Backdoor Sets of Quantified Boolean Formulas

Marko Samer · Stefan Szeider

Abstract We generalize the notion of backdoor sets from propositional formulas to quantified Boolean formulas (QBF). This allows us to obtain hierarchies of tractable classes of quantified Boolean formulas with the classes of quantified Horn and quantified 2CNF formulas, respectively, at their first level, thus gradually generalizing these two important tractable classes. In contrast to known tractable classes based on bounded treewidth, the number of quantifier alternations of our classes is unbounded. As a side product of our considerations we develop a theory of variable dependency which is of independent interest.

Keywords Quantified Boolean formulas · backdoor sets · variable dependencies · parameterized complexity

1 Introduction

Many important computational tasks like planning, verification, and several questions of knowledge representation and automated reasoning can be naturally encoded as the evaluation problem of *quantified Boolean formulas* [10, 24, 27, 29], a generalization of the propositional satisfiability problem (SAT). In recent years quantified Boolean formulas have become a very active research area. The evaluation of quantified Boolean formulas constitutes a PSPACE-complete problem and is therefore believed to be computationally harder than the NP-complete propositional satisfiability problem [17, 26, 33]. In the sequel we make the common assumption that for a given formula all variables are quantified (i.e., there are no free variables) and that the formula is in prenex normal form with the propositional part (the matrix) in conjunctive normal form; we will refer to such formulas as *QCNF formulas*.

Marko Samer
Department of Computer Science
TU Darmstadt, Germany
E-mail: samer@cs.tu-darmstadt.de

Stefan Szeider
Department of Computer Science
University of Durham, UK
E-mail: stefan.szeider@durham.ac.uk

Only a few tractable classes of quantified Boolean formulas are known where the number of *quantifier alternations* is unbounded. For example, the time needed to solve QCNF formulas of bounded treewidth grows non-elementarily in the number of quantifier alternations, as recently shown by Pan and Vardi [25]. Two prominent tractable classes with *unbounded* quantifier alternations are QHORN (clauses contain at most one positive literal) and Q2CNF (clauses contain at most two literals). QHORN formulas and Q2CNF formulas can be evaluated in polynomial time due to classic results of Kleine Büning, Karpinski, and Flögel [16] and of Aspvall, Plass, and Tarjan [2], respectively.

In this paper we define hierarchies of tractable classes of QCNF formulas of the form $\mathcal{C}_0 \subseteq \mathcal{C}_1 \subseteq \mathcal{C}_2 \subseteq \cdots$ where the first class $\mathcal{C}_0$ is either QHORN or Q2CNF, and every QCNF formula belongs to some $\mathcal{C}_k$ for k large enough. We develop algorithms which render membership in $\mathcal{C}_k$ as well as evaluation of formulas in $\mathcal{C}_k$ feasible in polynomial time where the order of the polynomial is the same for all values of k. In other words, our algorithms are *fixed-parameter algorithms* (we will briefly review some basic concepts of fixed-parameter algorithms in Section 2.2). Such time complexity admits an efficient processing of large instances as long as the parameter k is kept reasonably small.

Backdoor Sets

Our approach is based on the generalization of the concept of backdoor sets from propositional satisfiability to quantified Boolean formulas. Backdoor sets for SAT (and similarly for constraint satisfaction) were introduced by Williams, Gomes, and Selman as a tool for analyzing the performance of SAT algorithms [38,39]. Backdoor sets have recently received a lot of attention in satisfiability research [13–15,19,22,23,28,35]. The idea is to consider a *base class* $\mathcal{C}$ of CNF formulas for which membership and satisfiability are both decidable in polynomial time. A set B of variables of an arbitrary CNF formula F is a *strong $\mathcal{C}$-backdoor set* if all formulas that can be obtained from F by instantiating the variables in B and simplifying the resulting formula F' belong to the base class $\mathcal{C}$ (in the sequel we will also discuss the notion of a *weak $\mathcal{C}$-backdoor set* which is, however, less relevant for our considerations). If a strong backdoor set B is known, we can efficiently decide the satisfiability of F by checking the satisfiability of $2^{|B|}$ CNF formulas that belong to the tractable class $\mathcal{C}$. The parameterized complexity of finding strong backdoor sets of CNF formulas has been studied for various base classes including Horn and 2CNF formulas [22], formulas that can be decided by polynomial-time DLL subsolvers [34], variable-disjoint unions of hitting formulas (clustering formulas) [23], and formulas of small maximum deficiency [36].

In this paper we generalize the concepts of weak and strong backdoor sets to the more general set-up of QCNF formulas. In the following we discuss some basic principles of our approach, using the class (Q)HORN of (quantified) Horn formulas as base class. Consider the CNF formula

$$F = (\neg x \vee y \vee \neg w) \wedge (x \vee \neg y \vee w) \wedge (\neg y \vee z) \wedge (y \vee \neg z).$$

The set $B = \{x\}$ is a strong HORN-backdoor set of F since for $x = 0$ we obtain the clauses $(\neg y \vee w)$, $(\neg y \vee z)$, and $(y \vee \neg z)$ and for $x = 1$ we obtain the clauses $(y \vee \neg w)$, $(\neg y \vee z)$, and $(y \vee \neg z)$ which are all Horn. Now let us quantify the variables so that we obtain the QCNF formula

$$\mathcal{F} = \forall y \, \forall z \, \exists x \, \exists w \, F.$$

Obviously, the variable x cannot be isolated anymore in a backdoor set as above since the truth value of x apparently depends on the truth values of y and z. In other words, we cannot reduce the evaluation of $\mathcal{F}$ to the evaluation of some simpler QHORN formula obtained by fixing the truth value of x while y and z remain universally quantified. Hence, for QCNF formulas we require that strong backdoor sets are closed with respect to the dependency of variables: if x belongs to a backdoor set B, also all variables on which x depends belong to B. In Section 4 we present fixed-parameter algorithms for finding strong backdoor sets with respect to the base classes QHORN and Q2CNF. The algorithms take into account variable dependencies that are provided as additional input (below we will address the problem of computing the dependencies). Our algorithms make use of known fixed-parameter algorithms for vertex cover and hitting set [7, 21]. Once a strong backdoor set is found, the formula can be evaluated by considering all truth assignments to the variables in the backdoor set. Thus, if we take $\mathcal{C}_k$ as the class of QCNF formulas that have strong QHORN-backdoor sets (strong Q2CNF-backdoor sets) of size at most k, then we have indeed an infinite hierarchy of tractable classes of QCNF formulas with the base class QHORN (Q2CNF) at its first level. Each QCNF formula belongs to some $\mathcal{C}_k$ for k large enough, and every class $\mathcal{C}_k$ contains formulas with arbitrarily many quantifier alternations.

Thus, we have fixed-parameter tractability results for a problem that is PSPACE-hard in the non-parameterized sense. Here, the gain due to parameterization is even more drastic than it is for most of the known fixed-parameter tractability results where the non-parameterized problems are "only" NP-complete.

Variable Dependencies

So far we have left open the exact meaning of "x depends on y." Clearly it would be safe to assume that a variable x depends on all variables that are quantified on the left of x. Thus, in the above example, $\{x, y, z\}$ certainly constitutes a strong QHORN-backdoor set of $\mathcal{F}$. However, a closer look at the formula reveals that we can do better. Although z is quantified left of x we can actually swap the quantification of x and z, revealing that x does not depend on z. Namely, the matrix F can be split into two parts $F_1 = (\neg x \vee y \vee \neg w) \wedge (x \vee \neg y \vee w)$ and $F_2 = (\neg y \vee z) \wedge (y \vee \neg z)$ such that x and w occur only in F_1 and z occurs only in F_2. Thus, we can rewrite $\mathcal{F}$ equivalently as

$$\forall y \forall z \exists x \exists w \, (F_1 \wedge F_2) \;\Leftrightarrow\; \forall y \left((\exists x \exists w \, F_1) \wedge (\forall z \, F_2) \right) \;\Leftrightarrow\; \forall y \exists x \exists w \forall z \, (F_1 \wedge F_2),$$

thus shifting x to the left and so showing that x does not depend on z. Consequently, we can actually form the smaller backdoor set $\{x, y\}$. With a more sophisticated reasoning that we will describe in Section 3, we can shift x to the left of z even if x occurs in F_2, as long as it occurs only positively or only negatively. For the general case we need to take into account whether variables are connected to each other in a certain way.

Along these lines we develop a *scheme of variable dependency* that allows us to limit the blow-up of strong backdoor sets caused by variable dependencies. Variable dependencies have been studied in a slightly different context by Ayari and Basin [3], Biere [5], and Bubeck and Kleine Büning [6]; of related interest is the work of Egly, Tompits, and Woltran on quantifier shiftings [11] and Benedetti's work on quantifier trees [4]. For a variable dependency scheme one needs to compromise between tractability and generality: we show in Section 3 that identifying minimal variable dependencies is PSPACE-hard. We propose two tractable dependency schemes that are

reasonably general: the *standard dependency scheme* which is based on ideas of Ayari and Basin [3], Biere [5], and Bubeck and Kleine Büning [6], and the *triangle dependency scheme* which generalizes the standard dependency scheme without increasing the asymptotic worst-case runtime.

We formulate our schemes strictly in terms of QCNF formulas, allowing a direct implementation within the data structures used by QCNF-based solvers. The application of our dependency schemes is not limited to backdoor set optimization; we think that it is also useful for other aspects of the evaluation of quantified Boolean formulas. In the works of Ayari and Basin [3], Biere [5], and Bubeck and Kleine Büning [6], variable dependencies are used to identify clauses that have to be duplicated when eliminating universal variables by expansion. In particular, those clauses that contain variables that depend on the expanded variable (more precisely, variables for which it is unknown whether they are independent) have to be duplicated. Since it is desirable to keep the number of such duplications small in order to avoid memory overflow, it is important to identify variable dependencies as accurately as possible. For example, consider the formula

$$\forall z \, \forall x \, \exists y_1 \, \ldots \, \exists y_n \, (x \vee \neg y_1) \wedge (y_1 \vee \neg y_2) \wedge \cdots \wedge (y_{n-1} \vee \neg y_n) \wedge (y_n \vee z).$$

Following the approaches of [3,5,6], as formalized in the standard dependency scheme, one has to duplicate the whole matrix when expanding x, since the standard dependency scheme is not able to identify any of the variables $y_1, \ldots, y_n$ as independent from x. The triangle dependency scheme, on the other hand, identifies all variables $y_1, \ldots, y_n$ as independent from x and thus x can be shifted to the rightmost position in the quantifier prefix. Since universal variables with no existential variables in their scope can be eliminated, x can actually be eliminated without expansion.

Note, however, that this superiority of the triangle dependency scheme does not necessarily hold if we want to expand universal variables that have other universal variables in their scope. The reason for this are indirect dependencies that have to be taken into account when shifting variables within the quantifier prefix but that are irrelevant for expansion. A refined notion of the triangle dependency scheme that overcomes this problem as well as a generalization have been recently developed [30].

2 Background

2.1 Quantified Boolean Formulas

We consider propositional formulas in conjunctive normal form (CNF). We identify each CNF formula with the set of its clauses, e.g., the formula $(\neg x \vee y \vee z) \wedge (\neg y \vee \neg z) \wedge (x \vee \neg y)$ is identified with the set $\{\{\neg x, y, z\}, \{\neg y, \neg z\}, \{x, \neg y\}\}$. Moreover, we consider quantified Boolean formulas in quantified CNF (QCNF), for example,

$$\mathcal{F} = \forall x \, \exists y \, \forall z \, F = \forall x \, \exists y \, \forall z \, (\neg x \vee y \vee z) \wedge (\neg y \vee \neg z) \wedge (x \vee \neg y).$$

We refer to F as the *matrix* of $\mathcal{F}$. We assume that all variables occurring in the matrix are bounded by some quantifier, i.e., there are no free variables in $\mathcal{F}$, and that all variables bounded by some quantifier occur in the matrix. Each clause in F is a finite set of *literals*, and a literal is a negated or unnegated propositional *variable*. For a literal ℓ we denote by $\overline{\ell}$ the literal of opposite polarity, i.e., $\overline{x} = \neg x$ and $\overline{\neg x} = x$;

moreover, for a set X of literals, we put $\overline{X} = \{\,\overline{\ell} : \ell \in X\,\}$. For a clause C we assume that if $\ell \in C$ then $\overline{\ell} \notin C$ and we denote by $var(C)$ the set of variables that occur (negated or unnegated) in C. For a QCNF formula $\mathcal{F}$ and its matrix F we put $var(\mathcal{F}) = var(F) = \bigcup_{C \in F} var(C)$.

For a CNF formula F and a variable $x \in var(F)$, we put $F - x = \{\, C \setminus \{x, \overline{x}\} : C \in F\,\}$; moreover, for a set $X \subseteq var(F)$, we put $F - X = \{\, C \setminus (X \cup \overline{X}) : C \in F\,\}$. For a QCNF formula $\mathcal{F} = \mathsf{Q}_1 x_1 \ldots \mathsf{Q}_n x_n\, F$ and a variable $x_p \in var(\mathcal{F})$, we denote by $\mathcal{F} - x_p$ the QCNF formula obtained from $\mathcal{F}$ by replacing the matrix F by $F - x_p$ and removing the superfluous quantification $\mathsf{Q}_p x_p$; moreover, we generalize this notation in a straight-forward way to $\mathcal{F} - X$ for sets $X \subseteq var(\mathcal{F})$. We define the *depth* of x_p in $\mathcal{F}$ as $\delta_{\mathcal{F}}(x_p) = p$ and we put $q_{\mathcal{F}}(x_p) = \mathsf{Q}_p$. Moreover, we define $var_\forall(\mathcal{F}) = \{x \in var(\mathcal{F}) : q_{\mathcal{F}}(x) = \forall\}$ and $var_\exists(\mathcal{F}) = \{x \in var(\mathcal{F}) : q_{\mathcal{F}}(x) = \exists\}$. A QCNF formula $\mathcal{F}'$ is obtained from $\mathcal{F}$ by *quantifier reordering*, if there is a permutation $i_1, \ldots, i_n$ of $1, \ldots, n$ such that $\mathcal{F}' = \mathsf{Q}_{i_1} x_{i_1} \ldots \mathsf{Q}_{i_n} x_{i_n}\, F$.

A *truth assignment* is a mapping $\tau : X \to \{0,1\}$ defined on some set X of variables. We extend τ to literals by setting $\tau(\neg x) = 1 - \tau(x)$ for $x \in X$. For a truth assignment $\tau : \{x\} \to \{0,1\}$ we simply write "$x = 0$" and "$x = 1$" respectively. For a truth assignment τ and a CNF formula F, we denote by $F[\tau]$ the CNF formula obtained from F by removing all clauses which contain a literal ℓ with $\tau(\ell) = 1$ and by removing literals ℓ with $\tau(\ell) = 0$ from the remaining clauses; moreover, for a QCNF formula $\mathcal{F} = \mathsf{Q}_1 x_1 \ldots \mathsf{Q}_n x_n\, F$, we denote by $\mathcal{F}[\tau]$ the QCNF formula obtained from $\mathcal{F}$ by replacing the matrix F by $F[\tau]$ and removing all superfluous quantifications. A truth assignment τ *satisfies* a CNF formula F if $F[\tau] = \emptyset$.

The evaluation function $\nu : \mathcal{F} \mapsto \{0,1\}$ on QCNF formulas $\mathcal{F}$ is recursively defined by $\nu(\exists x\, \mathcal{F}) = \max(\nu(\mathcal{F}[x = 0]), \nu(\mathcal{F}[x = 1]))$, $\nu(\forall x\, \mathcal{F}) = \min(\nu(\mathcal{F}[x = 0]), \nu(\mathcal{F}[x = 1]))$, and, if $\mathcal{F}$ has no variables, $\nu(\mathcal{F}) = 1$ if $\mathcal{F} = \emptyset$ and $\nu(\mathcal{F}) = 0$ otherwise. A QCNF formula $\mathcal{F}$ is *true* (or *satisfiable*) if $\nu(\mathcal{F}) = 1$; otherwise it is *false* (or *unsatisfiable*). Two QCNF formulas $\mathcal{F}$ and $\mathcal{F}'$ are *equivalent* if $\nu(\mathcal{F}) = \nu(\mathcal{F}')$.

A clause is called *Horn* if it contains at most one positive literal and it is called *binary* if it contains at most two literals. A CNF/QCNF formula is called Horn or binary if all its clauses are Horn or binary, respectively. The classes of Horn and binary CNF formulas are denoted by HORN and 2CNF, respectively; the classes of Horn and binary QCNF formulas are denoted by QHORN and Q2CNF, respectively.

2.2 Parameterized Complexity

An instance of a parameterized problem is a pair (I, k) where I is the *main part* and k is the *parameter*; the latter is usually a non-negative integer. A parameterized problem is *fixed-parameter tractable* if it can be solved by a fixed-parameter algorithm, i.e., if instances (I, k) can be solved in time $\mathcal{O}(f(k)\, n^c)$, where f is a computable function of k, c is a constant, and n is the size of I. FPT denotes the class of all fixed-parameter tractable decision problems [9, 12, 20].

Parameterized complexity offers a *completeness theory*, similar to the theory of NP-completeness, that allows the accumulation of strong theoretical evidence that a parameterized problem is *not* fixed-parameter tractable. This completeness theory is based on the *weft hierarchy* of complexity classes W[t], $t \geq 1$. Each class contains all parameterized decision problems that can be reduced to a certain parameterized satisfiability problem under parameterized reductions. For example, for $t = 2$, the

corresponding satisfiability problem asks whether a given CNF formula has a satisfying truth assignment that sets exactly k variables to 1. Parameterized reductions are straightforward extensions of polynomial-time many-to-one reductions that ensure a parameter of one problem maps into a parameter of another problem [9,12,20]. If we know that a parameterized problem is W[t]-hard (under parameterized reductions) for some $t \geq 1$, then it is very unlikely that the problem is fixed-parameter tractable. Fixed-parameter tractability of the problem would imply that the Exponential Time Hypothesis fails [12] (i.e., the existence of a $2^{o(n)}$ algorithm for n-variable 3SAT). For example, Vertex Cover is fixed-parameter tractable, while Clique is W[1]-complete and Set Cover is W[2]-complete with the size of the solution as parameter [9].

3 Dependency Schemes

As already mentioned in the introduction, we consider *dependency schemes* in order to obtain smaller backdoor sets. Since we will define dependency schemes as binary relations, we will need the following notations: For a binary relation $\mathcal{R}$ over some set V we write $\overline{\mathcal{R}}$ to denote the relation *inverse* to $\mathcal{R}$, i.e., $\overline{\mathcal{R}} = \{(y,x) : (x,y) \in \mathcal{R}\}$, and we write $\mathcal{R}^*$ to denote the reflexive and transitive *closure* of $\mathcal{R}$, i.e., the smallest set $\mathcal{R}^*$ such that $\mathcal{R}^* = \mathcal{R} \cup \{(x,x) : x \in V\} \cup \{(x,y) : \exists z \text{ such that } (x,z) \in \mathcal{R}^* \text{ and } (z,y) \in \mathcal{R}\}$. Moreover, we put $\mathcal{R}(x) = \{y : (x,y) \in \mathcal{R}\}$ for $x \in V$ and $\mathcal{R}(X) = \bigcup_{x \in X} \mathcal{R}(x)$ for $X \subseteq V$.

For example, let $\mathcal{R} = \{(w,w),(x,z),(y,x),(z,w)\}$. Then $\overline{\mathcal{R}} = \{(w,w),(w,z),(x,y),(z,x)\}$ and $\mathcal{R}^* = \{(w,w),(x,w),(x,x),(x,z),(y,w),(y,x),(y,y),(y,z),(z,w),(z,z)\}$. Moreover, $\mathcal{R}^*(x) = \{w,x,z\}$.

We will also need the following binary relations $R_{\mathcal{F}}$ and $R^{\diamond}_{\mathcal{F}}$ over $var(\mathcal{F})$:

- $R_{\mathcal{F}} = \{\, (x,y) : x,y \in var(\mathcal{F}),\ \delta_{\mathcal{F}}(x) < \delta_{\mathcal{F}}(y) \,\}$
- $R^{\diamond}_{\mathcal{F}} = \{\, (x,y) : x,y \in var(\mathcal{F}),\ \exists z \in R_{\mathcal{F}}(x),\ q_{\mathcal{F}}(z) \neq q_{\mathcal{F}}(x),\ \delta_{\mathcal{F}}(z) \leq \delta_{\mathcal{F}}(y) \,\}$

In other words, $R_{\mathcal{F}}$ assigns to each variable x the variables on the right of x in the quantifier prefix and $R^{\diamond}_{\mathcal{F}}$ assigns to each variable x the variables on the right of x starting at the first variable (from left to right) with different quantification. We will also use the shorthands $L_{\mathcal{F}} = \overline{R}_{\mathcal{F}}$ and $L^{\diamond}_{\mathcal{F}} = \overline{R^{\diamond}_{\mathcal{F}}}$.

Definition 1 (Shifting) Let $\mathcal{F}$ be a QCNF formula and $X \subseteq var(\mathcal{F})$. We say the QCNF formula $\mathcal{F}'$ is obtained from $\mathcal{F}$ by *down-shifting* (*up-shifting*) X, in symbols $\mathcal{F}' = S^{\downarrow}(\mathcal{F}, X)$ ($\mathcal{F}' = S^{\uparrow}(\mathcal{F}, X)$), if $\mathcal{F}'$ is obtained from $\mathcal{F}$ by quantifier reordering such that the following holds:

1. $X = R_{\mathcal{F}'}(x)$ ($X = L_{\mathcal{F}'}(x)$) for some $x \in var(\mathcal{F}) = var(\mathcal{F}')$ and
2. $\delta_{\mathcal{F}'}(x) < \delta_{\mathcal{F}'}(y)$ if and only if $\delta_{\mathcal{F}}(x) < \delta_{\mathcal{F}}(y)$ for all $x,y \in X$ and
3. $\delta_{\mathcal{F}'}(x) < \delta_{\mathcal{F}'}(y)$ if and only if $\delta_{\mathcal{F}}(x) < \delta_{\mathcal{F}}(y)$ for all $x,y \in var(\mathcal{F}) \setminus X$.

For example, recall the QCNF formula $\mathcal{F} = \forall y \forall z \exists x \exists w\, F$ from the introduction and let $X = \{x,y\}$. Then we have $S^{\downarrow}(\mathcal{F}, X) = \forall z \exists w \forall y \exists x\, F$ and $S^{\uparrow}(\mathcal{F}, X) = \forall y \exists x \forall z \exists w\, F$. Note that neither down-shifting nor up-shifting preserves equivalence, in general, and that the result of shifting is always unique.

Definition 2 (Dependency scheme) A *dependency scheme* D assigns to each QCNF formula $\mathcal{F}$ a binary relation $D_{\mathcal{F}} \subseteq R_{\mathcal{F}}$ such that $\mathcal{F}$ and $S^{\downarrow}(\mathcal{F}, D^*_{\mathcal{F}}(x))$ are equivalent. A dependency scheme D is *tractable* if $D_{\mathcal{F}}$ can always be computed in time that is polynomial in $\mathcal{F}$.

Intuitively, a dependency scheme assigns to each variable x the set of variables that depend on x. Actually, the assigned sets contain those variables for which we cannot prove independence. The following property of dependency schemes will be crucial for our backdoor set algorithms in Section 4.

Definition 3 (Cumulative) A dependency scheme D is *cumulative* if for every QCNF formula $\mathcal{F}$ and set $X \subseteq var(\mathcal{F})$, $\mathcal{F}$ and $S^{\downarrow}(\mathcal{F}, D^*_{\mathcal{F}}(X))$ are equivalent.

In general, a dependency scheme does not need to be cumulative. For example, consider the formula

$$\mathcal{F} = \forall x \forall y \exists u \exists v \, (x \vee u \vee \neg v) \wedge (\neg x \vee \neg u \vee v) \wedge (y \vee u \vee v) \wedge (\neg y \vee \neg u \vee \neg v),$$

and let D be a dependency scheme such that $D_{\mathcal{F}}(x) = D_{\mathcal{F}}(y) = \{v\}$ and $D_{\mathcal{F}'}(z) = R_{\mathcal{F}'}(z)$ for all other combinations of QCNF formulas $\mathcal{F}'$ and $z \in var(\mathcal{F}')$. It is easy to verify that $\mathcal{F}$, $S^{\downarrow}(\mathcal{F}, D^*_{\mathcal{F}}(x))$, and $S^{\downarrow}(\mathcal{F}, D^*_{\mathcal{F}}(y))$ are all true, i.e., D is indeed a dependency scheme. However, D is not cumulative as $S^{\downarrow}(\mathcal{F}, D^*_{\mathcal{F}}(\{x, y\}))$ is false.

Since all dependency schemes that we will define in this paper are cumulative, we are allowed to use them for up-shifting as shown next.

Proposition 1 *Let D be a cumulative dependency scheme, $\mathcal{F}$ be a QCNF formula, and $X \subseteq var(\mathcal{F})$. Then $\mathcal{F}$ and $S^{\uparrow}(\mathcal{F}, \overline{D}^*_{\mathcal{F}}(X))$ are equivalent.*

Proof Let $\mathcal{F}$ be a QCNF formula and $X \subseteq var(\mathcal{F})$. Consider the set $Y = var(\mathcal{F}) \setminus \overline{D}^*_{\mathcal{F}}(X)$. It holds that $Y = D^*_{\mathcal{F}}(Y)$; otherwise, there exists $y \in Y$ and $z \in D^*_{\mathcal{F}}(y) \cap \overline{D}^*_{\mathcal{F}}(X)$, which implies $y \in \overline{D}^*_{\mathcal{F}}(X)$. Consequently, since D is cumulative, $\mathcal{F}$ and $S^{\downarrow}(\mathcal{F}, Y)$ are equivalent. Since $Y \cap \overline{D}^*_{\mathcal{F}}(X) = \emptyset$ and $Y \cup \overline{D}^*_{\mathcal{F}}(X) = var(\mathcal{F})$, it is easy to verify that $S^{\downarrow}(\mathcal{F}, Y)$ and $S^{\uparrow}(\mathcal{F}, \overline{D}^*_{\mathcal{F}}(X))$ are syntactically identical. Hence, $\mathcal{F}$ and $S^{\uparrow}(\mathcal{F}, \overline{D}^*_{\mathcal{F}}(X))$ are equivalent. □

Our aim in the following is to find tractable dependency schemes D such that the sets $D_{\mathcal{F}}$ are as small as possible. We say that dependency scheme D is *more general* than dependency scheme D' if always $D_{\mathcal{F}} \subseteq D'_{\mathcal{F}}$ and the inclusion is strict in some cases.

A very simple example of a tractable dependency scheme is R as defined above, since always $\mathcal{F} = S^{\downarrow}(\mathcal{F}, R^*_{\mathcal{F}}(x))$. A slightly improved but still very simple tractable dependency scheme is the following.

Definition 4 (Trivial dependency scheme) The *trivial dependency scheme* D^{trv} assigns to each QCNF formula $\mathcal{F}$ the binary relation $D^{\mathrm{trv}}_{\mathcal{F}} = R^{\diamond}_{\mathcal{F}}$.

It is easy to see that D^{trv} is indeed a dependency scheme since the only difference between $\mathcal{F}$ and $S^{\downarrow}(\mathcal{F}, {D^{\mathrm{trv}}}^*(x))$ is that the position of x has changed within the same quantifier block, which trivially preserves equivalence.

The following proposition shows that when we want a dependency scheme to be tractable, we cannot expect it to be a most general one.

Proposition 2 *Let $\mathcal{F}$ be a QCNF formula and $x, y \in var(\mathcal{F})$. The problem of deciding whether there exists a dependency scheme D such that $(x, y) \notin D_{\mathcal{F}}$ is* PSPACE*-complete.*

Proof The problem belongs to PSPACE as polynomial space suffices to go through all QCNF formulas $\mathcal{F}'$ obtained from $\mathcal{F}$ by quantifier reordering such that $y \notin R_{\mathcal{F}'}(x)$ and to check whether $\mathcal{F}$ and $\mathcal{F}'$ are equivalent. For showing PSPACE-hardness, we reduce the problem of deciding whether a given QCNF formula is true, which is known to be PSPACE-complete [33]. Let $\mathcal{G} = \mathsf{Q}_1 v_1 \ldots \mathsf{Q}_n v_n\, G$ be an arbitrary QCNF formula and $x, y \notin var(\mathcal{G})$ be two new variables. Moreover, let $F = G \wedge (x \vee y) \wedge (\neg x \vee \neg y)$ and $\mathcal{F} = \mathsf{Q}_1 v_1 \ldots \mathsf{Q}_n v_n \forall x \exists y\, F$. We define a mapping D in the following way: $D_{\mathcal{F}}(x) = \emptyset$ and $D_{\mathcal{F}'}(z) = R_{\mathcal{F}'}(z)$ for all other combinations of QCNF formulas $\mathcal{F}'$ and $z \in var(\mathcal{F}')$. Note that D is a dependency scheme if and only if there exists a dependency scheme D' with $y \notin D'_{\mathcal{F}}(x)$. This follows immediately from the definition of a dependency scheme. Now we have $S^{\downarrow}(\mathcal{F}, D^*_{\mathcal{F}}(x)) = \mathsf{Q}_1 v_1 \ldots \mathsf{Q}_n v_n \exists y \forall x\, F$ and $S^{\downarrow}(\mathcal{F}', D^*_{\mathcal{F}'}(z)) = \mathcal{F}'$ for all other combinations of QCNF formulas $\mathcal{F}'$ and $z \in var(\mathcal{F}')$. Since $\forall x \exists y\, (x \vee y) \wedge (\neg x \vee \neg y)$ is true, we know that $\mathcal{F}$ is true if and only if $\mathcal{G}$ is true. Moreover, since $\exists y \forall x\, (x \vee y) \wedge (\neg x \vee \neg y)$ is false, we know that $S^{\downarrow}(\mathcal{F}, D^*_{\mathcal{F}}(x))$ is false. Hence, it follows immediately that $\mathcal{F}$ and $S^{\downarrow}(\mathcal{F}, D^*_{\mathcal{F}}(x))$ are equivalent (i.e., our mapping D is a dependency scheme) if and only if $\mathcal{G}$ is false. □

We are now going to define our *standard dependency scheme* D^{std}, which is tractable and more general than the trivial dependency scheme D^{trv}. The standard dependency scheme is based on an approach of variable dependency used by Biere [5] for expanding universally quantified variables. Biere defined two variables $x, y \in var(\mathcal{F})$ ($y \in R_{\mathcal{F}}(x)$, $q_{\mathcal{F}}(x) = \forall$, $q_{\mathcal{F}}(y) = \exists$) to be *locally connected* if they occur in the same clause; the relation *connected* is the transitive closure of *locally connected* when considering only variables in $R^{\Diamond}_{\mathcal{F}}(x)$. Then y depends on x if x and y are connected. This definition was motivated by the parse tree of the formula where the quantifiers are shifted down as far as possible (corresponding quantifier shifting rules were investigated by Egly et al. [11].) From this point of view, y depends on x if x occurs on the path from y to the root of the parse tree.

We also include an observation of Bubeck and Kleine Büning [6] into our standard dependency scheme. Since universally quantified variables do not propagate the change of truth values, the authors suggested to ignore all universally quantified variables when building the transitive closure of "locally connected" in Biere's definition of "connected". Evidently this generalizes Biere's approach.

Our definition in the following combines these ideas in a more general setting which admits further generalizations. Let us first define some basic notions.

Definition 5 (Connected) Let $\mathcal{F}$ be a QCNF formula with matrix F. An *X-path*, $X \subseteq var(\mathcal{F})$, between two clauses $C, C' \in F$ is a sequence $C_1, \ldots, C_n$ of clauses in F with $C = C_1$ and $C' = C_n$ such that $var(C_i) \cap var(C_{i+1}) \cap X \neq \emptyset$ for all $1 \leq i < n$. Two clauses $C, C' \in F$ are *connected* with respect to $X \subseteq var(\mathcal{F})$ if there is an X-path between them.

Definition 6 (Dependency pair) Let $\mathcal{F}$ be a QCNF formula with matrix F and let $x, y \in var(\mathcal{F})$ such that $q_{\mathcal{F}}(x) \neq q_{\mathcal{F}}(y)$. An *$(x, y)$-dependency pair* with respect to $X \subseteq var(\mathcal{F})$ is a tuple $(C_1, C_2) \in F \times F$ of clauses such that (i) C_1 and C_2 are connected with respect to X and (ii) $x \in var(C_1)$ and $y \in var(C_2)$.

To illustrate these definitions, consider the following QCNF formula $\mathcal{F}$:

$$\forall u \exists v \forall w \exists x \forall y \exists z\, (u \vee \neg v \vee x) \wedge (u \vee \neg x) \wedge (v \vee z) \wedge (v \vee \neg z) \wedge (w \vee x \vee y) \wedge (y \vee \neg z)$$

For example, the clauses $u \vee \neg v \vee x$ and $y \vee \neg z$ are connected with respect to $\{v, z\}$ and with respect to $\{x, y\}$. Moreover, there is a (u, z)-dependency pair with respect to $\{v\}$ (by choosing $C_1 = u \vee \neg v \vee x$ and $C_2 = v \vee z$) and a (w, x)-dependency pair with respect to $\emptyset$ (by choosing $C_1 = C_2 = w \vee x \vee y$).

Definition 7 (Standard dependency scheme) The *standard dependency scheme* D^{std} assigns to each QCNF formula $\mathcal{F}$ the relation $D^{\mathrm{std}}_{\mathcal{F}} = \{\, (x, y) \in R_{\mathcal{F}} : \mathcal{F}$ contains an (x, y)-dependency pair with respect to $R_{\mathcal{F}}(x) \setminus var_{\forall}(\mathcal{F}) \,\}$.

For example, recall the QCNF formula $\mathcal{F}$ from above. It holds that $D^{\mathrm{std}}_{\mathcal{F}}(u) = \{v, x, z\}$, $D^{\mathrm{std}}_{\mathcal{F}}(v) = \{w, y\}$, $D^{\mathrm{std}}_{\mathcal{F}}(w) = \{x\}$, $D^{\mathrm{std}}_{\mathcal{F}}(x) = \{y\}$, $D^{\mathrm{std}}_{\mathcal{F}}(y) = \{z\}$, and $D^{\mathrm{std}}_{\mathcal{F}}(z) = \emptyset$.

Theorem 1 *The standard dependency scheme is indeed a dependency scheme and it is even cumulative.*

Proof Let $\mathcal{F}$ be a QCNF formula and $X \subseteq var(\mathcal{F})$. Moreover, let $\mathcal{F}'$ denote $S^{\downarrow}(\mathcal{F}, D^{\mathrm{std}^*}_{\mathcal{F}}(X))$. We have to show that $\mathcal{F}$ and $\mathcal{F}'$ are equivalent. To this aim, note that $\mathcal{F}'$ is obtained from $\mathcal{F}$ by quantifier reordering, i.e., by a permutation of the quantifications in the quantifier prefix. It is well known that every permutation of elements can be achieved by successively swapping adjacent elements such that each pair is swapped at most once [18]. In particular, this means that we can transform $\mathcal{F}$ into $\mathcal{F}'$ by successively swapping adjacent quantifications of variables $v \in D^{\mathrm{std}^*}_{\mathcal{F}}(X)$ with variables $w \in R_{\mathcal{F}}(X) \setminus D^{\mathrm{std}^*}_{\mathcal{F}}(X)$, since the relative ordering of variables within these two sets remains unchanged according to the definition of shifting. Thus, it suffices to show that each such elementary transformation step preserves equivalence.

Let $v \in D^{\mathrm{std}^*}_{\mathcal{F}}(X)$ and $w \in R_{\mathcal{F}}(X) \setminus D^{\mathrm{std}^*}_{\mathcal{F}}(X)$ be two variables with adjacent quantifications in the quantifier prefix that have to be swapped. If $q_{\mathcal{F}}(v) = q_{\mathcal{F}}(w)$, the equivalence follows trivially. Otherwise, if $q_{\mathcal{F}}(v) \neq q_{\mathcal{F}}(w)$, let $\mathcal{G}$ denote the formula before v and w are swapped, i.e., $\mathcal{G} = \cdots \mathsf{Q}_i v \, \mathsf{Q}_j w \cdots G$. Note that $w \notin D^{\mathrm{std}}_{\mathcal{F}}(v)$; otherwise, we obtain $w \in D^{\mathrm{std}^*}_{\mathcal{F}}(X)$, which contradicts our assumption. Thus, we know by Definition 7 that there is no (v, w)-dependency pair with respect to $R_{\mathcal{F}}(v) \setminus var_{\forall}(\mathcal{F}) \supseteq R_{\mathcal{G}}(v) \setminus var_{\forall}(\mathcal{F})$. This implies that the set G of clauses can be partitioned into two subsets G_1 and G_2 such that $v \in var(G_1) \setminus var(G_2)$, $w \in var(G_2) \setminus var(G_1)$, and $var(G_1) \cap var(G_2) \subseteq var_{\forall}(\mathcal{F}) \cup L_{\mathcal{G}}(v)$. Now consider a partial truth assignment to the variables in $L_{\mathcal{G}}(v)$ and let $\mathcal{G}'$ be the resulting formula obtained from $\mathcal{G}$ after such a partial truth assignment has been applied. Moreover, let G'_1 and G'_2 be the corresponding sets of clauses obtained from G_1 and G_2 respectively. Thus, we know that $var(G'_1) \cap var(G'_2) \subseteq var_{\forall}(\mathcal{F})$. Hence, since universal quantifiers are distributive over conjunction, we can shift all remaining quantifiers of $\mathcal{G}'$ in front of G'_1 and G'_2 respectively, which yields the formula $\mathcal{G}'_1 \wedge \mathcal{G}'_2$. Thus, the evaluation of $\mathcal{G}'$ can be reduced to the evaluation of the two independent formulas $\mathcal{G}'_1$ and $\mathcal{G}'_2$, where v occurs only in $\mathcal{G}'_1$ amd w occurs only in $\mathcal{G}'_2$. Consequently, swapping v and w in the original quantifier prefix cannot effect the truth value of the formula. □

Proposition 3 *The standard dependency scheme is tractable. Given a QCNF formula $\mathcal{F}$ of length n and $x \in var(\mathcal{F})$, we can compute $D^{\mathrm{std}}_{\mathcal{F}}(x)$ in time $\mathcal{O}(n)$.*

Proof Consider the incidence graph of the formula, i.e., the graph with the variables and clauses as vertices; a clause C and a variable x are joined by an edge if $x \in var(C)$. For

a variable $x \in var(\mathcal{F})$ we traverse the graph starting at x while ignoring variables not in $R_{\mathcal{F}}(x) \setminus var_{\forall}(\mathcal{F})$. We put a variable $y \in R_{\mathcal{F}}(x)$ with $q_{\mathcal{F}}(y) \neq q_{\mathcal{F}}(x)$ into $D^{\mathrm{std}}_{\mathcal{F}}(x)$ if it can be reached in this way. It is easy to see that the search procedure can be accomplished in time linear in n. □

Remark 1 Note that we only require from a dependency scheme that the set $D^{*}_{\mathcal{F}}(x)$ can be shifted down without affecting the truth value of the formula. If we want to identify dependencies in order to eliminate universal variables x by expansion (recall the introduction), this implies that it suffices to consider only existential variables in $D^{*}_{\mathcal{F}}(x)$, but, in general, it does *not* imply that we only need to consider variables in $D_{\mathcal{F}}(x) \subseteq D^{*}_{\mathcal{F}}(x)$. However, in the case of the standard dependency scheme, it follows immediately from a result of Bubeck and Kleine Büning [6], that, for expansion, it indeed suffices to consider only variables in $D^{\mathrm{std}}_{\mathcal{F}}(x)$. An intuitive reason for this is that additional variables in $D^{\mathrm{std}^{*}}_{\mathcal{F}}(x) \setminus D^{\mathrm{std}}_{\mathcal{F}}(x)$ are connected to x via universal variables; a connection via existential variables only would have caused the variables to be put into $D^{\mathrm{std}}_{\mathcal{F}}(x)$. Thus, since universal variables do not propagate changes of truth values, such indirect dependencies do not need to be taken into account in the case of expansion. We refer the interested reader to [30] for an explicit definition of independence, which also makes the difference between shifting and expansion clearer.

Next we define our *triangle dependency scheme* that improves upon the standard dependency scheme; in fact, we show that the improvement can be arbitrarily large.

Definition 8 (Dependency triple) Let $\mathcal{F}$ be a QCNF formula with matrix F and let $x, y \in var(\mathcal{F})$ such that $q_{\mathcal{F}}(x) \neq q_{\mathcal{F}}(y)$. An (x, y)-*dependency triple* with respect to $X \subseteq var(\mathcal{F})$ is a triple $(C_1, C_2, C_3) \in F \times F \times F$ of clauses such that

1. If $q_{\mathcal{F}}(x) = \forall$ and $q_{\mathcal{F}}(y) = \exists$, then
   - C_1 and C_2 as well as C_1 and C_3 are connected with respect to $X \cup \{x\}$
   - $x \in var(C_1)$, $y \in C_2$, and $\neg y \in C_3$
2. If $q_{\mathcal{F}}(x) = \exists$ and $q_{\mathcal{F}}(y) = \forall$, then
   - C_1 and C_2 as well as C_1 and C_3 are connected with respect to $X \cup \{y\}$
   - $y \in var(C_1)$, $x \in C_2$, and $\neg x \in C_3$

For instance, consider again the following QCNF formula $\mathcal{F}$:

$$\forall u\, \exists v\, \forall w\, \exists x\, \forall y\, \exists z\, (u \vee \neg v \vee x) \wedge (u \vee \neg x) \wedge (v \vee z) \wedge (v \vee \neg z) \wedge (w \vee x \vee y) \wedge (y \vee \neg z)$$

There is a (u, z)-dependency triple with respect to $\{v\}$ (by choosing $C_1 = u \vee \neg v \vee x$, $C_2 = v \vee z$, and $C_3 = v \vee \neg z$) and a (u, x)-dependency triple with respect to $\emptyset$ (by choosing $C_1 = C_2 = u \vee \neg v \vee x$ and $C_3 = u \vee \neg x$).

Definition 9 (Triangle dependency scheme) The *triangle dependency scheme* $D^{\triangle}$ assigns to each QCNF formula $\mathcal{F}$ the relation $D^{\triangle}_{\mathcal{F}} = \{\, (x, y) \in R_{\mathcal{F}} : \mathcal{F}$ contains an (x, y)-dependency triple with respect to $R_{\mathcal{F}}(x) \setminus (var_{\forall}(\mathcal{F}) \cup \{y\})\,\}$.

For example, recall the QCNF formula $\mathcal{F}$ from above. It holds that $D^{\triangle}_{\mathcal{F}}(u) = \{x, z\}$, $D^{\triangle}_{\mathcal{F}}(v) = \{y\}$, and $D^{\triangle}_{\mathcal{F}}(w) = D^{\triangle}_{\mathcal{F}}(x) = D^{\triangle}_{\mathcal{F}}(y) = D^{\triangle}_{\mathcal{F}}(z) = \emptyset$.

Theorem 2 *The triangle dependency scheme is indeed a dependency scheme and it is even cumulative.*

Proof Let $\mathcal{F}$ be a QCNF formula and $X \subseteq var(\mathcal{F})$. Moreover, let $\mathcal{F}'$ denote $S^{\downarrow}(\mathcal{F}, D^{\triangle}_{\mathcal{F}}{}^{*}(X))$. Similar as in the proof of Theorem 1, it suffices to show that swapping adjacent quantifications preserves equivalence. Let $v \in D^{\triangle}_{\mathcal{F}}{}^{*}(X)$ and $w \in R_{\mathcal{F}}(X) \setminus D^{\triangle}_{\mathcal{F}}{}^{*}(X)$ be two variables with adjacent quantifications in the quantifier prefix that have to be swapped. If $q_{\mathcal{F}}(v) = q_{\mathcal{F}}(w)$, the equivalence follows trivially. Otherwise, if $q_{\mathcal{F}}(v) \neq q_{\mathcal{F}}(w)$, let us first assume that $q_{\mathcal{F}}(v) = \forall$ and $q_{\mathcal{F}}(w) = \exists$. Let $\mathcal{G}$ denote the formula before v and w are swapped and let $\mathcal{H}$ denote the formula after v and w have been swapped. In particular, that means $\mathcal{G} = \cdots \forall v \exists w \cdots G$ and $\mathcal{H} = \cdots \exists w \forall v \cdots G$. Thus, $\mathcal{H}$ trivially implies $\mathcal{G}$. For the other direction, note that $w \notin D^{\triangle}_{\mathcal{F}}(v)$; otherwise, we obtain $w \in D^{\triangle}_{\mathcal{F}}{}^{*}(X)$, which contradicts our assumption. Thus, we know by Definition 9 that there is no (v, w)-dependency triple with respect to $R_{\mathcal{F}}(v) \setminus (var_{\forall}(\mathcal{F}) \cup \{w\}) \supseteq R_{\mathcal{G}}(v) \setminus (var_{\forall}(\mathcal{F}) \cup \{w\})$. This implies that the set G of clauses can be partitioned into two subsets G_1 and G_2 such that $v \in var(G_1) \setminus var(G_2)$, $\{w, \neg w\} \nsubseteq \bigcup G_1$, and $var(G_1) \cap var(G_2) \subseteq var_{\forall}(\mathcal{F}) \cup L_{\mathcal{G}}(v) \cup \{w\}$. Now assume for the sake of contradiction that the truth value of w depends on the truth value of v when evaluating $\mathcal{G}$, i.e., there exists a partial truth assignment to the variables in $L_{\mathcal{G}}(v)$ such that the remaining formula evaluates to true if and only if w is assigned different truth values for different truth values of v. Let $\mathcal{G}'$ be the resulting formula obtained from $\mathcal{G}$ after such a partial truth assignment has been applied. Moreover, let G'_1 and G'_2 be the corresponding sets of clauses obtained from G_1 and G_2 respectively. Thus, we know that $var(G'_1) \cap var(G'_2) \subseteq var_{\forall}(\mathcal{F}) \cup \{w\}$. Hence, since universal quantifiers are distributive over conjunction, we can shift almost all remaining quantifiers of $\mathcal{G}'$ in front of G'_1 and G'_2 respectively, which yields the formula $\forall v \exists w (\mathcal{G}'_1 \wedge \mathcal{G}'_2)$. Now, by our assumption, we know that this formula evaluates to true if and only if the truth value of w changes with the truth value of v. Hence, we know that $\mathcal{G}'_2$ must evaluate to true for both truth values assigned to w, i.e, $\forall w \, \mathcal{G}'_2$ must be true. Thus, we can rewrite $\mathcal{G}'$ as $(\forall v \exists w \, \mathcal{G}'_1) \wedge (\forall w \, \mathcal{G}'_2)$. Moreover, since $\{w, \neg w\} \nsubseteq \bigcup G_1 \supseteq \bigcup G'_1$, i.e., since w is pure in G'_1, we know that the clauses in G'_1 must be satisfiable for a fixed truth value assigned to w, i.e., if $w \in \bigcup G'_1$ then w is assigned 1 and if $\neg w \in \bigcup G'_1$ then w is assigned 0. Thus, we know that $\forall v \exists w \, \mathcal{G}'_1$ and $\exists w \forall v \, \mathcal{G}'_1$ are equivalent. So we can rewrite $\mathcal{G}'$ as $(\exists w \forall v \, \mathcal{G}'_1) \wedge (\forall w \, \mathcal{G}'_2)$, which implies $\exists w \forall v \, (\mathcal{G}'_1 \wedge \mathcal{G}'_2)$. Hence, the truth value of w can be chosen independently from the truth value of v, which contradicts our assumption. Consequently, swapping v and w in the quantifier prefix of $\mathcal{G}$ does not affect its truth value. Thus, we know that $\mathcal{G}$ implies $\mathcal{H}$. The case $q_{\mathcal{F}}(v) = \exists$ and $q_{\mathcal{F}}(w) = \forall$ is symmetric. □

Since the search for a dependency triple is not significantly more expensive than the search for a dependency pair, we obtain the same worst-case runtime complexity as for the standard dependency scheme.

Proposition 4 *The triangle dependency scheme is tractable. Given a QCNF formula $\mathcal{F}$ of length n and $x \in var(\mathcal{F})$, we can compute $D^{\triangle}_{\mathcal{F}}(x)$ in time $\mathcal{O}(n)$.*

Proof Let $G = (V, E)$ be the graph whose vertices are the clauses and the literals of $\mathcal{F}$; each literal ℓ is adjacent to its complementary literal $\bar{\ell}$ and to all clauses that contain ℓ. Note that $|V| + |E| = \mathcal{O}(n)$, and so G can be constructed in time $\mathcal{O}(n)$. In order to compute $D^{\triangle}_{\mathcal{F}}(x)$, we distinguish between the following two cases:

(i) If x is existential, observe that $D^{\triangle}_{\mathcal{F}}(x)$ consists of all variables $y \in R_{\mathcal{F}}(x) \cap var_{\forall}(\mathcal{F})$ such that G contains a path between x and y that avoids $\neg x$, and a path between $\neg x$ and y that avoids x; both paths also have to avoid literals with variables

in $X = L_{\mathcal{F}}(x) \cup (var_\forall(\mathcal{F}) \setminus \{y\})$, i.e., we consider paths in $G - (X \cup \overline{X})$. We compute $D^{\triangle}_{\mathcal{F}}(x)$ by traversing G on such paths twice, first starting at x and then starting at $\neg x$. Evidently, $y \in D^{\triangle}_{\mathcal{F}}(x)$ if and only if y can be reached in this way from both x and $\neg x$. Hence, we can compute $D^{\triangle}_{\mathcal{F}}(x)$ in time $\mathcal{O}(n)$.

(ii) If x is universal, observe that $D^{\triangle}_{\mathcal{F}}(x)$ consists of all variables $y \in R_{\mathcal{F}}(x) \setminus var_\forall(\mathcal{F})$ such that G contains a path between x and y that avoids $\neg y$, and a path between x and $\neg y$ that avoids y; both paths also have to avoid literals with variables in $X = L_{\mathcal{F}}(x) \cup (var_\forall(\mathcal{F}) \setminus \{x\})$, i.e., we consider paths in $G - (X \cup \overline{X})$. We compute $D^{\triangle}_{\mathcal{F}}(x)$ as follows: Starting from x we perform depth-first search (dfs) in G; let T be the corresponding search-tree with root x. Whenever we visit a literal ℓ whose complement $\bar{\ell}$ has not yet been visited, we visit $\bar{\ell}$ next (if it exists). This ensures that complementary literals (if reachable from x) are adjacent in T. If $v \in V$ is the i-th vertex visited, then we put $n(v) = i$; we put $n(v) = \infty$ if v never gets visited. Now let u, v be nodes of T. We say that u is the *low point* of v and write $lp(v) = n(u)$ if u is the node with smallest $n(u)$ that can be reached from v by traversing zero or more edges downwards in T followed by at most one edge of G that is not in T. The concept of low points is due to Tarjan [37] who showed that one can compute the numbers $lp(v)$ for all nodes v in time $\mathcal{O}(n)$, and that the following property holds: If $u \neq x$ is a node of T and v is a child of u, then $lp(v) \geq u$ if and only if the removal of u separates v from x in G. It follows that $y \in D^{\triangle}_{\mathcal{F}}(x)$ if and only if either (a) $n(y) = n(\neg y) - 1$ and $lp(\neg y) < n(y)$ or (b) $n(\neg y) = n(y) - 1$ and $lp(y) < n(\neg y)$. Hence, we can compute $D^{\triangle}_{\mathcal{F}}(x)$ in time $\mathcal{O}(n)$. □

Remark 2 Note that in the case of the triangle dependency scheme we need for both shifting and expansion the closure $D^{\triangle}_{\mathcal{F}}{}^*(x)$. In particular, for expansion we have to duplicate all existential variables in $D^{\triangle}_{\mathcal{F}}{}^*(x)$ and the clauses containing these variables. The equivalence of the resulting formula follows then immediately from our proofs. The following example demonstrates that it is not sufficient to duplicate variables in $D^{\triangle}_{\mathcal{F}}(x)$. To this aim, consider the following formula $\mathcal{F}$ [30]:

$$\forall x\, \exists u\, \forall y\, \exists v\, (x \vee y \vee \neg v) \wedge (\neg x \vee \neg y \vee \neg v) \wedge (u \vee y \vee v) \wedge (\neg u \vee y \vee \neg v) \wedge (\neg u \vee \neg y \vee v) \wedge (u \vee \neg y \vee \neg v)$$

It is easy to verify that this formula is true and that $D^{\triangle}_{\mathcal{F}}(x) = \{u\}$ (while $D^{\triangle}_{\mathcal{F}}{}^*(x) = \{x, y, u, v\}$). Thus, expansion of x based on $D^{\triangle}_{\mathcal{F}}(x)$ results in $\mathcal{F}' = \exists u\, \exists u'\, \forall y\, \exists v\, F_{x=0} \wedge F_{x=1}$ with $F_{x=0} = (y \vee \neg v) \wedge (u \vee y \vee v) \wedge (\neg u \vee y \vee \neg v) \wedge (\neg u \vee \neg y \vee v) \wedge (u \vee \neg y \vee \neg v)$ and $F_{x=1} = (\neg y \vee \neg v) \wedge (u' \vee y \vee v) \wedge (\neg u' \vee y \vee \neg v) \wedge (\neg u' \vee \neg y \vee v) \wedge (u' \vee \neg y \vee \neg v)$. However, $\mathcal{F}'$ is false, thus not equivalent to $\mathcal{F}$. Consequently, we actually need time $\mathcal{O}(n^2)$ to compute the required set $D^{\triangle}_{\mathcal{F}}{}^*(x)$. In fact, in time $\mathcal{O}(n^2)$ we can compute the sets $D^{\triangle}_{\mathcal{F}}{}^*(x)$ for all $x \in var(\mathcal{F})$. For the standard dependency scheme, on the other hand, the sets $D^{\mathrm{std}}_{\mathcal{F}}(x)$ are sufficient in the case of expansion (recall Remark 1).

Remark 3 Moreover, note that the proof of Theorem 1 still goes through if we replace the set $R_{\mathcal{F}}(x) \setminus var_\forall(\mathcal{F})$ in Definition 7 by $R^{\diamond}_{\mathcal{F}}(x) \setminus var_\forall(\mathcal{F})$. However, the proof of Theorem 2 does not work if we replace $R_{\mathcal{F}}(x) \setminus (var_\forall(\mathcal{F}) \cup \{y\})$ in Definition 9 by $R^{\diamond}_{\mathcal{F}}(x) \setminus (var_\forall(\mathcal{F}) \cup \{y\})$ as the following example demonstrates: Let D denote the dependency scheme obtained from Definition 9 by such a replacement and consider the formula $\mathcal{F} = \exists u\, \exists v\, \forall x\, (\neg u \vee \neg v) \wedge (u \vee \neg x) \wedge (v \vee x)$. It is easy to verify that $\mathcal{F}$ is false and that $D_{\mathcal{F}}(u) = D_{\mathcal{F}}(v) = \emptyset$ since $R^{\diamond}_{\mathcal{F}}(u) = R^{\diamond}_{\mathcal{F}}(v) = \{x\}$. However, D is not cumulative as $S^{\downarrow}(\mathcal{F}, D^*_{\mathcal{F}}(\{u, v\})) = \forall x\, \exists u\, \exists v\, (\neg u \vee \neg v) \wedge (u \vee \neg x) \wedge (v \vee x)$ is true.

Next we will show that the triangle dependency scheme is indeed more general than the standard dependency scheme. In particular we will also show that the difference of the sizes of the sets assigned to each variable by the dependency schemes can be arbitrarily large.

Proposition 5 *Let $\mathcal{F}$ be a QCNF formula and $x \in var(\mathcal{F})$. The difference in size of the sets $D^{\mathrm{std}}_{\mathcal{F}}(x)$ and $D^{\triangle}_{\mathcal{F}}(x)$ can be arbitrarily large, and the difference in size of the sets $D^{\mathrm{std}^*}_{\mathcal{F}}(x)$ and $D^{\triangle^*}_{\mathcal{F}}(x)$ can be arbitrarily large.*

Proof Let n be an arbitrarily large non-negative integer and let $\mathcal{F} = \forall z \forall x \exists y_1 \exists y_2 \cdots \exists y_n (x \vee \neg y_1) \wedge (y_1 \vee \neg y_2) \wedge \cdots \wedge (y_{n-1} \vee \neg y_n) \wedge (y_n \vee z)$. Then $D^{\mathrm{std}}_{\mathcal{F}}(x) = \{y_1, y_2, \ldots, y_n\}$ and $D^{\triangle}_{\mathcal{F}}(x) = \emptyset$, as well as $D^{\mathrm{std}^*}_{\mathcal{F}}(x) = \{x, y_1, y_2, \ldots, y_n\}$ and $D^{\triangle^*}_{\mathcal{F}}(x) = \{x\}$. Hence, $|D^{\mathrm{std}}_{\mathcal{F}}(x)| - |D^{\triangle}_{\mathcal{F}}(x)| = |D^{\mathrm{std}^*}_{\mathcal{F}}(x)| - |D^{\triangle^*}_{\mathcal{F}}(x)| = n$. □

Proposition 6 *The triangle dependency scheme is more general than the standard dependency scheme.*

Proof We have to show that (i) $D^{\triangle}_{\mathcal{F}} \subseteq D^{\mathrm{std}}_{\mathcal{F}}$ for all QCNF formulas $\mathcal{F}$ and that (ii) the inclusion is strict in some cases. Part (ii) follows immediately from part (i) and Proposition 5. For the proof of part (i), let $y \in D^{\triangle}_{\mathcal{F}}(x)$, i.e., there is an (x, y)-dependency triple with respect to $R_{\mathcal{F}}(x) \setminus (var_\forall(\mathcal{F}) \cup \{y\})$. By definition, this immediately implies that there is an (x, y)-dependency pair with respect to $R_{\mathcal{F}}(x) \setminus var_\forall(\mathcal{F})$, i.e., $y \in D^{\mathrm{std}}_{\mathcal{F}}(x)$. □

Note that the sets assigned to each variable by the triangle dependency scheme may still be larger than necessary. Of course, this is not surprising in consideration of Proposition 2. However, we believe that there is a considerable potential for future research to determine dependency schemes which can be computed in a reasonable amount of time and which are more general than the triangle dependency scheme.

In the following section we will have to shift up several variables when computing backdoor sets of QCNF formulas. For this purpose we will use a cumulative dependency scheme (recall Definition 3) to identify variables that have also to be shifted up in order to preserve equivalence.

4 Backdoor Sets

In the remainder of this paper, we consider an arbitrary but fixed cumulative dependency scheme D; the definitions of partial assignment trees and backdoor sets are subject to the choice of D.

Partial truth assignments are key features for defining backdoor sets of propositional CNF formulas. In the following we introduce the concept of assignment trees which allows us to extend the notions of partial truth assignments and backdoor sets to the quantified setting. We roughly follow a concept of Samulowitz and Bacchus [32].

An *assignment tree* $\mathcal{T} = (T, \lambda)$ is a pair of a rooted binary tree T and a node labeling λ with the following properties. The labeling λ assigns to every node t (except the root) of T a pair $\lambda(t) = (x, \varepsilon)$, where x is a variable and $\varepsilon \in \{0, 1\}$. Every node has at most two children. Nodes at the same depth (i.e., distance from the root) are labeled with the same variable and have the same number of children. A variable does

not appear at different levels. If a node has two children t_1 and t_2, then $\lambda(t_1) = (x, \varepsilon)$ and $\lambda(t_2) = (x, 1 - \varepsilon)$. This completes the definition of an assignment tree.

Let $\mathcal{T} = (T, \lambda)$ be an assignment tree. We denote by $var(\mathcal{T})$ the set of variables occurring in labels of $\mathcal{T}$, and for $x \in var(\mathcal{T})$ we denote by $\delta_{\mathcal{T}}(x)$ the depth of x in T. A variable $x \in var(\mathcal{T})$ is *existential* or *universal* in $\mathcal{T}$ if the nodes of T at depth $\delta_{\mathcal{T}}(x) - 1$ have one or two children, respectively. Every leaf t of T corresponds to a truth assignment $\tau : var(\mathcal{T}) \to \{0, 1\}$ consisting of the assignments made along the path from the root to t. We simply write $\tau \in \mathcal{T}$ if τ is such a truth assignment.

Definition 10 (Partial assignment tree) Let $\mathcal{F}$ be a QCNF formula and $\mathcal{T}$ an assignment tree. Then $\mathcal{T}$ is a *partial assignment tree of* $\mathcal{F}$ if (i) $var(\mathcal{T}) \subseteq var(\mathcal{F})$ and existential (universal) variables of $\mathcal{T}$ are existentially (universally) quantified variables in $\mathcal{F}$, (ii) $\delta_{\mathcal{T}}(x) < \delta_{\mathcal{T}}(y)$ if and only if $\delta_{\mathcal{F}}(x) < \delta_{\mathcal{F}}(y)$ holds for every pair $x, y \in var(\mathcal{T})$, and (iii) $var(\mathcal{T}) = \overline{D}^*_{\mathcal{F}}(var(\mathcal{T}))$.

We define backdoor sets with respect to some base class $\mathcal{C}$ of QCNF formulas. We think of $\mathcal{C}$ as a class that can be recognized in polynomial time and for which satisfiability can be decided in polynomial time.

Definition 11 (Weak backdoor set) Let $\mathcal{F}$ be a QCNF formula. The set $B = \overline{D}^*_{\mathcal{F}}(X)$ for some $X \subseteq var(\mathcal{F})$ is a *weak backdoor set* of $\mathcal{F}$ with respect to $\mathcal{C}$ (or a *weak $\mathcal{C}$-backdoor set*, for short) if there exists a partial assignment tree $\mathcal{T}$ of $\mathcal{F}$ with $var(\mathcal{T}) = B$ such that $\mathcal{F}[\tau]$ is true and belongs to $\mathcal{C}$ for all $\tau \in \mathcal{T}$.

Proposition 7 *Assume that the dependency scheme under consideration is tractable. Let $\mathcal{C} \in \{\text{QHORN}, \text{Q2CNF}\}$ and $k \geq 0$ be a constant. For a given QCNF formula $\mathcal{F}$ we can decide in polynomial time whether $\mathcal{F}$ has a weak $\mathcal{C}$-backdoor set of size at most k. If the answer is affirmative, then $\mathcal{F}$ is true.*

Proof We go through all sets $X \subseteq var(\mathcal{F})$ of size at most k; for $|var(\mathcal{F})| = n$ there are $\mathcal{O}(n^k)$ such sets. For each X we can check in polynomial time whether it gives rise to a weak $\mathcal{C}$-backdoor set $B = \overline{D}^*_{\mathcal{F}}(X)$ of $\mathcal{F}$ for the following reasons: (i) since the dependency scheme is tractable, we can compute B in polynomial time; (ii) the number of partial assignment trees $\mathcal{T}$ with $var(\mathcal{T}) = B$ is a function of k and therefore a constant; (iii) whether $\mathcal{F}[\tau] \in \mathcal{C}$ can obviously be checked in polynomial time; (iv) if $\mathcal{F}[\tau] \in \mathcal{C}$, we can check if $\mathcal{F}[\tau]$ is true in polynomial time by means of the known algorithms [2,16]. Finally note that in general if a QCNF formula has a weak backdoor set, then the formula is true. □

The runtime of the algorithm outlined in the previous proof is polynomial, but the order of the polynomial depends on the size of the backdoor set. Thus, the algorithm is not a fixed-parameter algorithm.

WEAK $\mathcal{C}$-BACKDOOR
Instance: A QCNF formula $\mathcal{F}$ and a non-negative integer k.
Parameter: k.
Question: Does $\mathcal{F}$ have a weak $\mathcal{C}$-backdoor set of size at most k?

The problem WEAK $\mathcal{C}$-BACKDOOR for CNF formulas is just a special case of the corresponding problem for QCNF formulas. Hence, the W[2]-hardness result of Nishimura et al. [22] establishes the following proposition.

Proposition 8 *Let $\mathcal{C} \in \{\text{QHorn}, \text{Q2CNF}\}$. The problem* Weak $\mathcal{C}$-Backdoor *is* W[2]*-hard and thus unlikely to be fixed-parameter tractable.*

For the definition of strong backdoor sets, we do not need partial assignment trees as in the case of weak backdoor sets.

Definition 12 (Strong backdoor set) Let $\mathcal{F}$ be a QCNF formula. The set $B = \overline{D}^*_{\mathcal{F}}(X)$ for some $X \subseteq var(\mathcal{F})$ is a *strong backdoor set* of $\mathcal{F}$ with respect to $\mathcal{C}$ (or a *strong $\mathcal{C}$-backdoor set*, for short) if for all truth assignments $\tau : B \to \{0, 1\}$ it holds that $\mathcal{F}[\tau]$ belongs to $\mathcal{C}$.

Lemma 1 *Let $\mathcal{F}$ be a QCNF formula and let $B = \overline{D}^*_{\mathcal{F}}(X)$ for some $X \subseteq var(\mathcal{F})$ be a strong $\mathcal{C}$-backdoor set of $\mathcal{F}$. Then the following holds:*

1. *For all partial assignments trees $\mathcal{T}$ of $\mathcal{F}$ with $var(\mathcal{T}) = B$ it holds that $\mathcal{F}[\tau]$ belongs to $\mathcal{C}$ for all $\tau \in \mathcal{T}$.*
2. *$\mathcal{F}$ is true if and only if there exists a partial assignment tree $\mathcal{T}$ of $\mathcal{F}$ with $var(\mathcal{T}) = B$ such that $\mathcal{F}[\tau]$ is true for all $\tau \in \mathcal{T}$.*

By taking the size of the backdoor set as the parameter, we obtain the following parameterized decision problem for an arbitrary base class $\mathcal{C}$ of QCNF formulas.

Strong $\mathcal{C}$-Backdoor
Instance: A QCNF formula $\mathcal{F}$ and a non-negative integer k.
Parameter: k.
Question: Does $\mathcal{F}$ have a strong $\mathcal{C}$-backdoor set of size at most k?

For certain important base classes it suffices to consider the following variant of backdoor sets.

Definition 13 (Deletion backdoor set) Let $\mathcal{F}$ be a QCNF formula. The set $B = \overline{D}^*_{\mathcal{F}}(X)$ for some $X \subseteq var(\mathcal{F})$ is a *deletion backdoor set* of $\mathcal{F}$ with respect to $\mathcal{C}$ (or a *deletion $\mathcal{C}$-backdoor set*, for short) if $\mathcal{F} - B \in \mathcal{C}$.

The next result follows analogously to the corresponding result for propositional CNF formulas shown by Crama et al. [8] and Nishimura et al. [22].

Lemma 2 *Let $\mathcal{F}$ be a QCNF formula and $\mathcal{C} \in \{\text{QHorn}, \text{Q2CNF}\}$. Then a set $B \subseteq var(\mathcal{F})$ is a strong $\mathcal{C}$-backdoor set of $\mathcal{F}$ if and only if B is a deletion $\mathcal{C}$-backdoor set of $\mathcal{F}$.*

We state the corresponding parameterized problem:

Deletion $\mathcal{C}$-Backdoor
Instance: A QCNF formula $\mathcal{F}$ and a non-negative integer k.
Parameter: k.
Question: Does $\mathcal{F}$ have a deletion $\mathcal{C}$-backdoor set of size at most k?

Note that the various definitions of backdoor sets in this section coincide with their propositional analogs. The problems of detecting (weak or strong) backdoor sets can also be considered as traditional "non-parameterized" problems by taking the parameter as part of the input. These non-parameterized problems are NP-complete, justifying our parameterized approach. Membership follows immediately from Lemma 2 and hardness follows by trivial reduction from the non-quantified propositional versions, which have been shown by Crama et al. [8] and Nishimura et al. [22] to be NP-complete.

Using a similar construction as for Proposition 5, we can show that the difference between the sizes of the smallest strong backdoor set based on the standard dependency scheme and the smallest strong backdoor set based on the triangle dependency scheme can be arbitrarily large.

Theorem 3 *Let $\mathcal{C} \in \{\text{QHORN}, \text{Q2CNF}\}$ and assume that the dependency scheme D under consideration is cumulative and tractable. The evaluation of QCNF formulas is fixed-parameter tractable with the size of a smallest strong $\mathcal{C}$-backdoor set as parameter.*

The reminder of this section is devoted to proving this result. The following considerations will allow us to apply known vertex cover and hitting set algorithms to the detection of backdoor sets.

Let $S = \{X_1, \ldots, X_m\}$ be a set of finite sets over a universe $V(S) = \bigcup_{i=1}^{m} X_i$ of elements. We will refer to S as a *set system*. A set $H \subseteq V(S)$ is a *hitting set* of S if $H \cap X_i \neq \emptyset$ for all $1 \leq i \leq m$. A hitting set is *minimal* if none of its proper subsets is a hitting set. If $|X_i| = 2$ for all $1 \leq i \leq m$ then the set system represents a graph and a hitting set of S is a *vertex cover* of this graph.

To a QCNF formula $\mathcal{F}$ with matrix F we associate the following two set systems:

- $S_{\text{HORN}}(\mathcal{F}) = \{\, \{u, v\} : u, v \in C \text{ with } u \neq v \text{ for some clause } C \in F \,\}$,
- $S_{\text{2CNF}}(\mathcal{F}) = \{\, X : X \subseteq var(C) \text{ with } |X| = 3 \text{ for some clause } C \in F \,\}$.

The following is a direct consequence of Lemma 2 and Definition 13. Note that in the remainder of this section we write for simplicity $D(x)$ and $D(X)$ to denote $\overline{D}^*_{\mathcal{F}}(x)$ and $\overline{D}^*_{\mathcal{F}}(X)$, respectively.

Lemma 3 *Let $\mathcal{C} \in \{\text{QHORN}, \text{Q2CNF}\}$. For each QCNF formula $\mathcal{F}$, a set $B \subseteq var(\mathcal{F})$ is a strong $\mathcal{C}$-backdoor set of $\mathcal{F}$ if and only if $B = D(B)$ and B is a hitting set of the set system $S_{\mathcal{C}}(\mathcal{F})$.*

By means of the following construction we can encode the condition $B = D(B)$ directly in the hitting set instance: Given a set system S and for each $x \in V(S)$ a subset $D(x) \subseteq V(S)$. We obtain a set system S_D with $V(S_D) = V(S)$ by applying the following saturation rule as often as possible:

Let $X \in S$, $x \in X$, $x' \in D(x) \setminus X$, and $X' = (X \setminus \{x\}) \cup \{x'\}$.
If $X' \notin S$, then add X' to S.

Lemma 4 *Let S and S_D be as above. For each $B \subseteq V(S)$, the following two properties are equivalent:*

1. *B is a hitting set of S such that $D(x) \subseteq B$ for all $x \in B$, and each proper subset B' of B is either not a hitting set of S or contains an element x with $D(x) \not\subseteq B'$.*
2. *B is a minimal hitting set of S_D.*

Proof (a) Assume B satisfies Property 1 of the lemma. Let S' be obtained from S by one application of the saturation rule, i.e., $S' = S \cup \{X'\}$ for $X \in S$, $x \in X$, $x' \in D(x) \setminus X$, and $X' = (X \setminus \{x\}) \cup \{x'\}$. Assume for the sake of contradiction that B is not a hitting set of S'. Consequently, $B \cap X' = \emptyset$; thus $x' \notin B$. However, since $B \cap X \neq \emptyset$, $x \in B$. Since $x' \in D(x)$ we have derived a contradiction. Thus B is indeed a hitting set of S'. It follows by induction on $|S_D \setminus S|$ (i.e., the number of times the saturation rule has been applied) that B is a hitting set of S_D.

(b) Assume B satisfies Property 2 of the lemma. Clearly B is a hitting set of S, since $S \subseteq S_D$. Assume for the sake of contradiction that there are elements $x, x' \in V(S)$ such that $x \in B$, $x' \notin B$, and $x' \in D(x)$. We observe that there is some $X \in S_D$ such that $X \cap B = \{x\}$, since otherwise $B \setminus \{x\}$ were a hitting set of S_D but B is assumed to be minimal. By construction of S_D, we have $X' = (X \setminus \{x\}) \cup \{x'\} \in S_D$. However, $X' \cap B = \emptyset$, thus B is not a hitting set of S_D, a contradiction. Thus, $D(x) \subseteq B$ for all $x \in B$.

(a′) Assume B satisfies Property 1 of the lemma. By (a) there is a set $B_0 \subseteq B$ that is a minimal hitting set of S_D. By (b) it follows that B_0 is a hitting set of S and $D(x) \subseteq B_0$ for all $x \in B_0$. Since B satisfies the minimality condition of Property 1, it follows that $B = B_0$, hence B satisfies Property 2.

(b′) Assume B satisfies Property 2 of the lemma. By (b) there is a set $B_0 \subseteq B$ that satisfies Property 1. By (a) it follows that B_0 is a hitting set of S_D. Since B is a minimal hitting set of S_D, $B = B_0$ follows, hence B satisfies Property 1. □

We are now in the position to complete the proof of Theorem 3. Let $\mathcal{C} \in \{\text{QHorn}, \text{Q2CNF}\}$ and assume that we are given a QCNF formula $\mathcal{F}$ with n variables and an integer $k \geq 0$. We obtain in polynomial time the set system $S = S_{\mathcal{C}}(F)$ (recall the definitions given directly above Lemma 3). As the dependency scheme D under consideration is assumed to be tractable, all the sets $D(x)$ can be computed in polynomial time. Next we obtain the set system S_D. This can be accomplished in polynomial time as well since S_D has at most $\binom{n}{2}$ (for $\mathcal{C} = \text{QHorn}$) or $\binom{n}{3}$ (for $\mathcal{C} = \text{Q2CNF}$) elements. Now we search for a minimal hitting set B of S_D with $|B| \leq k$. For this task we can use known fixed-parameter algorithms for Vertex Cover or 3-Hitting Set, respectively. Currently, the fastest known fixed-parameter algorithm for the former problem is due to Chen, Kanj, and Xia [7] and runs in time $\mathcal{O}(1.2738^k + kn)$. The fastest known fixed-parameter algorithm for the latter problem is due to Niedermeier and Rossmanith [21] and runs in time $\mathcal{O}(2.270^k + n)$; an alternative algorithm is due to Abu-Khzam [1]. If no hitting set of size at most k is found, then we know by Lemmas 3 and 4 that $\mathcal{F}$ has no strong $\mathcal{C}$-backdoor set of size at most k, and we can reject the instance. Otherwise, let B be a minimal $\mathcal{C}$-backdoor set of S_D as produced by the applied fixed-parameter algorithm. Again, by Lemmas 3 and 4, it follows that B is a strong $\mathcal{C}$-backdoor set of $\mathcal{F}$. The second part of Lemma 1 allows us to reduce the satisfiability of $\mathcal{F}$ to the satisfiability of at most $2^{|B|} \leq 2^k$ QCNF formulas $\mathcal{F}[\tau] \in \mathcal{C}$. The satisfiability of each $\mathcal{F}[\tau] \in \mathcal{C}$ can be decided in polynomial time by the known results [2,16]. This completes the proof of Theorem 3.

5 Conclusion

In this paper we introduced the notion of backdoor sets for quantified Boolean formulas, generalizing the notion from propositional formulas. To this aim, we introduced the notion partial assignment trees, a generalization of partial truth assignments of propositional formulas. An essential part in this paper was devoted to the investigation of dependency schemes which indicate the dependency among quantified variables. We proposed a dependency scheme that is both tractable and more powerful than dependency schemes that can be obtained by known methods. We presented fixed-parameter algorithms for detecting strong backdoor sets with respect to quantified Horn and quantified 2CNF formulas. As a consequence, we obtained infinite hierarchies of classes of

QCNF formulas that can be recognized and evaluated in polynomial time, with quantified Horn and quantified 2CNF formulas, respectively, at their first level.

Acknowledgements This research was supported by the EPSRC, project EP/E001394/1, and was carried out during the first author's postdoc position at the University of Durham. Material from the preliminary version [31] is reused with kind permission of Springer Science and Business Media. We thank the anonymous referees for their helpful comments.

References

1. F. N. Abu-Khzam. Kernelization algorithms for d-Hitting Set problems. In *Proc. 10th Int. Workshop on Algorithms and Data Structures (WADS'07)*, volume 4619 of *LNCS*, pages 434–445. Springer-Verlag, 2007.
2. B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.
3. A. Ayari and D. Basin. Qubos: Deciding quantified Boolean logic using propositional satisfiability solvers. In *Proc. 4th Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD'02)*, volume 2517 of *LNCS*, pages 187–201. Springer-Verlag, 2002.
4. M. Benedetti. Quantifier trees for QBFs. In *Proc. 8th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'05)*, volume 3569 of *LNCS*, pages 378–385. Springer-Verlag, 2005.
5. A. Biere. Resolve and expand. In *Proc. 7th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'04)*, volume 3542 of *LNCS*, pages 59–70. Springer-Verlag, 2005.
6. U. Bubeck and H. Kleine Büning. Bounded universal expansion for preprocessing QBF. In *Proc. 10th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *LNCS*, pages 244–257. Springer-Verlag, 2007.
7. J. Chen, I. A. Kanj, and G. Xia. Improved parameterized upper bounds for vertex cover. In *Proc. 31st Int. Symp. on Mathematical Foundations of Computer Science (MFCS'06)*, volume 4162 of *LNCS*, pages 238–249. Springer-Verlag, 2006.
8. Y. Crama and O. Ekin and P. L. Hammer. Variable and term removal from Boolean formulae. *Discrete Applied Mathematics*, 75(3):217–230, 1997.
9. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.
10. U. Egly, T. Eiter, H. Tompits, and S. Woltran. Solving advanced reasoning tasks using quantified Boolean formulas. In *Proc. 17th AAAI Conf. on Artificial Intelligence (AAAI'00)*, pages 417–422. AAAI Press, 2000.
11. U. Egly, H. Tompits, and S. Woltran. On quantifier shifting for quantified Boolean formulas. In *Proc. SAT'02 Workshop on Theory and Applications of Quantified Boolean Formulas*, pages 48–61. Informal Proceedings, 2002.
12. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.
13. J. Hoffmann, C. Gomes, and B. Selman. Structure and problem hardness: Goal asymmetry and DPLL proofs in SAT-based planning. In *Proc. 16th Int. Conf. on Automated Planning and Scheduling (ICAPS'06)*, pages 284–293. AAAI Press, 2006.
14. Y. Interian. Backdoor sets for random 3-SAT. In *Proc. 6th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 231–238. Informal Proceedings, 2003.
15. P. Kilby, J. K. Slaney, S. Thiébaux, and T. Walsh. Backbones and backdoors in satisfiability. In *Proc. 20th AAAI Conf. on Artificial Intelligence (AAAI'05)*, pages 1368–1373. AAAI Press, 2005.
16. H. Kleine Büning, M. Karpinski, and A. Flögel. Resolution for quantified Boolean formulas. *Information and Computation*, 117(1):12–18, 1995.
17. H. Kleine Büning and T. Lettman. *Propositional logic: Deduction and algorithms*. Cambridge University Press, 1999.
18. D. E. Knuth. *The Art of Computer Programming*, volume 3: Sorting and Searching, chapter 5.2.2 Sorting by Exchanging, pages 106–110. Addison-Wesley, 1973.
19. I. Lynce and J. P. Marques-Silva. Hidden structure in unsatisfiable random 3-SAT: An empirical study. In *Proc. 16th IEEE Int. Conf. on Tools with Artificial Intelligence (ICTAI'04)*, pages 246–251. IEEE Computer Society, 2004.
20. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

21. R. Niedermeier and P. Rossmanith. An efficient fixed-parameter algorithm for 3-hitting set. *Journal of Discrete Algorithms*, 1(1):89–102, 2003.
22. N. Nishimura, P. Ragde, and S. Szeider. Detecting backdoor sets with respect to Horn and binary clauses. In *Proc. 7th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'04)*, pages 96–103. Informal Proceedings, 2004.
23. N. Nishimura, P. Ragde, and S. Szeider. Solving #SAT using vertex covers. *Acta Informatica*, 44(7-8):509–523, 2007.
24. C. Otwell, A. Remshagen, and K. Truemper. An effective QBF solver for planning problems. In *Proc. Int. Conf. on Modeling, Simulation and Visualization Methods and Int. Conf. on Algorithmic Mathematics and Computer Science (MSV/AMCS'04)*, pages 311–316. CSREA Press, 2004.
25. G. Pan and M. Y. Vardi. Fixed-parameter hierarchies inside PSpace. In *Proc. 21st Annual IEEE Symp. on Logic in Computer Science (LICS'06)*, pages 27–36. IEEE Computer Society, 2006.
26. C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
27. J. Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.
28. Y. Ruan, H. A. Kautz, and E. Horvitz. The backdoor key: A path to understanding problem hardness. In *Proc. 19th AAAI Conf. on Artificial Intelligence (AAAI'04)*, pages 124–130. AAAI Press, 2004.
29. A. Sabharwal, C. Ansotegui, C. Gomes, J. Hart, and B. Selman. QBF modeling: Exploiting player symmetry for simplicity and efficiency. In *Proc. 9th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'06)*, volume 4121 of *LNCS*, pages 382–395. Springer-Verlag, 2006.
30. M. Samer. Variable Dependencies of Quantified CSPs. In *Proc. 15th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'08)*, volume 5330 of *LNCS*. Springer-Verlag, 2008.
31. M. Samer and S. Szeider. Backdoor sets of quantified Boolean formulas. In *Proc. 10th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *LNCS*, pages 230–243. Springer-Verlag, 2007.
32. H. Samulowitz and F. Bacchus. Binary clause reasoning in QBF. In *Proc. 9th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'06)*, volume 4121 of *LNCS*, pages 353–367. Springer-Verlag, 2006.
33. L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proc. 5th Annual ACM Symp. on Theory of Computing (STOC'73)*, pages 1–9. ACM Press, 1973.
34. S. Szeider. Backdoor sets for DLL subsolvers. *Journal of Automated Reasoning*, 35(1-3):73–88, 2005.
35. S. Szeider. Generalizations of matched CNF formulas. *Annals of Mathematics and Artificial Intelligence*, 43(1-4):223–238, 2005.
36. S. Szeider. Matched formulas and backdoor sets. In *Proc. 10th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *LNCS*, pages 94–99. Springer-Verlag, 2007.
37. R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2):146–160, 1972.
38. R. Williams, C. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proc. 18th Int. Joint Conf. on Artificial Intelligence (IJCAI'03)*, pages 1173–1178. Morgan Kaufmann, 2003.
39. R. Williams, C. Gomes, and B. Selman. On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search. In *Proc. 6th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 222–230. Informal Proceedings, 2003.