \wedge \big(C \cup \{k\}\big)$ results from $\varphi \wedge C$ by weakening a clause. Thus, the hard part is to prove that satisfiability of $Q : \varphi \wedge \big(C \cup \{k\}\big)$ implies satisfiability of $Q : \varphi \wedge C$. This holds, if it is always possible to construct Skolem functions for $Q : \varphi \wedge C$ from Skolem functions of $Q : \varphi \wedge \big(C \cup \{k\}\big)$. Exactly this is shown in the following Lemma 10. □

Just as Lemma 8, Lemma 10 is formulated based on the notion of an "outer formula", see Definition 12.

Lemma 10 (Covered literal addition) *Let $\psi = Q : \varphi$ with $\varphi = \tilde{\varphi} \wedge C$ be DQBF and k a covered literal for C w. r. t. an existential literal ℓ. Let $\psi' = Q : \varphi'$ with $\varphi' = \tilde{\varphi} \wedge \big(C \cup \{k\}\big)$.*

If $(s'_y)_{y \in V_\exists^\psi}$ are Skolem functions for ψ', then $(s_y)_{y \in V_\exists^\psi}$ are Skolem functions for ψ where

$$s_y = \begin{cases} s'_y, & \text{if } y \neq \mathrm{var}(\ell), \\ \mathrm{ITE}\big(\mathcal{OF}(\psi,\ell)[s'_z/z \text{ for } z \in V_\exists^\psi],\ \mathrm{sgn}(\ell),\ s'_y\big), & \text{if } y = \mathrm{var}(\ell). \end{cases}$$

Proof. Remember that ℓ is existential. Let $y^* = \mathrm{var}(\ell)$. For the proof that the Skolem function s_{y^*} for y^* is admissible, see the proof for blocked clause elimination (Lemma 8).

We show that $(s_y)_{y \in V_\exists^\psi}$ is a set of Skolem functions for ψ.

Let $\tilde{\nu} : V_\forall^\psi \to \mathbb{B}$ be an arbitrary variable assignment of the universal variables. For ψ' we extend $\tilde{\nu}$ to a complete assignment of all variables in $V_\forall^\psi \cup V_\exists^\psi$ by setting $\nu'(x) = \tilde{\nu}(x)$ for $x \in V_\forall^\psi$ and $\nu'(y) = s'_y(\tilde{\nu}_{|\mathrm{dep}_\psi(y)})$ for $y \in V_\exists^\psi$. In the same way we extend $\tilde{\nu}$ for ψ to a complete assignment for $V_\forall^\psi \cup V_\exists^\psi$ by $\nu(x) = \tilde{\nu}(x)$ for $x \in V_\forall^\psi$ and $\nu(y) = s_y(\tilde{\nu}_{|\mathrm{dep}_\psi(y)})$ for $y \in V_\exists^\psi$. Since $(s'_y)_{y \in V_\exists^\psi}$ are Skolem functions for ψ', we know that $\nu'(\varphi') = 1$ and have to show that $\nu(\varphi) = 1$.

We partition the set φ into three subsets of clauses:

$$\varphi^{\ell} = \{D \in \varphi \,|\, \ell \in D\},$$
$$\varphi^{\neg\ell} = \{D \in \varphi \,|\, \neg\ell \in D\},$$
$$\varphi^{\emptyset} = \varphi \setminus (\varphi^{\ell} \cup \varphi^{\neg\ell}).$$

We distinguish the following two cases:

- **Case 1:** $\nu(\mathcal{OF}(\psi, \ell)) = 1$:
  In this case $\nu(y^*) = s_{y^*}(\nu) = \mathrm{sgn}(\ell)$. All clauses in $\varphi^{\neg\ell}$ are satisfied, because $\mathcal{OF}(\psi, \ell)$ is satisfied by ν. The clauses in $\varphi^{\emptyset}$ are satisfied, because they contain neither y^* nor $\neg y^*$ and ν coincides with ν' for all variables but y^*. Finally, the clauses in φ^{ℓ} are satisfied because they contain ℓ and $\nu(\ell) = 1$.

- **Case 2:** $\nu(\mathcal{OF}(\psi, \ell)) = 0$:
  In this case $\nu(y^*) = s_{y^*}(\nu) = s'_{y^*}(\nu) = \nu'(y^*)$. As $\mathcal{OF}(\psi, \ell)$ is not satisfied by ν, there is a clause $D \in \varphi$ such that $\neg\ell \in D$ and $\mathcal{OC}(\psi, D, \neg\ell)$ is violated by ν. As $\nu = \nu'$, ν' satisfies all clauses in φ', and $\varphi \setminus \varphi' = \{C\}$, we just have to show that ν satisfies C.

  We distinguish two cases:

  - **Case 2.1:** $D \in R_{\psi}(C, \ell)$:
    Since k is a covered literal for C w. r. t. ℓ, we have that $k \in D \setminus \{\neg\ell\}$ and $\mathrm{dep}_{\psi}(k) \subseteq \mathrm{dep}_{\psi}(\ell)$, i. e., $k \in \mathcal{OC}(D, \neg\ell)$. Since $\mathcal{OC}(D, \neg\ell)$ is violated, $\nu(k) = 0$. We have $\nu(v) = \nu'(v)$ for all $v \in V$, i. e., $\nu(k) = \nu'(k)$, and ν' satisfies $C \cup \{k\}$. This implies that ν satisfies C.
  - **Case 2.2:** $D \in \varphi^{\neg\ell} \setminus R_{\psi}(C, \ell)$:
    This implies that there is a variable $v \in V$ such that $\{v, \neg v\} \in C \otimes_{\ell} D$ and $\mathrm{dep}_{\psi}(v) \subseteq \mathrm{dep}_{\psi}(\ell)$. W. l. o. g. we assume $v \in D$ and $\neg v \in C$. Since $\mathrm{dep}_{\psi}(v) \subseteq \mathrm{dep}_{\psi}(\ell)$, we have $v \in \mathcal{OC}(\psi, D, \neg\ell)$. Since ν does not satisfy $\mathcal{OC}(\psi, D, \neg\ell)$, we have $\nu(v) = 0$ and therefore $\nu(\neg v) = 1$. As $\neg v \in C$, C is satisfied by ν.

  This concludes the proof.

□

In order to reduce the size of the formula, we determine for each clause C the set H of hidden and the set K of covered literals. Then we check if $C \cup H \cup K$ is blocked, subsumed, or tautological. If this is the case, C is removed; otherwise C remains unchanged. This is iterated until we reach a fixed point.

Note that if a hidden or covered literal is universal, its addition can be helpful not only because it can make a clause blocked. If a CNF-based solver core uses elimination of universal variables to decide the formula, all clauses which contain an existential variable that depends on the eliminated universal variable have to be doubled [27]. If the clause contains the universal variable to be eliminated, one of these copies is satisfied and can therefore be omitted (cf. [36]).

4.4 Clause Strengthening Routines

Clause strengthening routines try to eliminate literals from a clause while preserving the truth value of the formula. We identify three main ways to do so.

4.4.1 Universal Reduction

Universal reduction removes a universal variable from a clause if the clause does not contain any existential variable which depends upon it. This technique has already been generalized to DQBF in [1, 21].

Lemma 11 (Universal reduction, [1, 21]) *Let $Q : \varphi \wedge C$ be a DQBF and $\ell \in C$ a universal literal such that for all $k \in C$ with $k \neq \ell$ we have $\mathrm{var}(\ell) \notin \mathrm{dep}_\psi(k)$. Then $Q : \varphi \wedge C$ and $Q : \varphi \wedge \big(C \setminus \{\ell\}\big)$ are equivalent.*

4.4.2 Self-subsuming Resolution

Self-subsuming resolution [17] identifies two clauses C_1 and C_2 with $\ell \in C_1$, $\neg\ell \in C_2$ and $C_2 \setminus \{\neg\ell\} \subseteq C_1 \setminus \{\ell\}$, i. e., C_2 "almost subsumes" C_1 with the exception of exactly one literal ℓ, which is contained in the opposite polarity. Resolution of C_1 and C_2 with the pivot literal ℓ leads to $C_\mathrm{r} = C_1 \setminus \{\ell\}$. By adding C_r to the formula, C_1 is "self-subsumed" by C_r; therefore C_1 can be removed after this addition. Our implementation simply removes ℓ from C_1, which has the same effect. This technique leads to a logically equivalent matrix and is therefore independent of the quantification type and the dependencies of the variables; hence it can be applied to QBF and DQBF without any restrictions.

4.4.3 Vivification

A further technique which replaces the matrix with a logically equivalent formula is called vivification [56]. It extensively uses Boolean constraint propagation (BCP). For a set ϕ of clauses and a set D of literals, $\mathrm{BCP}_\phi(D)$ returns the set of literals that are implied by assigning the value true to the literals in D. $\mathrm{BCP}_\phi(D) = \bot$ denotes that the matrix becomes unsatisfiable when assigning the value true to the literals in D.

Let $C = \{\ell_1, \ldots, \ell_k\} \in \varphi$ be a clause and $\emptyset \neq D \subsetneq C$. We distinguish three cases, depending on the result of calling $\mathrm{BCP}_{\varphi\setminus\{C\}}(\neg D)$:

1. If $\mathrm{BCP}_{\varphi\setminus\{C\}}(\neg D) = \bot$, we can replace C by D, because one of the literals in D must be true in order to avoid the conflict. Therefore we can add the clause D and remove C because D subsumes C.

2. If $\mathrm{BCP}_{\varphi\setminus\{C\}}(\neg D) = L$ with some $\ell_j \in L \cap (C \setminus D)$, we can replace C by $D \cup \{\ell_j\}$. For each satisfying assignment of φ either a literal in D is true and therefore $D \cup \{\ell_j\}$, or all literals in D are false, therefore ℓ_j is true and thus $D \cup \{\ell_j\}$ is true. In this case we can alternatively choose not to strengthen C, but to eliminate the clause C, since for each satisfying assignment of $\varphi \setminus \{C\}$ either a literal in D is true and therefore also C, or all literals in D are false, but then C is satisfied by ℓ_j.

3. If $\mathrm{BCP}_{\varphi\setminus\{C\}}(\neg D) = L$ with $\neg\ell_j \in L$ for some $\ell_j \in C \setminus D$, then we can delete ℓ_j from C. For each satisfying assignment of φ either a literal in D is true and therefore $C \setminus \{\ell_j\}$, or all literals in D are false, therefore ℓ_j is false and thus $C \setminus \{\ell_j\}$ has to be satisfied as well.

Since vivification replaces the formula's matrix by an equivalent matrix, it can be applied to SAT, QBF, and DQBF.

4.5 Reduction of Dependency Sets

In a DQBF, a universal variable $x \in V_\forall^\psi$ may be contained in the dependency set D_y^ψ of an existential variable $y \in V_\exists^\psi$, but actually, due to the structure of the matrix, the (potential) Skolem functions for y do not need to exploit the information about x's value to satisfy the formula. If such a situation is detected, x can be removed from D_y^ψ. At the first sight removing universal variables from dependency sets does not simplify the DQBF, but if we consider this operation when universal expansion according to Theorem 7 is used for solving a DQBF, it potentially reduces the number of copies of existential variables.

Definition 15 *An existential variable $y \in V_\exists^\psi$ is* independent *of a universal variable $x \in V_\forall^\psi$ if either $x \notin D_y^\psi$ or replacing D_y^ψ by $D_y^\psi \setminus \{x\}$ does not change the truth value of ψ.*

An example for a situation when dependency sets may be reduced is when a circuit is transformed into CNF by Tseitin transformation. The dependency set D_y^ψ of a Tseitin variable y can be an arbitrary superset of the universal variables in its cone-of-influence. The variables in D_y^ψ that are not in the cone-of-influence of y can be removed from D_y^ψ without affecting the truth value of the formula.

Deciding whether two variables are independent has the same complexity as deciding the DQBF itself:

Theorem 11 *Let $\psi = Q : \varphi$ be a DQBF, $x \in V_\forall^\psi$, and $y \in V_\exists^\psi$ with $x \in D_y^\psi$. Deciding whether y is independent of x is NEXPTIME-complete.*

Proof. Checking whether y is independent of y can be done by first solving ψ and then ψ without the dependency of y on x. Iff both formulas have the same truth value, y is independent of x. Since solving a DQBF is in NEXPTIME, deciding independence is in NEXPTIME as well.

For showing the NEXPTIME-hardness, we reduce deciding satisfiability of a DQBF to checking independence of two variables. Let $\psi = Q : \varphi$ be the DQBF whose satisfiability is to be determined. Let a, b be fresh variables that do not appear in ψ and consider the formula $\psi' := \forall a\, Q \exists b(a) : \varphi \wedge (a \equiv b)$. We check whether a and b are independent. If ψ is unsatisfiable, then ψ' is unsatisfiable as well – independent of whether b depends on a or not. Therefore a and b are independent. If ψ is satisfiable, then ψ' is satisfied if we use $s_b(a) = a$ as additional Skolem function for b, i. e., if b is allowed to depend on a. In this case a and b are not independent. So we can use the check for independence to decide satisfiability of DQBFs. □

The corresponding theorem for QBF – deciding independence of two variables in a QBF is PSPACE-complete – has been shown with the same argumentation in [64].

Because of the high complexity one resorts to sufficient criteria to show independence. Here we consider two methods for reducing dependency sets: the detection of gate definitions and dependency schemes.

4.5.1 Detecting Gate Definitions for Dependency Set Reduction

Suppose the DQBF matrix in CNF encodes a relationship $y \equiv f(\ell_1, \ldots, \ell_k)$ and the conditions of Theorem 2 are satisfied, i. e., $y \in V_\exists^\psi$ and $\mathrm{dep}_\psi(\ell_i) \subseteq \mathrm{dep}_\psi(y)$ for $i = 1, \ldots, k$. If $\bigcup_{i=1}^k \mathrm{dep}_\psi(\ell_i) \subsetneq D_y^\psi$, then D_y^ψ can be replaced by $\bigcup_{i=1}^k \mathrm{dep}_\psi(\ell_i)$. The proof that we can replace D_y^ψ by $\bigcup_{i=1}^k \mathrm{dep}_\psi(\ell_i)$ is easy: If there are Skolem functions for the DQBF ψ, then the

constraint $y \equiv f(\ell_1, \ldots, \ell_k)$ enforces that s_y results from applying f to (possibly negated) Skolem functions $s_{\mathrm{var}(\ell_i)}$ $(1 \leq i \leq k)$.[7] Thus, the Skolem function s_y only depends on $\bigcup_{i=1}^{k} \mathrm{dep}_\psi(\ell_i)$.

4.5.2 Dependency Schemes

Another simple sufficient criterion to show independence is based on the incidence graph of the matrix:

Definition 16 *The* variable-clause incidence graph $G_{V,\varphi} = (V \cup \varphi, E)$ *of the formula is an undirected graph with* $E = \big\{\{v, C\} \in V \times \varphi \,\big|\, v \in C \vee \neg v \in C\big\}$.

The following theorem generalizes a theorem from [64]:

Theorem 12 (Standard dependency scheme) *An existential variable $y \in V_\exists^\psi$ is independent of a universal variable $x \in V_\forall^\psi$ if there is no path in $G_{V,\varphi}$ between x and y, visiting only variables in $\{z \in V_\exists^\psi \,|\, x \in D_z^\psi\}$ in between.*

Instead of providing a direct proof of Theorem 12, we will conclude its correctness from the more general Theorem 13 given below.

In the QBF context more powerful dependency schemes have been devised, which can possibly identify more variables as independent, see, e. g., [64, 63, 48, 24, 70, 72]. Among these dependency schemes the "reflexive quadrangle resolution path dependency scheme" [72, 80] is currently the most effective dependency scheme that is sound for both QBF and DQBF. Thus, this scheme is used in HQSpre for reducing dependency sets and saving variable copies during universal expansion.

For defining the reflexive quadrangle resolution path dependency scheme we have to define "connections" between clauses which go beyond paths in $G_{V,\varphi}$:

Definition 17 (Resolution path connected) *Let $\psi = Q : \varphi$ be a DQBF. A* resolution Z-path *for $Z \subseteq V$ between two clauses $C, C' \in \varphi$ is a sequence $C_1, \ldots, C_n$ of clauses with $C = C_1$, $C' = C_n$ such that for all $1 \leq i < n$ there is a literal l_i with $\mathrm{var}(l_i) \in Z$, $l_i \in C_i$, $\neg l_i \in C_{i+1}$, and for all $1 \leq i < n-1$ we have $\mathrm{var}(l_i) \neq \mathrm{var}(l_{i+1})$.*[8] *The sequence $\mathrm{var}(l_1), \ldots, \mathrm{var}(l_{n-1})$ is called a* connecting sequence *of the resolution Z-path. Two clauses $C, C' \in \varphi$ are* resolution path connected *w. r. t. Z (written $C \overset{Z}{\underset{\mathrm{rp}}{\longleftrightarrow}} C'$) if there is a resolution Z-path between C and C'.*

Definition 18 (Reflexive Quadrangle Resolution Path Dependency Scheme) *Let ψ be a DQBF. Furthermore, let $x^* \in V_\forall^\psi$ and $y^* \in V_\exists^\psi$ such that $x^* \in D_{y^*}^\psi$. We set $Z_{x^*} := \{z \in V_\exists^\psi \,|\, x^* \in D_z^\psi\}$.*

[7] If $\mathrm{var}(\ell_j)$ is universal, we define $s_{\mathrm{var}(\ell_j)} = \mathrm{var}(\ell_j)$.

[8] In [24] there is an additional constraint "the resolvent of C_i and C_{i+1} w. r. t. l_i is non-tautological". This constraint has to be removed according to [70, 71]. If it is not removed, resolution path dependencies are not sound.

For the reflexive quadrangle resolution path dependency scheme, $(x^*, y^*) \in \mathrm{rqdep}^{\mathrm{rp}}(\psi)$ *holds iff there are clauses* $C_1, C_2, C_3, C_4 \in \varphi$ *with* $x^* \in C_1$, $\neg x^* \in C_2$, $y^* \in C_3$, *and* $\neg y^* \in C_4$ *such that*

$$(C_1 \xleftrightarrow[\mathrm{rp}]{Z_{x^*}} C_3 \wedge C_2 \xleftrightarrow[\mathrm{rp}]{Z_{x^*}} C_4) \vee (C_1 \xleftrightarrow[\mathrm{rp}]{Z_{x^*}} C_4 \wedge C_2 \xleftrightarrow[\mathrm{rp}]{Z_{x^*}} C_3).$$

The next step is to prove that $\mathrm{rqdep}^{\mathrm{rp}}$ is sound for DQBF.

Theorem 13 *Let* $x^* \in V_\forall^\psi$ *and* $y^* \in V_\exists^\psi$ *such that* $x^* \in D_{y^*}^\psi$. *If* $(x^*, y^*) \notin \mathrm{rqdep}^{\mathrm{rp}}(\psi)$, *then* y^* *is independent of* x^*.

For the proof, we assume that y^* is *not* independent of x^* and show that then $(x^*, y^*) \in \mathrm{rqdep}^{\mathrm{rp}}(\psi)$. The main idea of the proof consists of using universal expansion on the DQBF until the following Theorem 14 on the "quadrangle resolution path dependency scheme" $\mathrm{qdep}^{\mathrm{rp}}$ for QBFs becomes applicable. This result then implies $(x^*, y^*) \in \mathrm{rqdep}^{\mathrm{rp}}(\psi)$.

The quadrangle resolution path dependency scheme[9.] is similar to the reflexive quadrangle resolution path dependency scheme with the only difference that the considered resolution paths are defined w. r. t. $Z_{x^*} \setminus \{y^*\}$ instead of Z_{x^*}, i. e., the connecting sequences of those paths are not allowed to contain y^*. Thus, the quadrangle resolution path dependency scheme is defined as follows:

Definition 19 (Quadrangle Resolution Path Dependency Scheme)
Let ψ *be a DQBF. Furthermore, let* $x^* \in V_\forall^\psi$ *and* $y^* \in V_\exists^\psi$ *such that* $x^* \in D_{y^*}^\psi$. *We set* $Z_{x^*} := \{z \in V_\exists^\psi \mid x^* \in D_z^\psi\}$.

For the quadrangle resolution path dependency scheme, $(x^*, y^*) \in \mathrm{qdep}^{\mathrm{rp}}(\psi)$ *holds iff there are clauses* $C_1, C_2, C_3, C_4 \in \varphi$ *with* $x^* \in C_1$, $\neg x^* \in C_2$, $y^* \in C_3$, *and* $\neg y^* \in C_4$ *such that*

$$(C_1 \xleftrightarrow[\mathrm{rp}]{Z_{x^*}\setminus\{y^*\}} C_3 \wedge C_2 \xleftrightarrow[\mathrm{rp}]{Z_{x^*}\setminus\{y^*\}} C_4) \vee (C_1 \xleftrightarrow[\mathrm{rp}]{Z_{x^*}\setminus\{y^*\}} C_4 \wedge C_2 \xleftrightarrow[\mathrm{rp}]{Z_{x^*}\setminus\{y^*\}} C_3).$$

The following theorem for QBFs [24, 70] is then needed for proving Theorem 13:

Theorem 14 ([24, 70]) *Let* $\psi = Q : \varphi$ *be a* ***QBF****. Let* $x^* \in V_\forall^\psi$ *and* $y^* \in V_\exists^\psi$ *such that* $x^* \in D_{y^*}^\psi$. *If* $(x^*, y^*) \notin \mathrm{qdep}^{\mathrm{rp}}(\psi)$ *and the DQBF which results from* ψ *by removing* x^* *from* $D_{y^*}^\psi$ *is a* ***QBF*** *as well, then* y^* *is independent of* x^*.

Before we finally come to the proof of Theorem 13, we have to consider the following technical lemma on DQBFs resulting from universal expansion:

Lemma 12 *Let* $\psi = Q : \varphi$ *be a DQBF and* $\psi' = Q' : \varphi'$ *a DQBF derived from* ψ *by universally expanding some variables in* ψ. *If there exists a resolution* Z*-path from* $C_1 \in \varphi'$ *to* $C_2 \in \varphi'$, *then the connecting sequence* $y_1, \ldots, y_n$ *(see Definition 17) does not include a pair* (y_i, y_{i+1}) *where* y_i *and* y_{i+1} *are two copies of the same existential variable in* ψ.

[9.] The quadrangle resolution path dependency scheme has originally been introduced as "resolution path dependency scheme" in [24, 70, 71]. For a clearer categorization, however, we prefer to call it the "quadrangle resolution path dependency scheme" $\mathrm{qdep}^{\mathrm{rp}}$.

Proof. Assume that the resolution Z-path has a connecting sequence with a pair (y_i, y_{i+1}) where y_i and y_{i+1} are two copies of the same existential variable in ψ. According to the definition of resolution Z-paths, $y_i \neq y_{i+1}$, i. e., y_i and y_{i+1} are two *different* copies of the same existential variable in ψ. Then there exists a clause $C \in \varphi'$ with $y_i, y_{i+1} \in \mathrm{var}(C)$, $y_i \neq y_{i+1}$. This contradicts the definition of universal expansion (see Theorem 7), since clauses in a universal expansion can contain at most one copy of the same original variable. □

Now we come to the proof of Theorem 13. In the proof we use the notion $\psi \ominus (x, y)$ for $x \in V_\forall^\psi$ and $y \in V_\exists^\psi$. $\psi \ominus (x, y)$ denotes the formula that results from ψ by removing the dependency of y on x, i. e., by replacing D_y with $D_y \setminus \{x\}$.

Proof. (Theorem 13) Let $\psi = \forall x_1 \ldots \forall x_n \exists y_1(D_{y_1}) \ldots \exists y_m(D_{y_m}) : \varphi$ be a DQBF. W. l. o. g. assume that $x_1 \in D_{y_1}$ and y_1 is not independent of x_1. We have to prove that $(x_1, y_1) \in \mathrm{rqdep}^{\mathrm{rp}}(\psi)$.

Since, in ψ, y_1 is not independent of x_1, ψ has to be satisfiable and $\psi \ominus (x_1, y_1)$ is unsatisfiable. In ψ, we universally expand on the remaining universal variables $x_2, \ldots, x_n$. Let $\boldsymbol{y}_{\mathbf{indep}}$ be the copies of existential variables y_i ($i \in \{2, \ldots, m\}$) with $x_1 \notin D_{y_i}$, $\boldsymbol{y}_{\mathbf{dep}}$ be the copies of existential variables y_i ($i \in \{2, \ldots, m\}$) with $x_1 \in D_{y_i}$, and $y_1^1, \ldots, y_1^k$ the copies of y_1 with $k = 2^{|D_{y_1}|-1}$. Thus, universal expansion results in the following **QBF** ψ':

$$\psi' = \exists \boldsymbol{y}_{\mathbf{indep}} \forall x_1 \exists y_1^1 \ldots \exists y_1^k \exists \boldsymbol{y}_{\mathbf{dep}} : \varphi'.$$

Now we start to move copies y_1^i from the right of x_1 to the left of x_1. Since $\psi \ominus (x_1, y_1)$ is unsatisfiable, ψ' will get unsatisfiable when *all* $y_1^1 \ldots y_1^k$ have been moved to the left of x_1. So there exists a maximal subset $\boldsymbol{y}_{\mathbf{1}}^{\mathbf{mov}} \subsetneq \{y_1^1, \ldots, y_1^k\}$ (with $\boldsymbol{y}_{\mathbf{1}}^{\mathbf{stay}} = \{y_1^1, \ldots y_1^k\} \setminus \boldsymbol{y}_{\mathbf{1}}^{\mathbf{mov}}$ containing the remaining variables from $\{y_1^1 \ldots y_1^k\}$) with the property that

$$\psi'' = \exists \boldsymbol{y}_{\mathbf{indep}} \exists \boldsymbol{y}_{\mathbf{1}}^{\mathbf{mov}} \forall x_1 \exists \boldsymbol{y}_{\mathbf{1}}^{\mathbf{stay}} \exists \boldsymbol{y}_{\mathbf{dep}} : \varphi'$$

is satisfiable and, in ψ'', each variable from $\boldsymbol{y}_{\mathbf{1}}^{\mathbf{stay}}$ is *not* independent of x_1. That means that moving an arbitrary variable from $\boldsymbol{y}_{\mathbf{1}}^{\mathbf{stay}}$ to the left of x_1 will turn ψ'' from satisfiable to unsatisfiable. The set $\boldsymbol{y}_{\mathbf{1}}^{\mathbf{stay}}$ is not empty, since otherwise y_1 would be independent of x_1 in ψ. Choose an arbitrary existential variable y_1^j in $\boldsymbol{y}_{\mathbf{1}}^{\mathbf{stay}}$. Since ψ'' is a QBF where y_1^j is not independent of x_1, we have $(x_1, y_1^j) \in \mathrm{qdep}^{\mathrm{rp}}(\psi'')$ due to Theorem 14, i. e., $\exists C_1, C_2, C_3, C_4 \in \varphi'$ with $x_1 \in C_1$, $\neg x_1 \in C_2$, $y_1^j \in C_3$, $\neg y_1^j \in C_4$, and – w. l. o. g. – $C_1 \xleftrightarrow[\mathrm{rp}]{Z_{x_1} \setminus \{y_1^j\}} C_3$ and $C_2 \xleftrightarrow[\mathrm{rp}]{Z_{x_1} \setminus \{y_1^j\}} C_4$ with $Z_{x_1} = \boldsymbol{y}_{\mathbf{1}}^{\mathbf{stay}} \cup \boldsymbol{y}_{\mathbf{dep}}$.

Now we make use of a simple property of the process of universal expansion in order to turn the two constructed resolution $(Z_{x_1} \setminus \{y_1^j\})$-paths for ψ'' into suitable resolution paths for the original DQBF ψ proving that $(x_1, y_1) \in \mathrm{rqdep}^{\mathrm{rp}}(\psi)$: Each clause C in a universal expansion can be mapped back to a clause C^{orig} of the original formula "it results from" by (1) adding universal literals which have been removed by the universal expansion, since they are unsatisfied in the copy of the clause at hand, and by (2) replacing copies of existential literals ℓ by the existential literals ℓ^{orig} in the original formula. Now consider one of the constructed resolution $(Z_{x_1} \setminus \{y_1^j\})$-paths $D_1, \ldots, D_n$ with $D_1 = C_1$ and $D_n = C_3$ (or $D_1 = C_2$ and $D_n = C_4$). For all $1 \leq i < n$ there is a literal ℓ_i with $\mathrm{var}(\ell_i)$ from $Z_{x_1} \setminus \{y_1^j\} = \boldsymbol{y}_{\mathbf{dep}} \cup (\boldsymbol{y}_{\mathbf{1}}^{\mathbf{stay}} \setminus \{y_1^j\})$, $\ell_i \in C_i$, $\neg \ell_i \in C_{i+1}$. It is easy to see that $D_1^{\mathrm{orig}}, \ldots, D_n^{\mathrm{orig}}$ is a resolution Z'_{x_1}-path for ψ with $Z'_{x_1} = \{y \in V_\exists^\psi \mid x_1 \in D_y\}$: For all $1 \leq i < n$ there is a literal ℓ_i^{orig} with $\mathrm{var}(\ell_i^{\mathrm{orig}}) \in Z'_{x_1}$, $\ell_i^{\mathrm{orig}} \in C_i$, $\neg \ell_i^{\mathrm{orig}} \in C_{i+1}$. (Note that ℓ_i^{orig} may also be y_1 or $\neg y_1$, since the connecting sequence of $D_1, \ldots, D_n$ may contain a copy of y_1 which is different from y_1^j.) Due to Lemma 12, $\mathrm{var}(\ell_i)$ and $\mathrm{var}(\ell_{i+1})$ are copies of *different* existential variables and thus $\mathrm{var}(\ell_i^{\mathrm{orig}}) \neq \mathrm{var}(\ell_{i+1}^{\mathrm{orig}})$ for all $1 \leq i < n$. Altogether we have proven that there are two resolution Z'_{x_1}-paths for ψ with $Z'_{x_1} = \{y \in V_\exists^\psi \mid x_1 \in D_y\}$ leading from $C_1^{\mathrm{orig}} \in \varphi$ with $x_1 \in C_1^{\mathrm{orig}}$ to

$C_3^{\text{orig}} \in \varphi$ with $y_1 \in C_3^{\text{orig}}$ and from $C_2^{\text{orig}} \in \varphi$ with $\neg x_1 \in C_2^{\text{orig}}$ to $C_4^{\text{orig}} \in \varphi$ with $\neg y_1 \in C_4^{\text{orig}}$. Thus $(x_1, y_1) \in \text{rqdep}^{\text{rp}}(\psi)$.[10.] □

Now we can reduce the proof of Theorem 12 to Theorem 13:

Proof. (Theorem 12) The correctness of Theorem 12 follows easily from Theorem 13. According to Definition 18 of the reflexive quadrangle resolution path dependency scheme and Definition 17 of "resolution path connected", $(x^*, y^*) \in \text{rqdep}^{\text{rp}}(\psi)$ directly implies that there is a path in $G_{V,\varphi}$ from x^* to y^* that visits only variables in $\{z \in V_\exists^\psi \mid x^* \in D_z^\psi\}$ in between. That means, in case that there is no such path, we conclude $(x^*, y^*) \notin \text{rqdep}^{\text{rp}}(\psi)$ and thus y^* is independent of x^* according to Theorem 13. □

5. Implementation of HQSpre

While the previous section provided the theoretical background of preprocessing for (D)QBF, this section looks into the question how the presented techniques are combined into an efficient tool. For an efficient implementation several factors are essential: Appropriate heuristics which determine whether, when, and how intensively certain techniques are applied, an appropriate order of the techniques, efficient data structures, and an adaption to different solver cores, to name just a few criteria.

5.1 Data structures

We start with some remarks on the data structures used in our implementation.

A clause is essentially stored as a sorted array of literals. For an efficient access to all clauses in which a literal ℓ occurs, we keep complete occurrence lists for each literal. Furthermore, we redundantly hold for each literal ℓ a list of all binary clauses in which $\neg\ell$ occurs, since many of our syntactic methods, such as gate and equivalence detection, employ binary clauses.

We re-use unused variable IDs, i. e., whenever a variable was removed, we mark the index as "open" such that it can be re-used. This avoids very large variable IDs and gaps in the data structure, which is crucial during universal expansion where many existential variables are newly introduced as a copy.

We tested different data structures for manipulating clauses. Structures based on `std::set` have the advantage of sorted ranges, which is beneficial for, e. g., subsumption and hidden/covered literal addition, but comes with the downside of more expensive access and insertion costs. On the other hand, a `std::vector` has constant access time, but checking the occurrence of a literal in a clause gets more expensive. To overcome this issue we implemented a data structure which marks already occurring literals in the current clause. This "seen" data structure is also implemented as a `std::vector` with the length of the maximal literal ID. By doing so, we have efficient access to clause data, and checking whether a literal occurs in the clause or whether a resolvent is a tautology becomes very cheap (linear in the length of the involved clauses). By using this structure, we have measured a speed-up compared to `std::set`-based clauses of up to a factor of 4.

[10.] The construction does *not* lead to $(Z'_{x_1} \setminus \{y_1\})$-paths and thus $(x_1, y_1) \in \text{qdep}^{\text{rp}}(\psi)$ cannot be proven.

Algorithm 1 Outline of the main preprocessing routine. Note that after each routine, a fast formula simplification procedure is called. Universal reduction and subsumption checks are performed for each added or modified clause.

Preprocess((D)QBF $\psi = Q : \varphi$)
begin
    simplify(ψ) (1)
    iteration $\leftarrow$ 1 (2)
    repeat (3)
        if iteration $\leq$ 2 **then** (4)
            gates $\leftarrow$ gateDetection(ψ) (5)
            gateSubstitutionAndRewriting(ψ, gates); simplify(ψ) (6)
        end if (7)
        eliminateClauses(ψ); simplify(ψ) $\triangleleft$ *Hidden/covered TE/SE/BCE* (8)
        selfsubsumingResolution(ψ); simplify(ψ) (9)
        vivify(ψ); simplify(ψ) (10)
        variableEliminationByResolution(ψ); simplify(ψ) (11)
        syntacticConstants(ψ); simplify(ψ) (12)
        if first iteration **then** (13)
            semanticConstants(ψ); simplify(ψ) (14)
            trivialMatrixChecks(ψ) (15)
        end if (16)
        universalExpansion(ψ); simplify(ψ) (17)
        iteration $\leftarrow$ iteration + 1 (18)
    until ψ has not been changed anymore or ψ is decided (19)
    return ψ (20)
end

5.2 Algorithmic overview

Now we have a closer look into the actual preprocessing routine.

First of all, we always apply tautology checks, universal reduction, and backward subsumption checks when we insert a clause C into the clause database or when we strengthen a clause C as a result of some preprocessing step. Backward subsumption looks for clauses C' with $C \subseteq C'$ that are already included in the clause database. Apparently, we can restrict the search for such a clause C' to the shortest occurrence list among the literals in C.

In Algorithm 1 we give an overview of the main preprocessing routine, which calls the different techniques in a loop until the formula does not change anymore.

The basic (syntactic) detection and propagation of constant and pure literals as well as equivalent literals (see Section 4.2.1) can be (and were) implemented very efficiently and turned out to be a necessary feature to let preprocessing scale. Hence, in our implementation we apply these three methods[11.] after each and every more complex technique until a fixed-point is reached. This is referred to as the `simplify()` method in Algorithm 1. `simplify()` is also called in the very beginning of the preprocessing algorithm.

[11.] The syntactic constant detection using transitive implication chains is *not* included.

After eliminating unit, pure, and equivalent literals, we start the main preprocessing loop by applying gate detection (see Algorithm 1, line 5 and Section 4.1). By looking for certain clause patterns, we identify definitions of logical gates within the formula. In particular, we seek for AND gates with an arbitrary number of inputs, 2-input XOR gates [57], and multiplexers. For all of them we allow arbitrary negations on both inputs and output.[12.] As many (D)QBF instances result from applications with circuits, the number of detectable gates can be very large. It is important that this technique is applied early, since many other methods, in particular the clause elimination methods, might eliminate gate defining clauses. It is also important to note that the exploitation of gate definitions has to be completely different depending on the solver used after preprocessing:

- If the subsequent solver works on a matrix in CNF representation, we make use of gate substitution followed by re-transformation into CNF. As already mentioned in Section 4.2.3, substitution and transformation into CNF is simulated by a series of resolutions w. r. t. the defined variable [17]. Since this step may lead to a huge number of resolvents, it is only performed if the formula does not grow above a user-given bound. More precisely, we proceed as follows, when we have to resolve (w. r. t. ℓ) all clauses from a clause set S_0 with all clauses from a clause set S_1, followed by removing S_0 and S_1: First of all, we remember the size $size_{\text{orig}}$ of $S_0 \cup S_1$, counted by the number of literals in $S_0 \cup S_1$. Then we estimate the size $size_{\text{est}}$ of $S_0 \otimes_\ell S_1$ from above just by computing the number of literals which would result from producing all resolvents of pairs in S_0 and S_1. If $size_{\text{est}} > \varepsilon_0 \cdot size_{\text{orig}} + c_0$, we do not perform resolutions at all and leave S_0 and S_1 unchanged in the CNF (in our implementation we use $\varepsilon_0 = 1.0$ and $c_0 = 200$). Otherwise we have a closer look into the resolution process: Tentatively, we perform resolutions between S_0 and S_1 without counting literals in tautological clauses or literals that will be removed by universal reduction. If the current size $size_{\text{current}}$ during this process exceeds $\varepsilon_1 \cdot size_{\text{orig}} + c_1$, we stop again. If we can handle all pairs in $S_0 \times S_1$ in that way without exceeding $\varepsilon_1 \cdot size_{\text{orig}} + c_1$, we add the resolvents to the CNF and remove S_0 and S_1 (in our implementation we use $\varepsilon_1 = 1.0$ and $c_1 = 100$). If the elimination of gate definitions failed due to size limits, gate rewriting is used instead as mentioned in Section 4.2.3.

- If the subsequent solver works with a circuit (e. g., AIG) representation instead of CNF, the detected gate definitions are used to transform the CNF into a circuit representation after preprocessing. For this reason we have to protect the detected gate-defining clauses from being removed by subsequent preprocessing steps. This is done by applying the concept of frozen variables and clauses [46] for gate definitions, i. e., these variables and clauses are excluded from elimination methods (with the exception of unit, pure, and equivalent literal detection).

The next step in the algorithm applies all clause elimination routines (tautology elimination, subsumption elimination, and blocked clause elimination supported by hidden / covered literal addition, see Section 4.3) in a loop until a fixed-point is reached, i. e., until no further changes to the formula can be made (see Algorithm 1, line 8). To do so, we keep a queue of clause candidates, which are updated after removing a clause from the formula.

[12.] Note, this covers also OR gates with arbitrarily negated inputs and output.

Whenever a clause $c = \{\ell_1, \ldots, \ell_n\}$ has been removed, every clause in which at least one of the literals $\ell_1, \neg\ell_1, \ldots, \ell_n, \neg\ell_n$ occurs becomes a new candidate to be removed by one of the above methods.

Afterwards, the algorithm applies self-subsuming resolution (Algorithm 1, line 9, see also Section 4.4.2). In our implementation, we iterate over all clauses in order to identify such self-subsumptions until a fixed-point is reached. To do so efficiently, we keep a queue of candidates that is updated after deleting a literal. Whenever a literal ℓ_i is removed from a clause $C = \{\ell_1, \ldots, \ell_n\}$, each clause containing at least one of $\neg\ell_1, \ldots, \neg\ell_{i-1}, \neg\ell_{i+1}, \ldots, \neg\ell_n$ is potentially self-subsuming with C.

Subsequently, we try to further strengthen the clauses one after the other by applying vivification (cf. Section 4.4.3). We only consider clauses having a minimum length (in our experiments: at least 4 literals). For efficiency reasons, we implemented BCP for vivification using the two watched literals scheme [53], which is standard in modern SAT solvers. It determines all unit clauses and detects conflicts resulting from the assignment of a subset of the formula's variables without modifying the formula. As an alternative to SAT-based BCP, one could use the watched literal schemes that have been developed for QBF [25] and DQBF [20], which take universal reduction into account. They can generally identify more implications, but are also more costly.

When vivification considers a clause $C \in \varphi$, we select literals from C in a random order (in fact, we use a fixed random seed in order to ensure reproducibility) and propagate one at a time until BCP's result either allows us to delete or shorten C or all literals of C have been considered.

The third clause strengthening method, universal reduction, does not appear explicitly in Algorithm 1, since it is applied eagerly for every added clause as well as for every clause that was strengthened.

The remaining operations in lines 11 to 17 of the main loop in Algorithm 1 are variable elimination routines which are able to fix the truth values of variables v and propagate them through φ using BCP. In contrast to the BCP implementation used in vivification, which does not modify the formula, here we simplify the formula by deleting satisfied clauses and unsatisfied literals.

The first one is variable elimination by resolution. Variable elimination by resolution is not performed for every eligible existential variable according to Theorem 6, since it may lead to a considerable blow-up of the CNF size. Rather, we use exactly the same size estimation as described for the special case of gate substitution by resolution and apply it only when the size of the formula does not grow beyond a threshold.

We exploit a special case of resolution which is efficiently detected and always leads to a smaller CNF. Therefore, we perform these resolutions more frequently – namely during BCP and blocked clause elimination. If an existential literal $\ell^\exists$ only occurs in exactly one binary clause $\{\ell^\exists, \ell\}$, then resolution of the pivot literal $\ell^\exists$ yields resolvents in which $\neg\ell^\exists$ is replaced by ℓ w. r. t. the original clauses. In our implementation we simply remove the clause $\{\ell^\exists, \ell\}$ and replace $\neg\ell^\exists$ with ℓ in every clause. According to Theorem 6, elimination of $\ell^\exists$ by resolution is sound as long as $\mathrm{var}(\ell)$ is also existential and $D_{\mathrm{var}(\ell)} \subseteq D_{\mathrm{var}(\ell^\exists)}$ or $\mathrm{var}(\ell)$ is universal and $\mathrm{var}(\ell^\exists)$ depends on it. Interestingly, for soundness $\neg\ell^\exists$ can occur in an arbitrary number of clauses and together with arbitrary literals, since we could prove that

either condition 1 *or* condition 2 of Theorem 6 has to hold, not both as in QBF versions of the theorem known from the literature [5].

In the next step (line 12 of Algorithm 1), the syntactic constant detection or backbone detection using transitive implication chains in binary clauses (based on Lemma 4) is performed, followed by eliminating unit, pure, and equivalent literals as usual.

We detect "semantic constants" (line 14), i. e., backbones and monotonic variables, using incremental SAT solver calls as described in Section 4.2.1 only in the first iteration of the main loop (due to the cost of this operation).

Due to the needed cost, the SAT-based checks for trivial satisfiability and unsatisfiability (see Section 3.1 and line 15 of the algorithm) are performed only in the first main loop iteration as well.

As a last step in the main loop we apply universal expansion (Section 4.2.4 and line 17). In our DQBF benchmark set, the number of depending existential variables is often very large and therefore we obtain a huge blow-up of the formula. Hence, in our implementation we do not apply universal expansion for DQBF.[13.] In contrast, in QBF many benchmark classes have quite small universal quantifier blocks. In this case, it turns out that the elimination of a complete universal block is often very beneficial, whereas expansion of single variables in large blocks does have a rather small impact. Therefore, we try to expand blocks with small sizes (< 20). We always try to expand the whole block as long as the blow-up of the formula is at most 50 % per variable and less than 100 % in total. After each expansion step we also apply variable elimination by resolution in order to reduce the potential number of copied existential variables in the next steps as suggested in [11]. During universal expansion, we utilize the reflexive quadrangle resolution path dependency scheme (Section 4.5), which is currently the most effective dependency scheme that is sound for both QBF and DQBF. Before expanding a universal variable x, we identify its pseudo-dependencies, i. e., the dependencies that can be removed without changing the satisfiability of the formula. All existential variables forming a pseudo-dependency with x do not have to be copied and neither have the clauses to be doubled in which only existential variables with pseudo-dependencies and variables independent of x occur. This often leads to significantly smaller formulas after the expansion. Finally, if a formula does not contain any universal variables after universal expansion, we immediately employ a SAT solver for deciding the resulting formula.

The main loop of Algorithm 1 is finished whenever no further changes in the formula arise during the latest iteration. Moreover, we immediately exit the loop and return the result whenever a routine was able to decide the (D)QBF. Otherwise, if the formula is undecided and not a QBF, Algorithm 1 is followed by the QBF-based filter, which was described in Section 3.2.

As mentioned above, some preprocessing operations are restricted by the amount of additional memory we are allowed to use for representing the matrix resulting from the operation (in particular in case of resolution and universal expansion). Moreover, we also apply restrictions in terms of "run time" invested into the different preprocessing operations: For each potentially expensive operation like vivification, blocked clause elimination, and self-subsuming resolution, we have a counter that is initialized with the maximum number

[13.] Depending on the subsequent solver used after preprocessing, universal expansions are moved to the solver core, however.

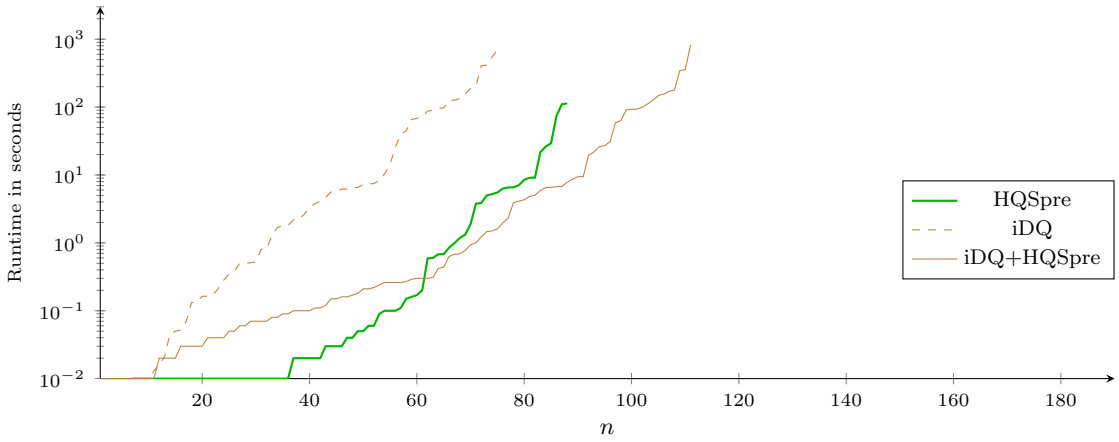

Figure 2. iDQ with and without preprocessing using HQSpre.

of steps allowed for the execution of the operation. For each step that the operation does (e. g., computing the resolvent of two clauses during blocked clause elimination), the counter is decreased. When the counter value becomes negative, the operation is aborted. Having a limit on the number of steps instead of a time limit for the different operations has the advantage that the result is reproducible. The initial values for the counters are chosen such that potentially expensive operations are aborted after a few seconds on an average computer.

6. Experimental Results

We have implemented the described techniques in C++ in the preprocessor HQSpre, which supports both QBF and DQBF preprocessing. To support different back-end solvers, it is able to write the resulting formula into a file in DQDIMACS format, which is the standard file format of the DQBF track at the QBF Evaluation.

As benchmark instances we use the 334 DQBF instances from the DQBF track at the QBFEVAL'18[14.] and the 463 QBF instances from the PCNF track.

All experiments were run on one Intel Xeon E5-2650v2 core at 2.60 GHz, running Ubuntu Linux 16.04 in 64-bit mode as operating system. We aborted all experiments whose computation time exceeded 900 seconds or which required more than 4 GB of memory.

For solving DQBFs, we use the three publicly available solvers iDQ [21], iProver [44], and HQS [78, 28] and run them with and without preprocessing using HQSpre. Figures 2, 3, and 4 show cactus plots comparing the computation times including loading, preprocessing, and solving the formulas.

It is interesting to note that HQSpre alone already solves 88 out of 334 DQBF instances. iDQ without preprocessing solves 75 instances, but with preprocessing it solves 36 instances (48.0 %) more. The situation for iProver is similar. iProver without preprocessing solves some instances more than iDQ without preprocessing (101 instead of 75 instances) and with preprocessing by HQSpre it again solves 20 instances (19.8 %) more. In both cases we

[14.] see http://www.qbflib.org/qbfeval18.php

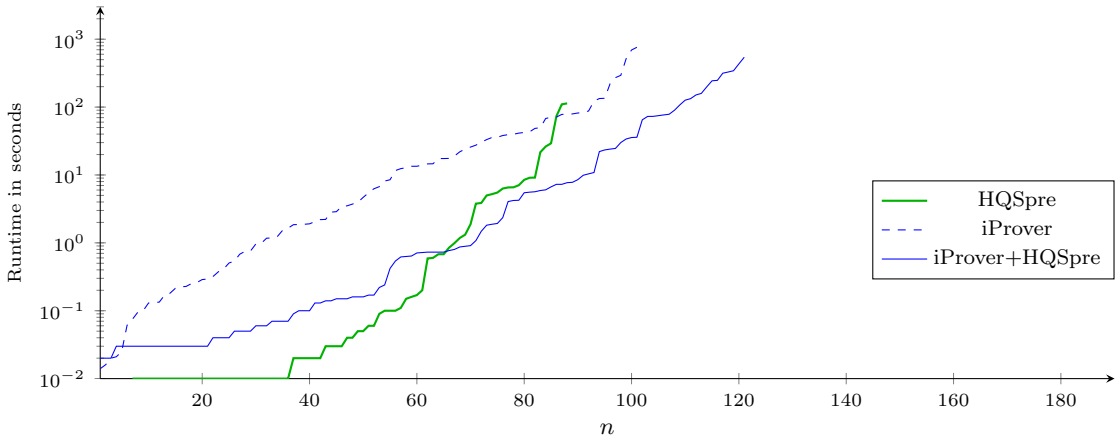

Figure 3. iProver with and without preprocessing using HQSpre.

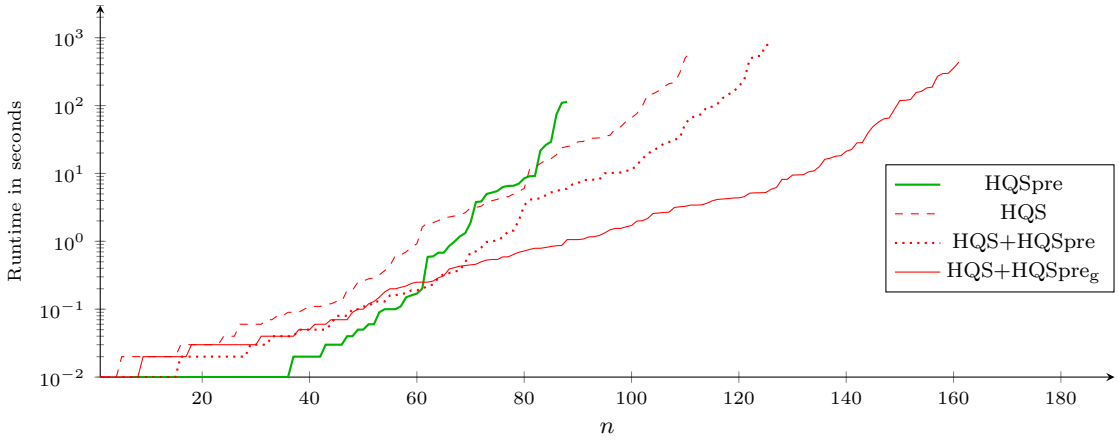

Figure 4. HQS with and without preprocessing using HQSpre / HQSpre$_g$.

used HQSpre configured for CNF-based preprocessing. In contrast, we tried two variants of HQSpre for the solver core HQS: The original HQSpre and a special version HQSpre$_g$ which preserves gate definitions for a later extraction of circuit structure from the CNF, see Section 5.2. HQS without preprocessing solves 111 instances, HQS with the CNF-based preprocessor HQSpre 126 instances (13.5 % more), and HQS with HQSpre$_g$ solves 161 instances, i. e., 45.0 % more instances than HQS without preprocessing. In both cases, the solver core HQS tries to extract circuit structure from the CNF. The original CNF-based HQSpre has advantages and disadvantages for the solver HQS: The positive effect of helping to solve more instances by simplifying the formula still predominates, but on the other hand HQSpre often destroys gate definitions such that a later circuit extraction does not work as well as before. For that reason HQSpre$_g$ is much more successful. It simplifies the formula, but without destroying gate definitions. Since HQS works on AIGs as its internal data structure and heavily relies on the success of extracting circuit information, the combination

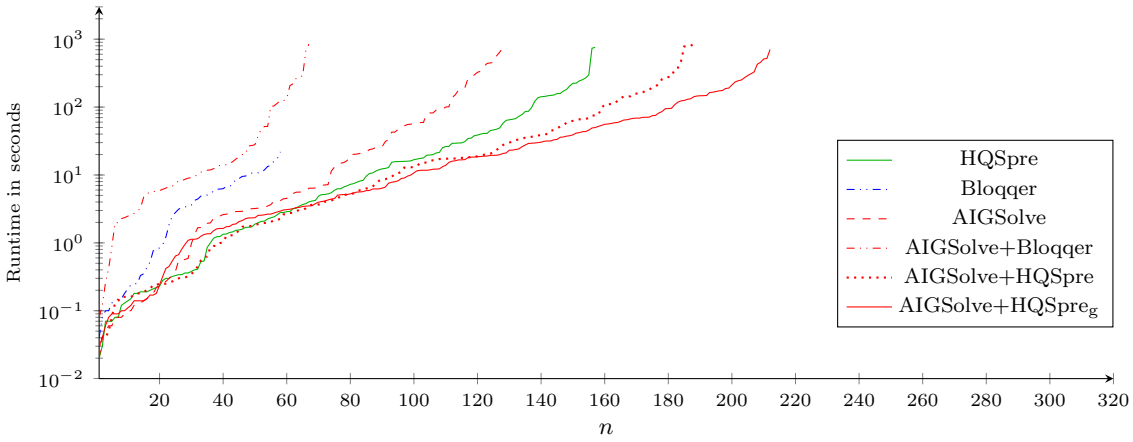

Figure 5. AIGSolve with and without preprocessing using Bloqqer and HQSpre / HQSpre$_g$.

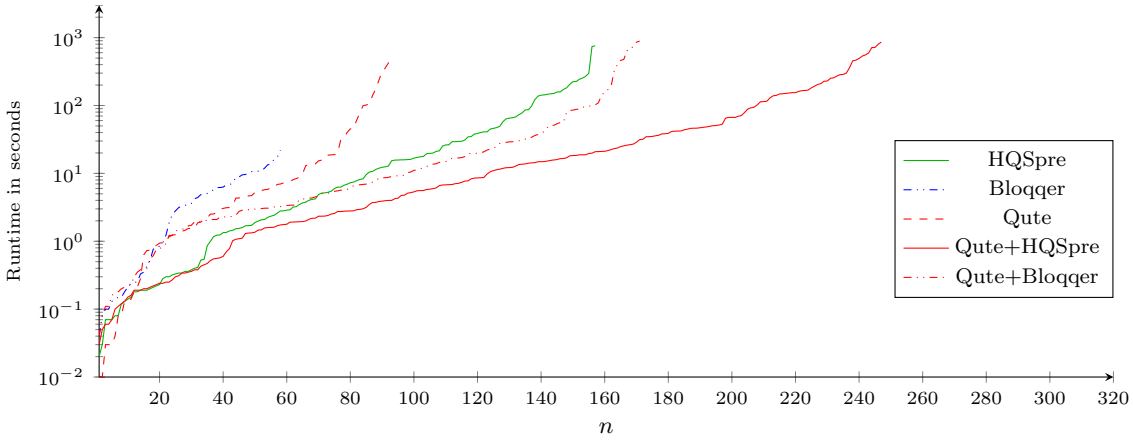

Figure 6. Qute with and without preprocessing using Bloqqer and HQSpre.

of HQS with HQSpre$_g$ is clearly the most successful. To conclude, HQSpre turned out to be highly successful as a preprocessor for DQBF solving. This is true for all considered solvers. The experiment with HQS showed that for obtaining best results different preprocessing techniques should be applied, depending on the solver core used after preprocessing.

For solving QBFs, we use the QBF solvers AIGSolve [57, 58], QUTE [54][15.], DepQBF 6.0.3 [49], QESTO 1.0 [40], RAReQS 1.1 [38], and CAQE [74][15.] We compare their running times alone and in combination with the preprocessors Bloqqer 37 [6] and HQSpre. The results are given in Figs. 5 to 10.

First of all, we observe that Bloqqer by itself already solves 58 out of 463 QBF instances. With a number of 157, HQSpre solves considerably more instances.

[15.] Version from the GitHub repository, checked out at November 7, 2018

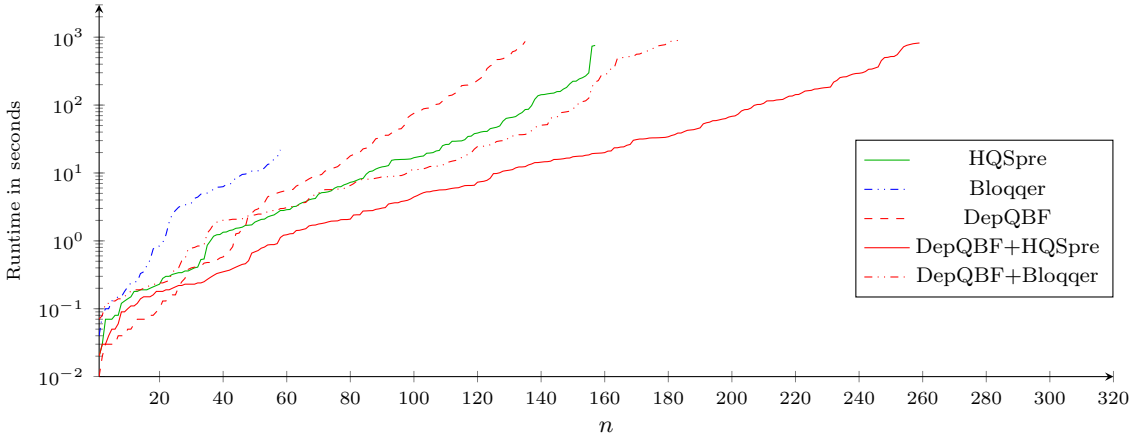

Figure 7. DepQBF with and without preprocessing using Bloqqer and HQSpre.

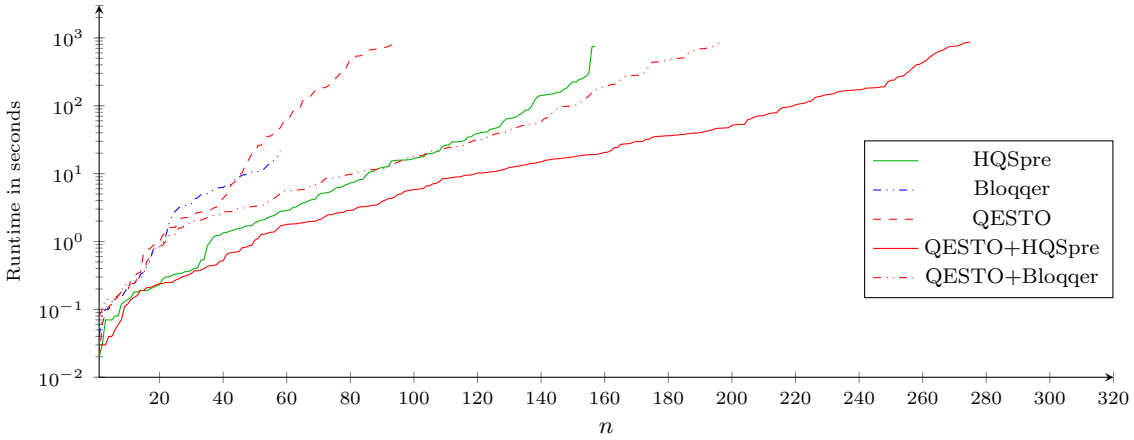

Figure 8. QESTO with and without preprocessing using Bloqqer and HQSpre.

In Fig. 5 we consider the results of the solver AIGSolve, which relies on circuit extraction, like the DQBF solver HQS. Therefore we also try HQSpre$_g$ in addition to Bloqqer and HQSpre. AIGSolve without preprocessing solves 128 instances. Fig. 5 shows that surprisingly the application of the preprocessor Bloqqer before AIGSolve has a negative effect. The combination of Bloqqer and AIGSolve solves 61 instances (47.7 %) fewer instances than AIGSolve alone. Apparently, for AIGSolve the preprocessor Bloqqer is too aggressive in destroying gate definitions, e. g., by blocked clauses elimination. On the other hand, even the CNF-based version HQSpre of our preprocessor has clearly a positive effect on AIGSolve in sum. Using the gate-preserving version HQSpre$_g$ the combination is even more successful and solves 84 (65.6 %) instances more than AIGSolve alone.

All other QBF solvers are CNF based. Thus, we only compare the solvers without preprocessing to the solvers with Bloqqer and HQSpre. For all considered solvers we have a consistent picture: Using Bloqqer is beneficial and using HQSpre is even more beneficial.

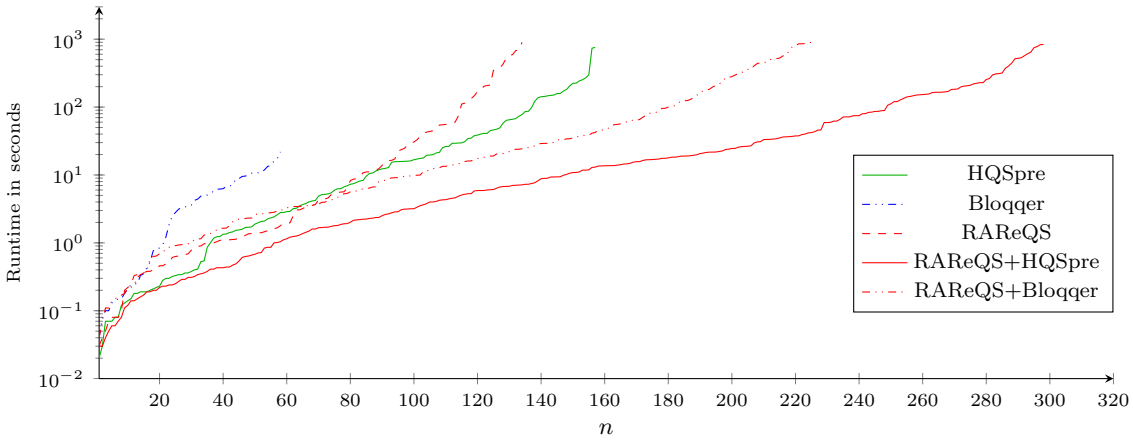

Figure 9. RAReQS with and without preprocessing using Bloqqer and HQSpre.

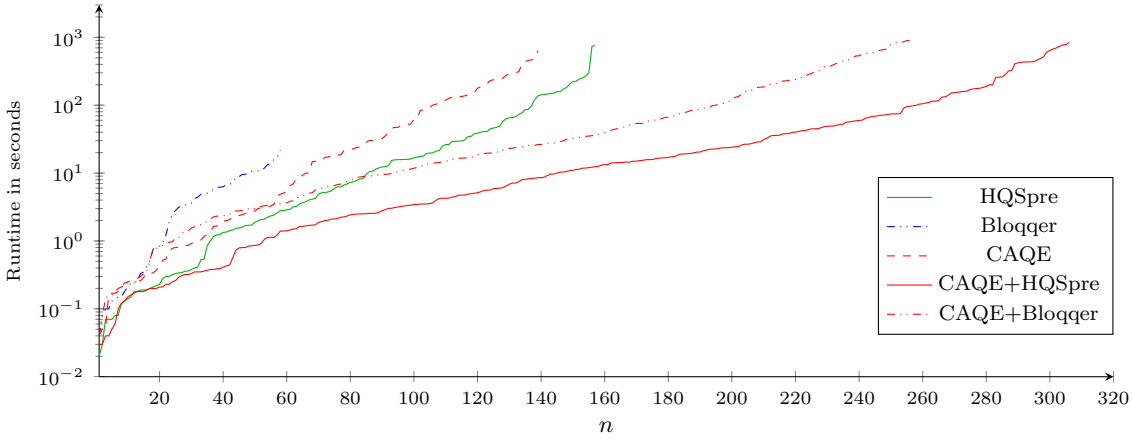

Figure 10. CAQE with and without preprocessing using Bloqqer and HQSpre.

Qute (see Fig. 6) solves 93 instances without preprocessing, together with Bloqqer it solves 78 (83.9 %) instances more, with HQSpre even 154 (165.6 %) instances more. Without preprocessing DepQBF solves 135 instances, with Bloqqer 48 (35.6 %) more, with HQSpre 124 (91.9 %) more (see Fig. 7). Qesto (Fig. 8) solves 94 instances without preprocessing and 102 (108.5 %) instances more when using Bloqqer, 181 (192.6 %) instances more when using HQSpre. Similarly, the number of 134 solved instances for RAReQS (Fig. 9) is increased by 91 (67.9 %) using Bloqqer and 164 (122.4 %) using HQSpre. Finally, considering CAQE (Fig. 10) we observe that Bloqqer increases the number of 139 solved instances by 117 (84.2 %), whereas HQSpre increases it by 167 (120.1 %).

The experiments also show that the quality of a solver is highly determined by the question of how well it may be assisted by preprocessing. For instance, CAQE, the winner of the competition QBFEVAL18, solves only 4 instances more than DepQBF (139 vs. 135

instances) without preprocessing, but with HQSpre as preprocessor it solves 47 instances more (306 vs. 259 instances).

Taken together, we observe that preprocessing is essential also for QBF solving. Whereas Bloqqer is already successful by increasing the number of solved instances by up to 84.2 %, depending on the solver core used, HQSpre clearly outperforms Bloqqer and increases the solved instances by up to 192.6 %.

It is difficult to make reliable statements about the *exact* reasons for the advantage of HQSpre without accessing the internals of *both* HQSpre and Blogger. We believe that the advantage of HQSpre even for QBF instances can be explained by the facts that HQSpre implements almost all known preprocessing techniques for QBF (and DQBF), in particular exploiting gate definitions and strong dependency schemes allow to reduce the cost of variable elimination. HQSpre applies generalized preprocessing techniques based on the specialization of DQBF techniques to QBF, and it benefits from a well-coordinated combination of all individual techniques.

7. Conclusion

We have shown how preprocessing techniques for SAT and QBF can be generalized to DQBF. For all techniques we provided rigorous proofs. For some techniques, the theoretical parts of the paper even advance the field of QBF preprocessing by generalizing known results, by proving results which have been given without proof in the literature, and by disproving wrong statements made in the literature. Based on the theoretical results we presented the preprocessor HQSpre together with implementation details contributing to the efficiency of the preprocessor. Experiments demonstrate that our preprocessor successfully reduces the running time of the actual solving process to a great extent, both for CNF-based and non-CNF-based solver cores. This observation holds both for DQBF and for QBF solving.

In the future, we plan to further enhance the preprocessing techniques. Moreover, based on a better understanding of how different preprocessing techniques assist solver cores working with different solution paradigms, we will try to tune our preprocessor for different solvers, leading to an even better co-operation between preprocessor and solver cores. A further topic for future work is to extend the current version of HQSpre by integrating the computation of Skolem functions for the input formula, given Skolem functions for the formula obtained after preprocessing. This will be based on work already presented in [76] which has to be extended by a rigorous result validation in case of unsatisfiable DQBF formulas as well.

References

[1] Valeriy Balabanov, Hui-Ju Katherine Chiang, and Jie-Hong R. Jiang. Henkin quantifiers and Boolean formulae: A certification perspective of DQBF. *Theoretical Computer Science*, **523**:86–100, 2014.

[2] Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In Benjamin Kuipers and Bonnie L. Webber, editors, *National Conf. on Artificial Intelligence / Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI)*, pages 203–208, Providence, Rhode Island, USA, July 1997. AAAI Press / The MIT Press.

[3] Armin Biere. Resolve and expand. In Holger H. Hoos and David G. Mitchell, editors, *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, **3542** of *LNCS*, pages 59–70, Vancouver, BC, Canada, May 2004. Springer.

[4] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in Computers*, **58**:117–148, 2003.

[5] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, **185** of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2008.

[6] Armin Biere, Florian Lonsing, and Martina Seidl. Blocked clause elimination for QBF. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Int'l Conf. on Automated Deduction (CADE)*, **6803** of *LNCS*, pages 101–115, Wroclaw, Poland, 2011. Springer.

[7] Roderick Bloem, Robert Könighofer, and Martina Seidl. SAT-based synthesis methods for safety specs. In Kenneth L. McMillan and Xavier Rival, editors, *Int'l Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, **8318** of *LNCS*, pages 1–20, San Diego, CA, USA, January 2014. Springer.

[8] Ronen I. Brafman. A simplifier for propositional formulas with many binary clauses. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, **34**(1):52–59, 2004.

[9] Uwe Bubeck. *Model-based transformations for quantified Boolean formulas*. PhD thesis, University of Paderborn, Paderborn, Germany, 2010.

[10] Uwe Bubeck and Hans Kleine Büning. Dependency quantified Horn formulas: Models and complexity. In Armin Biere and Carla P. Gomes, editors, *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, **4121** of *LNCS*, pages 198–211, Seattle, WA, USA, August 2006. Springer.

[11] Uwe Bubeck and Hans Kleine Büning. Bounded universal expansion for preprocessing QBF. In João Marques-Silva and Karem A. Sakallah, editors, *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, **4501** of *LNCS*, pages 244–257, Lisbon, Portugal, May 2007. Springer.

[12] Marco Cadoli, Marco Schaerf, Andrea Giovanardi, and Massimo Giovanardi. An algorithm to evaluate quantified Boolean formulae and its experimental evaluation. *Journal of Automated Reasoning*, **28**(2):101–142, 2002.

[13] Krishnendu Chatterjee, Thomas A. Henzinger, Jan Otop, and Andreas Pavlogiannis. Distributed synthesis for LTL fragments. In *Int'l Conf. on Formal Methods in Computer Aided Design (FMCAD)*, pages 18–25, Portland, OR, USA, October 2013. IEEE.

[14] Edmund M. Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, **19**(1):7–34, 2001.

[15] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–158, Shaker Heights, OH, USA, May 1971. ACM Press.

[16] Alejandro Czutro, Ilia Polian, Matthew D. T. Lewis, Piet Engelke, Sudhakar M. Reddy, and Bernd Becker. Thread-parallel integrated test pattern generator utilizing satisfiability analysis. *Int'l Journal of Parallel Programming*, **38**(3–4):185–202, 2010.

[17] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, **3569** of *LNCS*, pages 61–75, St. Andrews, UK, June 2005. Springer.

[18] Stephan Eggersglüß and Rolf Drechsler. A highly fault-efficient SAT-based ATPG flow. *IEEE Design & Test of Computers*, **29**(4):63–70, 2012.

[19] Bernd Finkbeiner and Leander Tentrup. Fast DQBF refutation. In Carsten Sinz and Uwe Egly, editors, *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, **8561** of *LNCS*, pages 243–251, Vienna, Austria, July 2014. Springer.

[20] Andreas Fröhlich, Gergely Kovásznai, and Armin Biere. A DPLL algorithm for solving DQBF. In *Int'l Workshop on Pragmatics of SAT (POS)*, Trento, Italy, 2012.

[21] Andreas Fröhlich, Gergely Kovásznai, Armin Biere, and Helmut Veith. iDQ: Instantiation-based DQBF solving. In Daniel Le Berre, editor, *Int'l Workshop on Pragmatics of SAT (POS)*, **27** of *EPiC Series*, pages 103–116, Vienna, Austria, 2014. EasyChair.

[22] Aile Ge-Ernst, Christoph Scholl, and Ralf Wimmer. Localizing quantifiers for DQBF. In Clark Barrett and Jin Yang, editors, *Int'l Conf. on Formal Methods in Computer-Aided Design (FMCAD)*. IEEE, October 2019.

[23] Allen Van Gelder. Toward leaner binary-clause reasoning in a satisfiability solver. *Annals of Mathematics and Artificial Intelligence*, **43**(1):239–253, 2005.

[24] Allen Van Gelder. Variable independence and resolution paths for quantified Boolean formulas. In Jimmy Ho-Man Lee, editor, *Int'l Conf. on Principles and Practice of Constraint Programming (CP)*, **6876** of *LNCS*, pages 789–803, Perugia, Italy, September 2011. Springer.

[25] Ian P. Gent, Enrico Giunchiglia, Massimo Narizzano, Andrew G. D. Rowley, and Armando Tacchella. Watched data structures for QBF solvers. In Enrico Giunchiglia and Armando Tacchella, editors, *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, **2919** of *LNCS*, pages 25–36, Santa Margherita Ligure, Italy, May 2003. Springer.

[26] Roman Gershman and Ofer Strichman. Cost-effective hyper-resolution for preprocessing CNF formulas. In Fahiem Bacchus and Toby Walsh, editors, *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, **3569** of *LNCS*, pages 423–429, St. Andrews, UK, June 2005. Springer.

[27] Karina Gitina, Sven Reimer, Matthias Sauer, Ralf Wimmer, Christoph Scholl, and Bernd Becker. Equivalence checking of partial designs using dependency quantified Boolean formulae. In *IEEE Int'l Conf. on Computer Design (ICCD)*, pages 396–403, Asheville, NC, USA, October 2013. IEEE Computer Society.

[28] Karina Gitina, Ralf Wimmer, Sven Reimer, Matthias Sauer, Christoph Scholl, and Bernd Becker. Solving DQBF through quantifier elimination. In Wolfgang Nebel and David Atienza, editors, *Int'l Conf. on Design, Automation & Test in Europe (DATE)*, pages 1617–1622, Grenoble, France, March 2015. IEEE.

[29] Enrico Giunchiglia, Paolo Marin, and Massimo Narizzano. sQueezeBF: An effective preprocessor for QBFs based on equivalence reasoning. In Ofer Strichman and Stefan Szeider, editors, *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, **6175** of *LNCS*, pages 85–98, Edinburgh, UK, July 2010. Springer.

[30] Leon Henkin. Some remarks on infinitely long formulas. In *Infinitistic Methods: Proc. of the 1959 Symp. on Foundations of Mathematics*, pages 167–183, Warsaw, Panstwowe, September 1961. Pergamon Press.

[31] Marijn Heule, Matti Järvisalo, and Armin Biere. Clause elimination procedures for CNF formulas. In Christian G. Fermüller and Andrei Voronkov, editors, *Int'l Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, **6397** of *LNCS*, pages 357–371, Yogyakarta, Indonesia, October 2010. Springer.

[32] Marijn Heule, Matti Järvisalo, and Armin Biere. Covered clause elimination. In Andrei Voronkov, Geoff Sutcliffe, Matthias Baaz, and Christian G. Fermüller, editors, *Int'l Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR) (Short papers)*, **13** of *EPiC Series*, pages 41–46, Yogyakarta, Indonesia, October 2010. EasyChair.

[33] Marijn Heule, Matti Järvisalo, and Armin Biere. Efficient CNF simplification based on binary implication graphs. In Karem A. Sakallah and Laurent Simon, editors, *Int'l*

Conf. on Theory and Applications of Satisfiability Testing (SAT), **6695** of *LNCS*, pages 201–215, Ann Arbor, MI, USA, June 2011. Springer.

[34] Marijn Heule, Matti Järvisalo, Florian Lonsing, Martina Seidl, and Armin Biere. Clause elimination for SAT and QSAT. *Journal of Artificial Intelligence Research*, **53**:127–168, 2015.

[35] Marijn Heule, Martina Seidl, and Armin Biere. Efficient extraction of Skolem functions from QRAT proofs. In *Int'l Conf. on Formal Methods in Computer Aided Design (FMCAD)*, pages 107–114, Lausanne, Switzerland, October 2014. IEEE.

[36] Marjin Heule, Martina Seidl, and Armin Biere. Blocked literals are universal. In Klaus Havelund, Gerard Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods Symposium (NFM)*, **9058** of *LNCS*, pages 436–442, Pasadena, CA, USA, April 2015. Springer.

[37] Franjo Ivancic, Zijiang Yang, Malay K. Ganai, Aarti Gupta, and Pranav Ashar. Efficient SAT-based bounded model checking for software verification. *Theoretical Computer Science*, **404**(3):256–274, 2008.

[38] Mikolás Janota, William Klieber, João Marques-Silva, and Edmund M. Clarke. Solving QBF with counterexample guided refinement. In Alessandro Cimatti and Roberto Sebastiani, editors, *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, **7317** of *LNCS*, pages 114–128, Trento, Italy, June 2012. Springer.

[39] Mikolás Janota, Inês Lynce, and Joao Marques-Silva. Algorithms for computing backbones of propositional formulae. *AI Communications*, **28**(2):161–177, 2015.

[40] Mikolás Janota and João Marques-Silva. Solving QBF by clause selection. In Qiang Yang and Michael Wooldridge, editors, *Int'l Joint Conf. on Artificial Intelligence (IJCAI)*, pages 325–331, Buenos Aires, Argentina, July 2015. AAAI Press.

[41] Matti Järvisalo, Armin Biere, and Marijn Heule. Blocked clause elimination. In Javier Esparza and Rupak Majumdar, editors, *Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, **6015** of *LNCS*, pages 129–144, Paphos, Cyprus, March 2010. Springer.

[42] Philip Kilby, John K. Slaney, Sylvie Thiébaux, and Toby Walsh. Backbones and backdoors in satisfiability. In Manuela M. Veloso and Subbarao Kambhampati, editors, *National Conference on Artificial Intelligence / Int'l Conf. on Innovative Applications of Artificial Intelligence (IAAI)*, pages 1368–1373, Pittsburgh, Pennsylvania, USA, July 2005. AAAI Press / The MIT Press.

[43] Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified boolean formulas. *Information and Computation*, **117**(1):12–18, 1995.

[44] Konstantin Korovin. iProver – An instantiation-based theorem prover for first-order logic (system description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Int'l Joint Conf. on Automated Reasoning (IJCAR)*, **5195** of *LNCS*, pages 292–298, Sydney, Australia, August 2008. Springer.

[45] Oliver Kullmann. On a generalization of extended resolution. *Discrete Applied Mathematics*, **96-97**:149–176, 1999.

[46] Stefan Kupferschmid, Matthew Lewis, Tobias Schubert, and Bernd Becker. Incremental preprocessing methods for use in BMC. *Formal Methods in System Design*, **39**(2):185–204, 2011.

[47] Harry R. Lewis. Complexity results for classes of quantificational formulas. *Journal of Computer and System Sciences*, **21**(3):317–353, 1980.

[48] Florian Lonsing and Armin Biere. Efficiently representing existential dependency sets for expansion-based QBF solvers. *Electronic Notes in Theoretical Computer Science*, **251**:83–95, 2009.

[49] Florian Lonsing and Uwe Egly. DepQBF 6.0: A search-based QBF solver beyond traditional QCDCL. In Leonardo de Moura, editor, *Int'l Conf. on Automated Deduction (CADE)*, **10395** of *LNCS*, pages 371–384, Gothenburg, Sweden, August 2017. Springer.

[50] Norbert Manthey. Coprocessor 2.0 – A flexible CNF simplifier (Tool presentation). In Alessandro Cimatti and Roberto Sebastiani, editors, *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, **7317** of *LNCS*, pages 436–441, Trento, Italy, June 2012. Springer.

[51] Albert R. Meyer and Larry J. Stockmeyer. Word problems requiring exponential time: Preliminary report. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–9, Austin, TX, USA, April 1973. ACM Press.

[52] Ilya Mironov and Lintao Zhang. Applications of SAT solvers to cryptanalysis of hash functions. In Armin Biere and Carla P. Gomes, editors, *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, **4121** of *LNCS*, pages 102–115, Seattle, WA, USA, August 2006. Springer.

[53] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *ACM/IEEE Design Automation Conference (DAC)*, pages 530–535, Las Vegas, NV, USA, June 2001. ACM.

[54] Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Dependency learning for QBF. In Serge Gaspers and Toby Walsh, editors, *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, **10491** of *LNCS*, pages 298–313, Melbourne, Australia, August 2017. Springer.

[55] Gary Peterson, John Reif, and Salman Azhar. Lower bounds for multiplayer non-cooperative games of incomplete information. *Computers & Mathematics with Applications*, **41**(7–8):957–992, April 2001.

[56] Cédric Piette, Youssef Hamadi, and Lakhdar Sais. Vivifying propositional clausal formulae. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikolaos M.

Avouris, editors, *European Conf. on Artificial Intelligence (ECAI)*, **178** of *Frontiers in Artificial Intelligence and Applications*, pages 525–529, Patras, Greece, July 2008. IOS Press.

[57] Florian Pigorsch and Christoph Scholl. Exploiting structure in an AIG based QBF solver. In Luca Benini, Giovanni De Micheli, Bashir M. Al-Hashimi, and Wolfgang Müller, editors, *Int'l Conf. on Design, Automation & Test in Europe (DATE)*, pages 1596–1601, Nice, France, April 2009. IEEE.

[58] Florian Pigorsch and Christoph Scholl. An AIG-based QBF-solver using SAT for preprocessing. In Sachin S. Sapatnekar, editor, *ACM/IEEE Design Automation Conference (DAC)*, pages 170–175, Anaheim, CA, USA, July 2010. ACM Press.

[59] David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, **2**(3):293–304, 1986.

[60] Markus N. Rabe. A resolution-style proof system for DQBF. In Serge Gaspers and Toby Walsh, editors, *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, **10491** of *LNCS*, pages 314–325, Melbourne, VIC, Australia, 2017. Springer.

[61] Markus N. Rabe and Leander Tentrup. Synthesis of Boolean functions with clausal abstraction. *arXiv*, **abs/1808.08759**, 2018.

[62] Jussi Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research*, **10**:323–352, 1999.

[63] Marko Samer. Variable dependencies of quantified CSPs. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *Int'l Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, **5330** of *LNCS*, pages 512–527, Doha, Qatar, November 2008. Springer.

[64] Marko Samer and Stefan Szeider. Backdoor sets of quantified Boolean formulas. *Journal of Automated Reasoning*, **42**(1):77–97, 2009.

[65] Christoph Scholl and Bernd Becker. Checking equivalence for partial implementations. In *ACM/IEEE Design Automation Conference (DAC)*, pages 238–243, Las Vegas, NV, USA, June 2001. ACM Press.

[66] Christoph Scholl, Jie-Hong Roland Jiang, Ralf Wimmer, and Aile Ge-Ernst. A PSPACE subclass of dependency quantified Boolean formulas and its effective solving. In Pascal Van Hentenryck and Zhi-Hua Zhou, editors, *AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, January 2019.

[67] Martina Seidl, Florian Lonsing, and Armin Biere. qbf2epr: A tool for generating EPR formulas from QBF. In Pascal Fontaine, Renate A. Schmidt, and Stephan Schulz, editors, *3rd Workshop on Practical Aspects of Automated Reasoning (PAAR)*, **21** of *EPiC Series in Computing*, pages 139–148, Manchester, UK, June 2012. EasyChair.

[68] João P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, **48**(5):506–521, 1999.

[69] Carsten Sinz, Andreas Kaiser, and Wolfgang Küchlin. Formal methods for the validation of automotive product configuration data. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM)*, **17**(1):75–97, 2003.

[70] Friedrich Slivovsky and Stefan Szeider. Computing resolution-path dependencies in linear time. In Alessandro Cimatti and Roberto Sebastiani, editors, *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, **7317** of *LNCS*, pages 58–71, Trento, Italy, June 2012. Springer.

[71] Friedrich Slivovsky and Stefan Szeider. Quantifier reordering for QBF. *Journal of Automated Reasoning*, **56**(4):459–477, 2016.

[72] Friedrich Slivovsky and Stefan Szeider. Soundness of Q-resolution with dependency schemes. *Theoretical Computer Science*, **612**:83–101, 2016.

[73] Robert E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, **1**(2):146–160, 1972.

[74] Leander Tentrup and Markus N. Rabe. CAQE: A certifying QBF solver. In Roope Kaivola and Thomas Wahl, editors, *Int'l Conf. on Formal Methods in Computer Aided Design (FMCAD)*, pages 136–143, Austin, TX, USA, September 2015. IEEE.

[75] Grigorij Samuilovich Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*, **Part 2**:115–125, 1970.

[76] Karina Wimmer, Ralf Wimmer, Christoph Scholl, and Bernd Becker. Skolem functions for DQBF. In Cyrille Artho, Axel Legay, and Doron Peled, editors, *Int'l Symposium on Automated Technology for Verification and Analysis (ATVA)*, **9938** of *LNCS*, pages 395–411, Chiba, Japan, October 2016. Springer.

[77] Ralf Wimmer, Karina Gitina, Jennifer Nist, Christoph Scholl, and Bernd Becker. Preprocessing for DQBF. In Marijn Heule and Sean Weaver, editors, *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, **9340** of *LNCS*, pages 173–190, Austin, TX, USA, September 2015. Springer.

[78] Ralf Wimmer, Andreas Karrenbauer, Ruben Becker, Christoph Scholl, and Bernd Becker. From DQBF to QBF by dependency elimination. In Serge Gaspers and Toby Walsh, editors, *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, **10491** of *LNCS*, pages 326–343, Melbourne, Australia, August 2017. Springer.

[79] Ralf Wimmer, Sven Reimer, Paolo Marin, and Bernd Becker. HQSpre – An effective preprocessor for QBF and DQBF. In Axel Legay and Tiziana Margaria, editors, *Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Part I*, **10205** of *LNCS*, pages 373–390, Uppsala, Sweden, April 2017. Springer.

[80] Ralf Wimmer, Christoph Scholl, Karina Wimmer, and Bernd Becker. Dependency schemes for DQBF. In Nadia Creignou and Daniel Le Berre, editors, *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, **9710** of *LNCS*, pages 473–489, Bordeaux, France, 2016. Springer.