Preprint from 10th International Conference on Theory and Applications of Satisfiability Testing (SAT), September 2015, Austin, USA

Preprocessing for DQBF

Ralf Wimmer(✉), Karina Gitina, Jennifer Nist, Christoph Scholl, and Bernd Becker

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

Abstract. For SAT and QBF formulas many techniques are applied in order to reduce/modify the number of variables and clauses of the formula, before the formula is passed to the actual solving algorithm. It is well known that these preprocessing techniques often reduce the computation time of the solver by orders of magnitude. In this paper we generalize different preprocessing techniques for SAT and QBF problems to dependency quantified Boolean formulas (DQBF) and describe how they need to be adapted to work with a DQBF solver core. We demonstrate their effectiveness both for CNF- and non-CNF-based DQBF algorithms.

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 verification and test of hard- and software [1,2] to planning [3], product configuration [4], and cryptanalysis [5]. 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.

In this paper, we consider the more general, still practically relevant formalism of *dependency quantified Boolean formulas* (DQBF). "Standard" quantified Boolean formulas (in prenex normal form) 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 at the cost of a higher complexity for the decision problem – for SAT it is NP-complete [6], for QBF PSPACE-complete [7], and for DQBF it is NEXPTIME-complete [8]. However, some applications like the verification of incomplete circuits [9] or the synthesis of safe controllers [10] require the higher expressiveness of DQBF. Therefore, first solvers for DQBF have been presented recently: iDQ [11] reduces the solution of a DQBF to the solution of a series of SAT instantiations. HQS [12] applies quantifier elimination to solve the formula.

This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center AVACS (SFB/TR 14).

© Springer International Publishing Switzerland 2015
M. Heule and S. Weaver (Eds.): SAT 2015, LNCS 9340, pp. 173–190, 2015.
DOI: 10.1007/978-3-319-24318-4_13

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; e. g., 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 [13,14], which often reduces computation times considerably.

For SAT and QBF, efficient and effective preprocessing tools are available like SatELite [15], Coprocessor [16] for SAT and squeezeBF [17], bloqqer [18] for QBF. Both available DQBF solvers, however, still lack a preprocessing phase before the actual solving process. 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 [19,20], 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 [9,21] 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, in this paper for the first time preprocessing techniques are made available for DQBF solving. We generalize successful preprocessing techniques for QBF to DQBF like blocked clause elimination (BCE) [18,22], equivalence reasoning [17], structure extraction [23], and variable elimination by resolution [24]. All correctness proofs are available in an extended version of this paper [25]. We present experimental results which show the effectiveness of these techniques for DQBF. We demonstrate that the applied techniques have to be chosen depending on the solving techniques applied in the solver core. For example, BCE prevents an effective undoing of Tseitin transformation [26], 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. The experiments show that preprocessing both reduces the computation times and significantly increases the number of solved instances of both solvers, iDQ and HQS.

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 describes the preprocessing techniques for DQBF that we apply in our tool to simplify the DQBF at hand. Section 5 gives an experimental evaluation of the described techniques, and Section 6 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., $\mathcal{A}(V') = \{\nu \mid \nu : V' \to \{0,1\}\}$.

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 $\psi := \forall x_1 \forall x_2 \ldots \forall x_n \exists y_1(D^\psi_{y_1}) \exists y_2(D^\psi_{y_2}) \ldots \exists y_m(D^\psi_{y_m}) : \varphi$ 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 Boolean formula over V, the* matrix *of ψ.*

We often write $\psi = Q : \varphi$ with the quantifier prefix Q and the matrix φ. Throughout the whole paper we assume, unless explicitly stated differently, that a DQBF $\psi = Q : \varphi$ as in Definition 1 with φ in CNF is given. 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* satisfied *(written $\vDash \psi$) 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 s_{y_i} is called a* Skolem function *for y_i.*

Two DQBFs ψ_1 and ψ_2 are *equivalent* iff $\vDash \psi_1 \Leftrightarrow \vDash \psi_2$ holds.

Definition 3 (QBF). *A* quantified Boolean formula *(QBF)*[1] *is a DQBF ψ such that $D^\psi_y \subseteq D^\psi_{y'}$ or $D^\psi_{y'} \subseteq D^\psi_y$ holds for any pair $y, y' \in V^\psi_\exists$ of existential variables.*

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

In the following we assume 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 $\mathrm{dep}_\psi(\ell) = \mathrm{dep}_\psi\big(\mathrm{var}(\ell)\big)$. Moreover, we define the "sign" sgn 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 [26] 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.

Definition 4 (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 [27].

Currently, three solvers for DQBF have been proposed: An extension of the DPLL algorithm, typically applied for solving SAT and QBF formulas, has been described in [28]. However, no implementation thereof is available. The second solver is iDQ [11], which relies on a formula in CNF and uses instantiation-based solving, i. e., it reduces deciding a DQBF to deciding a series of SAT problems. Finally, there is the solver HQS [12], 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. Although HQS reads the same CNF-based input format as iDQ, its back-end can handle Boolean formulas of arbitrary structure. We use both iDQ and HQS for the evaluation of the preprocessing techniques presented in the following.

3 Incomplete, but Cheap Decision Procedures

Before we present our preprocessing techniques for DQBF, we review an incomplete, but cheap decision procedure (called "filter") for 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 the filter technique, and

only if it finishes with an inconclusive result, we apply the solver core. Experiments showed that it is beneficial to use a filter before the solving process.

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 [9]. This is the case if $D_y^{\Psi^{\uparrow}} \supseteq D_y^{\psi}$ for all $y \in V_{\exists}^{\psi}$. 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 to show the unsatisfiability of DQBFs. For the formal definitions of the approximations we refer the reader to [9].

Finkbeiner and Tentrup [21] improve these over-approximations by constructing a series of more and more precise QBF formulas. To make this possible they modify both the sets of variables and the matrix of the DQBF: The idea is to use $k \geq 1$ copies of the matrix and its variables. 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. Since the sizes of the QBF instances grow considerably with increasing values of k, in most cases only values $k \leq 3$ are beneficial. For more details we refer the reader to [21].

4 Preprocessing Techniques for DQBF

In this section we describe techniques which can be applied to preprocess a DQBF. The proofs of the main theorems and lemmas are given in the extended version [25] of this paper.

4.1 Backbones, Monotonic and Equivalent Variables

Here we describe techniques which reduce both the number of variables in the formula and the number of clauses.

Unit and pure variables are well-known concepts from SAT and QBF solving. They can be replaced by constant values without influencing the formula's truth value. Typically a variable is defined as unit if the matrix contains a clause consisting only of this variable. A variable is pure if it occurs in the whole matrix either only positive or only negative:

Definition 5 (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).

For DQBF 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 6 (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 theorem states how we can exploit backbones and monotonic variables to reduce the size of the formula:

Theorem 1. *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 equivalent to 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.*

This theorem has been proven formally in [29]. 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. It is easy to show that unit literals are backbones and pure literals are monotonic.

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

Definition 7. *Let $\varphi^2 = \{C \in \varphi \,|\, |C| = 2\}$ be the set of binary clauses. The* binary implication graph *of ψ is the directed graph $\mathrm{BIP}(\psi) = (L, E)$ with the set $L = \{v, \neg v \,|\, v \in V\}$ of literals as its set of nodes and $E = \{(v, \neg w), (\neg v, w) \,|\, \{v, w\} \in \varphi^2\}$ the set of edges.*

Then the following lemma holds:

Lemma 1. *A literal ℓ is a backbone if there is a path in $\mathrm{BIP}(\psi)$ from $\neg\ell$ to ℓ.*

If there is a path from literal ℓ to literal ℓ', we can derive the clause $\{\neg\ell, \ell'\}$ by resolution. In case of the lemma, the path from $\neg\ell$ to ℓ implies that we can derive the clause $\{\neg\neg\ell, \ell\} = \{\ell\}$. Since this is a resolvent of clauses in φ, it may be added to φ. Then we can apply Definition 5 to obtain the result.

Unit and pure literals, according to Definition 5, and backbones according to Lemma 1, 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 6) for backbones and monotonic variables, which are based on solving a sequence of SAT problems. For backbones in the QBF context this observation has been made in [30].

Definition 8 (Equivalent literals). *The literals ℓ and μ are* equivalent *w. r. t. a propositional formula φ iff φ is equivalent to $\varphi \wedge (\ell \equiv \mu)$.*

Theorem 2. *Let ℓ and μ be equivalent literals. We assume, w. l. o. g., that* $\operatorname{sgn}(\ell) = 1$. *If* $\operatorname{var}(\ell), \operatorname{var}(\mu) \in V_\forall^\psi$, *then ψ is unsatisfiable. Otherwise, we assume w. l. o. g. that* $\operatorname{var}(\ell) \in V_\exists^\psi$. *If* $\operatorname{var}(\mu) \in V_\forall^\psi$ *and* $\operatorname{var}(\mu) \notin D^\psi_{\operatorname{var}(\ell)}$, *then ψ is unsatisfiable. If* $\operatorname{var}(\mu) \in V_\forall^\psi$ *and* $\operatorname{var}(\mu) \in D^\psi_{\operatorname{var}(\ell)}$, *then ψ is equivalent to* $Q \setminus \{\operatorname{var}(\ell)\} : \varphi[\mu/\ell]$. *If* $\operatorname{var}(\ell), \operatorname{var}(\mu) \in V_\exists^\psi$, *then ψ is equivalent to*

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

A proof can be found in the extended version [25] of this paper.

To detect equivalent literals, we exploit the following lemma:

Lemma 2. *Two literals ℓ, μ are equivalent if there is a path in* $\mathrm{BIP}(\psi)$ *from ℓ to μ and vice versa.*

We decompose $\mathrm{BIP}(\psi)$ into strongly connected components (SCCs) using Tarjan's SCC algorithm [31]. SCCs have the property that there is a path between each pair of nodes in an SCC. Therefore literals within one SCC are equivalent. They are replaced by one representative by applying Theorem 2. This procedure was described e. g., in [16,32–35] for SAT preprocessing. Further equivalent literals can be found using structure extraction (see Section 4.5). Of course, even SAT checks based on Definition 8 may be beneficial in the DQBF context.

4.2 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 Skolem function for y does 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. This potentially reduces the number of copies of variables, if universal expansion according to Theorem 5 is used for solving a DQBF.

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.

Definition 9. *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.*

Deciding whether two variables are independent has the same complexity as deciding the DQBF itself [36]. Therefore one resorts to sufficient criteria to show independence. The most simple ones are based on the incidence graph of the matrix:

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

Theorem 3 (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 $G_{V,\varphi}$ from x to y, visiting only variables in $\{z \in V_\exists^\psi \mid x \in D_z^\psi\}$ in between.*

For a proof for this theorem, which generalizes a theorem from [36], see [25].

In the QBF context more powerful dependency schemes have been developed which can possibly identify more variables as independent, see, e. g., [36–40]. A generalization of these techniques will have an immediate benefit for DQBF solving by increasing the potential to save variable copies during universal expansion.

4.3 Universal Reduction, Resolution, and Universal Expansion

Universal reduction, *resolution*, and *universal expansion* are well-known techniques used during the solution of QBFs. 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 [11,41].

Lemma 3 (Universal reduction, [11,41]). *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.*

For QBF resolution and universal reduction together are able to derive the empty clause iff the formula is unsatisfiable. This does not hold for DQBF [41]. Resolution in QBF formulas allows to eliminate an existential variable by replacing the clauses containing this variable with their resolvents. While adding resolvents is sound for DQBF as well, eliminating existential variables by resolution [15] only works under certain conditions. 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 [26], variable elimination by resolution is applicable to a large subset of the formula's existential variables.

Theorem 4 (Variable elimination by resolution). *Let $y \in V_\exists^\psi$ be an existential variable of ψ. We partition φ into the sets $\varphi^y = \{C \in \varphi \mid y \in C\}$, $\varphi^{\neg y} = \{C \in \varphi \mid \neg y \in C\}$, and $\varphi^\emptyset = \varphi \setminus (C^y \cup C^{\neg y})$.*
If one of the following conditions is satisfied:

- *for all $C \in \varphi^y$ and all $k \in C$ we have $\mathrm{dep}_\psi(k) \subseteq \mathrm{dep}_\psi(y)$,*
- *for all $C' \in \varphi^{\neg y}$ and all $k \in C'$ we have $\mathrm{dep}_\psi(k) \subseteq \mathrm{dep}_\psi(y)$, or*
- *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. Sec. 4.5),*

then ψ is equivalent to $\psi' := Q \setminus \{y\} : \varphi^\emptyset \wedge \bigwedge_{C \in \varphi^y} \bigwedge_{C' \in \varphi^{\neg y}} C \otimes_y C'$.

Proof sketch. Resolvents are implied by the matrix, i. e., adding resolvents to the matrix yields an equivalent formula. If ψ is satisfied, then removing the clauses in φ^y and $\varphi^{\neg y}$ cannot make the formula unsatisfied, i. e., ψ' is satisfied.

Assume that ψ' is satisfied by Skolem functions s_z for $z \in V_\exists^\psi \setminus \{y\}$. We define $s_y := \neg\varphi^y\big[0/y\big]\big[s_z/z \text{ for } z \in V_\exists^\psi \setminus \{y\}\big]$ in the first case, $s_y := \neg\varphi^{\neg y}\big[1/y\big]\big[s_z/z \text{ for } z \in V_\exists^\psi \setminus \{y\}\big]$ in the second case, and $s_y := f(V')\big[s_z/z \text{ for } z \in V_\exists^\psi \setminus \{y\}\big]$ in the third case. It is not hard to show that s_y is an admissible Skolem function for y and that $\varphi[s_v/v \text{ for } v \in V_\exists^\psi]$ is indeed a tautology. Details can be found in the extended version [25] of this paper. □

Theorem 4 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 4 is in general not feasible because the number of clauses can grow considerably during elimination. We first create a list of variables that may be eliminated. For each such variable y we estimate the cost c_y of elimination, i. e., $c_y := \frac{|\varphi^\emptyset|+|\varphi^y|\cdot|\varphi^{\neg y}|}{|\varphi|}$. We eliminate one variable y with minimum cost provided that c_y is less than a user-specified factor $\varepsilon > 1$. After resolving variables we check for subsumed clauses, i. e., clauses C such that there is a clause C' with $C' \subseteq C$. Then C can be deleted [27].

Universal expansion [9,41–43] is the corresponding method for eliminating universal variables. It is the main operation which the solver HQS [12] uses to transform the DQBF at hand into an equivalent QBF. This QBF can be solved by an arbitrary QBF solver.

Theorem 5 (Universal expansion). *Let* $x_i \in V_\forall^\psi$, *and* $E_{x_i}^\psi = \big\{y_i \in V_\exists^\psi \,\big|\, x_i \in \mathrm{dep}_\psi(y_j)\big\}$. *Then* ψ *is equivalent to*

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

A formal proof of this theorem is given, e. g., in [9]. In order to avoid unnecessary variable copies, we check using the standard dependency scheme (cf. Theorem 3) which existential variables actually depend on the expanded universal variable.

4.4 Blocked Clause Elimination

The concept of blocked clauses was introduced by Järvisalo et al. for SAT in [22] and later generalized to QBF by Biere et al. in [18]. Blocked clauses can be removed from a formula without changing its truth value. Before checking for blockedness, clauses can be extended by so-called hidden and covered literals [18,44,45]. This does not change the truth value of the formula, but increases the chance that a 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. Then 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 equivalent 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[2] and the innermost universal quantifiers (upon which no existential variable depends) by universal reduction until all quantifiers have been removed [24]. 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 10 (Blocked clauses). *Let $Q : \varphi \wedge C$ be a DQBF and C a clause with $\ell \in C$. 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 6 (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 equivalent.*

Proof sketch. The theorem can be shown by induction on the number $|\mathrm{dep}_\psi(\ell)|$ of ℓ's dependencies. The base case $\mathrm{dep}_\psi(\ell) = \emptyset$ works analogously to the QBF case, see [18]. For the induction step, we choose an arbitrary universal variable $x \in \mathrm{dep}_\psi(\ell)$ and eliminate it by universal expansion (see Theorem 5). In the resulting formula, ℓ and its copy ℓ' depend on one variable less. One can show that both copies of C in this formula are either blocked or tautological. Therefore they can be removed by the induction assumption. Un-doing the expansion step yields the result. A more detailed proof can be found in [25]. □

The purpose of the following techniques is to extend clauses by redundant literals. This increases the chance that the clause is blocked and can be deleted. If the extended clause is not blocked, the additional literals are removed again.

Definition 11 (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 $\{\ell_1, \ldots, \ell_n\} \subseteq C$.*

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

Theorem 7 (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* [15]. 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.

This is in contrast to the '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 12 (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 $R_\psi(C, \ell) = \big\{C' \in \varphi \,\big|\, \neg\ell \in C' \wedge \forall v \in V : (\{v, \neg v\} \subseteq C \otimes_\ell C' \;\Rightarrow\; \mathrm{dep}_\psi(v) \not\subseteq \mathrm{dep}_\psi(\ell))\big\}$.*

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 8 (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 equivalent.*

Proof sketch. Assume that k is a covered literal for C w. r. t. ℓ. We show the theorem by induction on the number $|\mathrm{dep}_\psi(\ell)|$ of dependencies of ℓ. The induction base where $\mathrm{dep}_\psi(\ell) = \emptyset$ is similar to the QBF case (cf. [18]). For the induction step, we apply universal expansion of an arbitrary variable in $\mathrm{dep}_\psi(\ell)$ (see Theorem 5) to obtain a formula in which ℓ and its copy ℓ' both depend on one variable less. It is rather technical to show that adding k (k') to the copies of C in this formula leads to an equivalent formula, since these copies are either tautological or k (k') is a covered literal. By undoing the expansion step we obtain the desired result. For a detailed proof we refer to [25]. □

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 equivalent 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 $R_\psi(C, \ell) = \big\{C' \in \varphi \,\big|\, \neg\ell \in C' \wedge \forall v \in V : (\{v, \neg v\} \subseteq C \otimes_\ell C' \;\Rightarrow\; \mathrm{dep}_\psi(v) \not\subseteq \mathrm{dep}_\psi(\ell))\big\}$ instead of $R_\psi(C, \ell) = \big\{C' \in \varphi \,\big|\, \neg\ell \in C' \wedge \nexists v \in V : \{v, \neg v\} \subseteq C \otimes_\ell C'\big\}$ in order to be able to lead the (rather involved) proof of Theorem 8, see [25].

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 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 [9]. If the clause contains the universal variable to be eliminated, one of these copies is satisfied and can therefore be omitted (cf. [46]).

4.5 Structure Extraction

The DQBF's matrix in CNF is often created from a circuit or a Boolean expression by Tseitin transformation [26], where a new existential variable v_e is created for each sub-expression e (or gate output). Clauses encoding the relationship $v_e \equiv e$ are added and the sub-expression e is replaced by the variable v_e. If a solver (like HQS) does not rely on a matrix in CNF, this transformation step can be undone. This removes all artificially introduced variables. Structure extraction is used in the QBF solver AIGsolve [23].

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 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*.

Theorem 9. *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 equivalent to $Q \setminus \{y\} : (\varphi \setminus \varphi^f)[f(\ell_1, \ldots, \ell_k)/y]$ if $y \in V_\exists^\psi$ and for $i = 1, \ldots, k$ we have $\text{dep}_\psi(\ell_i) \subseteq \text{dep}_\psi(y)$.*

Our implementation checks for defining clauses for (multi-input) (N)AND gates and 2-input XOR gates, both with arbitrarily negated inputs. We do not extract definitions that lead to cyclic dependencies.

Gate detection can be used as the last step of the preprocessing routine. If a relationship $y \equiv f(\ell_1, \ldots, \ell_k)$ is detected which does not lead to cyclic dependencies, we remove y from the prefix and the defining clauses from the matrix. We additionally use a data structure 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, we convert the remaining clause into an AIG and then substitute the defined variables by their definitions.

The same structure extraction procedure can also be used to identify equivalent variables and unnecessary variable dependencies. For both purposes, the relationships are only detected, but neither are the defining clauses removed nor is the data structure that stores the relationships updated. Therefore this can also be used if the solver back-end requires a matrix in CNF: If there is the relationship $y \equiv f(\ell_1, \ldots, \ell_k)$ and $\bigcup_{i=1}^k \text{dep}_\psi(\ell_i) \subsetneq D_y^\psi$, then D_y^ψ can be replaced by $\bigcup_{i=1}^k \text{dep}_\psi(\ell_i)$. If two defined variables y, y' with the same definition are detected, i. e., $y \equiv f(\ell_1, \ldots, \ell_k)$ and $y' \equiv f(\ell_1, \ldots, \ell_k)$, then y and y' are equivalent and Theorem 2 can be applied to remove one of them.

5 Experimental Results

We have implemented the described techniques in C++ as a preprocessor for our DQBF solver HQS. To support other back-end solvers, too, it is able to write the resulting formula into a file in DQDIMACS format, which can be read by the currently only competing solver iDQ [11].

As benchmark instances we use 4381 formulas, resulting from the verification of incomplete circuits [9,11,21] and controller synthesis [10]. The synthesis benchmarks are those shipped with the tool *Demiurge 1.1.0* [10]. We used the encoding described in [10] to create a DQBF formulation.

All experiments were run on one Intel Xeon E5-2650v2 core at 2.60 GHz with 64 GB of main memory, running Ubuntu Linux 12.04 in 64-bit mode as operating system. We aborted all experiments whose computation time exceeded 900 seconds or which required more than 8 GB of memory. For solving QBFs, we use DepQBF 4.0 [47,48] with the QBF preprocessor bloqqer [18] (version 35) if the matrix is in CNF, and AIGsolve [23] if the matrix is given as an AIG.

We used two parameter settings for preprocessing, in the following called V_1 and V_2. Both use the detection of backbones (by syntactic and semantic checks), monotonic variables (by syntactic checks), and equivalent variables (both using the binary implication graph and structure extraction). We reduce the dependency sets of the existential variables using the standard dependency scheme and structure extraction. For these operations, the functional definitions are only detected, but neither are the defined variables replaced by their definition nor are the defining clauses removed.

- V_1 additionally enables structure extraction, which replaces the defined variables by their definitions. V_1 does not yield a CNF representation, but rather an And-Inverter Graph (AIG) [49] for the formula. Since iDQ requires a CNF representation of the matrix, V_1 can only be combined with HQS.
- V_2 applies BCE after adding hidden and covered literals and variable elimination by resolution ($\varepsilon = 1.1$), but disables structure extraction. V_2 yields a matrix in CNF; therefore it can be combined with both iDQ and HQS.

Table 1. Effect of preprocessing

Solver	Filter	Preproc.	Solved
none	$k = 1$	none	935
none	$k = 1$	V_1	2459
none	$k = 2$	V_1	2733
none	$k = 1$	V_2	2240
HQS	none	none	1537
HQS	none	V_1	3629
HQS	$k = 1$	V_1	3752
HQS	$k = 1$	V_1 + BCE	2174
HQS	$k = 2$	V_1	3737
HQS	$k = 1$	V_2	3542
iDQ	none	none	1073
iDQ	none	V_2	1378
iDQ	$k = 1$	none	1359
iDQ	$k = 1$	V_2	2714

Table 1 shows the number of *solved instances* (out of 4381) for different combinations of preprocessing, filtering (see Section 3), and the HQS or iDQ solver cores. Preprocessing alone can only solve a small fraction of all instances (80 for V_1 and 57 for V_2). The filter solves already 935 instances for $k = 1$ (slightly more with higher values of k). The combination of preprocessing V_1 with the filter allows to decide 2459 instances (2240 with V_2). In spite of using bloqqer as preprocessor for simplifying the QBF over-approximations

for filtering, doing DQBF preprocessing before reduces the solving times for the QBFs. Without DQBF preprocessing, solving the QBF approximation runs into a timeout frequently.

For HQS as solver back-end, the trend is similar: without preprocessing and filtering, HQS is able to solve 1537 instances, with V_1 preprocessing this number increases to 3629 instances, and if filtering is used thereafter, 3752 instances can be solved. We can also see that BCE largely prevents structure extraction: if all described techniques are enabled, only 2174 instances can be solved successfully. Increasing the value of k to 2 does not seem beneficial at least if a time limit of 15 min is used. For larger time limit, $k = 2$ can slightly increase the number of solved instances. Finally, if we combine V_2 with filtering ($k = 1$) and HQS, we can also observe a positive effect on the number of solved instances (3542); however, it is not as strong as with V_1, which includes structure extraction instead of BCE.

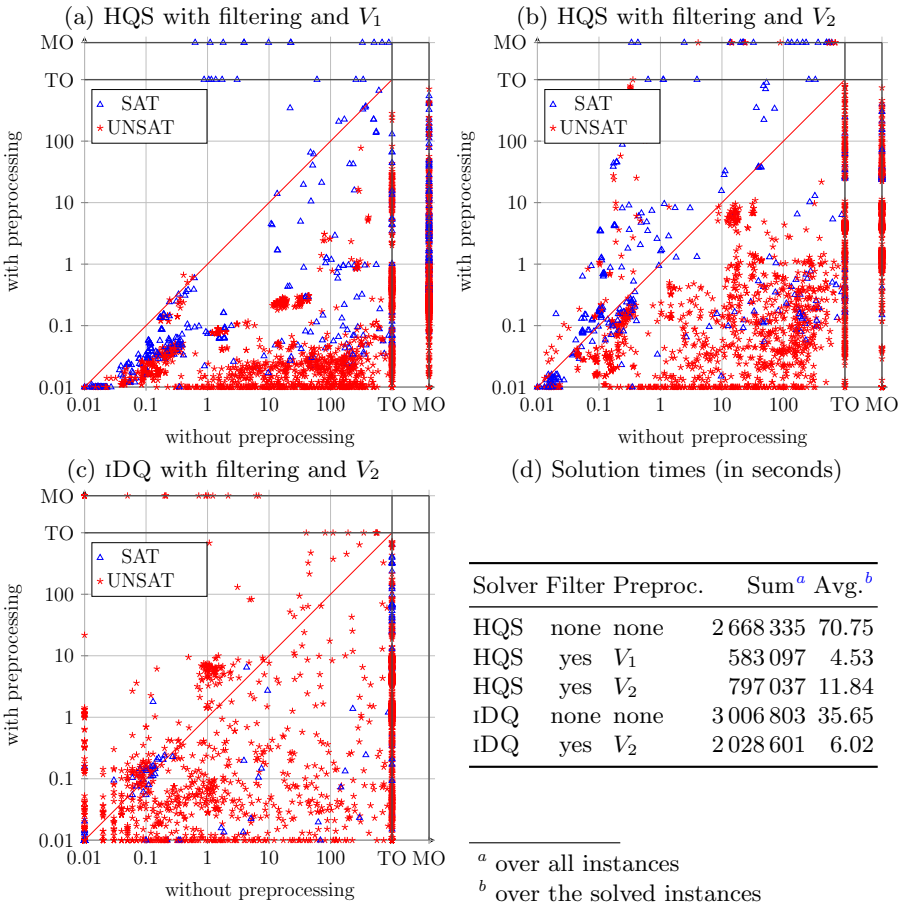
iDQ without filtering and preprocessing solves 1073 instances. This number is increased to 1378 by preprocessing (V_2) and to 1359 instances by filtering ($k = 1$). The combination with filtering and preprocessing yields 2714 solved instances.

In summary, the combination of filtering and preprocessing significantly increases the number of solved instances by a factor of up to 2.44 (for HQS) and 2.52 (for iDQ). The best results are obtained if the preprocessing techniques are chosen according to the solver core.

Now we focus on the *size of the instances* before and after preprocessing. Preprocessing variant V_2 reduces the number of clauses by 64 % on average, the number of existential variables by 76 % on average, but leaving the number of universal variables essentially unchanged. As preprocessing variant V_1 does not yield a CNF representation, we cannot compare the number of clauses. Instead we compare the size of the AIG representation of the matrix before and after preprocessing. V_1 reduces the number of existential variables by 97 % on average (including all Tseitin variables), the number of AIG nodes by 84 %, leaving the number of universal variables almost unchanged, too.

If the CNF structure of the matrix needs to be preserved (as in V_2) not all Tseitin variables can be removed by identifying functional definitions and by elimination by resolution, since this leads to a significant increase in size of the CNF. This effect is lessened by BCE, in particular if HLA and CLA are enabled.

Finally, we take a closer look at the *solving times* of the instances. For the instances which were solved with or without preprocessing and filtering, Fig. 1 compares the computation times when using only the solver core and when using the solver core after preprocessing and filtering. The times include everything from reading the input files to termination. The upper two pictures show HQS with V_1 (Fig. 1(a)) and V_2 (Fig. 1(b)) and filtering using $k = 1$, compared to HQS without preprocessing and filtering. Fig. 1(c) shows iDQ with V_2, compared to iDQ without preprocessing. In Fig. 1(d) we present the accumulated running times over all instances (unsolved instances contributing the time limit of 900 seconds) and the average running time of the solved instances.



Solver	Filter	Preproc.	Sum[a]	Avg.[b]
HQS	none	none	2 668 335	70.75
HQS	yes	V_1	583 097	4.53
HQS	yes	V_2	797 037	11.84
iDQ	none	none	3 006 803	35.65
iDQ	yes	V_2	2 028 601	6.02

[a] over all instances
[b] over the solved instances

Fig. 1. Running times (in seconds) for HQS and iDQ with and without preprocessing.

In all three cases, preprocessing and filtering reduce the computation times for the vast majority of instances significantly, often by orders of magnitude. The very few exceptions in case of iDQ are instances that are very easy to solve such that the overhead for preprocessing exceeds the solving time. We can also observe that many instances, for which the solver core alone ran into a time out or memory out, can be solved successfully after preprocessing and filtering.

6 Conclusion

We have shown how preprocessing techniques for SAT and QBF can be generalized to DQBF. Experiments have demonstrated that they can reduce the running time of the actual solving process by orders of magnitude, both for CNF-based and non-CNF-based solver cores.

In future we want to investigate more powerful dependency schemes and how the flexibility in the dependency sets can be exploited when choosing sets of universal variables to eliminate in order to obtain a QBF.

References

1. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers **58**, 117–148 (2003)
2. Czutro, A., Polian, I., Lewis, M.D.T., Engelke, P., Reddy, S.M., Becker, B.: TIGUAN: thread-parallel integrated test pattern generator utilizing satisfiability analysis. In: International Conference on VLSI Design, pp. 227–232. IEEE Computer Society, New Delhi, India (2009)
3. Rintanen, J.: Constructing conditional plans by a theorem-prover. Journal of Artificial Intelligence Research **10**, 323–352 (1999)
4. Sinz, C., Kaiser, A., Küchlin, W.: Formal methods for the validation of automotive product configuration data. AI EDAM **17**(1), 75–97 (2003)
5. Mironov, I., Zhang, L.: Applications of SAT solvers to cryptanalysis of hash functions. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 102–115. Springer, Heidelberg (2006)
6. Cook, S.A.: The complexity of theorem-proving procedures. In: Annual ACM Symposium on Theory of Computing (STOC), ACM Press, pp. 151–158 (1971)
7. Meyer, A.R., Stockmeyer, L.J.: Word problems requiring exponential time: Preliminary report. In: Annual ACM Symposium on Theory of Computing (STOC), pp. 1–9. ACM Press (1973)
8. Peterson, G., Reif, J., Azhar, S.: Lower bounds for multiplayer non-cooperative games of incomplete information. Computers and Mathematics with Applications **41**(7–8), 957–992 (2001)
9. Gitina, K., Reimer, S., Sauer, M., Wimmer, R., Scholl, C., Becker, B.: Equivalence checking of partial designs using dependency quantified Boolean formulae. In: IEEE Int'l Conf. on Computer Design (ICCD), Asheville, NC, USA, IEEE Computer Society, pp. 396–403 (2013)
10. Bloem, R., Könighofer, R., Seidl, M.: SAT-based synthesis methods for safety specs. In: McMillan, K.L., Rival, X. (eds.) VMCAI 2014. LNCS, vol. 8318, pp. 1–20. Springer, Heidelberg (2014)
11. Fröhlich, A., Kovásznai, G., Biere, A., Veith, H.: iDQ: Instantiation-based DQBF solving. In: Berre, D.L. (ed.) Int'l Workshop on Pragmatics of SAT (POS). EPiC Series, vol. 27, pp. 103–116. Vienna, Austria, EasyChair (2014)
12. Gitina, K., Wimmer, R., Reimer, S., Sauer, M., Scholl, C., Becker, B.: Solving DQBF through quantifier elimination. In: Int'l Conf. on Design, Automation and Test in Europe (DATE), Grenoble, France, IEEE (2015)
13. Jr., R.J.B., Schrag, R.: Using CSP look-back techniques to solve real-world SAT instances. In: Kuipers, B., Webber, B.L. (eds.): National Conference on Artificial Intelligence / Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI), Providence, Rhode Island, USA, AAAI Press / The MIT Press, pp. 203–208 (1997)
14. Silva, J.P.M., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Transactions on Computers **48**(5), 506–521 (1999)
15. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 61–75. Springer, Heidelberg (2005)

16. Manthey, N.: Coprocessor 2.0 – a flexible CNF simplifier. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 436–441. Springer, Heidelberg (2012)
17. Giunchiglia, E., Marin, P., Narizzano, M.: sQueezeBF: an effective preprocessor for QBFs based on equivalence reasoning. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 85–98. Springer, Heidelberg (2010)
18. Biere, A., Lonsing, F., Seidl, M.: Blocked clause elimination for QBF. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS, vol. 6803, pp. 101–115. Springer, Heidelberg (2011)
19. Kilby, P., Slaney, J.K., Thiébaux, S., Walsh, T.: Backbones and backdoors in satisfiability. In: Veloso, M.M., Kambhampati, S. (eds.): National Conference on Artificial Intelligence / Int'l Conf. on Innovative Applications of Artificial Intelligence (IAAI), Pittsburgh, Pennsylvania, USA, AAAI Press / The MIT Press, pp. 1368–1373 (2005)
20. Janota, M., Lynce, I., Marques-Silva, J.: Algorithms for computing backbones of propositional formulae. AI Communications **28**(2), 161–177 (2015)
21. Finkbeiner, B., Tentrup, L.: Fast DQBF refutation. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 243–251. Springer, Heidelberg (2014)
22. Järvisalo, M., Biere, A., Heule, M.: Blocked clause elimination. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 129–144. Springer, Heidelberg (2010)
23. Pigorsch, F., Scholl, C.: Exploiting structure in an AIG based QBF solver. In: Conf, I. (ed.) on Design, Automation and Test in Europe (DATE), pp. 1596–1601. IEEE, Nice, France (2009)
24. Biere, A.: Resolve and expand. In: H. Hoos, H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 59–70. Springer, Heidelberg (2005)
25. Wimmer, R., Gitina, K., Nist, J., Scholl, C., Becker, B.: Preprocessing for DQBF (extended version). Reports of SFB/TR 14 AVACS number 110 (2015). http://www.avacs.org
26. Tseitin, G.S.: On the complexity of derivation in propositional calculus. Studies in Constructive Mathematics and Mathematical Logic Part **2**, 115–125 (1970)
27. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. vol. 185 of Frontiers in Artificial Intelligence and Applications. IOS Press (2008)
28. Fröhlich, A., Kovásznai, G., Biere, A.: A DPLL algorithm for solving DQBF. In: Int'l Workshop on Pragmatics of SAT (POS), Trento, Italy (2012)
29. Gitina, K., Wimmer, R., Reimer, S., Sauer, M., Scholl, C., Becker, B.: Solving DQBF through quantifier elimination. Reports of SFB/TR 14 AVACS 107 (2015). http://www.avacs.org
30. Pigorsch, F., Scholl, C.: An AIG-based QBF-solver using SAT for preprocessing. In: Sapatnekar, S.S. (ed.) ACM/IEEE Design Automation Conference (DAC), pp. 170–175. ACM Press, Anaheim, CA, USA (2010)
31. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM Journal on Computing **1**(2), 146–160 (1972)
32. Brafman, R.I.: A simplifier for propositional formulas with many binary clauses. IEEE Transactions on Systems, Man, and Cybernetics, Part B **34**(1), 52–59 (2004)
33. Gelder, A.V.: Toward leaner binary-clause reasoning in a satisfiability solver. Ann. Math. Artif. Intell. **43**(1), 239–253 (2005)
34. Gershman, R., Strichman, O.: Cost-effective hyper-resolution for preprocessing CNF formulas. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 423–429. Springer, Heidelberg (2005)

35. Heule, M.J.H., Järvisalo, M., Biere, A.: Efficient CNF simplification based on binary implication graphs. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 201–215. Springer, Heidelberg (2011)
36. Samer, M., Szeider, S.: Backdoor sets of quantified Boolean formulas. Journal of Automated Reasoning **42**(1), 77–97 (2009)
37. Samer, M.: Variable dependencies of quantified CSPs. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 512–527. Springer, Heidelberg (2008)
38. Lonsing, F., Biere, A.: Efficiently representing existential dependency sets for expansion-based QBF solvers. Electronic Notes in Theoretical Computer Science **251**, 83–95 (2009)
39. Van Gelder, A.: Variable independence and resolution paths for quantified boolean formulas. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 789–803. Springer, Heidelberg (2011)
40. Slivovsky, F., Szeider, S.: Computing resolution-path dependencies in linear time. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 58–71. Springer, Heidelberg (2012)
41. Balabanov, V., Chiang, H.K., Jiang, J.R.: Henkin quantifiers and Boolean formulae: A certification perspective of DQBF. Theoretical Computer Science **523**, 86–100 (2014)
42. Bubeck, U., Kleine Büning, H.: Dependency quantified horn formulas: models and complexity. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 198–211. Springer, Heidelberg (2006)
43. Bubeck, U.: Model-based transformations for quantified Boolean formulas. Ph.D. thesis, University of Paderborn (2010)
44. Heule, M., Järvisalo, M., Biere, A.: Clause elimination procedures for CNF formulas. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR-17. LNCS, vol. 6397, pp. 357–371. Springer, Heidelberg (2010)
45. Heule, M., Järvisalo, M., Biere, A.: Covered clause elimination. In: Voronkov, A., Sutcliffe, G., Baaz, M., Fermüller, C.G. (eds.): Int'l Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR) (Short papers). vol. 13 of EPiC Series, Yogyakarta, Indonesia, EasyChair, pp. 41–46 (2010)
46. Heule, M.J.H., Seidl, M., Biere, A.: Blocked literals are universal. In: Havelund, K., Holzmann, G., Joshi, R. (eds.) NFM 2015. LNCS, vol. 9058, pp. 436–442. Springer, Heidelberg (2015)
47. Lonsing, F., Egly, U.: Incremental QBF solving by DepQBF. In: Hong, H., Yap, C. (eds.) ICMS 2014. LNCS, vol. 8592, pp. 307–314. Springer, Heidelberg (2014)
48. Lonsing, F., Biere, A.: DepQBF: A dependency-aware QBF solver. Journal on Satisfiability, Boolean Modelling and Computation **7**(2–3), 71–76 (2010)
49. Kuehlmann, A., Paruthi, V., Krohm, F., Ganai, M.K.: Robust Boolean reasoning for equivalence checking and functional property verification. IEEE Transactions on CAD of Integrated Circuits and Systems **21**(12), 1377–1394 (2002)