A Compact Representation for Syntactic Dependencies in QBFs

Florian Lonsing and Armin Biere

Institute for Formal Models and Verification
Johannes Kepler University, Linz, Austria
http://fmv.jku.at/

Abstract. Different quantifier types in Quantified Boolean Formulae (QBF) introduce variable dependencies which have to be taken into consideration when deciding satisfiability of a QBF. In this work, we focus on dependencies based on syntactically connected variables. We generalize our previous ideas for efficiently representing dependency sets of universal variables to existential ones. We obtain a dependency graph which is applicable to arbitrary QBF solvers. The core part of our work is the formulation and correctness proof of a static and compact, tree-shaped connection relation over equivalence classes of existential variables. In practice, this relation is constructed once from a given QBF and allows to share connection information among all variables. We report on practical aspects and demonstrate the effectiveness of our approach in experiments on structured formulae from QBF competitions. Further, we show by example that the common approach of quantifier scope analysis is not optimal among syntactic methods for dependency computation.

1 Introduction

In the logic of Quantified Boolean Formulae (QBF), variables can be existentially or universally quantified. This extends propositional logic (SAT), where all variables are existentially quantified, and renders the decision problem of QBF PSPACE-complete [26]. Whereas QBF is not more expressive than SAT, relevant problems from formal verification [6, 11, 19] often can be encoded more compactly in QBF than in SAT.

The two quantifier types in QBF introduce dependencies between differently quantified variables. For example, if (the value of) an existential variable y depends on (the value of) a universal variable x, then a search-based QBF solver must not assign y before x to ensure soundness.

Example 1. In the satisfiable QBF $\forall x \exists y.\ (x \vee \neg y) \wedge (\neg x \vee y)$, y depends on x. If erroneously y is assigned before x then satisfiability can not be concluded.

Dependencies limit the solver's freedom to assign variables and thus influence its performance negatively and complicate the integration of unit propagation and learning as reported in [16–18, 21, 28]. The problem of determining smallest

possible dependency sets is therefore closely related to the practical applicability of QBF solvers. This also applies to memory-bound solvers which eliminate variables, for example by expansion [7, 8] or skolemization [5, 20].

Identifying dependencies in QBFs has been addressed in various ways in previous approaches. Most QBF solvers process formulae in prenex conjunctive normal form (PCNF), where all quantifiers occur in the quantifier prefix and the quantifier-free part of the formula is in CNF. For example, in search-based solvers like [10, 14, 28], dependencies are given by the total linear quantifier ordering in the prefix. Strategies for converting QBFs into PCNF were suggested in [12] to produce optimal prefixes with respect to the number of quantifier alternations. As a more powerful approach, *mini-scoping* was used in [2] to minimize quantifier scopes by shifting quantifiers from the prefix into the formula. Mini-scoping results in a tree shaped dependency relation, which follows the formula structure.

By a similar approach in [4], syntactic quantifier trees were extracted from a PCNF to be used instead of the linear prefix. In expansion-based solvers like [7, 8], dependencies are identified by variable connections. A partial quantifier ordering was derived in [18] by analyzing the quantifier scope structure in non-PCNF formulae prior to conversion into PCNF. Again this results in a tree-shaped prefix which leaves more freedom for choosing decision variables. The same method can implicitly be applied in non-PCNF solvers [13]. All of these approaches mentioned so far are based on syntactic analysis of the QBF.

Informally, y depends on x in a QBF if reordering the quantifiers of x and y in the prefix changes satisfiability. For example, the formula in Ex. 1 becomes unsatisfiable under the prefix $\exists y \forall x$. Dependencies were formalized in [25] in terms of *dependency schemes*. A dependency scheme for a QBF is a binary relation D on the set of variables where $(x, y) \in D$ if y depends on x. In practice, D must be computed according to some strategy which influences the quality of D. Trivially D could be defined to correspond to the prefix: $(x, y) \in D$ if y occurs to the right of x in the prefix and is quantified differently. Such trivial dependency scheme is usually too restrictive. The goal is to minimize dependencies.

Since the problem of computing the optimal, that is the smallest, dependency scheme is PSPACE-hard [25], a trade-off has to be found between efficiency (polynomial time computation) and optimality (non-optimal over-approximation). In this work we focus on dependency computation for QBFs in PCNF by the *standard dependency scheme* D^{std} defined in [25], which is another syntactic approach based on variable connections [7, 8]. As we show, D^{std} can be efficiently represented as a compact graph. This first result gives a structural characterization of the standard dependency scheme. We then show how this graph can be constructed and give experimental results.

Before elaborating our ideas, we review dependency computation by mini-scoping [2, 4] and point out two drawbacks compared to our approach using D^{std}. While considering QBFs in PCNF, we argue that our results can be extended to QBFs with tree-shaped prefixes. Thus they are also applicable to solvers using quantifier scope analysis [4, 13, 18]. Again, using a less restrictive (that is smaller) dependency relation provides more flexibility.

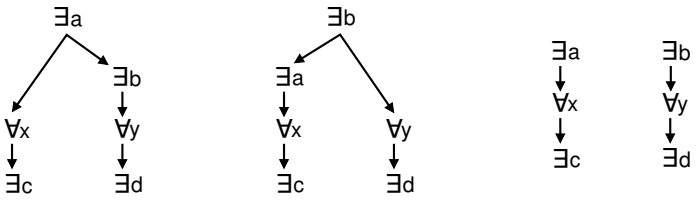

Fig. 1. Two possible quantifier trees for the QBF $\exists a, b \forall x, y \exists c, d.\ (a \vee x \vee c) \wedge (a \vee b) \wedge (b \vee d) \wedge (y \vee d)$ obtained by mini-scoping (left and middle) and dependencies by the standard dependency scheme D^{std} (right). See also Ex. 2 and 4.

1.1 Motivation

Mini-scoping was applied in various contexts as a syntactic method for dependency computation [2, 4, 5, 7, 8, 12]. By rule $(Qx.\ (\phi \wedge \psi)) \equiv (Qx.\ \phi) \wedge \psi$ where $x \notin Var(\psi), Q \in \{\forall, \exists\}$, quantifiers are shifted from the prefix into the formula. Their scopes are reduced to a subset of clauses. This produces a syntactic quantifier tree (parse tree) similar to [4]. For a quantifier tree and a variable x, all differently quantified descendants of x are regarded as depending on x.

Example 2. Consider the QBF $\exists a, b \forall x, y \exists c, d.\ (a \vee x \vee c) \wedge (a \vee b) \wedge (b \vee d) \wedge (y \vee d)$. Minimizing $\exists c$, $\exists d$, $\forall x$ and $\forall y$ yields $\exists a, b.\ (\forall x \exists c.\ (a \vee x \vee c)) \wedge (a \vee b) \wedge (\forall y \exists d.\ (b \vee d) \wedge (y \vee d))$. Now there is the non-deterministic choice whether to first minimize $\exists a$ and then $\exists b$ or vice versa. Fig. 1 shows the quantifier trees for the two alternatives. Dependency schemes resulting from the trees (left and middle) are $D^l = \{(a, x), (x, c), (a, y), (b, y), (y, d)\}$ and $D^m = \{(b, x), (a, x), (x, c), (b, y), (y, d)\}$.

Apart from non-determinism, which has already been reported in [4, 12, 13, 18], mini-scoping as well as quantifier scope analysis [13, 18] is not optimal among syntactic methods for dependency computation. At this point, we informally introduce D^{std} and report its advantage over mini-scoping and scope analysis.

The standard dependency scheme D^{std}, which is the focus of our work, was defined in [25] and is based on ideas from expansion-based solvers [7, 8]. Dependencies are identified by analyzing connections between variables in a PCNF over sequences of clauses as follows.

Definition 1 (X-path). *For $x, y \in V$, where V is the set of variables in the PCNF, and $X \subseteq V$, an X-path between x and y is a sequence $C_1, \ldots, C_k$ of clauses such that $x \in C_1$, $y \in C_k$ and $C_i \cap C_{i+1} \cap X \neq \emptyset$ for $1 \leq i < k$.*

Example 3. For the formula from Ex. 2, there are X-paths between b and y for $X = \{d\}$ and clauses $(b \vee d)$ and $(y \vee d)$, and between a and y for $X = \{b, d\}$ and clauses $(a \vee b)$, $(b \vee d)$ and $(y \vee d)$.

Definition 2 (D^{std} informally). *$(x, y) \in D^{\mathrm{std}}$ whenever x and y are quantified differently and there is an X-path between x and y where X is the set of existential variables to the right of, but not adjacent to x in the quantifier prefix.*

A correctness proof of D^{std} is given in [25] and a formal definition in Def. 5.

Example 4. For the formula from Ex. 2, $D^{\text{std}} = \{(a, x), (x, c), (b, y), (y, d)\}$.

Note that in Ex. 4 $(a, y) \notin D^{\text{std}}$ and $(b, x) \notin D^{\text{std}}$, hence y does not depend on a and x not on b by D^{std}. By Def. 2, a and b are excluded from X, and there are no X-paths for $X = \{c, d\}$ between a, y and b, x in the QBF from Ex. 2.

Comparing dependencies from Ex. 2 and 4 shows a crucial difference between mini-scoping or scope analysis and D^{std}. Dependencies by D^{std} can be strictly less restrictive: no matter which of the two non-deterministically constructed quantifier trees (Fig. 1) are taken for dependency computation, either (a, y) or (b, x) is included in the resulting dependency set, but neither in D^{std}. The same applies to scope analysis like in [13, 18] because any tree-shaped prefix of non-PCNF formulae can in principle be obtained by mini-scoping.

Because of non-determinism and more restrictive dependencies when using mini-scoping or scope analysis, we focus on D^{std}. Our motivation is two-fold. First, we want to extract a static graph representation of D^{std} from a QBF in PCNF. By traversing clauses in a QBF ϕ, $D^{\text{std}}(x)$ for *one* variable $x \in \mathit{Var}(\phi)$ can be computed in $O(|\phi|)$ time [25] where $|\phi|$ is the length of ϕ. However, computing $D^{\text{std}}(x)$ for *all* variables x by the same approach requires $O(|\mathit{Var}(\phi)|.|\phi|)$ time. We construct a directed acyclic graph (DAG) for D^{std}, which has the same worst-case time complexity but can be done efficiently in practice. The idea is similar to quantifier trees by mini-scoping [4] but does not suffer from non-determinism and, as shown, results in a less restrictive dependency relation.

Example 5. Search-based solvers profit from D^{std} because variables can be assigned earlier. In Fig. 1, both a and b have to be assigned before y (left tree) and before x (middle). By D^{std} (right), x and y can be assigned as soon as a, respectively b has been assigned.

Second, we aim at compactness in practice. We take advantage of properties of the connection relations from [7, 8] which allow to merge existential variables into equivalence classes. A static connection relation over equivalence classes is defined which is shared between all variables, thus contributing to compactness.

In this work, we extend our ideas from [22] to existential variables, thus making our work applicable to arbitrary QBF solvers. We develop a formal background for a graph representation of D^{std} in Sec. 3 including proofs. Based on this theoretical part, practical aspects concerning dependency computation and graph construction are reported in Sec. 4. In Sec. 5, experimental results on structured formulae demonstrate the effectiveness of our approach.

2 Preliminaries

For a set of propositional variables V, a *literal* is either a variable $x \in V$ or its negation $\neg x$ where $v(x) = x$ and $v(\neg x) = x$ denotes the variable of a literal. A *clause* is a disjunction over literals. A propositional formula is in *conjunctive normal form (CNF)* if it consists of a conjunction over clauses.

A quantified boolean formula (QBF) $S_1 \dots S_n.\, \phi$ in *prenex conjunctive normal form (PCNF)* consists of a propositional formula ϕ in CNF over a set of variables V and a *quantifier prefix* $S_1 \dots S_n$. The quantifier prefix is a linearly ordered set of *scopes* S_i where $S_1 < \dots < S_n$, which forms a partition on the set of variables: $V = S_1 \cup \dots \cup S_n$ where $S_i \neq \emptyset$ and $S_i \cap S_j = \emptyset$ for $1 \leq i, j \leq n$ and $i \neq j$.

A scope S_i is *existential* if it is associated with an existential quantifier, written as $q(S_i) = \exists$ and *universal* otherwise where $q(S_i) = \forall$. The set of existential and universal variables is denoted by $V_\exists = \bigcup S_i$ for $q(S_i) = \exists$ and $V_\forall = \bigcup S_i$ for $q(S_i) = \forall$, respectively. For a variable $x \in S_i$, $s(x) = S_i$ is the scope of x and $q(x) = q(s(x))$ the *type* of x. For two adjacent scopes S_i and S_{i+1} where $1 \leq i < n$, $q(S_i) \neq q(S_{i+1})$. Given a QBF with n scopes, there are $n - 1$ *quantifier alternations*.

For a scope S_i and literal l, $\delta(S_i) = i$ and $\delta(l) = \delta(s(v(l)))$ denote the *level* of S_i and of l, respectively. For scopes S_i, S_j and literals l, k, S_j is *larger* than S_i and k is larger than l if $\delta(S_i) < \delta(S_j)$ and $\delta(l) < \delta(k)$, respectively.

Let $R \subseteq V \times V$ be a binary relation on the set of variables V. The *reflexive and transitive closure* of R is the smallest reflexive and transitive $R' \subseteq V \times V$ such that $R \subseteq R'$. The *reflexive and transitive reduction* of R is the smallest $R' \subseteq V \times V$ such that R and R' have the same reflexive and transitive closure.

In the following, we consider QBFs in PCNF where for all clauses $C = (l_1 \vee \dots \vee l_k)$, $v(l_i) \neq v(l_j)$ and $\delta(l_i) \leq \delta(l_j)$ for $1 \leq i < j \leq k$ and $q(v(l_k)) = \exists$. A clause neither contains multiple nor complementary literals of one and the same variable, all literals are sorted ascendingly according to their level and the largest literal is existential. *Universal reduction* [7, 9] can be applied to remove literals l_k for which $q(v(l_k)) = \forall$. Furthermore, we assume that there occurs at least one literal for each $x \in V$ in the formula.

3 Theoretical Background

The goal of our work is a compact graph representation of the standard dependency scheme D^{std}. In this section we pick up our ideas from [22]. We first define a connection relation over equivalence classes of existential variables. A directed and reduced variant of this relation is tree-shaped and, as we prove, can be used for dependency computation by D^{std}. For reasons of space and conciseness we omit detailed proofs when appropriate. In definitions we explicitly state the types of variables since this is crucial particularly for connection relations.

Definition 3. *For $x \in V$, if $q(x) = \exists$ then $\overline{q(x)} = \forall$ and $\overline{q(x)} = \exists$ otherwise.*

Definition 4. *For a QBF and $q \in \{\exists, \forall\}$, $V_{q,i} = \{y \in V_q \mid \delta(y) \geq i\}$.*

Definition 5 (Standard Dependency Scheme). *For $x \in V, i = \delta(x) + 1$:* $D^{\text{std}}(x) = \{y \in V_{\overline{q(x)},i} \mid \text{there is an } X\text{-path between } x \text{ and } y \text{ for } X = V_{\exists,i}\}$.

By setting $i = \delta(x) + 1$ and $X = V_{\exists,i}$, universal variables as well as variables from the scope of x are excluded from X as already informally in Def. 2. [1]

[1] The correctness proof of D^{std} in [25] is given for $i = \delta(x)$ and, according to the author's remarks, also works when $i = \delta(x) + 1$ as for our purposes.

i	$q(S_i)$	S_i	(a2, e5, e9)
1	∀	a1, a2	(e5, e9, e15)
2	∃	e3, e4, e5	(e3, e8, e13)
3	∀	a6, a7	(e4, a7, e10)
4	∃	e8, e9, e10	(e4, e13, e14)
5	∀	a11, a12	(a1, a6, e8, e14)
6	∃	e13, e14, e15	(a11, a12, e13)

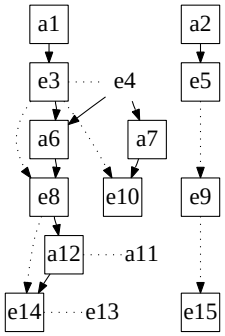

Fig. 2. QBF example. The table on the left shows the levels, quantifiers and variables for each scope in the first three columns and clauses as lists of literals in the last column. Variables and literals are uniquely identified by integers as in QDIMACS format [24]. Identifier prefixes "e" and "a" indicate types ∃ and ∀, respectively. The graph on the right shows a compact representation of D^{std} for the QBF (see also Ex. 10).

Example 6. For the QBF in Fig. 2, $\mathsf{e13} \in D^{\text{std}}(\mathsf{a1})$ by clauses $(\mathsf{a1}, \mathsf{a6}, \mathsf{e8}, \mathsf{e14})$ and $(\mathsf{e3}, \mathsf{e8}, \mathsf{e13})$, and $X = V_{\exists,2} = \{\mathsf{e3}, \mathsf{e4}, \mathsf{e5}, \mathsf{e8}, \mathsf{e9}, \mathsf{e10}, \mathsf{e13}, \mathsf{e14}, \mathsf{e15}\}$.

Different from [7, 8, 25], the following definition of connections is scope-aware.

Definition 6 (Connection). *For $x, y \in V$, x is* connected *to y with respect to scope S_i, written as $x \to_i y$, if, and only if $y \in V_{\exists,i}$ and there is a clause C such that $x \in C$ and $y \in C$. $\to_i^*$ denotes the reflexive and transitive closure of $\to_i$.*

Relation $\to_i^*$ is defined with respect to some scope S_i: if $x \to_i^* y$, then x is connected to y over existential variables from scopes larger than or equal to S_i only. There is a close correspondence between X-paths and $\to_i^*$.

Corollary 1. *For $x, y \in V$, if $x \to_i^* y$, then there is an X-path between x and y for $X = V_{\exists,i}$.*

Due to Def. 6 the converse of Cor. 1 does not hold in general. For example, if there is an X-path between $x \in V_\exists$ and $y \in V_\forall$ then $x \not\to_i^* y$ for all i. A weaker variant can be stated as follows.

Corollary 2. *For $x \in V, y \in V_\exists$, if there is an X-path between x and y for $X = V_{\exists,i}$ and $i \leq min(\delta(x), \delta(y))$, then $x \to_i^* y$.*

Connections with respect to a scope S_j are preserved for any smaller scope S_i.

Corollary 3. *For $x, y \in V, i \leq j$: if $x \to_j^* y$, then also $x \to_i^* y$.*

For proper values of i, connections between existential variables are symmetric because X-paths resulting from Cor. 1 can be reversed.

Lemma 1. *For $x, y \in V_\exists$ and $i \leq min(\delta(x), \delta(y))$: if $x \to_i^* y$ then $y \to_i^* x$.*

Example 7. For the QBF in Fig. 2, $\mathsf{e3} \to_4 \mathsf{e8}$ but $\mathsf{e3} \not\to_5 \mathsf{e8}$, $\mathsf{e8} \to_6 \mathsf{e14}$ and by Cor. 3 also $\mathsf{e8} \to_1 \mathsf{e14}$, further $\mathsf{e3} \to_2^* \mathsf{e14}$ and by Lem. 1 $\mathsf{e14} \to_2^* \mathsf{e3}$.

As a first step towards a compact representation of D^{std} we want to take advantage of situations where two variables can be regarded as equivalent.

Definition 7 (Equivalence). *For $x, y \in V$, x is* equivalent *to y, written as $x \approx y$, if, and only if either (1) $x = y$ or (2) $q(x) = q(y) = \exists$, $\delta(x) = \delta(y) = i$ and $x \to_i^* y$.*

Variables x and y are equivalent if $x = y$ or both are from the same existential scope S_i and are connected by existential variables larger than or equal to S_i.

Theorem 1. *$\approx$ is an equivalence relation. For $x \in V$, $[x]$ is the class of x.*

Proof. Reflexivity is trivial since $x \approx x$ for $x \in V$ by Def. 7. If *not* $q(x) = q(y) = \exists$ then by Def. 7 $x \approx y$ if, and only if $x = y$. Since $=$ is an equivalence relation, symmetry and transitivity of $\approx$ follow immediately. Otherwise, assume $q(x) = q(y) = \exists$. If $x \approx y$ and $x = y$, then also $y \approx x$ by Def. 7. If $x \approx y$ and $x \neq y$ then by Def. 6 and Def. 7 $\delta(x) = \delta(y)$ and $x \to_i^* y$ for $i = \delta(x) = \delta(y)$. Then by Lem. 1 also $y \to_i^* x$ and hence $y \approx x$. Therefore $\approx$ is symmetric. To show transitivity, assume $x \approx y'$ and $y' \approx y$ for $y' \in V$. Then more precisely $y' \in V_\exists$ (because otherwise $x \not\approx y'$ and $y' \not\approx y$) and by Def. 7 also $x \to_i^* y'$, $y' \to_i^* y$ for $i = \delta(x) = \delta(y') = \delta(y)$ and $q(x) = q(y') = q(y)$. By $x \to_i^* y'$, $y' \to_i^* y$ and transitivity of $\to_i^*$, also $x \to_i^* y$, hence $x \approx y$. □

Example 8. For the QBF in Fig. 2: $\mathsf{e3} \approx \mathsf{e4}$ since $q(\mathsf{e3}) = q(\mathsf{e4}) = \exists$, $\delta(\mathsf{e3}) = \delta(\mathsf{e4}) = 2$ and $\mathsf{e3} \to_2^* \mathsf{e4}$ by $\mathsf{e3} \to_2 \mathsf{e8} \to_2 \mathsf{e14} \to_2 \mathsf{e4}$. Also $\mathsf{e13} \approx \mathsf{e14}$ since $\mathsf{e13} \to_6 \mathsf{e14}$ but $\mathsf{e5} \not\approx \mathsf{e4}$ because $\mathsf{e5} \not\to_2^* \mathsf{e4}$. Trivially $\mathsf{a11} \approx \mathsf{a11}$ and $\mathsf{e3} \not\approx \mathsf{e14}$.

Relation $\to_i^*$ is compatible with $\approx$: if two variables are connected then so are all members of their respective classes and vice versa as stated in Lem. 2.

Lemma 2. *Let $x, y \in V, i \leq min(\delta(x), \delta(y))$. Then $x \to_i^* y$ if, and only if $x' \to_i^* y'$ for all $x' \in [x], y' \in [y]$.*

Proof. The proof works regardless of the types of x and y by Def. 6 (reflexivity of $\to_i^*$), Cor. 3 and Def. 7. Trivial cases arise for $V_\forall$. Assume $x \to_i^* y$ for $x, y \in V$ and $i \leq min(\delta(x), \delta(y))$. Then for $x' \in [x], y' \in [y]$, $x' \to_i^* x$ and $y \to_i^* y'$ by Cor. 3 and Def. 7. Since $x' \to_i^* x$, $x \to_i^* y$ (by assumption), $y \to_i^* y'$, also $x' \to_i^* y'$ by transitivity of $\to_i^*$. The other direction can be shown similarly by Lem. 1. □

When regarding $[x]$ as an arbitrary class member, we may write, for example, $[x] \to_i^* [y]$ by Lem. 2. This notation denotes connections between classes.

Lem. 2 would not hold for arbitrary values of i. For example, if $\delta(x) < i$ then $x \not\to_i^* x'$ for $x' \in [x]$, which contradicts Def. 7. The following variant of Lem. 2 does not refer to $[x]$ and holds for arbitrary values of i.

Lemma 3. *Let $x, y \in V$ with $\delta(x) \leq \delta(y)$. Then $x \to_i^* y$ if, and only if $x \to_i^* y'$ for all $y' \in [y]$.*

Example 9. For the QBF in Fig. 2, $\mathsf{e3} \approx \mathsf{e4}$, $\mathsf{e10} \approx \mathsf{e10}$, where $[\mathsf{e10}]$ is a singleton class, and $\mathsf{e4} \to_2^* \mathsf{e10}$ because $\mathsf{e4} \to_2 \mathsf{e10}$. By Lem. 2, also $\mathsf{e3} \to_2^* \mathsf{e10}$ because $\mathsf{e3} \to_2 \mathsf{e8} \to_2 \mathsf{e14} \to_2 \mathsf{e4} \to_2 \mathsf{e10}$.

Besides considering classes in $\rightarrow_i^*$ by Lem. 2, the following relation additionally allows to share information about connections, which is pointed out in Sec. 4.1.

Definition 8 (Directed Connection). $\rightsquigarrow^*$ *denotes the* directed connection relation. *For* $x \in V, y \in V_\exists$, $[x] \rightsquigarrow^* [y]$ *if, and only if,* $\delta(x) \leq \delta(y)$ *and* $x \rightarrow_i^* y$ *for* $i = \delta(x)$. *The reflexive and transitive reduction of* $\rightsquigarrow^*$ *is denoted by* $\rightsquigarrow$.

Corollary 4. *For* $x, y \in V$: *if* $[x] \rightsquigarrow^* [y]$ *then either* $[x] = [y]$ *or* $\delta(x) < \delta(y)$.

Relation $\rightsquigarrow^*$ is defined on classes only and respects the scope ordering. If $[x] \rightsquigarrow^* [y]$ then variables smaller than x are excluded in the connection between x and y. By Cor. 4, if $[x] \rightsquigarrow^* [y]$ then either x and y are in the same class or in different classes but from different scopes. We now prove that our definitions can be used to compute D^{std}.

Theorem 2 (Dependency Computation). *For* $x \in V, i = \delta(x) + 1$:

$$D^{\mathrm{std}}(x) = \{y \in V_{\overline{q(x)},i} \mid \exists w \in V_{\exists,i} : x \rightarrow_i^* w \text{ and } y \rightarrow_i^* w\} \tag{1}$$

$$= \{y \in V_{\overline{q(x)},i} \mid \exists w \in V_{\exists,i} : x \rightarrow_i^* [w] \text{ and } [y] \rightarrow_i^* [w]\} \tag{2}$$

$$= \{y \in V_{\overline{q(x)},i} \mid \exists w \in V_{\exists,i} : x \rightarrow_i^* [w] \text{ and } [y] \rightsquigarrow^* [w]\} \tag{3}$$

Proof. Equivalence of left (LHS) and right-hand sides (RHS) of Eqn. 1 to 3.

- LHS(1) = RHS(1): Assume X-path P between x and y by clauses $C_1, \ldots, C_k$ where $y \in V_{\overline{q(x)},i}$. P can be split into P_1 between x, w for clauses $C_1, \ldots, C_j$ where $w \in C_j, 1 \leq j \leq k, w \in V_{\exists,i}$ and P_2 between w, y by clauses $C_j, \ldots, C_k$. By P_1 and Cor. 2 also $x \rightarrow_i^* w$ and by reversing P_2 and Cor. 2, also $y \rightarrow_i^* w$ and hence $y \in$ RHS(1). For the other direction, assume $x \rightarrow_i^* w$ and $y \rightarrow_i^* w$. Then by Cor. 1, there are X-paths P_1 between x, w and P_2 between y, w for $X = V_{\exists,i}$. An X-path P between x, y can be constructed by combining P_1 with reversed P_2, thus $y \in$ LHS(1).
- RHS(1) = RHS(2): Assume $x \rightarrow_i^* w$ and $y \rightarrow_i^* w$. Since $w \in V_{\exists,i}$, also $\delta(x) \leq \delta(w)$ and hence by Lem. 3 and Def. 7 also $x \rightarrow_i^* [w]$. Further, because $i \leq \delta(y)$ and $i \leq \delta(w)$ and hence $i \leq min(\delta(y), \delta(w))$, also $[y] \rightarrow_i^* [w]$ by Lem. 2 and Def. 7. Since $x \rightarrow_i^* [w]$ and $[y] \rightarrow_i^* [w]$, also $y \in$ RHS(2). For the other direction, assume $x \rightarrow_i^* [w]$ and $[y] \rightarrow_i^* [w]$. Similar arguments apply to derive $x \rightarrow_i^* w$ and $y \rightarrow_i^* w$ by Lem. 2, Lem. 3 and Def. 7. Hence $y \in$ RHS(1).
- RHS(2) = RHS(3): Assume $x \rightarrow_i^* [w]$ and $[y] \rightarrow_i^* [w]$. Since LHS(1) = RHS(1) = RHS(2), there is an X-path P between x, y for $X = V_{\exists,i}$ and clauses $C_1, \ldots, C_k$ where $y \in C_k$. Let l denote the largest literal in C_k. By assumptions in Sec. 2, $v(l) \in V_\exists$ and more precisely $\delta(y) \leq \delta(l)$ (if $q(y) = \forall$ then $\delta(y) < \delta(l)$). Assume that $w = v(l)$. Then $\delta(y) \leq \delta(w)$. By $y, w \in C_k$ also $y \rightarrow_j w$ for $j = \delta(y)$ and $y \rightarrow_j^* w$ by Def. 6. By $y \rightarrow_j^* w$ and $\delta(y) \leq \delta(w)$ also $[y] \rightsquigarrow^* [w]$. Since $x \rightarrow_i^* [w]$ and $[y] \rightsquigarrow^* [w]$ also $y \in$ RHS(3). For the other direction, Def. 8, Cor. 3 and Lem. 2 apply. □

4 Practical Application

In Thm. 2, Eqn. 1 is similar to computation by X-paths in Def. 5, Eqn. 2 refers to classes rather than individual variables, which is already an improvement. The step from Eqn. 2 to Eqn. 3 is the most interesting one for practical applications, yet this is not apparent from theory. Since $\rightsquigarrow^*$ is directed, it restricts the set of classes to be considered when connections of a variable are determined. In practice this contributes to compactness in addition to equivalence classes. In this section we first examine properties of $\rightsquigarrow^*$ over existential variables which allow to efficiently represent its reflexive and transitive reduction $\rightsquigarrow$ as a tree. This tree can be shared between all variables and is the basis for a graph data-structure representing D^{std}.

4.1 A Tree-Shaped Representation of $\rightsquigarrow$

Since $\rightsquigarrow^*$ is directed by Def. 8 and hence also antisymmetric and acyclic, its transitive reduction $\rightsquigarrow$ is unique [1]. The following lemma states a property of $\rightsquigarrow^*$ which accounts for the tree structure of $\rightsquigarrow$.

Lemma 4. *Let $x, y, z \in V_\exists$ where $\delta(x) \leq \delta(y)$. If $[x] \rightsquigarrow^* [z]$ and $[y] \rightsquigarrow^* [z]$ then $[x] \rightsquigarrow^* [y]$.*

Proof. Assume $[x] \rightsquigarrow^* [z]$ and $[y] \rightsquigarrow^* [z]$ where $\delta(x) \leq \delta(y)$. Then by Def. 8, $x \rightarrow_i^* z$ for $i = \delta(x)$ and $y \rightarrow_j^* z$ for $j = \delta(y)$ and $\delta(x) \leq \delta(y) \leq \delta(z)$. By Cor. 3 also $y \rightarrow_i^* z$ and by Lem. 1 $z \rightarrow_i^* y$. By Def. 6, $x \rightarrow_i^* z$ and $z \rightarrow_i^* y$, also $x \rightarrow_i^* y$ and $[x] \rightsquigarrow^* [y]$. □

If $[x] \rightsquigarrow^* [z]$ and $[y] \rightsquigarrow^* [z]$ for existential variables x, y, z and $\delta(x) \leq \delta(y)$ then by Lem. 4 $[x] \rightsquigarrow^* [z]$ is transitive. As a consequence $[x] \not\rightsquigarrow [z]$: at most one class is related to another one in $\rightsquigarrow$. Hence $\rightsquigarrow$ can directly be represented as a forest, that is a collection of trees.

Definition 9 (Connection Forest). *The* connection forest *(c-forest) for a QBF with m existential scopes is a collection of trees over $V_\exists$ with respect to $\approx$ with the following properties:*

1. *For $x, y \in V_\exists$: there is an edge $([x], [y])$ if, and only if $[x] \rightsquigarrow [y]$.*
2. *For $x, y \in V_\exists$: there is a path from $[x]$ to $[y]$ if, and only if $[x] \rightsquigarrow^* [y]$.*
3. *The maximum length (number of edges) of a path is $m - 1$ (by Cor. 4).*

4.2 Dependency Computation by Connection-Forests

The c-forest represents directed connections between existential variables. To compute $D^{\text{std}}(x)$ for arbitrary $x \in V$, a set of proper classes has to be found in the c-forest which exactly denote *all* connections of x to larger existential variables. Classes in such a set must be connected to x and be *minimal* with respect to the scope ordering since edges in the c-forest are directed. Descendants of such classes in the c-forest then comprise *all* connections of x by $\rightsquigarrow^*$.

Definition 10 (Smallest Ancestor). *For $y \in V_\exists, i \leq \delta(y)$ and the c-forest, let $h(i,[y]) = [y']$ such that $y' \in V_{\exists,i}, [y'] \rightsquigarrow^* [y]$ and there is no $y'' \in V_{\exists,i}$ with $i \leq \delta(y'') < \delta(y')$ and $[y''] \rightsquigarrow^* [y]$.*

Class $h(i,[y])$ is the smallest ancestor of $[y]$ which is larger than or equal to S_i, hence $h(i,[y])$ is minimal with respect to S_i and the scope ordering.

Definition 11 (Descendants). *For $x \in V$ and the c-forest, the set of descendants $H_i^*(x)$ with respect to scope S_i is defined as follows:*

1. $V_{C,i}(x) := \{[y] \mid y \in V_{\exists,i} \text{ and } x \rightarrow_i y\}$
2. $H_i(x) := \{[z] \mid [z] = h(i,[y]) \text{ for } [y] \in V_{C,i}(x)\}$
3. $H_i^*(x) := \{[y] \mid [z] \rightsquigarrow^* [y] \text{ for } [z] \in H_i(x)\}$

From clauses containing x, classes of existential variables larger than or equal to S_i are collected in $V_{C,i}(x)$. $H_i(x)$ contains smallest ancestors with respect to S_i for classes in $V_{C,i}(x)$. $H_i^*(x)$ comprises descendants of classes in $H_i(x)$ and represents all connections of x to existential variables larger than or equal to S_i.

Corollary 5. *For $x \in V$: if $[y] \in H_i^*(x)$ then $x \rightarrow_i^* y$.*

For $x \in V$, $H_i^*(x)$ exactly characterizes connections of x to existential variables. This is sufficient for computing $D^{\mathrm{std}}(x)$. Informally, there is a dependence between two differently quantified variables if their sets of descendants in the c-forest are not disjoint.

Theorem 3 (Dependency Computation). *For $x \in V, i = \delta(x) + 1$:*
$D^{\mathrm{std}}(x) = \{y \in V_{\overline{q(x)},i} \mid H_i^*(x) \cap H_j^*(y) \neq \emptyset \text{ for } j = \delta(y)\}$.

Proof. Assume $x \in V$ and $i = \delta(x) + 1$. Direction $\supseteq$ follows right from Def. 11, Cor. 5, Cor. 3 and Thm. 2. To show $\subseteq$, assume $y \in D^{\mathrm{std}}(x)$. Then there is an X-path P between x, y for $X = V_{\exists,i}$. Hence there are clauses $C_1, \ldots, C_k$ where $y, y_k \in C_k$ for some $y_k \in V_{\exists,i}$ with $\delta(y) \leq \delta(y_k)$. Such y_k always exists since by assumption the largest literal in a clause is existential. [2] Then P is also an X-path between x and y_k by $C_1, \ldots, C_k$ and hence $x \rightarrow_i^* y_k$ and $\delta(x) < \delta(y_k)$ since $i \leq \delta(y_k)$, $i = \delta(x) + 1$. We show that $[y_k] \in H_i^*(x) \cap H_j^*(y)$ for $j = \delta(y)$.

Since $y, y_k \in C_k$ by P, also $[y_k] \in V_{C,j}(y)$. Then $[z'] \in H_j(y)$ where $[z'] = h(j,[y_k])$ for $j = \delta(y)$. By Def. 10, $[z'] \rightsquigarrow^* [y_k]$, hence $[y_k] \in H_j^*(y)$.

Since P connects x and y_k, also $x, y_1 \in C_1$ for some $y_1 \in V_{\exists,i}$. Thus $[y_1] \in V_{C,i}(x)$ and $[z_1] \in H_i(x)$ for $[z_1] = h(i,[y_1])$. Then by Def. 10, $[z_1] \rightsquigarrow^* [y_1]$. P is also an X-path between y_1 and y_k by $C_1, \ldots, C_k$, hence $y_1 \rightarrow_i^* y_k$ and $\delta(x) < \delta(y_1), \delta(x) < \delta(y_k)$. Let w denote the smallest connecting variable in P between y_1, y_k: $m = \delta(w) = min(\{\delta(v) \mid v \in C_i \cap C_{i+1} \cap X, 1 \leq i < k\})$. Since m is minimal, also $y_1 \rightarrow_m^* w$, $w \rightarrow_m^* y_k$ and by Lem. 1 $w \rightarrow_m^* y_1$. By Def. 8 and since $m = \delta(w)$, also $[w] \rightsquigarrow^* [y_1]$, $[w] \rightsquigarrow^* [y_k]$. By Lem. 4, $[z_1] \rightsquigarrow^* [y_1]$ and $[w] \rightsquigarrow^* [y_1]$, also $[z_1] \rightsquigarrow^* [w]$. Then by $[z_1] \rightsquigarrow^* [w]$, $[w] \rightsquigarrow^* [y_k]$ and transitivity also $[z_1] \rightsquigarrow^* [y_k]$, hence $[y_k] \in H_i^*(x)$ because $[z_1] \in H_i(x)$. □

In contrast to Thm. 2, practical application follows right from Thm. 3. For a QBF, dependencies can be identified by checking descendants in the c-forest.

[2] If $x \in V_\forall$ then $y \in V_\exists$ and we may choose $y_k = y$.

4.3 A Graph Representation of D^{std}

We describe a static graph representation of D^{std} for a given QBF which is compact in practice. Representing each pair $(x, y) \in D^{\text{std}}$ as a separate edge yields a graph with $|V|^2$ edges in the worst case. Instead, this can often be avoided by building the c-forest once and inserting edges as follows.

First, if $x \in V_\forall$ then by Thm. 3 any member y' of a class $[y] \in H_i^*(x)$ for $i = \delta(x) + 1$ depends on x (see also Thm. 3 in [22]). Thus the c-forest and set $H_i(x)$ compactly represent $D^{\text{std}}(x)$ (see also Ex. 10). In the graph $H_i(x)$ is represented as edges from class $[x]$, which is singleton by Def. 7, to classes in the c-forest. After $H_i(x)$ for all $x \in V_\forall$ have been determined, universal classes $[y_1], [y_2]$ are merged whenever $H_j(y_1) = H_j(y_2)$ for $j = \delta(y_1)+1 = \delta(y_2)+1$ and either $H_j(y_1)$ or $H_j(y_2)$ is discarded. This reduces the number of edges in the graph. Such merging does *not* correspond to $\approx$ but is applied as post-processing.

Second, if $x \in V_\exists$ then edges for $y \in D^{\text{std}}(x)$ need to be inserted explicitly in the graph. For $x \in V_\exists$ and descendant $[y'] \in H_i^*(x)$ where $i = \delta(x) + 1$, there is an edge from variable x to $[y]$ for $y \in V_{\forall,i}$ if $[y'] \in H_j(y)$ for $j = \delta(y) + 1$. This amounts to checking descendants in $H_i^*(x)$ and sets $H_j(y)$ for $y \in V_{\forall,i}$. Since universal classes have been merged before, again the number of inserted edges is reduced. Edges corresponding to transitive dependencies are discarded.

Example 10. In the graph in Fig. 2, boxes denote class representatives. Dotted vertical pointers like from [e3] to [e8] correspond to $\rightsquigarrow$ and denoted edges in the c-forest, dotted horizontal edges like between e13 and e14 connect class members and solid vertical pointers indicate dependencies. The classes of a11 and a12 have been merged in post-processing. The dependency $\mathsf{e15} \in D^{\text{std}}(\mathsf{a2})$ is represented implicitly by the pointer from [a2] to [e5] and the path from [e5] to [e15]. Also $\mathsf{e13} \in D^{\text{std}}(\mathsf{a11})$ by the pointer from [a12] to [e14] and $\mathsf{a11} \in D^{\text{std}}(\mathsf{e8})$ by the pointer from [e8] to [a12]. Further $\mathsf{a7} \in D^{\text{std}}(\mathsf{e4})$ by the pointer from e4 to [a7], but $\mathsf{a7} \notin D^{\text{std}}(\mathsf{e3})$ since $[\mathsf{e10}] \notin H_3^*(\mathsf{e3})$.

5 Experimental Results

We have implemented a tool which constructs the graph representing D^{std} for a given QBF as described in Sec. 4.3. Tab. 1 shows experimental results with conclusions. In a first pass over the clauses, the c-forest is incrementally built by maintaining relation $\rightsquigarrow$ whenever pairs of existential literals l_1, l_2 are encountered in a clause. Additionally, sets $H_i(x)$ for $x \in V, i = \delta(x)+1$ are updated for literal pairs l_1, l_2 where $v(l_1) = x$, $\delta(l_1) < \delta(l_2)$ and either $q(v(l_1)) = q(v(l_2)) = \exists$ or $q(v(l_1)) = \forall$ and $q(v(l_2)) = \exists$. An efficient union-find data structure [27] is used to represent classes. In subsequent passes over the c-forest for $x \in V_\exists$, pointers representing dependencies of existential variables are inserted.

By using the c-forest over equivalence classes as basis for the graph of D^{std}, both time and memory requirements are kept small. For a given QBF ϕ, the graph can be constructed in $O(|V|.|\phi|)$ time and $O(|V|^2)$ space, when keeping edges for transitive dependencies. We observed that time required for removing

	QBFEVAL'05	*QBFEVAL'06*	*QBFEVAL'07*	*QBFEVAL'08*
size	211	216	1136	3328
total time	7.94	1.35	227.05	300.31
max. time	0.58	0.03	7.96	8.11
avg. time	0.04	0.01	0.2	0.09
$x \in V_\forall$				
max. $\lvert D^{\text{std}}(x)\rvert$	256535	9993	2177280	2177280
avg. $\lvert D^{\text{std}}(x)\rvert$	82055.87	4794.60	33447.6	19807
max. $\lvert H_i(x)\rvert$	256	1	518	518
avg. $\lvert H_i(x)\rvert$	3.26	0.98	2.02	1.14
max. $\lvert H_i^*(x)\rvert$	797	5	797	1872
avg. $\lvert H_i^*(x)\rvert$	19.51	1.12	39.06	8.24
avg. $\frac{\lvert\{[y]\in D^{\text{std}}(x)\}\rvert}{\lvert\{y\in D^{\text{std}}(x)\}\rvert}$	3.44%	0.04%	6.42%	1.21%
classes per variables	28.2%	10.23%	40.31%	21.29%
$x \in V_\exists$				
max. $\lvert D^{\text{std}}(x)\rvert$	5040	440	5040	22696
avg. $\lvert D^{\text{std}}(x)\rvert$	12.76	2.98	3.24	4
max. $\lvert H_i(x)\rvert$	24	7	490	490
avg. $\lvert H_i(x)\rvert$	0.14	0.13	0.17	0.13
max. $\lvert H_i^*(x)\rvert$	797	7	797	1872
avg. $\lvert H_i^*(x)\rvert$	5.16	0.16	1.32	1.31
avg. $\frac{\lvert\{[y]\in D^{\text{std}}(x)\}\rvert}{\lvert\{y\in D^{\text{std}}(x)\}\rvert}$	2.37%	0.4%	2.76%	2.09%
classes per variables	10.96%	4.99%	11.45%	7.11%

Table 1. Experimental results on publicly available, structured ("fixed" class) instances from QBF competitions 2005 to 2008 [15]. We did not include random instances. Experiments were run on 64-bit Ubuntu Linux 8.04, Intel® Q6700 at 2.66 GHz and 8 GB of memory. For reference, statistical data and a binary of our tool are available from `http://fmv.jku.at/qdag/`. For all formulae in the sets the graph for D^{std} has been built (see also Sec. 4.3 and 5 for comments on graph construction). The first line shows the numbers of formulae per set. Total run time, maximum over all formulae and average per formula are reported in seconds. Statistics are divided into two sections for existential and universal variables, respectively, and always $i = \delta(x)+1$. Maximum and average number of dependencies by D^{std} over all variables are shown. Compactness of the graph is indicated several times. For $x \in V_\forall$ classes in $H_i^*(x)$, which are reachable by ancestors in $H_i(x)$, efficiently represent $D^{\text{std}}(x)$. This becomes apparent when comparing $|H_i(x)|$, $|H_i^*(x)|$ and $|D^{\text{std}}(x)|$. For $x \in V_\exists$, $|H_i(x)|$ and $|H_i^*(x)|$ measure the effort for inserting dependency pointers since, starting from classes in $H_i(x)$, descendants in $H_i^*(x)$ are visited. Further, the average number of dependency classes per dependency for all $x \in V_\forall$ and $x \in V_\exists$, denoted by line $\frac{|\{[y]\in D^{\text{std}}(x)\}|}{|\{y\in D^{\text{std}}(x)\}|}$, is small. Note that classes result from $\approx$ for $x \in V_\exists$ and from post-processing for $x \in V_\forall$. The worst-case is 100%, where each dependency is in a singleton class. This is clearly not the case. The last line in each section shows the average number of classes per variable in each formula. Again, values are far below 100%, hence many variables can be regarded as equivalent.

transitive dependencies is negligible. As the results in Tab. 1 indicate, we achieve compaction of up to two orders of magnitude compared to a graph of D^{std} over variables rather than classes. This is due to the fact that connection information is shared between variables in the c-forest. To increase confidence in our implementation, we have run random tests and tests on formulae from Tab. 1 where we compared dependencies resulting from the graph to those from Def. 5.

6 Conclusion

Using less restrictive dependency schemes than those obtained from mini-scoping or scoping information readily available in structural formulae has the potential to boost performance of QBF solvers considerably. We gave a structural characterization of the simplest such formulation, based on the standard dependency scheme. The standard dependency scheme has so far only been applied in expansion based QBF solvers and preprocessing algorithms. As next step we want to incorporate our dependency analysis into search-based solvers, which currently are restricted to use tree-shaped prefixes. In a search-based solver it is prohibitive to recompute the dependency relation at each decision point. This also applies to static dependency representations based on mini-scoping such as quantifier trees [4]. As quantifier trees, our compact graph representation can be used as a precomputed approximation of actual dependencies. This can also be beneficial for expansion-based solvers.

Even though our algorithms can easily be extended to work on CNF with a tree-shaped prefix, it is not clear at this point how dependencies of variables introduced to encode structural QBF into CNF can be eliminated in order to lift our arguments to arbitrary structural QBF. This would also give us a way to experimentally show that less restrictive dependency schemes are useful for structural QBF solvers as well. As alternative one can try to generalize the concept of dependency schemes to structural formulas. Furthermore, we want to apply similar ideas to more advanced dependency schemes. Finally, we would like to thank Marko Samer for fruitful discussions on dependency schemes.

References

1. A. V. Aho, M. R. Garey, and J. D. Ullman. The Transitive Reduction of a Directed Graph. *SIAM J. Comput.*, 1(2):131–137, 1972.
2. A. Ayari and D. A. Basin. QUBOS: Deciding Quantified Boolean Logic Using Propositional Satisfiability Solvers. In M. Aagaard and J. W. O'Leary, editors, *FMCAD*, volume 2517 of *LNCS*, pages 187–201. Springer, 2002.
3. F. Bacchus and T. Walsh, editors. *Proceedings SAT'05*, volume 3569 of *LNCS*. Springer, 2005.
4. M. Benedetti. Quantifier Trees for QBFs. In Bacchus and Walsh [3], pages 378–385.
5. M. Benedetti. sKizzo: A Suite to Evaluate and Certify QBFs. In R. Nieuwenhuis, editor, *CADE*, volume 3632 of *LNCS*, pages 369–376. Springer, 2005.
6. M. Benedetti and H. Mangassarian. QBF-Based Formal Verification: Experience and Perspectives. *JSAT*, 5:133–191, 2008.

7. A. Biere. Resolve and Expand. In H. H. Hoos and D. G. Mitchell, editors, *SAT (Selected Papers)*, volume 3542 of *LNCS*, pages 59–70. Springer, 2004.
8. U. Bubeck and H. Kleine Büning. Bounded Universal Expansion for Preprocessing QBF. In Marques-Silva and Sakallah [23], pages 244–257.
9. H. Kleine Büning, M. Karpinski, and A. Flögel. Resolution for Quantified Boolean Formulas. *Inf. Comput.*, 117(1):12–18, 1995.
10. M. Cadoli, A. Giovanardi, and M. Schaerf. An Algorithm to Evaluate Quantified Boolean Formulae. In *AAAI/IAAI*, pages 262–267, 1998.
11. N. Dershowitz, Z. Hanna, and J. Katz. Bounded Model Checking with QBF. In Bacchus and Walsh [3], pages 408–414.
12. U. Egly, M. Seidl, H. Tompits, S. Woltran, and M. Zolda. Comparing Different Prenexing Strategies for Quantified Boolean Formulas. In E. Giunchiglia and A. Tacchella, editors, *SAT*, volume 2919 of *LNCS*, pages 214–228. Springer, 2003.
13. U. Egly, M. Seidl, and S. Woltran. A Solver for QBFs in Nonprenex Form. In G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, editors, *ECAI*, volume 141 of *FAIA*, pages 477–481. IOS Press, 2006.
14. E. Giunchiglia, M. Narizzano, and A. Tacchella. QUBE: A System for Deciding Quantified Boolean Formulas Satisfiability. In R. Goré, A. Leitsch, and T. Nipkow, editors, *IJCAR*, volume 2083 of *LNCS*, pages 364–369. Springer, 2001.
15. E. Giunchiglia, M. Narizzano, and A. Tacchella. QBF Solver Evaluation Portal, 2001-2009. `http://www.qbflib.org/index_eval.php`.
16. E. Giunchiglia, M. Narizzano, and A. Tacchella. Learning for Quantified Boolean Logic Satisfiability. In *AAAI/IAAI*, pages 649–654, 2002.
17. E. Giunchiglia, M. Narizzano, and A. Tacchella. Backjumping for Quantified Boolean Logic satisfiability. *Artif. Intell.*, 145(1-2):99–120, 2003.
18. E. Giunchiglia, M. Narizzano, and A. Tacchella. Quantifier Structure in Search-Based Procedures for QBFs. *TCAD*, 26(3):497–507, 2007.
19. T. Jussila and A. Biere. Compressing BMC Encodings with QBF. *ENTCS*, 174(3):45–56, 2007.
20. T. Jussila, A. Biere, C. Sinz, D. Kröning, and C. M. Wintersteiger. A First Step Towards a Unified Proof Checker for QBF. In Marques-Silva and Sakallah [23], pages 201–214.
21. R. Letz. Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas. In U. Egly and C. G. Fermüller, editors, *TABLEAUX*, volume 2381 of *LNCS*, pages 160–175. Springer, 2002.
22. F. Lonsing and A. Biere. Efficiently Representing Existential Dependency Sets for Expansion-based QBF Solvers. In *Proc. MEMICS*, pages 148–155, 2008.
23. J. Marques-Silva and K. A. Sakallah, editors. *Proceedings SAT'07*, volume 4501 of *LNCS*. Springer, 2007.
24. QBFLIB. QDIMACS Standard v1.1. `http://www.qbflib.org/qdimacs.html`.
25. M. Samer and S. Szeider. Backdoor Sets of Quantified Boolean Formulas. *Journal of Automated Reasoning (JAR)*, 42(1):77–97, 2009.
26. L. J. Stockmeyer and A. R. Meyer. Word Problems Requiring Exponential Time: Preliminary Report. In *STOC*, pages 1–9. ACM, 1973.
27. R. E. Tarjan. Efficiency of a Good But Not Linear Set Union Algorithm. *J. ACM*, 22(2):215–225, 1975.
28. L. Zhang and S. Malik. Conflict driven learning in a quantified Boolean Satisfiability solver. In L. T. Pileggi and A. Kuehlmann, editors, *ICCAD*, pages 442–449. ACM, 2002.