J Autom Reasoning
DOI 10.1007/s10817-015-9353-1

Quantifier Reordering for QBF

Friedrich Slivovsky[1] · **Stefan Szeider**[1]

Received: 31 October 2014 / Accepted: 2 November 2015

Abstract State-of-the-art procedures for evaluating quantified Boolean formulas often expect input formulas in prenex conjunctive normal form (PCNF). We study dependency schemes as a means of reordering the quantifier prefix of a PCNF formula while preserving its truth value. Dependency schemes associate each formula with a binary relation on its variables (the dependency relation) that imposes constraints on certain operations manipulating the formula's quantifier prefix. We prove that known dependency schemes support a stronger reordering operation than was previously known. We present an algorithm that, given a formula and its dependency relation, computes a compatible reordering with a minimum number of quantifier alternations. In combination with a dependency scheme that can be computed in polynomial time, this yields a polynomial time heuristic for reducing the number of quantifier alternations of an input formula. The resolution-path dependency scheme is the most general dependency scheme introduced so far. Using an interpretation of resolution paths as directed paths in a formula's implication graph, we prove that the resolution-path dependency relation can be computed in polynomial time.

Keywords Quantified Boolean formulas · Quantifier reordering · Variable dependencies · Q-resolution

1 Introduction

The past decade has seen a tremendous increase in the performance of propositional satisfiability (SAT) solvers, to a point where they are used as combinatorial reasoning engines powering state-of-the-art tools in formal verification [2,5,10] and planning [7]. Quantified

✉ Friedrich Slivovsky
fslivovsky@gmail.com

Stefan Szeider
stefan@szeider.net

[1] Institute of Computer Graphics and Algorithms, TU Wien, 1040 Vienna, Austria

Boolean formulas (QBFs) are a generalization of propositional formulas. The satisfiability problem of QBFs (QSAT) offers compact encodings for problems that are assumed to only admit SAT encodings of exponential size, such as unbounded model checking or conformant planning. Most QSAT solvers expect input formulas in prenex conjunctive normal form (PCNF), with quantifiers arranged in a single leading sequence (the quantifier prefix), followed by a purely propositional formula (the matrix) in conjunctive normal form. The number of *quantifier alternations* of a PCNF formula is generally a indicator of hardness for the associated instance of QSAT, as witnessed by the following results:

1. QSAT is complete for the kth level of the polynomial hierarchy when restricted to PCNF formulas with k alternations [15].
2. Without a bound on quantifier alternations, QSAT is PSPACE-hard even for formulas of bounded pathwidth [1].
3. QSAT is fixed-parameter tractable (FPT) if both the treewidth and the number of quantifier alternations are taken as parameters, but the function bounding the complexity in the parameters is a tower of exponentials with height corresponding to the number of quantifier alternations (unless P = NP) [11].

In this paper, we show how *dependency schemes* [12] can be used to manipulate the order of variables in the prefix of a PCNF formula and possibly reduce the number of quantifier alternations. A dependency scheme is a mapping that associates each PCNF formula with a binary relation on its variables (a *dependency relation*). The intuitive interpretation of a pair (x, y) in this relation is that y *may depend on* x, so that changing the order of x and y in the quantifier prefix may change the truth value of the formula. Conversely, if there is no pair (x, y) in the dependency relation, y *does not depend on* x and so y may precede x in a reordering of the quantifier prefix without this changing the truth value. One can define a pointwise partial order on dependency schemes in terms of their dependency relations. Several dependency schemes have been introduced in the literature so far. Among these, the so-called *resolution-path dependency scheme* [16] is minimal with respect to this partial order (see Fig. 1). That is, it imposes a minimal set of ordering constraints.

Previous work considered dependency schemes that support an operation on quantifier prefixes called *shifting* [12]. In this work, we study dependency schemes that support a more general operation: *any* reordering that simultaneously satisfies the constraints encoded in a dependency relation yields an equisatisfiable formula. Calling dependency schemes of this kind *permutation dependency schemes*, we prove the following result.

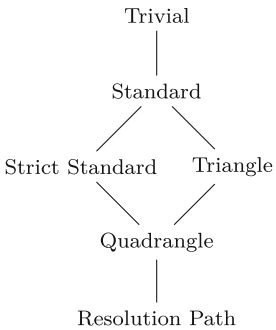

Fig. 1 Dependency schemes, partially ordered by pointwise inclusion of dependency relations

1. The resolution-path dependency scheme is a permutation dependency scheme (Theorem 4). In fact, every dependency scheme satisfying a mild technical condition we call *monotonicity* is a permutation dependency scheme (Proposition 2).

Permutation dependency schemes offer a high degree of freedom in reordering the prefix of a PCNF formula. If a permutation dependency scheme is *tractable*, that is, if the dependency relation can be computed in polynomial time for each PCNF formula, it can be used to efficiently find reorderings with desirable properties.

2. Let D be a tractable permutation dependency scheme. For every PCNF formula Φ, a reordering Ψ of Φ satisfying the following two properties can be computed in polynomial time: (1) Ψ observes the ordering constraints encoded in the dependency relation D_Φ, and (2) among reorderings with this property, Ψ has the fewest quantifier alternations (Theorem 2).

In the work introducing resolution-path dependencies, it was stated as an open problem whether the resolution-path dependency relation can be computed in polynomial time [16]. Observing that resolution paths correspond to paths in a formula's *implication graph*, we answer this question in the affirmative.

3. The resolution-path dependency scheme is tractable: given a PCNF formula, the associated resolution-path dependency relation can be computed in polynomial time (Theorem 5).

Dependency schemes were originally introduced in context with *backdoor sets* of QBFs [12]. QSAT is FPT with the size of a smallest strong backdoor set (to either the class of Horn or 2CNF formulas) taken as a parameter. A strong backdoor set of a PCNF formula is a set of variables with the following property: for every assignment to the variables in this set, the remaining formula belongs to a certain class (for the above result, Horn or 2CNF). For use in QSAT, backdoor sets must be closed under a dependency relation, and smaller (with respect to set inclusion) dependency relations lead to smaller backdoor sets. Since the resolution-path dependency scheme is tractable, it can be used to compute smaller backdoor sets, decreasing the parameter and thus improving the runtime bound of the above FPT result.

The remainder of this paper is organized as follows. In Sect. 2 we introduce basic notation and terminology. We present permutation dependency schemes in Sect. 3 and use them in Sect. 4 to minimize the number of quantifier alternations of PCNF formulas. We introduce the resolution-path dependency scheme in Sect. 5 and prove that it is tractable in Sect. 6. In Sect. 7 we present our conclusions. The Appendix contains a more detailed runtime analysis of the minimization algorithm from Sect. 4.

1.1 Previous Work

Some of the results in this work were published, in preliminary form, as part of the proceedings of SAT'12 and SAT'14. More specifically, a weaker version of Proposition 2, stating that every monotone transposition dependency scheme is a cumulative dependency scheme (rather than a permutation dependency scheme), was proved in [13]. The same paper contains our original proof of tractability for the resolution-path dependency scheme (Theorem 5). The simplified algorithm presented here is from [14].

2 Preliminaries

We will denote the set of natural numbers (that is, positive integers) by $\mathbb{N}$. For any $n \in \mathbb{N}$, we write $[n]$ for the set $\{1, \ldots, n\}$.

2.1 Permutations

Let $s = s_1 \ldots s_n$ be a sequence. By a *permutation* of s we mean a sequence $s_{f(1)} \ldots s_{f(n)}$, where $f : [n] \to [n]$ is a bijection. A *transposition* of s is a permutation $s_{f(1)} \ldots s_{f(n)}$ of s such that $f(i) = i + 1$ and $f(i + 1) = i$ for some $1 \leq i < n$, and $f(j) = j$ for every $j \neq i$.

2.2 Relations

Let $\mathcal{R}$ be a binary relation on a set S. For $s \in S$ we write $\mathcal{R}(s) = \{ t \in S : (s, t) \in \mathcal{R} \}$; for $S' \subseteq S$ we let $\mathcal{R}(S') = \cup_{s \in S'} \mathcal{R}(s)$. By $\mathcal{R}^*$ we denote the reflexive transitive closure of $\mathcal{R}$. That is $\mathcal{R}^*$ denotes the smallest (with respect to set inclusion) relation $\mathcal{S}$ such that $(s, s) \in \mathcal{S}$ for every $s \in S$, and such that $(s, u) \in \mathcal{S}$ whenever $(s, t) \in \mathcal{S}$ and $(t, u) \in \mathcal{S}$.

2.3 Graphs

A *directed graph*, or *digraph*, for short, is a pair $G = (V, E)$, where V is a finite set and $E \subseteq V \times V$ is an irreflexive binary relation. The elements of V and E are called the *vertices* and *edges* of G, respectively. An edge (v, w) of G is an *incoming edge* of w and an *outgoing edge* of v. Let $W \subseteq V$. The *subdigraph of G induced by W* is $G[W] = (W, E \cap (W \times W))$. A *walk (from v_1 to v_n)* in G is a sequence $w = v_1 \ldots v_n$ of vertices such that $(v_i, v_{i+1}) \in E$ for each $i \in [n - 1]$. The *length* of w is $n - 1$. If the v_i are pairwise distinct then w is a *path*. A *longest* (*shortest*) *path* (*from v to u*) in G is a path (from v to u) in G of maximum (minimum) length. A walk $v_1 \ldots v_n$ is *closed* if $v_1 = v_n$. The digraph G is *acyclic* if it does not contain a closed walk. The sets of *out-neighbors* and *in-neighbors* of a vertex v of G are as defined as $N_G^+(v) = \{ w \in V : (v, w) \in E \}$ and $N_G^-(v) = \{ w \in V : (w, v) \in E \}$, respectively.

A *rooted binary tree* is an acyclic digraph $T = (V, E)$ such that every vertex, or *node*, of T has at most one incoming and at most two outgoing edges, and such that there is a unique node r without incoming edges, called the *root* of T. The *parent* of a node $t \in V \setminus \{r\}$ is the unique in-neighbor of t. The out-neighbors of a node t are called *children* of t. Nodes without outgoing edges are called *leaves* of T. The *height* of T is the length of a longest path in T.

2.4 Formulas

A *literal* is a negated or unnegated variable. If x is a variable, we write $\overline{x} = \neg x$ and $\overline{\neg x} = x$, and let $var(x) = var(\neg x) = x$. If X is a set of literals, we write $\overline{X}$ for the set $\{ \overline{x} : x \in X \}$. A *clause* is a finite disjunction of literals, and a *CNF formula* is a finite conjunction of clauses. Whenever convenient, we treat a clause as a set of literals, and a CNF formula as a set of sets of literals. We let $var(C)$ be the set of variables occuring (negated or unnegated) in a clause C. For a CNF formula φ we let $var(\varphi) = \bigcup_{C \in \varphi} var(C)$. A *(quantifier) prefix* is a (possibly empty) sequence $\mathcal{Q} = Q_1 x_1 \ldots Q_n x_n$, where $Q_i \in \{\forall, \exists\}$, and the x_i are pairwise distinct variables. The set of variables occurring in the prefix $\mathcal{Q}$ is $var(\mathcal{Q}) = \{x_1, \ldots, x_n\}$. We let $q_{\mathcal{Q}}(x_i) = Q_i$ and define the relation $R_{\mathcal{Q}}$ as $R_{\mathcal{Q}} = \{ (x_i, x_j) : i < j \}$. We drop the

subscript from $R_{\mathcal{Q}}$ and $q_{\mathcal{Q}}$ whenever the prefix is understood. A *quantifier block* (of $\mathcal{Q}$) is a maximal (sub)sequence $Q_i x_i \dots Q_k x_k$ such that $1 \leq i < k \leq n$ and $Q_i = Q_j$ for each j with $i \leq j \leq k$. Relative to prefix $\mathcal{Q}$, variable x_i is called *existential* (*universal*) if $Q_i = \exists$ ($Q_i = \forall$). The set of existential (universal) variables occurring in a the prefix $\mathcal{Q}$ is denoted $var_\exists(\mathcal{Q})$ ($var_\forall(\mathcal{Q})$). A literal ℓ is existential (universal) relative to a prefix $\mathcal{Q}$ if $var(\ell)$ is existential (universal) relative to $\mathcal{Q}$. A *PCNF formula* is a pair $\Phi = \mathcal{Q} \cdot \varphi$ consisting of a prefix $\mathcal{Q}$ and a CNF formula φ (called the *matrix* of Φ) such that $var(\varphi) = var(\mathcal{Q})$. If $\Phi = \mathcal{Q} \cdot \varphi$ is PCNF formula, we let $var(\Phi) = var(\mathcal{Q})$, $var_\exists(\Phi) = var_\exists(\mathcal{Q})$, and $var_\forall(\Phi) = var_\forall(\mathcal{Q})$. Moreover, we let $lit(\Phi) = var(\Phi) \cup \overline{var(\Phi)}$ and $R_\Phi = R_{\mathcal{Q}}$. We call a clause *tautological* if it contains the same variable negated as well as unnegated. Unless otherwise stated, we assume that the matrix of a PCNF formula contains only non-tautological clauses. The *size* of a PCNF formula $\Phi = \mathcal{Q} \cdot \varphi$ is defined $|\Phi| = \sum_{C \in \varphi} |C|$.

For a set X of variables, a *truth assignment* is a mapping $\tau : X \to \{0, 1\}$. We extend τ to literals by letting $\tau(\neg x) = 1 - \tau(x)$. For a truth assignment $\tau : X \to \{0, 1\}$ and a CNF formula φ we define the result of applying τ to φ as

$$\varphi[\tau] = \left\{C \backslash \tau^{-1}(0) : C \cap \tau^{-1}(1) = \emptyset\right\}.$$

We extend this to PCNF formulas $\Phi = \mathcal{Q} \cdot \varphi$ by defining $\Phi[\tau] = \mathcal{Q}' \cdot \varphi[\tau]$, where $\mathcal{Q}'$ denotes the prefix obtained from $\mathcal{Q}$ by omitting variables (and their associated quantifiers) not occurring in $\varphi[\tau]$. We define an evaluation function eval on PCNF formulas as follows. Let $\Phi = Q_1 x_1 \dots Q_n x_n \cdot \varphi$ be a PCNF formula. If $\varphi = \emptyset$ then $\mathsf{eval}(\Phi) = 1$; if $\emptyset \in \varphi$ then $\mathsf{eval}(\Phi) = 0$; otherwise, we define

$$\mathsf{eval}(\Phi) = \begin{cases} \max\left(\mathsf{eval}\left(\Phi\left[x_1 \mapsto 0\right]\right), \quad \mathsf{eval}\left(\Phi\left[x_1 \mapsto 1\right]\right)\right) & \text{if } \ Q_1 = \exists; \\ \min\left(\mathsf{eval}\left(\Phi\left[x_1 \mapsto 0\right]\right), \quad \mathsf{eval}\left(\Phi\left[x_1 \mapsto 1\right]\right)\right) & \text{otherwise.} \end{cases}$$

Here, $x \mapsto \varepsilon$ denotes the assignment $\tau : \{x\} \to \{0, 1\}$ such that $\tau(x) = \varepsilon$, for $\varepsilon \in \{0, 1\}$. A PCNF formula Φ is *satisfiable* if $\mathsf{eval}(\Phi) = 1$ and *unsatisfiable* otherwise. Two formulas Φ and Ψ are *equisatisfiable*, in symbols $\Phi \equiv \Psi$, if $\mathsf{eval}(\Phi) = \mathsf{eval}(\Psi)$.

2.5 Q-resolution

Let $\Phi = \mathcal{Q} \cdot \varphi$ be PCNF formula. A *tree-like Q-resolution derivation* of clause C from Φ is a pair $\mathcal{D} = (T, \lambda)$, where T is a rooted binary tree T and λ is a labeling function satisfying the following properties. The labeling λ assigns a non-tautological clause to each node, and a variable to each edge. The leaves of T are labeled with clauses of φ, and the root of T is labeled with C. Whenever a node t has two children t' and t'', there is an existential literal ℓ such that $\ell \in \lambda(t')$ and $\overline{\ell} \in \lambda(t'')$, and such that the edges (t, t') and (t, t'') are labeled with $var(\ell)$. Moreover, $\lambda(t) = (\lambda(t') \cup \lambda(t'')) \backslash (\{\ell, \overline{\ell}\})$. We call $\lambda(t)$ the *(Q-)resolvent* of $\lambda(t')$ and $\lambda(t'')$, and say that $\lambda(t)$ is obtained by *resolution* of $\lambda(t')$ and $\lambda(t'')$ on variable $var(\ell)$. If a node t has a single child t', then $\lambda(t) = \lambda(t') \backslash \{\ell\}$ and $\lambda((t, t')) = var(\ell)$ for some *tailing* universal literal ℓ in $\lambda(t')$. A universal literal ℓ is tailing in $\lambda(t')$ if for all existential variables $x \in var(\lambda(t'))$, we have $(x, var(\ell)) \in R_\Phi$. The clause $\lambda(t)$ is the result of *universal reduction* of $\lambda(t')$ on variable $var(\ell)$. We call an instance of resolution or universal reduction in $\mathcal{D}$ a *derivation step* of $\mathcal{D}$. We say $\mathcal{D}$ is *prefix-ordered* if for every path $t_1 \dots t_k$ from the root of T to one of its leaves we have $R_\Phi(x_i, x_j)$ whenever $1 \leq i < j < |var(\Phi)|$, where x_i is the label of (t_i, t_{i+1}) and x_j is the label of (t_j, t_{j+1}). For a tree-like Q-resolution derivation $\mathcal{D} = (T, \lambda)$, we define the set of *resolved variables of* $\mathcal{D}$ as $resvar(\mathcal{D}) = \{\, y \in var_\exists(\Phi) :$ there is an edge $e \in E(T)$ such that $\lambda(e) = y \,\}$. The *height* of a tree-like Q-resolution

derivation $\mathcal{D} = (T, \lambda)$ is the height of T. A tree-like Q-resolution derivation of the empty clause from Φ is called a *Q-resolution refutation* of Φ.

Theorem 1 *A PCNF formula Φ is unsatisfiable if and only if it has a prefix-ordered tree-like Q-resolution refutation.*

Proof Soundness and completeness of Q-resolution was proved by Kleine Büning et al. [6]. The derivations appearing in their proof of completeness are prefix-ordered, and it is straightforward to turn any derivation into a tree-like derivation by copying subderivations. □

3 Dependency Schemes

We are going to consider formulas obtained by reordering the variables in the quantifier prefix of a PCNF formula. In general, reordering may affect satisfiable, and we would like to identify conditions under which reordering leads to an equisatisable formula. For a given formula, these conditions will take the form of a binary relation on its variables representing ordering constraints: a pair (x, y) will be interpreted as "x must precede y", and we require that a reordering observing all constraints be equisatisfiable. A *dependency scheme* is a mapping that associates each PCNF formula with such a set of constraints. It is readily verified that reordering variables within a quantifier block preserves (un)satisfiability. We can use this observation to define a simple dependency scheme as follows.

Definition 1 The *trivial dependency scheme* is the mapping D^{trv} that associates each PCNF formula Φ with the binary relation $D^{\mathrm{trv}}_{\Phi} = \{ (x, y) \in R_{\Phi} : q(x) \neq q(y) \}$ called the *trivial dependency relation* of formula Φ.

The remaining dependency schemes mentioned in Fig. 1 refine the trivial dependency scheme based on a syntactic analysis of input formulas. In principle, any refinement of the trivial dependency scheme could be considered, but not every refinement will be useful for the purpose of reordering quantifier prefixes. Accordingly, we refer to mappings refining the trivial dependency scheme as *proto-dependency schemes*.

Definition 2 A *proto-dependency scheme* is a mapping D that associates each PCNF formula Φ with a relation $D_{\Phi} \subseteq D^{\mathrm{trv}}_{\Phi}$ called the *dependency relation* of Φ with respect to D. A proto-dependency scheme D is *tractable* if D_{Φ} can be computed in polynomial time for every PCNF formula Φ.

We now define what we mean by a transposition or permutation observing ordering constraints encoded in a dependency relation.

Definition 3 Let $\Phi = \mathcal{Q} \cdot \varphi$ and $\Psi = \mathcal{Q}' \cdot \varphi$ be PCNF formulas such that $\mathcal{Q}'$ is a permutation of $\mathcal{Q}$. If $D_{\Phi} \subseteq R_{\Psi}$ we call Ψ a *D-permutation* of Φ. If, in addition, $\mathcal{Q}'$ is a transposition of $\mathcal{Q}$ then Ψ is a *D-transposition* of Φ.

Definition 4 (*Permutation Dependency Scheme*) A *permutation dependency scheme* (*transposition dependency scheme*) is a proto-dependency scheme D such that $\Phi \equiv \Psi$ for every PCNF formula Φ and every D-permutation (D-transposition) Ψ of Φ.

Example 1 Consider the PCNF formula $\Phi = \mathcal{Q} \cdot \varphi$, where

$$\mathcal{Q} = \forall x_1 \exists y_1 \forall x_2 \exists y_2 \forall x_3 \exists y_3, \quad \text{and}$$

$$\varphi = (\neg x_1 \vee y_1) \wedge (\neg y_1 \vee x_2) \wedge (\neg x_2 \vee y_2) \wedge (\neg y_2 \vee x_3) \wedge (\neg x_3 \vee y_3)\,.$$

The trivial dependency relation of this formula is

$$\{(x_1, y_1), (x_1, y_2), (x_1, y_3), (x_2, y_2), (x_2, y_3), (x_3, y_3), \\ (y_1, x_2), (y_1, x_3), (y_2, x_3)\}\,,$$

so the only D^{trv}-permutation of Φ is the formula Φ itself. On the other hand, for the so-called resolution-path dependency scheme D^{res}, to be defined in Sect. 5, the dependency relation is empty and the formula $\forall x_1 \forall x_2 \forall x_3 \exists y_1 \exists y_2 \exists y_3 \cdot \varphi$ is an example of a D^{res}-permutation of Φ.

Permutation dependency schemes are a special case of so-called *cumulative dependency schemes* which are defined in terms of quantifier *shifting* [12]. Shifting a set $X \subseteq var(\Phi)$ of variables amounts to moving X to the right in the quantifier prefix of Φ without changing the order within X or within the set $var(\Phi)\backslash X$ of remaining variables.

Definition 5 (*Shifting*) Let $\Phi = \mathcal{Q} \cdot \varphi$ be a PCNF formula and $X \subseteq var(\Phi)$. The PCNF formula Ψ is obtained from Φ by *(down)-shifting* X, in symbols $\Psi = S^{\downarrow}(\Phi, X)$, if $\Psi = \mathcal{Q}' \cdot \varphi$ and $\mathcal{Q}'$ is a permutation of $\mathcal{Q}$ such that the following conditions hold:

1. $X = R_\Psi(x)$ for some $x \in var(\Phi) = var(\Psi)$,
2. $(x, y) \in R_\Psi$ if and only if $(x, y) \in R_\Phi$ for all $x, y \in X$, and
3. $(x, y) \in R_\Psi$ if and only if $(x, y) \in R_\Phi$ for all $x, y \in var(\Phi)\backslash X$.

Definition 6 (*Dependency Scheme*) A proto-dependency scheme D is a *dependency scheme* if $\Phi \equiv S^{\downarrow}(\Phi, D^*_\Phi(x))$ for every PCNF formula Φ and every $x \in var(\Phi)$.

Definition 7 (*Cumulative*) A dependency scheme D is *cumulative* if the equivalence $\Phi \equiv S^{\downarrow}(\Phi, D^*_\Phi(X))$ holds for every PCNF formula Φ and $X \subseteq var(\Phi)$.

Cumulative dependency schemes were originally introduced in context with *backdoor sets* of PCNF formulas [12]. A strong backdoor set of a PCNF formula is a set of variables with the following property: for every assignment to the variables in this set, the remaining formula belongs to a certain class (for example, the class of PCNF formulas with Horn matrices). A strong backdoor set to a tractable PCNF class can be used algorithmically by instantiating the input formula with every possible assignment of the backdoor variables and solving the remaining (tractable) instance in polynomial time. If the backdoor set is (upward) closed under the dependency relation computed by a cumulative dependency scheme, we can first shift the corresponding variables towards the front of the quantifier prefix and this evaluation strategy is sound. If these backdoor sets can be found efficiently, this leads to a fast decision algorithm for PCNF formulas with small backdoor sets.

It is not difficult to prove that permutation dependency schemes are a special case of cumulative dependency schemes.

Proposition 1 *Every permutation dependency scheme is a cumulative dependency scheme.*

Proof Let D be a permutation dependency scheme and Φ a PCNF formula. Let $X \subseteq var(\Phi)$ and let $\Psi = S^{\downarrow}(\Phi, D^*_\Phi(X))$. We claim that Ψ is a D-permutation of Φ. To prove this, we have to show that $(x, y) \in R_\Psi$ whenever $(x, y) \in D_\Phi$. Let $(x, y) \in D_\Phi$. If $x \in D^*_\Phi(X)$ then also $y \in D^*_\Phi(X)$ and so $(x, y) \in R_\Psi$ by Condition 2 of Definition 5. Suppose $x \notin D^*_\Phi(X)$. If $y \notin D^*_\Phi(X)$ then $(x, y) \in R_\Psi$ by Condition 3. If $y \in D^*_\Phi(X)$ then there is a $z \in var(\Phi)$ such that $D^*_\Phi(X) = R_\Psi(z)$ by Condition 1. We have $x \notin R_\Psi(z)$ and thus $(x, z) \in R_\Psi$. Because

R_Ψ is transitive and $(z, y) \in R_\Psi$ this implies $(x, y) \in R_\Psi$. This proves the claim. Since D is a permutation dependency scheme it follows that $\Psi \equiv \Phi$. So D is a cumulative dependency scheme. □

Clearly, every permutation dependency scheme is a transposition dependency scheme. Though the converse does not hold in general, one can show that every *monotone* transposition dependency scheme is in fact permutation dependency scheme. Intuitively, a proto-dependency scheme D is monotone if the set $D_\Phi(x)$ does not depend on the exact order of variables in $R_\Phi(x)$, and moving x further towards the right in the prefix cannot result in new variables depending—according to D—on x. Most dependency schemes introduced in the literature so far have this property.

Definition 8 (*Monotone*) A proto-dependency scheme D is *monotone* if it satisfies the following condition for every PCNF formula Φ, every D-permutation Ψ of Φ, and every $x \in var(\Phi)$: if $R_\Psi(x) \subseteq R_\Phi(x)$ then $D_\Psi(x) \subseteq D_\Phi(x)$.

Proposition 2 *Every monotone transposition dependency scheme is a permutation dependency scheme.*

Proof Let D be a monotone transposition dependency scheme and let $\Phi = \mathcal{Q} \cdot \varphi$ and $\Psi = \mathcal{Q}' \cdot \varphi$ be PCNF formulas such that Ψ is a D-permutation of Φ. The prefix $\mathcal{Q}'$ can be obtained from $\mathcal{Q}$ by sorting subsequences of increasing length according to the order in $\mathcal{Q}'$, as follows. Assuming that the quantifier/variable pairs $Q_i x_i$ up to $Q_n x_n$ are already sorted according to $\mathcal{Q}'$, we insert the pair $Q_{i-1}x_{i-1}$ at the correct position (according to $\mathcal{Q}'$) by moving it to the right, variable by variable. Because D is monotone, neither reordering to the right of x_{i-1} nor moving x_{i-1} to the right can introduce new dependencies on x_{i-1}, so as long as x_{i-1} is not moved past a variable in $D_\Phi(x_{i-1})$, the fact that D is a transposition dependency scheme guarantees that (un)satisfiability is preserved in each step. Since the formula Ψ is a D-permutation of Φ we have $D_\Phi \subseteq R_\Psi$ and x_{i-1} must end up to the left of $D_\Phi(x_{i-1})$ in the prefix $\mathcal{Q}'$. As $Q_i x_i$ up to $Q_n x_n$ are already sorted according to $\mathcal{Q}'$, this means that x_{i-1} is not moved past an element of $D_\Phi(x_{i-1})$.

To make this more precise, let $\mathcal{Q}_i$ denote the prefix where $Q_i x_i$ up to $Q_n x_n$ are already sorted according to $\mathcal{Q}'$, let $\Phi_i = \mathcal{Q}_i \cdot \varphi$ be the corresponding PCNF formula, and let $R_i = R_{\Phi_i}$. Formally, $\mathcal{Q}_i$ is the unique permutation of $\mathcal{Q}$ such that $R_i = R'_i \cup R''_i \cup R'''_i$, where

$$\begin{aligned} R'_i &= R_\Phi \cap (\{x_1, \ldots, x_{i-1}\} \times \{x_1, \ldots, x_{i-1}\}), \\ R''_i &= R_\Psi \cap (\{x_i, \ldots, x_n\} \times \{x_i, \ldots, x_n\}), \quad \text{and} \\ R'''_i &= \{x_1, \ldots, x_{i-1}\} \times \{x_i, \ldots, x_n\}. \end{aligned}$$

That is, the first part of $\mathcal{Q}_i$ agrees with $\mathcal{Q}$, but the second part is ordered according to $\mathcal{Q}'$. We now show that $\Phi_i \equiv \Phi_{i-1}$ for every $1 < i \leq n$. Since $\Phi_n = \Phi$ and $\Phi_1 = \Psi$, the proposition then follows. So pick any i such that $1 < i \leq n$. We can obtain $\mathcal{Q}_{i-1}$ from $\mathcal{Q}_i$ by moving $Q_{i-1}x_{i-1}$ to the right as far as necessary, and this operation can in turn be represented as a sequence of transpositions where $Q_{i-1}x_{i-1}$ is moved one position to the right. Suppose k is the number of transpositions needed. For $0 \leq j \leq k$, let $\mathcal{Q}'_j$ be the prefix where $Q_{i-1}x_{i-1}$ has been moved j positions to the right in $\mathcal{Q}_i$, let $\Gamma_j = \mathcal{Q}'_j \cdot \varphi$, and let $R'_j = R_{\Gamma_j}$. We claim that $\Gamma_j \equiv \Gamma_{j+1}$ for $0 \leq j < k$. Let $0 \leq j < k$, and let y be the variable immediately to the right of x_{i-1} in $\mathcal{Q}'_j$, such that y trades places with x_{i-1} in $\mathcal{Q}'_{j+1}$. Then $(y, x_{i-1}) \in R_\Psi$ and so $(x_{i-1}, y) \notin D_\Phi$ because Ψ is a D-permutation of Φ. By construction of $\mathcal{Q}'_j$ we have $R'_j(x_{i-1}) \subseteq R_i(x_{i-1})$, and by construction of $\mathcal{Q}_i$ we have $R_i(x_{i-1}) = R_\Phi(x_{i-1})$. So

$R'_j(x_{i-1}) \subseteq R_\Phi(x_{i-1})$, and since D is monotone, this implies $D'_j(x_{i-1}) \subseteq D_\Phi(x_{i-1})$. In particular, $(x_{i-1}, y) \notin D(\Gamma_j)$. It follows from D being a transposition dependency scheme that $\Gamma_j \equiv \Gamma_{j+1}$. This proves the claim. Because $\Gamma_0 = \Phi_i$ and $\Gamma_k = \Phi_{i-1}$, we conclude that $\Phi_i \equiv \Phi_{i-1}$. □

4 Minimizing Quantifier Alternations with Dependency Schemes

The alternation depth of a PCNF formula is its number of quantifier blocks minus one. We use an equivalent graph-theoretic definition that relies on the following interpretation of dependency relations as directed acyclic graphs (DAGs).

Definition 9 (*Dependency DAG*) Let Φ be a PCNF formula and let D be a proto-dependency scheme. The *dependency DAG* of Φ with respect to D, denoted $G(\Phi, D)$, is the directed graph with vertex set $var(\Phi)$ and edge set D_Φ.

Since R_Φ is a total order and $D_\Phi \subseteq R_\Phi$, dependency DAGs are indeed acyclic.

Lemma 1 *Let D be a proto-dependency scheme and Φ a PCNF formula. The dependency DAG of Φ with respect D is acyclic.*

Definition 10 (*Alternation Depth*) Let D be a proto-dependency scheme. The *D-alternation depth* of a PCNF formula Φ is the length of a longest path in $G(\Phi, D)$. The *alternation depth* of a formula is its D^{trv}-alternation depth.

The minimum alternation depth of a D-permutation of Φ differs from the D-alternation depth of Φ by at most one. The following lemma shows that the latter is a lower bound on the alternation depth of D-permutations.

Lemma 2 *Let G be the dependency DAG of Φ with respect to a proto-dependency scheme D. If there is a path $x_1 \dots x_{k+1}$ in G, the alternation depth of a D-permutation of Φ is at least k. If there is another path $y_1 \dots y_{k+1}$ in G such that $q(x_1) \neq q(y_1)$, the alternation depth of a D-permutation of Φ is at least $k+1$.*

Proof Let $x_1 \dots x_{k+1}$ be a path in $G(\Phi, D)$ and let Ψ be a D-permutation of Φ. Since D is a proto-dependency scheme we have $D_\Phi \subseteq D^{\text{trv}}_\Phi$ and thus $q(x_i) \neq q(x_{i+1})$ for each $1 \leq i \leq k$. In combination with $D_\Phi \subseteq R_\Psi$ this yields $(x_i, x_{i+1}) \in D^{\text{trv}}_\Psi$ and thus (x_i, x_{i+1}) is an edge of $G(\Psi, D^{\text{trv}})$ for each $1 \leq i \leq k$. That is, $x_1 \dots x_{k+1}$ is a path in $G(\Psi, D^{\text{trv}})$, so the alternation depth of Ψ is at least k. Suppose there exists a path $y_1 \dots y_{k+1}$ in G such that $q(x_1) \neq q(y_1)$. Then $y_1 \dots y_{k+1}$ is a path in $G(\Psi, D^{\text{trv}})$ as well. Assume without loss of generality that $(y_1, x_1) \in R_\Psi$. Then $(y_1, x_1) \in D^{\text{trv}}_\Psi$ and (y_1, x_1) is an edge in $G(\Psi, D^{\text{trv}})$, so $y_1 x_1 \dots x_{k+1}$ is a path of length $k+1$ in $G(\Psi, D^{\text{trv}})$ and Ψ has alternation depth at least $k+1$. □

We now present an algorithm (Algorithm 1) that computes a D-permutation of minimum alternation depth, matching the lower bound of Lemma 2. Starting with an empty quantifier prefix, Algorithm 1 either removes all existential or all universal source vertices/variables (vertices without incoming edges) from the dependency DAG, appending them to the prefix in an arbitrary order. If there are both existentially and universally quantified source vertices/variables, the algorithm picks a quantifier type with a vertex that is the initial vertex of a longest path. This way, it computes a topological ordering of the dependency DAG

with as few quantifier alternations as possible. The dependency DAG can be constructed in linear time from the dependency relation. The body of the while loop is executed at most $|var(\Phi)|$ times, and each line in the body of the while loop can be implemented so as to run in polynomial time. Overall, we get a polynomial-time algorithm.

Algorithm 1 Minimizing quantifier alternations

Input: A PCNF formula $\Phi = \mathcal{Q} \cdot \varphi$ and a dependency relation D_Φ.
Output: A D-permutation of Φ.
1: $G \leftarrow G(\Phi, D)$
2: $\mathcal{Q}' \leftarrow$ empty quantifier prefix
3: **while** $V(G) \neq \emptyset$ **do**
4: $\quad x \leftarrow$ first vertex of a longest path in G
5: $\quad X \leftarrow \{ v \in V(G) : q(v) = q(x)$ and $N_G^-(v) = \emptyset \}$
6: $\quad \mathcal{Q}'' \leftarrow Qx_1 \dots Qx_{|X|}$ where $Q = q(x)$ and $X = \{x_1, \dots, x_{|X|}\}$
7: $\quad \mathcal{Q}' \leftarrow \mathcal{Q}'\mathcal{Q}''$
8: $\quad G \leftarrow G[V(G) \setminus X]$
9: **end while**
10: **return** $\mathcal{Q}' \cdot \varphi$

Lemma 3 Algorithm 1 *runs in polynomial time.*

A more detailed analysis, proving that the algorithm runs in linear time, is given in the Appendix.

Lemma 4 *Let Φ be a PCNF formula and let D be a proto-dependency scheme. On input (Φ, D_Φ),* Algorithm 1 *computes a D-permutation of Φ with minimum alternation depth.*

Proof Let $\Phi = \mathcal{Q} \cdot \varphi$. By Lemma 3, Algorithm 1 terminates and outputs a PCNF formula $\Psi = \mathcal{Q}' \cdot \varphi$ such that $\mathcal{Q}'$ is a permutation of $\mathcal{Q}$. Let $r > 0$ be the number of times the while loop is executed. For $1 \leq i \leq r$, we let X_i be the set of vertices removed from the dependency DAG during the ith execution of the loop, and set $V_i = \cup_{j=1}^{i} X_j$. Let $G_i = G(\Phi, D)[var(\Phi) \setminus V_i]$ denote the part of the dependency DAG remaining after i executions of the loop body. □

We first prove that Ψ is a D-permutation of Φ. Let $G'_i = G(\Phi, D)[V_i]$ be the part of the dependency DAG containing vertices removed during the first i executions of the loop. Let $E_i = E(G'_i)$ denote its edge set, let $\mathcal{Q}'_i$ be the subsequence of $\mathcal{Q}'$ constructed after i executions, and let R_i denote $R_{\mathcal{Q}'_i}$.

Claim $E_i \subseteq R_i$ for each $i \in [r]$.

Proof The set E_0 is empty so the claim trivially holds for $i = 0$. Let $0 < i \leq r$ and suppose that $E_{i-1} \subseteq R_{i-1}$. Since $R_{i-1} \subseteq R_i$ we immediately get $E_{i-1} \subseteq R_i$. Let $(v, w) \in E_i \setminus E_{i-1}$. We are going to show that $(v, w) \in R_i$. Observe that exactly one of v or w is removed during the ith execution of the main loop: at least one of the vertices is removed and it cannot be the case that both $v \in X_i$ and $w \in X_i$, since variables in X_i are associated with the same quantifier but $G(\Phi, D)$ only contains edges between variables associated with different quantifiers. Suppose $v \in X_i$ but $w \notin X_i$. Then $w \in X_j$ for some $1 \leq j < i$. By choice of X_j, variable w does not have incoming edges in G_{j-1}. Because $v \notin V_{j-1}$, variable v is a vertex of G_{j-1} and so $(v, w) \in E(G_{j-1})$, a contradiction. This only leaves the case where $v \notin X_i$ and $w \in X_i$. In this case v is removed before w, that is, $v \in X_j$ for some $j < i$. By construction, X_j appears before X_i in $\mathcal{Q}'_i$, so $(v, w) \in R_i$. We conclude that $E_i \subseteq R_i$. □

Since $E_r = D_\Phi$ and $R_r = R_\mathcal{Q}$, it follows from the above claim that Ψ is indeed a D-permutation of Φ. For $i \in [r]$, let l_i be the length of a longest path in G_{i-1}. In order to show that $\mathcal{Q}'$ has minimum alternation depth, we first prove another claim.

Claim The following holds for each $i \in [r]$: if every variable that is the initial vertex of a longest path in G_{i-1} is existential or every initial vertex of a longest path in G_{i-1} is universal then $l_i = r - i$; otherwise, $l_i + 1 = r - i$.

Proof Consider the case $i = r$ first. As X_r is the last set removed by the algorithm, all remaining variables must be of the same type, so we have to show that $l_r = r - r = 0$. Since $X_r = V(G_{r-1})$, for any two variables $v, w \in V(G_{r-1})$ we have $q(v) = q(w)$. But $G(\Phi, D)$ only contains edges between variables of associated with different quantifiers, so G_{r-1} must be edgeless and $l_r = 0$.

Now let $1 \leq i < r$ and suppose the claim holds for all j such that $i < j \leq r$. Assume first (1) that any two initial vertices of a maximum-length path in G_{i-1} are quantified in the same way. Initial vertices of longest paths in G_{i-1} cannot have incoming edges, which in combination with (1) implies that they must be contained in the set X_i removed during the ith execution of the loop body. As a consequence, a maximum-length path in G_i must be strictly shorter than l_i. Let $x_1 \dots x_{l_i+1}$ be a maximum-length path in G_{i-1}. The sequence $x_2 \dots x_{l_i+1}$ is a maximum-length path in G_i, so $l_{i+1} = l_i - 1$. Suppose there is a path $y_2 \dots y_{l_i+1}$ in G_i such that $q(y_2) \neq q(x_2)$. Then $q(y_2) = q(x_1)$ and there must be a variable $y_1 \in V(G_{i-1})$ such that $(y_1, y_2) \in E(G_{i-1})$ is an incoming edge of y_2, since otherwise $y_2 \in X_i$ and the vertex would not be contained in G_i. But then $y_1 \dots y_{l_i+1}$ is a (longest) path in G_{i-1} and $q(y_1) \neq q(x_1)$, in contradiction with (1). We conclude that the initial vertices of any two longest paths in G_i are associated with the same quantifier. Applying the induction hypothesis, we get $l_{i+1} = r - (i + 1)$, which in combination with $l_{i+1} = l_i - 1$ yields $l_i = r - i$.

Now suppose (2) there are longest paths $x_1 \dots x_{l_i+1}$ and $y_1 \dots y_{l_i+1}$ in G_{i-1} such that $q(x_1) \neq q(y_1)$. Without loss of generality assume that $x_1 \in X_i$, so that X_i contains all variables $x \in V(G_{i-1})$ without incoming edges and $q(x) = q(x_1)$. The sequence $y_1 \dots y_{l_k+1}$ is still a path in G_i because $\{y_1, \dots, y_{l_k+1}\} \cap X_i = \emptyset$. In fact, it is a longest path in G_i and $l_{i+1} = l_i$. If G_i would contain a path $z_1 \dots z_{l_i+1}$ with $q(z_1) \neq q(y_1)$ then this would have been a maximum-length path in G_{i-1} such that $q(z_1) = q(x_1)$ and $z_1 \notin X_i$, a contradiction. Applying the induction hypothesis, we get $l_{i+1} = r - (i + 1)$ and conclude that $l_i + 1 = r - i$. □

The alternation depth of Ψ is $r - 1$ by construction. Let k be the maximum length of a path in $G(\Phi, D)$. Note that l_1 is the length of a longest path in $G_0 = G(\Phi, D)$, so that $l_1 = k$. We distinguish two cases. If there are paths $x_1 \dots x_{k+1}$ and $y_1 \dots y_{k+1}$ in $G(\Phi, D)$ such that $q(x_1) \neq q(x_2)$ then by Lemma 2, the alternation depth of any D-permutation of Φ is at least $k + 1$, and by the above claim, $k + 1 = r - 1$. Otherwise, every path in $G(\Phi, D)$ of length k starts with a variable associated with the same quantifier. By Lemma 2, the alternation depth of any D-permutation of Φ is at least k, and $k = r - 1$ by the above claim. We conclude that Ψ is a D-permutation of Φ with minimum alternation depth. □

Theorem 2 *Let D be a tractable permutation dependency scheme. Given a PCNF formula Φ, a D-permutation of Φ with minimum alternation depth can be computed in polynomial time.*

Proof The algorithm first invokes a polynomial-time algorithm for computing D to obtain D_Φ and then runs Algorithm 1 on (Φ, D_Φ) in linear time (Lemma 3). By Lemma 4, this yields a D-permutation of Φ with minimum alternation depth. □

5 Resolution-Path Dependencies

In this section, we will use Van Gelder's notion of *resolution paths*[1] [16] to define the *resolution-path dependency scheme* and prove that it is a permutation dependency scheme.

Definition 11 (*Resolution Path*) Let $\Phi = \mathcal{Q} \cdot \varphi$ be a PCNF formula and let $X \subseteq var_\exists(\Phi)$. A *resolution path* (from ℓ_1 to ℓ_{2k}) via X (in Φ) is a sequence $\ell_1 \dots \ell_{2k}$ of literals satisfying the following properties:

1. For all $i \in [k]$, there is a $C_i \in \varphi$ such that $\ell_{2i-1}, \ell_{2i} \in C_i$.
2. For all $i \in [k]$, $var(\ell_{2i-1}) \neq var(\ell_{2i})$.
3. For all $i \in [k-1]$, $\{\ell_{2i}, \ell_{2i+1}\} \subseteq X \cup \overline{X}$.
4. For all $i \in [k-1]$, $\overline{\ell_{2i}} = \ell_{2i+1}$.

If $\ell_1 \dots \ell_{2k}$ is a resolution path in Φ via X, we say that ℓ_1 and ℓ_{2k} are *connected in* Φ (with respect to X).

Example 2 Consider the PCNF formula $\Phi = \mathcal{Q} \cdot \varphi$, where

$$\mathcal{Q} = \exists y_1 \exists y_2 \forall x_1 \exists y_3 \forall x_2, \quad \text{and}$$
$$\varphi = \underbrace{(x_1 \vee x_2 \vee y_2 \vee y_1)}_{C_1} \wedge \underbrace{(\neg x_1 \vee \neg y_2 \vee \neg y_1)}_{C_2} \wedge \underbrace{(\neg y_1 \vee \neg y_3)}_{C_3} \wedge \underbrace{(\neg y_1 \vee y_3)}_{C_4}.$$

The sequence $x_1 y_1 \neg y_1 y_3$ is a resolution path in Φ via $\{y_1\}$, so the literals x_1 and $\neg y_3$ are connected with respect to $\{y_1\}$. By contrast, the sequence $\neg x_1 \neg y_1 \neg y_3$ is not a resolution path in Φ because it violates the third condition of Definition 11.

Two resolution paths can be chained together if their first and last literals are different polarities of a single variable.

Lemma 5 *Let Φ be a PCNF formula and let $s = \ell_1 \dots, \ell_{2i} \ell_{2i+1} \dots \ell_{2k}$ be a sequence of literals, and let $s' = \ell_1 \dots \ell_{2i}$ as well as $s'' = \ell_{2i+1} \dots \ell_{2k}$. The following statements are equivalent:*

1. *s is a resolution path in Φ.*
2. *s' and s'' are resolution paths in Φ and $\overline{\ell_{2i}} = \ell_{2i+1}$.*

Proof Immediate from Definition 11. □

Resolution path dependencies are induced by a pair of resolution paths that connect two literals and their negations.

Definition 12 (*Dependency Pair*) Let Φ be a PCNF formula and $x, y \in var(\Phi)$. We say $\{x, y\}$ is a *resolution-path dependency pair* in Φ with respect to $X \subseteq var_\exists(\Phi)$ if at least one of the following conditions holds:

- x and y, as well as $\neg x$ and $\neg y$, are connected in Φ with respect to X.
- x and $\neg y$, as well as $\neg x$ and y, are connected in Φ with respect to X.

[1] Van Gelder [16] offers two definitions of resolution paths (Definitions 4.1 and 5.2). The first one does not lead to a dependency scheme, so we base our considerations on the second definition. We elaborate on this in Example 3.

Definition 13 (*Resolution-Path Dependency Scheme*) The *resolution-path dependency scheme* is the mapping D^{res} that assigns to each PCNF formula $\Phi = \mathcal{Q} \cdot \varphi$ the relation D^{res}_{Φ} defined as $D^{\mathrm{res}}_{\Phi} = \{ (x, y) \in D^{\mathrm{trv}}_{\Phi} : \{x, y\}$ is a resolution-path dependency pair in Φ with respect to $R_{\Phi}(x) \setminus (var_{\forall}(\Phi) \cup \{y\}) \}$.

In the formula Φ of Example 2, $\{y_1, x_1\}$ is resolution-path dependency pair with respect to $\emptyset$, and $\{x_1, y_3\}$ is a resolution-path dependency pair with respect to $\{y_1, y_2\}$. We have $(y_1, x_1) \in D^{\mathrm{res}}_{\Phi}$, but $(x_1, y_3) \notin D^{\mathrm{res}}_{\Phi}$, because $\neg x_1$ is not connected in Φ to either of y_3 or $\neg y_3$ with respect to $R_{\Phi}(x_1) \setminus (var_{\forall(\Phi)} \cup \{y_3\}) = \emptyset$.

The following lemma establishes a connection between resolution paths and Q-resolution derivations.

Lemma 6 *Let $\Phi = \mathcal{Q} \cdot \varphi$ be PCNF formula, let ℓ, ℓ' be distinct literals, and let $\mathcal{D} = (T, \lambda)$ be a tree-like Q-resolution derivation of a clause C from Φ such that $\ell, \ell' \in C$. Then ℓ and ℓ' are connected in Φ with respect to $resvar(\pi)$.*

Proof By induction on the height k of $\mathcal{D}$. For $k = 0$, the clause C must already be contained in φ, and $\ell\ell'$ is a resolution path in Φ via $\emptyset$. Assume the lemma holds for derivations of height up to $k - 1$. Let r denote the root of $\mathcal{D}$. We have to consider two cases. (1) If r has a single child t, then $\lambda(r) = C$ is the result of universal reduction of $\lambda(t)$, and $\lambda(t)$ must already contain ℓ and ℓ'. Let $\mathcal{D}' = (T', \lambda)$, where T' is the subtree of T rooted at t. It is readily verified that $\mathcal{D}'$ is a tree-like Q-resolution derivation of $\lambda(t)$ of height $k-1$, so we can apply the induction hypothesis and conclude that ℓ and ℓ' are connected in Φ with respect to $resvar(\mathcal{D}') \subseteq resvar(\mathcal{D})$. (2) Suppose r has two child nodes t' and t''. Then $\lambda(r) = C$ is the resolvent of $\lambda(t')$ and $\lambda(t'')$ on some variable $v = \lambda(rt') = \lambda(rt'')$. Let T' and T'' denote the subtrees of T rooted at t' and t'', respectively, and let $\mathcal{D}' = (T', \lambda)$, $\mathcal{D}'' = (T'', \lambda)$. If $\ell, \ell' \in \lambda(t')$ or $\ell, \ell' \in \lambda(t'')$, we can apply the same reasoning as in case (1). Otherwise, assume without loss of generality that $\ell, v \in \lambda(t')$ and $\neg v, \ell' \in \lambda(t'')$. Since $\mathcal{D}'$ and $\mathcal{D}''$ are tree-like Q-resolution derivations of height at most $k - 1$, we can apply the induction hypothesis to conclude that ℓ and v must be connected in Φ with respect to $resvar(\mathcal{D}')$, and that $\neg v$ and ℓ' must be connected in Φ with respect to $resvar(\mathcal{D}'')$. That means there must be a resolution path $s' = \ell_1 \dots \ell_{2i}$ via $resvar(\mathcal{D}')$ with $\ell_1 = \ell$ and $\ell_{2i} = v$, as well as a resolution path $s'' = \ell'_1 \dots \ell'_{2j}$ via $resvar(\mathcal{D}'')$ with $\ell'_1 = \neg v$ and $\ell'_{2j} = \ell'$. By Lemma 5, the sequence $s = s's'' = \ell_1 \dots \ell_{2i} \ell'_1 \dots \ell'_{2j}$ is a resolution path in Φ. Since $resvar(\mathcal{D}) = resvar(\mathcal{D}') \cup resvar(\mathcal{D}'') \cup \{v\}$ the literals ℓ and ℓ' are connected in Φ with respect to $resvar(\mathcal{D})$. □

The next result is due to Van Gelder [16, Theorem 4.7]. Since his notion of a *resolution path* is ambiguous between two definitions and the theorem only holds for one of them (see Example 3), we include our own proof below.

Theorem 3 *The resolution-path dependency scheme is a transposition dependency scheme.*

Proof Let $\Phi = \mathcal{Q} \cdot \varphi$ and let $\Psi = \mathcal{Q}' \cdot \varphi$ be a D^{res}-transposition of Φ. More specifically, let $\mathcal{Q} = Q_1 x_1 \dots Q_i x_i Q_{i+1} x_{i+1} \dots Q_n x_n$ and let $\mathcal{Q}' = Q_1 x_1 \dots Q_{i+1} x_{i+1} Q_i x_i \dots Q_n x_n$. It is sufficient to show that for every truth assignment $\tau : \{x_1, \dots, x_{i-1}\} \to \{0, 1\}$ the formula $\Phi[\tau]$ has a Q-resolution refutation if and only if $\Psi[\tau]$ has a Q-resolution refutation. If $Q_i = Q_{i+1}$ this clearly holds, so consider the case where $Q_i \neq Q_{i+1}$ and assume without loss of generality that $Q_i = \forall$ and $Q_{i+1} = \exists$. The "only if" direction is trivial since every Q-resolution refutation of $\Phi[\tau]$ is also a Q-resolution refutation of $\Psi[\tau]$.

For the "if" direction, let $\mathcal{D} = (T, \lambda)$ be a prefix-ordered, tree-like Q-resolution refutation of $\Psi[\tau]$. The only derivation step admissible in $\mathcal{D}$ that cannot occur in a refutation of $\Phi[\tau]$ is universal reduction on x_i of a clause that contains x_{i+1} or $\neg x_{i+1}$. If $\mathcal{D}$ contains no such step, $\mathcal{D}$ is already a refutation of $\Phi[\tau]$ and we are done. Otherwise, because $\mathcal{D}$ is prefix-ordered, the final derivation step has to be resolution on x_{i+1}. We will show that this refutation can be rewritten into a refutation of $\Phi[\tau]$. The idea is that we can swap resolution on x_{i+1} with universal reduction on x_i because the only situation in which this would lead to a tautological clause is one where $(x_i, x_{i+1}) \in D^{\mathrm{res}}_{\Phi}$.

Let t_1, t_2 be the children of T's root with $\lambda(t_1) = \{x_{i+1}\}$ and $\lambda(t_2) = \{\neg x_{i+1}\}$. If a universal reduction step on x_i occurs on a branch, it must be the penultimate derivation step. Suppose t_1 has a child t_1' such that $x_i \in var(\lambda(t_1'))$ and assume without loss of generality that $x_i \in \lambda(t_1')$, that is, $\lambda(t_1') = \{x_i, x_{i+1}\}$. We now distinguish three cases. (1) Suppose t_2 has a child t_2' such that $\neg x_i \in \lambda(t_2')$. Let T_1 and T_2 denote the subtrees of T rooted at t_1 and t_2. Then $\mathcal{D}_1 = (T_1, \lambda)$ and $\mathcal{D}_2 = (T_2, \lambda)$ are tree-like Q-resolution derivations and by Lemma 6, x_i and x_{i+1}, as well as $\neg x_i$ and $\neg x_{i+1}$, are connected in $\Psi[\tau]$ with respect to $resvar(T_1)$ and $resvar(T_2)$, respectively. That is, $\{x_i, x_{i+1}\}$ is a resolution-path dependency pair in $\Psi[\tau]$ with respect to $resvar(T_1) \cup resvar(T_2)$. Because every resolution path in $\Psi[\tau]$ is a resolution path in Φ, the pair $\{x_i, x_{i+1}\}$ is a resolution-path dependency pair in Φ with respect to $resvar(T_1) \cup resvar(T_2)$. Since $\mathcal{D}$ is prefix-ordered we have $resvar(T_1) \cup resvar(T_2) \subseteq R_{\Phi}(x_i) \backslash \{x_{i+1}\}$. Moreover, the set $resvar(T_1) \cup resvar(T_2)$ only contains existential variables, so we conclude that $(x_i, x_{i+1}) \in D^{\mathrm{res}}_{\Phi}$, in contradiction with our assumption that Ψ is a D^{res}-transposition of Φ. (2) Suppose t_2 has a child t_2' such that $x_i \in \lambda(t_2')$, that is, $\lambda(t_2') = \{x_i, \neg x_{i+1}\}$. We construct a prefix-ordered, tree-like Q-resolution refutation of $\Phi[\tau]$ by resolving the clauses $\lambda(t_1')$ and $\lambda(t_2')$ and then adding a universal reduction step on x_i. (3) Otherwise, t_2 does not have a child t_2' with $x_i \in var(\lambda(t_2'))$. Then a prefix-ordered, tree-like Q-resolution refutation of $\Phi[\tau]$ is obtained by resolving $\lambda(t_1')$ with $\lambda(t_2)$ and adding a universal reduction step on x_i. □

Example 3 Consider the PCNF formula $\Phi = \mathcal{Q} \cdot \varphi$, where

$$\mathcal{Q} = \forall u \exists e \exists v \forall x \exists y \exists z, \quad \text{and}$$
$$\varphi = \underbrace{(u \vee y)}_{C_1} \wedge \underbrace{(\neg y \vee \neg x \vee v)}_{C_2} \wedge \underbrace{(\neg v \vee x \vee z)}_{C_3} \wedge \underbrace{(\neg z \vee e)}_{C_4} \wedge \underbrace{(\neg u \vee \neg e)}_{C_5}.$$

The sequence $s = uy\neg yv\neg vz\neg ze$ is a resolution path in Φ via $R_{\Phi}(u) \backslash (var_{\forall}(\Phi) \cup \{e\}) = \{v, y, z\}$. In fact, it is the only resolution path via $\{v, y, z\}$ connecting u and e. The literals $\neg u$ and $\neg e$ are trivially connected, so $\{u, e\}$ is a resolution path dependency pair with respect to $\{v, y, z\}$ and $(u, e) \in D^{\mathrm{res}}_{\Phi}$. Indeed, changing the order of u and e in the prefix of Φ results in a formula Ψ that is unsatisfiable, while Φ is satisfiable. By Definition 11, if $\ell_1 \ldots \ell_{2k}$ is a resolution path there has to be a sequence of clauses $C_1 \ldots C_k$ such that $\ell_{2i-1}, \ell_{2i} \in C_i$ for each $i \in [k]$. Let us call such a sequence a *witnessing clause sequence*. For s we have $c = C_1C_2C_3C_4$ as a unique witnessing clause sequence. Suppose we were to restrict Definition 11 and require a resolution path to have a witnessing clause sequence such that every pair of consecutive clauses has a non-tautological resolvent (as in Definition 4.1 of [16]). The resulting proto-dependency scheme D would not be a transposition dependency scheme: the clauses C_2 and C_3 do not have a non-tautological resolvent, so s would not be a resolution path and $(u, e) \notin D_{\Phi}$; thus Ψ would be a D-transposition of Φ but $\Phi \not\equiv \Psi$.

Lemma 7 *The resolution path dependency scheme is monotone.*

Proof The result follows from the observation that every resolution path in a PCNF formula Φ via X is a resolution path in Φ via Y whenever $X \subseteq Y \subseteq var(\Phi)$. □

Theorem 4 *The resolution path dependency scheme is a permutation dependency scheme.*

Proof Immediate from Proposition 2 in combination with Theorem 3 and Lemma 7. □

We can define a pointwise partial order of proto-dependency scheme as follows. For two proto-dependendency scheme D^1 and D^1 we let $D^1 \leq D^2$ if $D^1_\Phi \subseteq D^2_\Phi$ for every PCNF formula Φ. The resulting Hasse diagram for known dependency schemes is shown in Fig. 1. If D^1 is a permutation dependency scheme and $D^1 \leq D^2$ then D^2 is a permutation dependency scheme as well, since every D^2-permutation of a PCNF formula is also a D^1-permutation. Accordingly, Theorem 4 implies that every dependency scheme appearing in Fig. 1 is a permutation dependency scheme.

Samer and Szeider [12] generalized the notion of a strong backdoor set from CNF formulas to PCNF formulas, by adding the requirement that the backdoor set be closed under a cumulative dependency scheme. They showed that evaluating PCNF formulas is FPT when parameterized by the size of a smallest strong backdoor set (with respect to the classes QHORN or Q2CNF) provided that the considered cumulative dependency scheme is tractable. By Theorems 4 and 5, one can use the resolution path dependency scheme here and thus get an FPT result that is stronger than the results achieved by using any of the other dependency schemes appearing in Fig. 1.

6 Computing Resolution-Path Dependencies

In this section we prove that the resolution path dependency scheme is tractable. To this end, we establish a correspondence between resolution paths and walks in the *implication graph* (or *associated graph* [8, p. 75]) of a formula.

Definition 14 (*Implication Graph*) Let $\Phi = \mathcal{Q} \cdot \varphi$ be a PCNF formula. The *implication graph* of Φ is the directed graph with vertex set $var(\Phi) \cup \overline{var(\Phi)}$ and edge set $\{ (\overline{\ell}, \ell') :$ there is a $C \in \varphi$ such that $\ell, \ell' \in C$ and $\ell \neq \ell' \}$.

Lemma 8 *Let Φ be a PCNF formula and let $\ell, \ell' \in var(\Phi) \cup \overline{var(\Phi)}$ be distinct literals. Let $X \subseteq var(\Phi)$ and let G denote the implication graph of Φ. The following statements are equivalent.*

1. *There is a resolution path from ℓ to ℓ' via X.*
2. *There is a walk from $\overline{\ell}$ to ℓ' in $G[X \cup \overline{X} \cup \{\overline{\ell}, \ell'\}]$.*

Proof Let $s = \ell_1 \dots \ell_{2k}$ be a resolution path from ℓ to ℓ' via X. Let $\langle \ell_{2i} \rangle_{i=1}^{k}$ denote the sequence obtained from s by taking every other element. We claim that the sequence $w = \overline{\ell_1} \langle \ell_{2i} \rangle_{i=1}^{k}$ is a walk in G. By Definition 11, for each $i \in [k]$ there has to be a clause C_i such that $\ell_{2i-1}, \ell_{2i} \in C_i$. In particular, $\ell_1, \ell_2 \in C_1$ so $(\overline{\ell_1}, \ell_2)$ is an edge of G. Moreover, $\overline{\ell_{2i}} = \ell_{2i+1}$ for $i \in [k-1]$. That is, $\overline{\ell_{2i}}, \ell_{2(i+1)} \in C_i$ so $(\ell_{2i}, \ell_{2(i+1)})$ is an edge of G for each $i \in [k-1]$. This proves the claim. Since $var(\ell_{2i}) \in X$ for $i \in [k-1]$ the walk w is even a walk in $G[X \cup \overline{X} \cup \{\overline{\ell}, \ell'\}]$. For the converse, let $p = \ell_1 \dots \ell_k$ be a shortest path from ℓ_1 to ℓ_k in $G[X \cup \overline{X} \cup \{\overline{\ell_1}, \ell_k\}]$. We claim the sequence $s = \overline{\ell_1} \ell_2 \overline{\ell_2} \dots \ell_{k-1} \overline{\ell_{k-1}} \ell_k$ is a resolution path in via X. We prove that the conditions of Definition 11 are satisfied. Because (ℓ_i, ℓ_{i+1}) is an edge of G there must be a clause $C_i \in \varphi$ such that $\overline{\ell_i}, \ell_{i+1} \in C_i$ for each $i \in [k-1]$.

This establishes Condition 1. Since p is of minimum length the ℓ_i must be pairwise distinct. This implies Conditions 2 and 3. Finally, Condition 4 is satisfied by construction. □

The implication graph of a formula can be constructed in time quadratic in the size of Φ and directed reachability can be decided in linear time. In combination with Lemma 8, these observations yield tractabiliy of the resolution-path dependency scheme.

Theorem 5 *The resolution-path dependency scheme is tractable: given a PCNF formula Φ, the dependency relation D^{res}_{Φ} can be computed in polynomial time.*

If the size of input clauses is bounded by a constant the implication graph can even be constructed in linear time, since every clause contributes a constant number of edges. We can exploit this by first converting an input formula into an equisatisfiable formula with bounded clause size using the following well-known trick [8, p. 40].

Lemma 9 *Let $\Phi = \mathcal{Q} \cdot \varphi$ be a PCNF formula and let $C \in \varphi$ be a clause of Φ such that $C = C_1 \cup C_2$ and $C_1 \cap C_2 = \emptyset$. Let $\Psi = \mathcal{Q}' \cdot \varphi'$ be a PCNF formula with $\mathcal{Q}' = \mathcal{Q}\exists x$ and $\varphi' = (\varphi \backslash \{C\}) \cup \{C_1 \cup \{x\}, C_2 \cup \{\neg x\}\}$ for a fresh variable $x \notin var(\Phi)$. Two literals ℓ, ℓ' are connected in Φ with respect to $X \subseteq var(\Phi)$ if and only if ℓ, ℓ' are connected in Ψ with respect to $X \cup \{x\}$.*

Proof Let G and H denote the implication graphs of Φ and Ψ, respectively, and let $G' = G[X \cup \overline{X} \cup \{\overline{\ell}, \ell'\}]$ and $H' = H[X \cup \overline{X} \cup \{\overline{\ell}, \ell', x, \neg x\}]$. We claim that there is a walk from $\overline{\ell}$ to ℓ' in G' if and only if there is a walk from $\overline{\ell}$ to ℓ' in H'. The lemma then follows from Lemma 8. Let $w = \ell_1 \dots \ell_k$ be a walk in G' from $\overline{\ell}$ to ℓ'. Suppose (ℓ_i, ℓ_{i+1}) is an edge of G' but not of H' for any $i \in [k-1]$. Then $\overline{\ell_i} \in C_1$ and $\ell_{i+1} \in C_2$, or $\overline{\ell_i} \in C_2$ and $\ell_{i+1} \in C_1$. Assume without loss of generality that the first case holds. Then (ℓ_i, x) and (x, ℓ_{i+1}) are edges of H'. By inserting x between every such pair $\ell_i \ell_{i+1}$ of subsequent literals in w we obtain a walk from $\overline{\ell}$ to ℓ' in H'. For the converse, consider a walk $w = \ell_1 \dots \ell_k$ in H' from $\overline{\ell}$ to ℓ'. Note that $x \notin var(\{\ell, \ell'\})$ because $x \notin var(\Phi)$. If $var(\ell_i) = x$ then (ℓ_{i-1}, ℓ_{i+1}) is an edge of G', so we obtain a walk from $\overline{\ell}$ to ℓ' simply by omitting occurrences of x and $\neg x$ from w. □

Lemma 10 *Given a PCNF formula Φ and a pair (x, y) of variables, one can decide whether $(x, y) \in D^{res}_{\Phi}$ in linear time.*

Proof If $(x, y) \notin D^{\text{trv}}_{\Phi}$ the algorithm immediately rejects. Otherwise, it uses Lemma 9 to split clauses of Φ and construct a PCNF formula Ψ such that each clause of Ψ contains at most three literals and such that $(x, y) \in D^{\text{res}}_{\Phi}$ if and only $(x, y) \in D^{\text{res}}_{\Psi}$. This takes time linear in the size of Φ. The algorithm then determines whether $\{x, y\}$ is a dependency pair with respect to $R_{\Psi}(x) \backslash (var_{\forall}(\Psi) \cup \{y\})$. By Lemma 8, this comes down to deciding at most four instances of directed reachability in induced subdigraphs of Ψ's implication graph. Since these subdigraphs can be constructed in linear time, the overall runtime is linear in the size of Φ. □

It follows that, given a PCNF formula, the resolution-path dependency relation can be computed in time cubic in the size of the formula.

Proof (*of Theorem 5*) Given a PCNF formula Φ, the algorithm computes D^{trv}_{Φ} and tests whether $(x, y) \in D^{\text{res}}_{\Phi}$ for each pair $(x, y) \in D^{\text{trv}}_{\Phi}$. Since $|D^{\text{trv}}_{\Phi}| \leq |var(\Phi)|^2$ and each test can be performed in linear time by Lemma 10, this takes $O(|\Phi|^3)$ time in total. □

Formulas encountered in applications frequently have encoding sizes that render the above algorithm ineffective. The main advantage of the (weaker) *standard dependency scheme* over the resolution-path dependency scheme is that it can be computed in linear time [3,9]. The characterization of resolution paths as directed paths in the implication graph not only substantially simplifies the proof of Theorem 5 compared to our original proof [13], but also points to relaxations of the resolution-path dependency scheme based on strongly connected components of the implication graph that can be computed in linear time.

7 Conclusion

We studied dependency schemes as a means of reordering the quantifier prefix of a PCNF formula, and proved that known dependency schemes allow for a very general kind of reordering. As an application, we presented an algorithm for finding reorderings with the fewest quantifier alternations. This algorithm could be used to minimize the alternation depth of a PCNF formula before passing it to a QSAT solver, but whether this leads to performance gains in practice remains to be seen. Tractable permutation dependency schemes like the resolution-path dependency scheme may also prove useful in tandem with preprocessing rules that are sensitive to the order of variables in the prefix, such as *universal reduction* or *quantified blocked clause elimination* [4]: given a PCNF formula where these rules cannot be applied, reordering may lead to an instance that can be further simplified. By introducing transposition dependency schemes and the notion of monotonicity, we further hope to facilitate future proofs to the effect that a certain proto-dependency scheme is indeed a (permutation) dependency scheme.

Acknowledgments This research was supported by the European Research Council (ERC), Project Complex Reason 239962.

Appendix: Linear-Time Implementation of Algorithm 1

Proof (*of Lemma 3*) Let D be a proto-dependency scheme and Φ be PCNF formula. Let $n = |var(\Phi)|$ and $m = |D_\Phi|$. Consider an execution of Algorithm 1 on input (Φ, D_Φ). We are going to assume that $G(\Phi, D)$ is represented by separate adjacency lists for outgoing and incoming edges (such a representation can be computed from Φ and D_Φ in linear time). By processing the vertices of $G(\Phi, D)$ in reverse-topological order, we can compute the number of incoming edges $in(x)$, as well as the length $l(x)$ of a maximum-length path in $G(\Phi, D)$ starting at x, for each $x \in var(\Phi)$. This can be done in time $O(n+m)$. We can then compute the sets $X_\forall = \{ x \in var_\forall(\Phi) : in(x) = 0 \}$ and $X_\exists = \{ x \in var_\exists(\Phi) : in(x) = 0 \}$ of existentially and universally quantified variables without incoming edges in time $O(n)$.

For each vertex x, we also store a flag $active(x) \in \{0, 1\}$ to encode the characteristic function of $V(G)$ for the remaining (induced) subdigraph G of $G(\Phi, D)$. To check whether $V(G) \neq \emptyset$, we maintain a counter c that we initially set to n. Initially, $active(x)$ is set to 1 for each $x \in var(\Phi)$. Finally, we sort the variables in Φ with respect to $l(x)$, in decreasing order. Because $0 \leq l(x) \leq n-1$ for each $x \in var(\Phi)$, we can do this in linear time by creating a bin B_l for each l with $0 \leq l \leq n-1$, putting each $x \in var(\Phi)$ in bin $B_{l(x)}$, and then emptying the bins in descending order. Let $x_1 \ldots x_n$ be the corresponding sequence of variables. We maintain a pointer p to an element in this sequence that we set to 1 initially. The following properties hold: (1) x_p is the initial vertex of a longest path in G, (2) $X_\forall$ and

$X_\exists$ are the sets of universally and existentially quantified variables without incoming edges in G, (3) $c = |V(G)|$, and (4) *active* encodes the characteristic function of $V(G)$. Moreover, (5) if $x \in V(G)$ then $l(x)$ is the length of a longest path in G starting from x, and (6) $in(x)$ is the number of incoming edges for x.

To maintain these properties, we implement the while loop in the following way. Let $Q = q(x_p)$. We set $X := X_Q$ as well as $X_Q := \emptyset$. We then go through X in an arbitrary order $y_1, \ldots, y_{|X|}$ and do the following for each y_i. We set $active(y_i) := 0$, $P' := P'Qy_i$, and decrement the counter c by one. For each $y \in N_G^+(y_i)$ we set $in(y) := in(y) - 1$. Whenever $in(y)$ reaches 0 we add y to $X_{q(y)}$. Finally, if $c > 0$, we increase p until $active(x_p) = 1$.

This implementation of the loop body maintains (3), (4), and (6). Let G and G' denote the remaining subdigraphs before and after an execution of the loop, respectively. Similarly, let α and α' denote the values of variable α before and after. Let $\overline{\exists} = \forall$ and $\overline{\forall} = \exists$. The set $X'_{\overline{q(x_p)}}$ contains the vertices in $X_{\overline{q(x_p)}}$ plus any vertex in G all whose incoming edges are contained in X, which is just the set of vertices of G' without incoming edges. There cannot be a vertex x in G' such that x has no incoming edges and $q(x) = q(x_p)$ (since the ones in G are contained in X and removal of X does not create new ones), so $X_{q(x_p)}$ is indeed the empty set and (2) is maintained. Moreover, if $x \in V(G')$, then $l(x)$ is still the length of a longest path in G' starting at x: since G' contains no additional edges, the length of a longest path starting from x cannot be bigger than $l(x)$. Let $xy_1 \ldots y_{l(x)}$ be a path in G. By assumption, $x \in V(G') \subseteq V(G)$, so each y_i with $i \in [l(x)]$ has an incoming edge in G and therefore $y_i \notin X$. This means $xy_1 \ldots y_{l(x)}$ is a path in G', so $l(x)$ is indeed the length of a longest path in G' starting from x, and (5) is preserved. Because $l(x_i) \geq l(x_j)$ for $1 \leq i < j \leq n$, the vertex $x_{p'}$ is the initial vertex of a longest path in G' (provided that $V(G) \neq \emptyset$), so (1) holds as well.

If G is acyclic then G'—which is a subdigraph of G—is also acyclic. Accordingly, since $G(\Phi, D)$ is acyclic by Lemma 1, at any time during the execution the remaining subdigraph of $G(\Phi, D)$ is acyclic. Thus the set X computed in the loop body and then removed from the digraph is always nonempty, so the algorithm must terminate. More specifically, every vertex $v \in G(\Phi, D)$ is put into either $X_\forall$ or $X_\exists$ exactly once, and each outgoing edge of v is considered once in updating *in*. The pointer p assumes each value in $\{0, \ldots, n-1\}$ at most once. We conclude that the overall runtime is in $O(n + m)$. □

References

1. Atserias, A., Oliva, S.: Bounded-width QBF is PSPACE-complete. In: Portier, N., Wilke, T. (eds.) 30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27–March 2, Kiel, Germany, Volume 20 of LIPIcs, pp. 44–54. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2013)
2. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without bdds. In: Cleaveland, R. (eds.) 5th International Conference in Tools and Algorithms for Construction and Analysis of Systems (TACAS '99), Held as Part of the European Joint Conferences on the Theory and Practice of Software (ETAPS '99), Amsterdam, The Netherlands, March 22–28, Proceedings, Volume 1579 of Lecture Notes in Computer Science, pp. 193–207. Springer, Berlin (1999)
3. Biere, A., Lonsing, F.: Integrating dependency schemes in search-based QBF solvers. In: Strichman, O., Szeider, S. (eds.) Theory and Applications of Satisfiability Testing—(SAT 2010), Volume 6175 of Lecture Notes in Computer Science, pp. 158–171. Springer, Berlin (2010)
4. Biere, A., Lonsing, F., Seidl, M.: Blocked clause elimination for QBF. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) International Conference on Automated Deduction—(CADE 23), Volume 6803 of Lecture Notes in Computer Science, pp. 101–115. Springer, Berlin (2011)

5. Bjesse, P., Leonard, T., Mokkedem, A.: Finding bugs in an alpha microprocessor using satisfiability solvers. In: Berry, G., Comon, H., Finkel, A. (eds.) Computer Aided Verification: Proceedings of the 13th International Conference (CAV 2001), Paris, France, July 18–22, pp. 454–464 (2001)
6. Kleine Büning, H., Karpinski, M., Flögel, A.: Resolution for quantified Boolean formulas. Inf. Comput. **117**(1), 12–18 (1995)
7. Kautz, H., Selman, B.: Pushing the envelope: planning, propositional logic, and stochastic search. In: Proceedings of the Thirteenth AAAI Conference on Artificial Intelligence (AAAI '96), pp. 1194–1201. AAAI Press, Palo Alto (1996)
8. Kleine Büning, H., Lettman, T.: Propositional Logic: Deduction and Algorithms. Cambridge University Press, Cambridge (1999)
9. Lonsing, F.: Dependency Schemes and Search-Based QBF Solving: Theory and Practice. Ph.D. thesis, Johannes Kepler University, Linz, Austria (2012)
10. Prasad, A.G.M., Biere, A.: A survey of recent advances in SAT-based formal verification. Softw. Tools Technol. Transf. **7**(2), 156–173 (2005)
11. Pan, G., Vardi, M.Y.: Fixed-parameter hierarchies inside PSPACE. In: Proceedings of the 21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12–15 August, Seattle, WA, USA, pp. 27–36. IEEE Computer Society, Los Alamitos (2006)
12. Samer, M., Szeider, S.: Backdoor sets of quantified Boolean formulas. J. Autom. Reason. **42**(1), 77–97 (2009)
13. Slivovsky, F., Szeider, S.: Computing resolution-path dependencies in linear time. In: Cimatti, A., Sebastiani, R. (eds.) Theory and Applications of Satisfiability Testing—SAT 2012, Volume 7317 of Lecture Notes in Computer Science, pp. 58–71. Springer, Berlin (2012)
14. Slivovsky, F., Szeider, S.: Variable dependencies and Q-resolution. In: Sinz, C., Egly, U. (eds.) Theory and Applications of Satisfiability Testing—SAT 2014, Volume 8561 of Lecture Notes in Computer Science, pp. 269–284. Springer, Berlin (2014)
15. Stockmeyer, L.J.: The polynomial-time hierarchy. Theor. Comput. Sci. **3**(1), 1–22 (1976)
16. Van Gelder, A.: Variable independence and resolution paths for quantified Boolean formulas. In: Lee, J. (eds.) Principles and Practice of Constraint Programming—CP 2011, Volume 6876 of Lecture Notes in Computer Science, pp. 789–803. Springer, Berlin (2011)