Technical Report AC-TR-17-003

April 2017

Long-Distance Q-Resolution with Dependency Schemes

Tomáš Peitl, Friedrich Slivovsky, and Stefan Szeider

Noname manuscript No.
(will be inserted by the editor)

Long-Distance Q-Resolution with Dependency Schemes

Tomáš Peitl · Friedrich Slivovsky · Stefan Szeider

the date of receipt and acceptance should be inserted later

Abstract Resolution proof systems for quantified Boolean formulas (QBFs) provide a formal model for studying the limitations of state-of-the-art search-based QBF solvers that use these systems to generate proofs. We study a proof system that combines two proof systems supported by the solver DepQBF: Q-resolution with generalized universal reduction according to a dependency scheme and long distance Q-resolution. We show that the resulting proof system—which we call long-distance Q(D)-resolution—is sound for the reflexive resolution-path dependency scheme—in fact, we prove that it admits strategy extraction in polynomial time. This comes as an application of a general result, by which we identify a whole class of dependency schemes for which long-distance Q(D)-resolution admits polynomial-time strategy extraction. As a special case, we obtain soundness and polynomial-time strategy extraction for long distance Q(D)-resolution with the standard dependency scheme. We report on experiments with a configuration of DepQBF that generates proofs in this system.

Keywords QBF · Q-resolution · dependency schemes · strategy extraction

Acknowledgements We would like to thank Florian Lonsing for helpful discussions and for pointing out how to modify DepQBF so that it generates LDQ(D^{std}) proofs. This research was partially supported by Austrian Science Fund (FWF) grants P27721 and W1255-N23.

Tomáš Peitl
E-mail: tomas.peitl@gmail.com
· Friedrich Slivovsky
Tel.: +43 (1) 58801 186104
E-mail: fslivovsky@gmail.com
· Stefan Szeider
E-mail: stefan@szeider.net

Institute of Computer Graphics and Algorithms, TU Wien, A-1040 Vienna, Austria

1 Introduction

Quantified Boolean Formulas (QBFs) offer succinct encodings for problems from domains such as formal verification, synthesis, and planning [5, 12, 15, 28, 35, 40]. Although the combination of (more verbose) propositional encodings with SAT solvers is still the state-of-the-art approach to many of these problems, QBF solvers are gaining ground. An arsenal of new techniques has been introduced over the past few years [10, 11, 13, 20, 21, 23, 24, 27, 30, 31, 33], and these advances in solver technology have been accompanied by the development of a better understanding of the underlying QBF proof systems and their limitations [4, 7–9, 17, 25, 39].

Search-based solvers based on the QDPLL algorithm [14] represent one of the principal state-of-the-art approaches in QBF solving. Akin to modern SAT solvers, these solvers rely on successive variable assignments in combination with fast constraint propagation and learning. Unlike SAT solvers, however, search-based QBF solvers are constrained by the variable dependencies induced by the quantifier prefix[1]: while SAT solvers can assign variables in any order, search-based QBF solvers can only assign variables from the leftmost quantifier block that contains unassigned variables, since the assignment of a variable further to the right might depend on the variable assignment to this block. In the most extreme case, this forces solvers into a fixed order of variable assignments, rendering decision variable heuristics ineffective.

The search-based solver DepQBF uses dependency schemes to partially bypass this restriction [10,29]. Dependency schemes can sometimes identify pairs of variables as independent, allowing the solver to assign them in any order. This gives decision heuristics more freedom and results in increased performance [10].

While this provides a strong motivation to use dependency schemes, their integration with QDPLL poses challenges of its own. Soundness of the proof system underlying QDPLL with the standard dependency scheme as implemented in DepQBF was shown only recently [39], and combining other state-of-the-art techniques with dependency schemes is often highly nontrivial. In this paper, we focus on two such issues:

(a) Long-distance Q-resolution permits the derivation of tautological clauses in certain cases [2, 42, 43]. This system can be used in constraint learning as an alternative to Q-resolution, leading to fewer backtracks during search and, sometimes, reduced runtime [18]. In addition, clause learning based on long-distance Q-resolution is substantially easier to implement. Currently, however, DepQBF does not permit learning based on long-distance Q-resolution in conjunction with dependency schemes, as the resulting proof system is not known to be sound.

(b) For applications in verification and synthesis, it is not enough for solvers to decide whether an input QBF is true or false—they also have to generate a certificate. Such certificates can be efficiently constructed from Q-resolution [2] and even long-distance Q-resolution proofs [3]. However, it is not clear whether this is possible for proofs generated by DepQBF with the standard dependency scheme, and proof generation with the standard dependency scheme is disabled by default.

We address (a) by showing that long-distance Q-resolution combined the reflexive resolution-path dependency scheme [39] is sound. In fact, we prove that this proof

[1] We consider QBFs in prenex normal form.

system allows for certificate extraction in polynomial time, thus resolving (b) as well. These results also hold for long-distance Q-resolution combined with the weaker standard dependency scheme.

Our proof relies on a familiar interpretation of Q-resolution refutations as winning strategies for the universal player in the evaluation game [22]. Defining LDQ(D) as the proof system consisting of long-distance Q-resolution with a dependency scheme D, we identify a natural property of dependency schemes D that not only allows for the interpretation of an LDQ(D)-refutation as a winning strategy for the universal player, but even implies certificate extraction in time $O(|\mathcal{P}| \cdot n)$ from an LDQ(D)-refutation $\mathcal{P}$ of a QBF with n variables. We then show that the reflexive resolution path dependency scheme in fact has this property.

One of our motivations for studying the combination of long-distance Q-resolution and dependency schemes is that it is already supported by DepQBF—by default, long-distance Q-resolution and the standard dependency scheme cannot be enabled at the same time because it was unclear whether the resulting solver configuration is sound. To complement our theoretical results, we performed experiments with a modified version of DepQBF that uses constraint learning based on LDQ(D) with the standard dependency scheme. Our experiments show that performance with this type of learning is on par with and—in some cases—even surpasses the performance of DepQBF with other configurations of constraint learning.

Organization Section 2 establishes basic notions used throughout this paper. In Section 3, we review dependency schemes and introduce the LDQ(D) proof system. Section 4 is split into two parts: in the first part, we define a property of dependency schemes D and prove that it is sufficient for soundness of LDQ(D); in the second part, we show that the reflexive resolution-path dependency scheme has this property. In Section 5, we report on experiments with a modified version of DepQBF that generates LDQ($\mathrm{D}^{\mathrm{std}}$)-proofs. In Section 6, we briefly discuss recently published related work. We conclude in Section 7 with some open questions.

2 Preliminaries

Formulas and Assignments. 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. We call a clause *tautological* if it contains the same variable negated as well as unnegated. A *CNF formula* is a finite conjunction of non-tautological clauses. Whenever convenient, we treat clauses as sets of literals, and CNF formulas as sets of sets of literals. We write $var(C)$ for the set of variables occuring (negated or unnegated) in a clause C, that is, $var(C) = \{\, var(\ell) : \ell \in C \,\}$. Moreover, we let $var(\varphi) = \bigcup_{C \in \varphi} var(C)$ denote the set of variables occurring in a CNF formula φ.

A *truth assignment* (or simply *assignment*) to a set X of variables is a mapping $\tau : X \to \{0, 1\}$. We write $[X]$ for the set of truth assignments to X, and extend $\tau : X \to \{0, 1\}$ to literals by letting $\tau(\neg x) = 1 - \tau(x)$ for $x \in X$. Let $\tau : X \to \{0, 1\}$ be a truth assignment. The restriction $C[\tau]$ of a clause C by τ is defined as follows: if there is a literal $\ell \in C \cap (X \cup \overline{X})$ such that $\tau(\ell) = 1$ then $C[\tau] = 1$. Otherwise, $C[\tau] = C \setminus (X \cup \overline{X})$. The restriction $\varphi[\tau]$ of a CNF formula φ by the assignment τ is defined $\varphi[\tau] = \{\, C[\tau] : C[\tau] \neq 1 \,\}$.

PCNF Formulas. A *PCNF formula* is denoted by $\Phi = \mathcal{Q}.\varphi$, where φ is a CNF formula and $\mathcal{Q} = Q_1X_1 \dots Q_nX_n$ is a sequence such that $Q_i \in \{\forall, \exists\}$, $Q_i \neq Q_{i+1}$ for $1 \leq i < n$, and the X_i are pairwise disjoint sets of variables. We call φ the *matrix* of Φ and $\mathcal{Q}$ the *(quantifier) prefix* of Φ, and refer to the X_i as *quantifier blocks*. We require that $var(\varphi) = X_1 \cup \dots \cup X_n$ and write $var(\Phi) = var(\varphi)$. We define a partial order $<_\Phi$ on $var(\varphi)$ as $x <_\Phi y \Leftrightarrow x \in X_i, y \in X_j, i < j$. We extend $<_\Phi$ to a relation on literals in the obvious way and drop the subscript whenever Φ is understood. For $x \in var(\Phi)$ we let $R_\Phi(x) = \{\, y \in var(\Phi) : x <_\Phi y \,\}$ and $L_\Phi(x) = \{\, y \in var(\Phi) : y <_\Phi x \,\}$ denote the sets of variables *to the right* and *to the left* of x in Φ, respectively. Relative to the PCNF formula Φ, variable x is called *existential* (*universal*) if $x \in X_i$ and $Q_i = \exists$ ($Q_i = \forall$). The set of existential (universal) variables occurring in Φ is denoted $var_\exists(\Phi)$ ($var_\forall(\Phi)$). The *size* of a PCNF formula $\Phi = \mathcal{Q}.\varphi$ is defined as $|\Phi| = \sum_{C \in \varphi} |C|$. If τ is an assignment, then $\Phi[\tau]$ denotes the PCNF formula $\mathcal{Q}'.\varphi[\tau]$, where $\mathcal{Q}'$ is the quantifier prefix obtained from $\mathcal{Q}$ by deleting variables that do not occur in $\varphi[\tau]$. *True* and *false* PCNF formulas are defined in the usual way.

Countermodels. Let $\Phi = \mathcal{Q}.\varphi$ be a PCNF formula. A *countermodel* of Φ is an indexed family $\{f_u\}_{u \in var_\forall(\Phi)}$ of functions $f_u : [L_\Phi(u)] \rightarrow \{0, 1\}$ such that $\varphi[\tau] = \{\emptyset\}$ for every assignment $\tau : var(\Phi) \rightarrow \{0, 1\}$ satisfying $\tau(u) = f_u(\tau|_{L_\Phi(u)})$ for $u \in var_\forall(\Phi)$.

Proposition 1 (Folklore) *A PCNF formula is false if, and only if, it has a countermodel.*

3 Dependency Schemes and LDQ(D)-Resolution

In this section, we introduce the proof system LDQ(D), which combines Q(D)-resolution [39] with long-distance Q-resolution [2]. Q-resolution is a generalization of propositional resolution to PCNF formulas [26]. Q-resolution is of practical interest due to its relation to search based QBF solvers that implement the QDPLL algorithm [14]: the trace of a QDPLL solver generated for a false PCNF formula corresponds to a Q-resolution refutation [19]. QDPLL generalizes the well-known DPLL procedure [16] from SAT to QSAT. In a nutshell, DPLL searches for a satisfying assignment of an input formula by propagating unit clauses and assigning pure literals until the formula cannot be simplified any further, at which point it picks an unassigned variable and branches on the assignment of this variable. Although any of the remaining variables can be chosen for assignment, the order of assignment can have significant effects on the runtime, and modern SAT solvers derived from the DPLL algorithm use sophisticated heuristics to determine what variable to assign next [32].

In QDPLL, the quantifier prefix imposes constraints on the order of variable assignments: a variable may be assigned only if it occurs in the leftmost quantifier block with unassigned variables. Often, this is more restrictive than necessary. For instance, variables from disjoint subformulas may be assigned in any order. Intuitively, a variable can be assigned as long as it *does not depend* on any unassigned variable. This is the intuition underlying a generalization of QDPLL implemented in the solver DepQBF [10, 29]. DepQBF uses a *dependency scheme* [36] to compute an overapproximation of variable dependencies. Dependency schemes are mappings that associate every PCNF formula with a binary relation on its variables that refines the order of variables in the quantifier prefix.[2]

Definition 1 (Dependency Scheme) A *dependency scheme* is a mapping D that associates each PCNF formula Φ with a relation $D_\Phi \subseteq \{\, (x, y) : x <_\Phi y \,\}$ called the *dependency relation* of Φ with respect to D.

The mapping which simply returns the prefix ordering of an input formula can be thought of as a baseline dependency scheme:

Definition 2 (Trivial Dependency Scheme) The *trivial dependency scheme* $\mathrm{D}^{\mathrm{trv}}$ associates each PCNF formula Φ with the relation $\mathrm{D}^{\mathrm{trv}}_\Phi = \{\, (x, y) : x <_\Phi y \,\}$.

DepQBF uses a dependency relation to determine the order in which variables can be assigned: if y is a variable and there is no unassigned variable x such that (x, y) is in the dependency relation, then y is considered ready for assignment. DepQBF also uses the dependency relation to generalize the ∀-reduction rule used in clause learning [10]. As a result of its use of dependency schemes, DepQBF generates proofs in a generalization of Q-resolution called Q(D)-resolution [39], a proof system that takes a dependency scheme D as a parameter.

Dependency schemes can be partially ordered based on their dependency relations: if the dependency relation computed by a dependency scheme D_1 is a subset of the dependency relation computed by a dependency scheme D_2, then D_1 is *more general* than D_2. The more general a dependency scheme, the more freedom DepQBF has in choosing decision variables. Currently, (aside from the trivial dependency scheme) DepQBF supports the so-called *standard dependency scheme* [36].[3] We will work with the more general *reflexive resolution-path dependency scheme* [39], a variant of the resolution-path dependency scheme [38, 41]. This dependency scheme computes an overapproximation of variable dependencies based on whether two variables are connected by a (pair of) resolution path(s).

Definition 3 (Resolution Path) Let $\Phi = \mathcal{Q}.\varphi$ be a PCNF formula and let X be a set of variables. 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 $\pi = \ell_1, \dots, \ell_{2k}$ is a resolution path in Φ via X, we say that ℓ_1 and ℓ_{2k} are *connected in* Φ (with respect to X). For every $i \in \{1, \dots, k\}$ we say that π *goes through* $var(\ell_{2i})$.

One can think of a resolution path as a potential chain of implications: if each clause C_i contains exactly two literals, then assigning ℓ_1 to 0 requires setting ℓ_{2k} to 1. If, in addition, there is such a path from $\overline{\ell_1}$ to $\overline{\ell_{2k}}$, then ℓ_1 and ℓ_{2k} have to be assigned the same value. Accordingly, the resolution path dependency scheme identifies variables connected by a pair of resolution paths as potentially dependent on each other.

[2] The original definition of dependency schemes [36] is more restrictive than the one given here, but the additional requirements are irrelevant for the purposes of this paper.

[3] Strictly speaking, it uses a refined version of the standard dependency scheme [29, p.49].

$$\frac{}{C} \text{ (input clause)} \qquad\qquad \frac{C_1 \vee e \qquad \neg e \vee C_2}{C_1 \vee C_2} \text{ (resolution)}$$

An *input clause* $C \in \varphi$ can be used as an axiom. From two clauses $C_1 \vee e$ and $\neg e \vee C_1$, where e is an existential variable, the *(long-distance) resolution* rule can derive the clause $C_1 \vee C_2$, provided that $(u, e) \notin D_\Phi$ for each universal variable u with $u \in C_1$ and $\overline{u} \in C_2$ (or vice versa).

$$\frac{C}{C \setminus \{u, \neg u\}} \text{ ($\forall$-reduction)}$$

The $\forall$-*reduction* rule derives the clause $C \setminus \{u, \neg u\}$ from C, where $u \in \mathit{var}(C)$ is a universal variable such that $(u, e) \notin D_\Phi$ for every existential variable $e \in \mathit{var}(C)$.

Fig. 1: Derivation rules of LDQ(D)-resolution for a PCNF formula $\Phi = \mathcal{Q}.\varphi$.

Definition 4 (Dependency Pair) Let Φ be a PCNF formula and $x, y \in \mathit{var}(\Phi)$. We say $\{x, y\}$ is a *resolution-path dependency pair* of Φ with respect to $X \subseteq \mathit{var}_\exists(\Phi)$ 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.

Definition 5 The *reflexive resolution-path dependency scheme* is the mapping $\mathrm{D}^{\mathrm{rrs}}$ that assigns to each PCNF formula $\Phi = \mathcal{Q}.\varphi$ the relation $\mathrm{D}^{\mathrm{rrs}}_\Phi = \{\, x <_\Phi y : \{x, y\}$ is a resolution-path dependency pair in Φ with respect to $R_\Phi(x) \setminus \mathit{var}_\forall(\Phi) \,\}$.

Both Q-resolution and Q(D)-resolution only allow for the derivation of non-tautological clauses, that is, clauses that do not contain a literal negated as well as unnegated. *Long-distance Q-resolution* is a variant of Q-resolution that admits tautological clauses in certain cases [2]. Variants of QDPLL that allow for learnt clauses to be tautological [42, 43] have been shown to generate proofs in long-distance Q-resolution [18].

In long-distance Q-resolution, when a tautological clause is created by resolution, a variable that appears in both polarities must be to the right of the pivot variable. We generalize this by requiring that the pivot be independent of a tautological variable to obtain *long-distance Q(D)-resolution* (LDQ(D)-resolution). The derivation rules of LDQ(D)-resolution are shown in Figure 1.[4] Here, as in the rest of the paper, D denotes an arbitrary dependency scheme.

A derivation in a proof system consists of repeated applications of the derivation rules to derive a clause from the clauses of an input formula. Here, derivations will be represented by node-labeled directed acyclic graphs (DAGs). More specifically, we require these DAGs to have a unique sink (that is, a node without outgoing edges) and each of their nodes to have at most two incoming edges. We further assume an ordering on the in-neighbors (or parents) of every node with two incoming edges—that is, each node has a “first” and a “second” in-neighbor. Referring to such DAGs as *proof DAGs*, we define the following two operations to represent resolution and $\forall$-reduction:

[4] The resolution rule as defined here is more general than the one considered in an earlier version of this paper [34], in that we admit complementary universal literals to be “merged” as long as the pivot is independent according to D (rather than $\mathrm{D}^{\mathrm{trv}}$). This definition—which we think is required to capture proofs generated by DepQBF—was proposed in (independent) work by Beyersdorff and Blinkhorn [6].

1. If ℓ is a literal and $\mathcal{P}_1$ and $\mathcal{P}_2$ are proof DAGs with distinct sinks v_1 and v_2, then $\mathcal{P}_1 \odot_\ell \mathcal{P}_2$ is the proof DAG consisting of the union of $\mathcal{P}_1$ and $\mathcal{P}_2$ along with a new sink v that has two incoming edges, the first one from v_1 and the second one from v_2. Moreover, if C_1 is the label of v_1 in $\mathcal{P}_1$ and C_2 is the label of v_2 in $\mathcal{P}_2$, then v is labeled with the clause $(C_1 \setminus \{\ell\}) \cup (C_2 \setminus \{\bar{\ell}\})$.
2. If u is a variable and $\mathcal{P}$ is a proof DAG with a sink w labeled with C, then $\mathcal{P} - u$ denotes the proof DAG obtained from $\mathcal{P}$ by adding an edge from w to a new node v such that v is labeled with $C \setminus \{u, \neg u\}$.

Definition 6 (Derivation) An *LDQ(D)-resolution derivation* (or *LDQ(D)-derivation*) of a clause C from a PCNF formula $\Phi = \mathcal{Q}.\varphi$ is a proof DAG $\mathcal{P}$ satisfying the following properties.

- Source nodes are labeled with input clauses from φ.
- If a node with label C has parents labeled C_1 and C_2 then C can be derived from C_1 and C_2 by (long-distance) resolution.
- If a node labeled with a clause C has a single parent with label C' then C can be derived from C' by $\forall$-reduction with respect to the dependency scheme D.

We refer to these nodes as *input nodes*, *resolution nodes*, and *$\forall$-reduction nodes*, respectively.

Let $\mathcal{P}$ be an LDQ(D)-derivation from a PCNF formula Φ. The (clause) label of the sink node is called the *conclusion* of $\mathcal{P}$, denoted $Cl(\mathcal{P})$. If the conclusion of $\mathcal{P}$ is the empty clause then we refer to $\mathcal{P}$ as an *LDQ(D)-refutation* of Φ. For a node v of $\mathcal{P}$, the *subderivation* (of $\mathcal{P}$) rooted at v is the proof DAG induced by v and its ancestors in $\mathcal{P}$. It is straightforward to verify that the resulting proof DAG is again an LDQ(D)-derivation from Φ. For convenience, we will identify (sub)derivations with their sinks. The *size* of $\mathcal{P}$, denoted $|\mathcal{P}|$, is the total number of literal occurrences in clause labels of $\mathcal{P}$.

4 Soundness of and Strategy Extraction for LDQ($\mathrm{D}^{\mathrm{rrs}}$)

A PCNF formula can be associated with an evaluation game played between an existential and a universal player. These players take turns assigning quantifier blocks in the order of the prefix. The existential player wins if the matrix evaluates to 1 under the resulting variable assignment, while the universal player wins if the matrix evaluates to 0. One can show that the formula is true (false) if and only if the existential (universal) player has a winning strategy in this game, and this winning strategy is a (counter)model.

Goultiaeva, Van Gelder and Bacchus [22] proved that a Q-resolution refutation can be used to compute winning moves for the universal player in the evaluation game. The idea is that universal maintains a "restriction" of the refutation by the assignment constructed in the evaluation game, which is a refutation of the restricted formula.

For assignments made by the existential player, the universal player only needs to consider each instance of resolution whose pivot variable is assigned: one of the premises is not satisifed and can be used to (re)construct a refutation.

When it is universal's turn, the quantifier block for which she needs to pick an assignment is leftmost in the restricted formula. This means that $\forall$-reduction of these variables is blocked by any of the remaining existential variables and can only be applied to a purely universal clause. In a Q-resolution refutation, these variables must therefore be reduced at the very end, and because Q-resolution does not permit tautological clauses, only one polarity of each universal variable from the leftmost block can appear in a refutation. It follows that universal can maintain a Q-resolution refutation by assigning variables from the leftmost block in such a way as to map the associated literals to 0, effectively deleting them from the remaining Q-resolution refutation.

In this manner, the universal player can maintain a refutation until the end of the game, when all variables have been assigned. At that point, a refutation can consist only of the empty clause, which means that the assignment chosen by the two players falsifies a clause of the original matrix and universal has won the game.

Egly, Lonsing, and Widl [18] observed that this argument goes through even in the case of long-distance Q-resolution, since a clause containing both u and $\neg u$ for a universal variable u can only be derived by resolving on an existential variable to the left of u, but no such existential variable exists if u is from the leftmost block.

In this section, we will prove that this argument can be generalized to LDQ($\mathrm{D}^{\mathrm{rrs}}$)-refutations. We illustrate this correspondence with an example:

Example 1 Consider the PCNF formula

$$\Phi = \exists x \, \forall u \, \exists e, y \quad (x \vee u \vee \overline{y}) \wedge (\overline{x} \vee \overline{u} \vee \overline{y}) \wedge (x \vee y) \wedge (\overline{x} \vee e) \wedge (\overline{u} \vee y) \wedge (\overline{y} \vee e)$$

Figure 2 shows an LDQ($\mathrm{D}^{\mathrm{rrs}}$)-refutation of Φ. The only universal variable is u, so a strategy for the universal player in the evaluation game associated with Φ has to determine an assignment to u given an assignment to x, the only (existential) variable preceding u. The figure illustrates how to compute the assignment to u for the two possible assignments $\tau : \{x\} \to \{0, 1\}$ from the restriction of the refutation by τ. In both cases, only one polarity of u occurs in the restricted refutation and therefore it is easy for universal to determine the correct assignment. Notice that in one of the cases, a generalized ∀-reduction node remains present in the restriction—this shows that we cannot limit ourselves to looking at the final reduction step in the proof when looking for the variables to assign (as is the case with ordinary Q-resolution refutations, cf. [22]).

In all of the above cases, the key property that allows universal to maintain a refutation is that universal variables from the leftmost quantifier block may appear in at most one polarity. We will show that, indeed, this property is sufficient for soundness of LDQ(D) when combined with a natural monotonicity property of dependency schemes.

Definition 7 A dependency scheme D is *monotone* if $D_{\Phi[\tau]} \subseteq D_{\Phi}$ for every PCNF formula Φ and every assignment τ to a subset of $var(\Phi)$. We say that D is *simple* if, for every PCNF formula $\Phi = \forall X \mathcal{Q}.\varphi$, every LDQ(D)-derivation $\mathcal{P}$ from Φ, and every universal variable $u \in X$, u or $\overline{u}$ does not appear in $\mathcal{P}$. A dependency scheme D is *normal* if it is both monotone and simple.

As in the case of Q-resolution, universal's move for a particular quantifier block can be computed from the assignment corresponding to the previous moves and the refutation in polynomial time. Since every polynomial-time algorithm can be implemented by a family of polynomially-sized circuits, and because these circuits

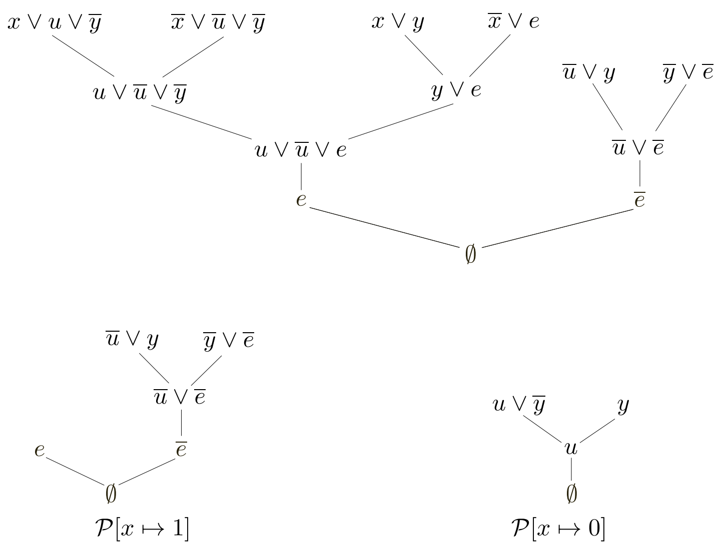

Fig. 2: An LDQ($\mathrm{D}^{\mathrm{rrs}}$)-refutation of the formula Φ from Example 1 (above) and two its restrictions (below).

can even be computed in polynomial time [1, p.109], it follows that LDQ(D) admits polynomial-time strategy extraction when D is normal. While the strategy extraction algorithm based on these general considerations is unlikely to be efficient, the algorithm for computing winning moves for universal is simple enough to be amenable to efficient simulation by a Boolean circuit. In Section 4.1, we give a direct construction that leads to the following result.

Theorem 1 *Let D be a normal dependency scheme. Then, there is an algorithm that computes a countermodel of a PCNF formula Φ with n variables from an LDQ(D)-refutation $\mathcal{P}$ of Φ in time $O(|\mathcal{P}| \cdot n)$.*

As an application of this general result, we will prove that the reflexive resolution-path dependency scheme is normal in Section 4.2.

Theorem 2 *D^{rrs} is normal.*

Corollary 1 *There is an algorithm that computes a countermodel of a PCNF formula Φ with n variables from an LDQ(D^{rrs})-refutation $\mathcal{P}$ of Φ in time $O(|\mathcal{P}| \cdot n)$.*

This result immediately carries over to the less general standard dependency scheme:

Corollary 2 *There is an algorithm that computes a countermodel of a PCNF formula Φ with n variables from an LDQ(D^{std})-refutation $\mathcal{P}$ of Φ in time $O(|\mathcal{P}| \cdot n)$.*

In combination with Proposition 1, these results imply soundness of both proof systems.

Corollary 3 *The systems LDQ(D^{std}) and LDQ(D^{rrs}) are sound.*

We begin by formally defining the "restriction" of an LDQ(D)-derivation by an assignment, which is a straightforward generalization of this operation for Q-resolution derivations [22].[5] The result of restricting a derivation is either a derivation or the object $\top$, which can be interpreted as representing the tautological clause containing every literal. Accordingly, we stipulate that $\ell \in \top$ for every literal ℓ.

Definition 8 (Restriction) Let Φ be a PCNF formula and let $\mathcal{P}$ be an LDQ(D)-derivation from Φ. Further, let $X \subseteq var(\Phi)$ and let $\tau : X \rightarrow \{0,1\}$ be a truth assignment. The *restriction of* $\mathcal{P}$ *by* τ, in symbols $\mathcal{P}[\tau]$, is defined as follows.

1. If $\mathcal{P}$ is an input node there are two cases. If $Cl(\mathcal{P})[\tau] = 1$ then $\mathcal{P}[\tau] = \top$. Otherwise, $\mathcal{P}[\tau]$ is the proof DAG consisting of a single node labeled with $Cl(\mathcal{P})[\tau]$.
2. If $\mathcal{P} = \mathcal{P}_1 \odot_\ell \mathcal{P}_2$, that is, if $\mathcal{P}$ is a resolution node, we distinguish four cases:
   (a) If $\ell \notin Cl(\mathcal{P}_1[\tau])$ then $\mathcal{P}[\tau] = \mathcal{P}_1[\tau]$.
   (b) If $\ell \in Cl(\mathcal{P}_1[\tau])$ and $\overline{\ell} \notin Cl(\mathcal{P}_2[\tau])$ then $\mathcal{P}[\tau] = \mathcal{P}_2[\tau]$.
   (c) If $\ell \in Cl(\mathcal{P}_1[\tau])$, $\overline{\ell} \in Cl(\mathcal{P}_2[\tau])$, and $\mathcal{P}_1[\tau] = \top$ or $\mathcal{P}_2[\tau] = \top$, we let $\mathcal{P}[\tau] = \top$.
   (d) If $\ell \in Cl(\mathcal{P}_1[\tau])$, $\overline{\ell} \in Cl(\mathcal{P}_2[\tau])$, $\mathcal{P}_1[\tau] \neq \top$, and $\mathcal{P}_2[\tau] \neq \top$, we define $\mathcal{P}[\tau] = \mathcal{P}_1[\tau] \odot_\ell \mathcal{P}_2[\tau]$.
3. If $\mathcal{P} = \mathcal{P}' - u$, that is, if $\mathcal{P}$ is a $\forall$-reduction node, we distinguish three cases:
   (a) If $\mathcal{P}'[\tau] = \top$ then $\mathcal{P}[\tau] = \top$.
   (b) If $\mathcal{P}'[\tau] \neq \top$ and $u \notin var(Cl(\mathcal{P}'[\tau]))$ then $\mathcal{P}[\tau] = \mathcal{P}'[\tau]$.
   (c) If $\mathcal{P}'[\tau] \neq \top$ and $u \in var(Cl(\mathcal{P}'[\tau]))$ then $\mathcal{P}[\tau] = \mathcal{P}'[\tau] - u$.

If D is a monotone dependency scheme, LDQ(D)-refutations are preserved under restriction by an existential assignment (cf. [22, Lemma 4]). This is stated in the following lemma, which can by proved by a straightforward induction on the structure of the LDQ(D)-derivation.

Lemma 1 *Let D be a monotone dependency scheme, let* $\mathcal{P}$ *be an LDQ(D)-derivation from a PCNF formula* Φ*, let* $E \subseteq var_\exists(\Phi)$*, and let* $\tau : E \rightarrow \{0,1\}$ *be an assignment. If* $\mathcal{P}[\tau] = \top$ *then* $Cl(\mathcal{P})[\tau] = 1$*. Otherwise,* $\mathcal{P}[\tau]$ *is an LDQ(D)-derivation from* $\Phi[\tau]$ *such that* $Cl(\mathcal{P}[\tau]) \subseteq Cl(\mathcal{P})[\tau]$*.*

Proof The proof is by induction on the structure of $\mathcal{P}$.

1. If $\mathcal{P}$ is an input node then $\mathcal{P}[\tau] = \top$ iff $Cl(\mathcal{P})[\tau] = 1$ and $Cl(\mathcal{P}[\tau]) = Cl(\mathcal{P})[\tau]$ otherwise, so the statement holds trivially.
2. If $\mathcal{P} = \mathcal{P}_1 \odot_\ell P_2$ is a resolution node we distinguish four cases:
   (a) If $\ell \notin Cl(\mathcal{P}_1[\tau])$, then $\mathcal{P}[\tau] = \mathcal{P}_1[\tau]$ and

   $$Cl(\mathcal{P}_1[\tau]) = Cl(\mathcal{P}_1[\tau]) \setminus \{\ell\} \subseteq Cl(\mathcal{P}_1)[\tau] \setminus \{\ell\} \subseteq Cl(\mathcal{P})[\tau],$$

   where the first inclusion holds by induction hypothesis and the second inclusion follows from the definition of the resolution rule.
   (b) If $\ell \in Cl(\mathcal{P}_1[\tau])$ and $\overline{\ell} \notin Cl(\mathcal{P}_2[\tau])$ then $\mathcal{P}[\tau] = \mathcal{P}_2[\tau]$ and the statement follows via a symmetric argument.

[5] Our definition slightly differs from the original for the resolution rule: if restriction removes the pivot variable from both premises, we simply pick the (restriction of the) first premise as the result (rather than the clause containing fewer literals). This simplifies the certificate extraction argument given below.

(c) If $\ell \in Cl(\mathcal{P}_1[\tau])$, $\bar{\ell} \in Cl(\mathcal{P}_2[\tau])$, and $\mathcal{P}_1[\tau] = \top$ or $\mathcal{P}_2[\tau] = \top$ then we have $\mathcal{P}[\tau] = \top$. Assume without loss of generality that $\mathcal{P}_1[\tau] = \top$. Then $Cl(\mathcal{P}_1)[\tau] = 1$ by induction hypothesis. Let $\ell' \in Cl(\mathcal{P}_1)$ be a literal such that $\tau(\ell') = 1$. We distinguish two cases. If $\ell \neq \ell'$ then $\ell' \in Cl(\mathcal{P})$ and $Cl(\mathcal{P})[\tau] = 1$. Otherwise, $\tau(\overline{\ell'}) = \tau(\bar{\ell}) = 0$, and we must have $\mathcal{P}_2[\tau] = \top$ since $\bar{\ell} \in Cl(\mathcal{P}_2[\tau])$. By induction hypothesis, there has to be another literal $\ell'' \neq \bar{\ell}$ such that $\ell'' \in Cl(\mathcal{P}_2)$ and $\tau(\ell'') = 1$. The literal ℓ'' is contained in $Cl(\mathcal{P})$ as well, so $Cl(\mathcal{P})[\tau] = 1$.

(d) If $\ell \in Cl(\mathcal{P}_1[\tau])$, $\bar{\ell} \in Cl(\mathcal{P}_2[\tau])$, $\mathcal{P}_1[\tau] \neq \top$, and $\mathcal{P}_2[\tau] \neq \top$, then $\mathcal{P}[\tau] = \mathcal{P}_1[\tau] \odot_\ell \mathcal{P}_2[\tau]$ and $\mathcal{P}[\tau] \neq \top$. By induction hypothesis, $\mathcal{P}_1[\tau]$ is an LDQ(D)-derivation from $\Phi[\tau]$ such that $Cl(\mathcal{P}_1[\tau]) \subseteq Cl(P_1)[\tau]$, and $\mathcal{P}_2[\tau]$ is an LDQ(D)-derivation from $\Phi[\tau]$ such that $Cl(\mathcal{P}_2[\tau]) \subseteq Cl(P_2)[\tau]$. Monotonicity of D ensures that after restriction, the resolution step is still sound and thus $\mathcal{P}[\tau]$ is an LDQ(D)-derivation from $\Phi[\tau]$ as well and

$$\begin{aligned} Cl(\mathcal{P}[\tau]) &= Cl(\mathcal{P}_1[\tau] \odot_\ell \mathcal{P}_2[\tau]) \\ &= Cl(\mathcal{P}_1[\tau]) \cup Cl(\mathcal{P}_2[\tau]) \setminus \{\ell, \bar{\ell}\} \\ &\subseteq Cl(\mathcal{P}_1)[\tau] \cup Cl(\mathcal{P}_2)[\tau] \setminus \{\ell, \bar{\ell}\} = Cl(\mathcal{P})[\tau]. \end{aligned}$$

3. If $\mathcal{P} = \mathcal{P}' - u$ is a reduction node, we have to distinguish two cases:
   (a) If $\mathcal{P}'[\tau] = \top$ then $\mathcal{P}[\tau] = \top$ by definition. By induction hypothesis $Cl(\mathcal{P}')[\tau] = 1$ and since τ does not assign u, we get $Cl(\mathcal{P})[\tau] = 1$ as well.
   (b) If $\mathcal{P}[\tau] \neq \top$ then $\mathcal{P}'[\tau] \neq \top$ by definition of the restriction operation. By induction hypothesis, $\mathcal{P}'[\tau]$ is an LDQ(D)-derivation from $\Phi[\tau]$ such that $Cl(\mathcal{P}'[\tau]) \subseteq Cl(\mathcal{P}')[\tau]$. If $u \notin var(Cl(\mathcal{P}'[\tau]))$ then $\mathcal{P}[\tau] = \mathcal{P}'[\tau]$ and the statement holds. Otherwise, if $u \in var(Cl(\mathcal{P}'[\tau]))$ then $\mathcal{P}[\tau] = \mathcal{P}'[\tau] - u$ and thus

$$\begin{aligned} Cl(\mathcal{P}[\tau]) &= Cl(\mathcal{P}'[\tau]) \setminus \{u, \neg u\} \\ &\subseteq Cl(\mathcal{P}')[\tau] \setminus \{u, \neg u\} = (Cl(\mathcal{P}') \setminus \{u, \neg u\})[\tau] = \mathcal{P}[\tau], \end{aligned}$$

   where the last but one equality holds because τ does not assign u. To see that $\mathcal{P}[\tau] = \mathcal{P}'[\tau] - u$ is a valid $\forall$-reduction node, note that $Cl(\mathcal{P}'[\tau]) \subseteq Cl(\mathcal{P}')$ by induction hypothesis and observe that $\mathrm{D}^{\mathrm{rrs}}_{\Phi[\tau]} \subseteq \mathrm{D}^{\mathrm{rrs}}_{\Phi}$.

$\square$

Above, we argued that the universal player can use an LDQ(D)-refutation for a normal dependency scheme D in order to compute winning moves in the evaluation game associated with a PCNF formula and that this can be used to compute a countermodel of the formula in polynomial time. We now prove this directly, by showing how to construct a circuit implementing a countermodel from an LDQ(D)-refutation.

We begin by describing auxiliary circuits simulating the restriction operation. Let $\Phi = Q_1 X_1 \dots Q_k X_k.\varphi$ be a PCNF formula and let $\mathcal{P}$ be a refutation of Φ. For each quantifier block X_i, each subderivation $\mathcal{S}$ of $\mathcal{P}$, and each literal ℓ, we will construct circuits $\textsc{top}^i_{\mathcal{S}}$ and $\textsc{contains}^i_{\mathcal{S},\ell}$ with inputs from $X = \bigcup_{j<i} X_j$ such that, for every assignment $\sigma : X \to \{0, 1\}$,

$$\textsc{top}^i_{\mathcal{S}}[\sigma] = 1 \iff \mathcal{S}[\sigma] = \top \tag{1}$$

$$\textsc{contains}^i_{\mathcal{S},\ell}[\sigma] = 1 \iff \ell \in Cl(\mathcal{S}[\sigma]) \tag{2}$$

We first describe our construction and then prove that it satisfies the above properties in Lemma 2. Let $\mathcal{S}$ be an input node. We let

$$\textsc{top}^1_{\mathcal{S}} := \bigvee\{\, Cl(\mathcal{S}) \cap (X_1 \cup \overline{X_1})\,\},$$

and define $\textsc{top}^i_{\mathcal{S}}$ for $1 < i \leq k$ as

$$\textsc{top}^i_{\mathcal{S}} := \textsc{top}^{i-1}_{\mathcal{S}} \vee \bigvee\{\, Cl(\mathcal{S}) \cap (X_i \cup \overline{X_i})\,\}.$$

Moreover, for $1 \leq i \leq k$ we define $\textsc{contains}^i_{\mathcal{S},\ell}$ as

$$\textsc{contains}^i_{\mathcal{S},\ell} = \begin{cases} 1 & \text{if } \ell \in Cl(\mathcal{S}) \setminus (X \cup \overline{X}), \\ \textsc{top}^i_{\mathcal{S}} & \text{otherwise.} \end{cases}$$

For non-input nodes, we proceed as follows. If $\mathcal{S} = \mathcal{S}_1 \odot_q \mathcal{S}_2$, we define $\textsc{top}^i_{\mathcal{S}}$ as

$$\textsc{top}^i_{\mathcal{S}} = (\textsc{contains}^i_{\mathcal{S}_1,q} \wedge \textsc{top}^i_{\mathcal{S}_2}) \vee (\textsc{contains}^i_{\mathcal{S}_2,\overline{q}} \wedge \textsc{top}^i_{\mathcal{S}_1}),$$

and if $\mathcal{S} = \mathcal{S}' - u$, we let

$$\textsc{top}^i_{\mathcal{S}} := \textsc{top}^i_{\mathcal{S}'}.$$

For the $\textsc{contains}^i_{\mathcal{S},\ell}$ circuit, we distinguish two cases. Let ℓ be a literal and $\mathcal{S}$ a derivation. If $\ell \notin Cl(\mathcal{S})$ we simply let

$$\textsc{contains}^i_{\mathcal{S},\ell} := \textsc{top}^i_{\mathcal{S}}.$$

Otherwise, if $\ell \in Cl(\mathcal{S})$, we have to consider two cases. First, if $\mathcal{S} = \mathcal{S}_1 \odot_q \mathcal{S}_2$, we let

$$\begin{aligned} \textsc{contains}^i_{\mathcal{S},\ell} = & \textsc{top}^i_{\mathcal{S}} \vee \\ & (\neg\textsc{contains}^i_{\mathcal{S}_1,q} \wedge \textsc{contains}^i_{\mathcal{S}_1,\ell}) \vee \\ & (\textsc{contains}^i_{\mathcal{S}_1,q} \wedge \neg\textsc{contains}^i_{\mathcal{S}_2,\overline{q}} \wedge \textsc{contains}^i_{\mathcal{S}_2,\ell}) \vee \\ & (\textsc{contains}^i_{\mathcal{S}_1,q} \wedge \textsc{contains}^i_{\mathcal{S}_2,\overline{q}} \wedge (\textsc{contains}^i_{\mathcal{S}_1,\ell} \vee \textsc{contains}^i_{\mathcal{S}_2,\ell})). \end{aligned}$$

Second, if $\mathcal{S} = \mathcal{S}' - u$, then

$$\textsc{contains}^i_{\mathcal{S},\ell} := \textsc{contains}^i_{\mathcal{S}',\ell}.$$

To implement the winning strategy for universal sketched above, we further construct circuits $\textsc{polarity}_{\mathcal{S},u}$ for each node $\mathcal{S}$ of $\mathcal{P}$ and each universal variable $u \in var_\forall(\Phi)$, such that, for each assignment $\tau : L_\Phi(u) \to \{0,1\}$,

$$\textsc{polarity}_{\mathcal{S},u}[\tau] = 1 \iff u \text{ occurs in } \mathcal{S}[\tau]. \tag{3}$$

Let $u \in X_i$ be a universal variable from the ith quantifier block. If $\mathcal{S}$ is an input node, we simply define

$$\textsc{polarity}_{\mathcal{S},u} := \textsc{contains}^i_{\mathcal{S},u},$$

and if $\mathcal{S} = \mathcal{S}' - u$ is a $\forall$-reduction node, we let

$$\textsc{polarity}_{\mathcal{S},u} := \textsc{polarity}_{\mathcal{S}',u}.$$

If $\mathcal{S} = \mathcal{S}_1 \odot_q \mathcal{S}_2$, then

$$\begin{aligned}
\textsc{polarity}_{\mathcal{S},u} :=& (\neg\textsc{contains}^i_{\mathcal{S}_1,q} \wedge \textsc{polarity}_{\mathcal{S}_1,u}) \vee \\
& (\textsc{contains}^i_{\mathcal{S}_1,q} \wedge \neg\textsc{contains}^i_{\mathcal{S}_2,\overline{q}} \wedge \textsc{polarity}_{\mathcal{S}_2,u}) \vee \\
& (\textsc{contains}^i_{\mathcal{S}_1,q} \wedge \textsc{contains}^i_{\mathcal{S}_2,\overline{q}} \wedge \\
& \qquad (\textsc{polarity}_{\mathcal{S}_1,u} \vee \textsc{polarity}_{\mathcal{S}_2,u})).
\end{aligned}$$

Lemma 2 *Let $\Phi = Q_1X_1 \dots Q_kX_k.\varphi$ be a PCNF formula and let $\mathcal{P}$ be an LDQ(D)-derivation from Φ. For each $1 \leq i \leq k$, each literal ℓ, every $u \in var_\forall(\Phi) \cap X_i$, and every truth assignment $\sigma : \bigcup_{j=1}^{i-1} X_j \to \{0,1\}$, $\textsc{top}^i_{\mathcal{P}}$ satisfies (1), $\textsc{contains}^i_{\mathcal{P},\ell}$ satisfies (2), and $\textsc{polarity}_{\mathcal{P},u}$ satisfies (3).*

Proof Let $X = X_1 \cup \dots \cup X_{i-1}$. As (1) and (2) are related, we will prove them first. We will use induction on the structure of $\mathcal{P}$, with the induction hypothesis that (1) and (2) hold. The inductive step will be carried out in two phases. In the first phase, we prove that (1) holds and in the second phase we use this additional information to prove that (2) holds as well.

1. Let $\mathcal{P}$ be an input node. By Definition 8 we have $\mathcal{P}[\sigma] = \top$ if, and only if, $Cl(\mathcal{P})[\sigma] = 1$. Since σ only assigns variables in X, this is the case if, and only if, $\textsc{top}^i_{\mathcal{P}}[\sigma] = 1$, so (1) holds.
2. Let $\mathcal{P} = \mathcal{P}_1 \odot_q \mathcal{P}_2$ such that (1) and (2) hold for $\mathcal{P}_1$ and $\mathcal{P}_2$. We distinguish several cases.
   (a) $q \notin Cl(\mathcal{P}_1[\tau])$. Then $\mathcal{P}[\tau] = \mathcal{P}_1[\tau]$. Since $q \notin Cl(\mathcal{P}_1[\tau])$, it cannot be the case that $\mathcal{P}_1[\tau] = \top$ and so $\mathcal{P}[\tau] \neq \top$ as well. By the induction hypothesis, we have $\textsc{contains}^i_{\mathcal{P}_1,q}[\tau] = 0$ and also $\textsc{top}^i_{\mathcal{P}_1}[\tau] = 0$ which means $\textsc{top}^i_{\mathcal{P}}[\tau] = 0$ as required.
   (b) $q \in Cl(\mathcal{P}_1[\tau])$ and $\overline{q} \notin Cl(\mathcal{P}_2[\tau])$. Then $\mathcal{P}[\tau] = \mathcal{P}_2[\tau]$. Since $\overline{q} \notin Cl(\mathcal{P}_2[\tau])$, we cannot have $\mathcal{P}_2[\tau] = \top$ and thus $\mathcal{P}[\tau] \neq \top$ as well. By the induction hypothesis, we have $\textsc{contains}^i_{\mathcal{P}_2,\overline{q}}[\tau] = 0$ and also $\textsc{top}^i_{\mathcal{P}_2}[\tau] = 0$ which means $\textsc{top}^i_{\mathcal{P}}[\tau] = 0$ as required.
   (c) $q \in Cl(\mathcal{P}_1[\tau])$ and $\overline{q} \in Cl(\mathcal{P}_2[\tau])$ and $\mathcal{P}_1[\tau] = \top$ or $\mathcal{P}_2[\tau] = \top$. Then $\mathcal{P}[\tau] = \top$ and by induction hypothesis, we have $\textsc{contains}^i_{\mathcal{P}_1,q}[\tau] = 1$ as well as $\textsc{contains}^i_{\mathcal{P}_2,\overline{q}}[\tau] = 1$, and $\textsc{top}^i_{\mathcal{P}_1}[\tau] = 1$ or $\textsc{top}^i_{\mathcal{P}_2}[\tau] = 1$. In any case, $\textsc{top}^i_{\mathcal{P}}[\tau] = 1$.
   (d) $q \in Cl(\mathcal{P}_1[\tau])$ and $\overline{q} \in Cl(\mathcal{P}_2[\tau])$ and $\mathcal{P}_1[\tau] \neq \top$ and $\mathcal{P}_2[\tau] \neq \top$. Then $\mathcal{P}[\tau] = \mathcal{P}_1[\tau] \odot_q \mathcal{P}_2[\tau] \neq \top$. By induction hypothesis, we have $\textsc{top}^i_{\mathcal{P}_1}[\tau] = 0$ and $\textsc{top}^i_{\mathcal{P}_2}[\tau] = 0$, which ensures $\textsc{top}^i_{\mathcal{P}}[\tau] = 0$.
3. Let $\mathcal{P} = \mathcal{P}' - u$. From the definitions, we can immediately see that $\mathcal{P}'[\tau] = \top \iff \mathcal{P}[\tau] = \top$ and $\textsc{top}^i_{\mathcal{P}'} = \textsc{top}^i_{\mathcal{P}}$ which proves (1).

We have proved that $\mathcal{P}[\tau] = \top \iff \textsc{top}^i_{\mathcal{P}}[\tau] = 1$, and it can be easily checked that, by definition, $\textsc{top}^i_{\mathcal{P}} \Rightarrow \textsc{contains}^i_{\mathcal{P},\ell}$ for every literal ℓ. Therefore, if $\mathcal{P}[\tau] = \top$, (2) holds and in the following, we can restrict ourselves to the cases when $\mathcal{P}[\tau] \neq \top$. Also, we can restrict ourselves to the cases when ℓ (the literal in question) actually belongs to $Cl(\mathcal{P})$, because otherwise $\textsc{contains}^i_{\mathcal{P},\ell} = \textsc{top}^i_{\mathcal{P}}$ and in that case (2) clearly holds.

1. Let $\mathcal{P}$ be an input node. We may assume $\mathcal{P}[\tau] \neq \top$ and $\ell \in Cl(\mathcal{P})$ by the above. By definition, we can easily see that $\textsc{contains}^i_{\mathcal{P},\ell}[\tau] = 1$ if, and only if, $\ell \in Cl(\mathcal{P}[\tau])$.
2. Let $\mathcal{P} = \mathcal{P}_1 \odot_q \mathcal{P}_2$ such that (1) and (2) hold for $\mathcal{P}_1$ and $\mathcal{P}_2$. We distinguish several cases.
   (a) $q \notin Cl(\mathcal{P}_1[\tau])$. By the induction hypothesis, we have $\textsc{contains}^i_{\mathcal{P}_1,q}[\tau] = 0$. Also $\mathcal{P}[\tau] = \mathcal{P}_1[\tau]$ and

   $$\ell \in Cl(\mathcal{P}[\tau]) \Longleftrightarrow \ell \in Cl(\mathcal{P}_1[\tau]) \Longleftrightarrow \textsc{contains}^i_{\mathcal{P}_1,\ell}[\tau],$$

   where the second equivalence holds by induction hypothesis. Since we have $\textsc{contains}^i_{\mathcal{P}_1,q}[\tau] = 0$, we can write

   $$\textsc{contains}^i_{\mathcal{P}_1,\ell}[\tau] \Longleftrightarrow \neg\textsc{contains}^i_{\mathcal{P}_1,q}[\tau] \wedge \textsc{contains}^i_{\mathcal{P}_1,\ell}[\tau].$$

   Because $\textsc{contains}^i_{\mathcal{P}_1,q}[\tau] = 0$ and $\textsc{top}^i_{\mathcal{P}}[\tau] = 0$, the only disjunct in the definition of $\textsc{contains}^i_{\mathcal{P},\ell}[\tau]$ that can possibly be satisfied is the second one, so that

   $$\textsc{contains}^i_{\mathcal{P},\ell}[\tau] \Longleftrightarrow \neg\textsc{contains}^i_{\mathcal{P}_1,q}[\tau] \wedge \textsc{contains}^i_{\mathcal{P}_1,\ell}[\tau],$$

   which establishes (2).
   (b) $q \in Cl(\mathcal{P}_1[\tau])$ and $\overline{q} \notin Cl(\mathcal{P}_2[\tau])$. By the induction hypothesis, we have $\textsc{contains}^i_{\mathcal{P}_1,q}[\tau] = 1$ and $\textsc{contains}^i_{\mathcal{P}_2,\overline{q}}[\tau] = 0$. An argument symmetric to the one for the preceding case can be used to show (2).
   (c) $q \in Cl(\mathcal{P}_1[\tau])$ and $\overline{q} \in Cl(\mathcal{P}_2[\tau])$ and $\mathcal{P}_1[\tau] = \top$ or $\mathcal{P}_2[\tau] = \top$. In this case $\mathcal{P}[\tau] = \top$ which has already been taken care of (see above).
   (d) $q \in Cl(\mathcal{P}_1[\tau])$ and $\overline{q} \in Cl(\mathcal{P}_2[\tau])$ and $\mathcal{P}_1[\tau] \neq \top$ and $\mathcal{P}_2[\tau] \neq \top$. Then $\mathcal{P}[\tau] = \mathcal{P}_1[\tau] \odot_q \mathcal{P}_2[\tau]$ and since we have restricted ourselves to the case when $\ell \in Cl(\mathcal{P})$ (see above), we have

   $$\begin{aligned}\ell \in Cl(\mathcal{P}[\tau]) &\Longleftrightarrow \ell \in Cl(\mathcal{P}_1[\tau]) \vee \ell \in Cl(\mathcal{P}_2[\tau]) \\ &\Longleftrightarrow \textsc{contains}^i_{\mathcal{P}_1,\ell}[\tau] \vee \textsc{contains}^i_{\mathcal{P}_2,\ell}[\tau],\end{aligned}$$

   where the final equivalence follows from the induction hypothesis. It is straightforward to verify that the last expression in turn is equivalent to the fourth disjunct in the definition of $\textsc{contains}^i_{\mathcal{P},\ell}$ being satisfied, and since this is the only disjunct that can be satisfied in this case, we conclude that (2) holds.

   By Definition 8, $\mathcal{P}[\sigma] = \top$ if, and only if, $\mathcal{P}'[\sigma] = \top$, and $\ell \in Cl(\mathcal{P}[\sigma])$ if, and only if, $\ell \in Cl(\mathcal{P}'[\sigma])$, for each literal $\ell \in Cl(\mathcal{P})$. Since (1) and (2) hold for $\mathcal{P}'$ by induction hypothesis, these properties must hold for $\mathcal{P}$ as well.

Let us now turn to the proof of (3).

1. If $\mathcal{P}$ is an input node we have

   $$u \in P[\tau] \Longleftrightarrow u \in Cl(\mathcal{P}[\tau]) \Longleftrightarrow \textsc{contains}^i_{\mathcal{P},u}[\tau] = \textsc{polarity}_{\mathcal{P},u}[\tau]$$

   by what we proved previously and the definition of $\textsc{polarity}_{\mathcal{P},u}$ for input nodes (and the fact that a literal appears in a derivation that consists of a single input node iff it occurs in the clause of associated with that node).
2. Let $\mathcal{P} = \mathcal{P}_1 \odot_q \mathcal{P}_2$.

(a) $q \notin Cl(\mathcal{P}_1[\tau])$. Then $\mathcal{P}[\tau] = \mathcal{P}_1[\tau]$ and by the induction hypothesis, we have

$$u \text{ appears in } \mathcal{P}[\tau] \iff u \text{ appears in } \mathcal{P}_1[\tau] \iff \textsc{polarity}_{\mathcal{P}_1,u}[\tau] = 1.$$

Using (2), it is readily verified that $\textsc{polarity}_{\mathcal{P},u}[\tau] = \textsc{polarity}_{\mathcal{P}_1,u}[\tau]$.

(b) $q \in Cl(\mathcal{P}_1[\tau])$ and $\overline{q} \notin Cl(\mathcal{P}_2[\tau])$. Here, (3) can be proved using an argument symmetric to one for the previous case.

(c) $q \in Cl(\mathcal{P}_1[\tau])$ and $\overline{q} \in Cl(\mathcal{P}_2[\tau])$ and $\mathcal{P}_1[\tau] = \top$ or $\mathcal{P}_2[\tau] = \top$. Then $\mathcal{P}[\tau] = \top$, so u appears in $\mathcal{P}[\tau]$. Without loss of generality, let $\mathcal{P}_1[\tau] = \top$. By the induction hypothesis we have $\textsc{polarity}_{\mathcal{P}_1,u}[\tau] = 1$, which, along with the assumptions for this case and (2), implies that $\textsc{polarity}_{\mathcal{P},u}$ is satisfied by the last disjunct.

(d) $q \in Cl(\mathcal{P}_1[\tau])$ and $\overline{q} \in Cl(\mathcal{P}_2[\tau])$ and $\mathcal{P}_1[\tau] \neq \top$ and $\mathcal{P}_2[\tau] \neq \top$. In this case u appears in $\mathcal{P}[\tau]$ if, and only if, it appears in $\mathcal{P}_1[\tau]$ or in $\mathcal{P}_2[\tau]$. Using the induction hypothesis and (2), one can verify that this is the case if, and only if, $\textsc{polarity}_{\mathcal{P},u}[\tau] = 1$.

□

These auxiliary circuits can be efficiently constructed in a top-down manner, from the input nodes to the conclusion. By a careful analysis, we obtain the following:

Lemma 3 *There is an algorithm that, given a PCNF formula Φ and an LDQ(D)-derivation $\mathcal{P}$ from Φ, computes the circuits $\textsc{polarity}_{\mathcal{P},u}$ for every universal variable u in time $O(|\mathcal{P}| \cdot n)$, where $n = |var(\Phi)|$.*

Proof The algorithm first sorts clauses according to a fixed order of literals. Let k be the number of quantifier blocks in the prefix of Φ. There is at most one circuit $\textsc{top}^i_{\mathcal{P}}$ for each node $\mathcal{S}$ of $\mathcal{P}$ and each $1 \leq i \leq k$. Similarly, there is at most one circuit $\textsc{contains}^i_{\mathcal{S},\ell}$ for each node $\mathcal{S}$ of $\mathcal{P}$, each $1 \leq i \leq k$, and each literal $\ell \in Cl(\mathcal{S})$.

Once $\textsc{top}^i_{\mathcal{S}}$ has been computed for each $1 \leq i \leq k$, the circuits $\textsc{contains}^i_{\mathcal{S},\ell}$ can easily be constructed for each $1 \leq i \leq k$ and every literal $\ell \in Cl(\mathcal{S})$. Overall, this can be done in time

$$O(|Cl(\mathcal{S})| \cdot k) \subseteq O(|Cl(\mathcal{S})| \cdot n).$$

Assume that the circuits $\textsc{contains}^i_{\mathcal{S},\ell}$ are stored in lists following the order of literals in $Cl(\mathcal{S})$. Then for each node $\mathcal{S}$, the circuits $\textsc{top}^i_{\mathcal{S}}$ and $\textsc{contains}^i_{\mathcal{S},\ell}$ associated with $\mathcal{S}$ can again be computed in time $O(|Cl(\mathcal{S})| \cdot n)$, so that overall, these circuits can be computed in time $O(|\mathcal{P}| \cdot n)$ for all nodes of $\mathcal{P}$. Having computed the circuits $\textsc{contains}$ and $\textsc{top}$, the circuits $\textsc{polarity}_{\mathcal{S},u}$ can be computed for each node $\mathcal{S}$ and each universal variable $u \in var_{\forall}(\Phi)$ in time $O(|\mathcal{P}| \cdot n)$. □

Using Lemma 1, we can spell out the argument sketched at the beginning of this section and prove that for normal dependency schemes D, the universal player can maintain an LDQ(D)-refutation throughout the evaluation game by successively restricting an initial LDQ(D)-refutation by both players' moves and assigning universal variables from the leftmost remaining block X so as to falsify the (unique) literals from X remaining the refutation. Lemma 2 tells us that the $\textsc{polarity}$ circuits can be used to implement this strategy. In order to put things together, we will need the following two lemmas, which tell us that successive restriction and bulk restriction in fact yield the same result.

Lemma 4 *Let $\mathcal{P}$ be an LDQ(D)-derivation from a PCNF formula Φ, let τ_1, τ_2 be two assignments to disjoint sets of variables. Then $\mathcal{P}[\tau_1][\tau_2] = \mathcal{P}[\tau_1 \cup \tau_2]$.*

Proof By induction on the structure of the derivation. If $\mathcal{P}$ is an input node, we have $Cl(\mathcal{P}[\tau]) = Cl(\mathcal{P})[\tau] = Cl(\mathcal{P})[\tau_1][\tau_2] = Cl(\mathcal{P}[\tau_1][\tau_2])$ and since both derivations consist of a single node with the same label, they are in fact equal. For derivations created by the operations, the equality is trivially preserved. □

Lemma 5 *Let D be a normal dependency scheme, let $\Phi = Q_1 X_1 \dots Q_k X_k.\varphi$ be a PCNF formula, let $\mathcal{P}$ be an LDQ(D)-refutation of Φ. Let X_i be a universal quantifier block and let $\tau : \bigcup_{j=1}^{i-1} X_j \to \{0,1\}$ be an assignment. If $\mathcal{P}[\tau]$ is an LDQ(D)-refutation of $\Phi[\tau]$, then $\mathcal{P}[\tau \cup \sigma]$ is an LDQ(D)-refutation of $\Phi[\tau \cup \sigma]$, where $\sigma : X_i \to \{0,1\}$ is the assignment such that $\sigma(u) = \neg\textsc{polarity}_{\mathcal{P},u}[\tau]$ for each $u \in X_i$.*

Proof Assume $\mathcal{P}[\tau]$ is an LDQ(D)-refutation of $\Phi[\tau]$. Let $u \in X_i$. Because D is simple, variable u appears in $\mathcal{P}[\tau]$ in at most one polarity. If u does not appear in $\mathcal{P}[\tau]$ at all, the restriction $\mathcal{P}[\tau][\sigma]$ does not depend on $\sigma(u)$. Otherwise, there is a unique literal ℓ with $var(\ell) = u$ that appears in $\mathcal{P}[\tau]$. By Lemma 2, $\textsc{polarity}_{\mathcal{P},u}[\tau] = 1$ iff u appears in $\mathcal{P}[\tau]$, so $\sigma(u) = \neg\textsc{polarity}_{\mathcal{P},u}[\tau] = 0$ if $\ell = u$ and $\sigma(u) = 1$ if $\ell = \neg u$. It is a straightforward consequence that $\mathcal{P}[\tau][\sigma]$ can be obtained from $\mathcal{P}[\tau]$ by deleting every occurrence of a variable $u \in X_i$ and omitting instances of $\forall$-reduction that become redundant as a result. Because D is monotone, the restriction $\mathcal{P}[\tau][\sigma]$ is an LDQ(D)-refutation of $\Phi[\tau \cup \sigma]$, and $\mathcal{P}[\tau][\sigma] = \mathcal{P}[\tau \cup \sigma]$ by Lemma 4. □

With that, we are ready to prove the final statement.

Lemma 6 *Let D be a normal dependency scheme, let $\mathcal{P}$ be an LDQ(D)-refutation of a PCNF formula Φ. Then the family $\{f_u\}_{u \in var_\forall(\Phi)}$ of functions $f_u = \neg\textsc{polarity}_{\mathcal{P},u}$ is a countermodel of Φ.*

Proof Let $\Phi = Q_1 X_1 \dots Q_k X_k.\varphi$ and let $\tau : var(\Phi) \to \{0,1\}$ be a truth assignment such that $\tau(u) = f_u\left(\tau|_{L_\Phi(u)}\right)$ for each universal variable u. Let $X_{<i} = \bigcup_{j=1}^{i-1} X_j$, and let $\tau_i = \tau|_{X_{<i}}$ for each $1 \le i \le k+1$. We claim that $\mathcal{P}[\tau_i]$ is an LDQ(D)-refutation of $\Phi[\tau_i]$ for $1 \le i \le k+1$. The assignment τ_1 is empty so $\mathcal{P}[\tau_1] = \mathcal{P}$ and $\Phi[\tau_1] = \Phi$ so the statement holds in that case. Suppose the claim holds for i such that $1 \le i \le k$. If $Q_i = \exists$, then $\mathcal{P}[\tau_i][\tau|_{X_i}]$ is an LDQ(D)-refutation of $\Phi[\tau_{i+1}]$ by Lemma 1, and $\mathcal{P}[\tau_i][\tau|_{X_i}] = \mathcal{P}[\tau_{i+1}]$ by Lemma 4. Otherwise, $Q_i = \forall$ and $\mathcal{P}[\tau_{i+1}]$ is an LDQ(D)-refutation of $\Phi[\tau_{i+1}]$ by Lemma 5. This completes the proof of the claim. In particular, we now have that $\mathcal{P}[\tau_{k+1}] = \mathcal{P}[\tau]$ is an LDQ(D)-refutation of $\Phi[\tau_{k+1}] = \Phi[\tau]$. Because $\Phi[\tau]$ does not contain any variables, the only way $\Phi[\tau]$ can have a refutation is that its matrix contains the empty clause, which means that $\varphi[\tau] = \{\emptyset\}$. □

Theorem 1 immediately follows from Lemma 3 and Lemma 6.

4.2 The Reflexive Resolution-Path Dependency Scheme is Normal

In order to prove Theorem 2 and show that $\mathrm{D}^{\mathrm{rrs}}$ is normal, we will need some insight into the relationship between resolution paths and LDQ($\mathrm{D}^{\mathrm{rrs}}$)-derivation. For a formula Φ and a universal variable u, we will denote by $T_u(\Phi)$ the set of existential literals to the right of u that are reachable from u by resolution-paths over existential variables to the right of u in Φ.

Lemma 7 *Let $\Phi = \forall X \mathcal{Q}.\varphi$ be a PNCF formula and $u \in X$ be a universal literal from the outermost block. Let $C_1, C_2 \in \Phi$ be clauses such that for some existential literal x, $x \in C_1$ and $\overline{x} \in C_2$, and let $C = C_1 \cup C_2 \setminus \{x, \overline{x}\}$. Then $T_u(\Phi) = T_u(\Phi \cup \{C\})$.*

Proof Let $\Phi' = \Phi \cup \{C\}$. Of course, by adding clauses to a formula, we preserve all existing resolution paths, so $T_u(\Phi) \subseteq T_u(\Phi')$. We will prove that the opposite inclusion holds as well. Let $e \in T_u(\Phi')$ and let π be a resolution path in Φ' certifying this. If π is also a resolution path in Φ, we are done. If it is not, it must be because it performs a *C-transition*, namely it contains a subsequence of two literals l_1, l_2 such that $var(l_1) \neq var(l_2)$, $l_1, l_2 \in C$, but $\{l_1, l_2\} \not\subseteq C_1$ and $\{l_1, l_2\} \not\subseteq C_2$. In this case, without loss of generality, we have $l_1 \in C_1$ and $l_2 \in C_2$. Let π_1 be the prefix of π up to and including l_1 and π_2 be the suffix of π starting with l_2. Let π' be the concatenation of π_1, x, $\overline{x}$, and π_2. It is clearly a valid resolution path and it uses one fewer C-transitions than π. Iterating this process, we can remove all C-transitions from π to obtain a resolution path in Φ. The resulting resolution path has the same endpoints and therefore certifies that $e \in T_u(\Phi)$. ☐

The previous lemma implies that when considering reachability from an outermost universal literal in a formula Φ, we can use clauses derived from Φ by LDQ($\mathrm{D}^{\mathrm{rrs}}$)-resolution as well. Indeed, adding clauses produced by the resolution rule does not change the set of reachable literals by Lemma 7, and adding clauses produced by universal reduction clearly does not even create new resolution paths. Particularly, if two literals ever appear together in a derived clause, there is a resolution path between them. This is summarized by the following corollary.

Corollary 4 *Let $\mathcal{P}$ be an LDQ(D^{rrs})-derivation from a PCNF formula $\Phi = \forall X \mathcal{Q}.\varphi$ and let $u \in X, u \in Cl(\mathcal{P})$. Then for all existential literals $e \in Cl(\mathcal{P})$, there is a resolution path from u to e in Φ.*

As a first step towards proving Theorem 2, we will prove that both polarities of an outermost universal literal cannot appear together in a single clause of a derivation.

Lemma 8 *Let $\mathcal{P}$ be an LDQ(D^{rrs})-derivation from a PCNF formula $\Phi = \forall X \mathcal{Q}.\varphi$ and let $u \in X$. Then $u \notin Cl(\mathcal{P})$ or $\neg u \notin Cl(\mathcal{P})$.*

Proof Towards a contradiction, suppose $u, \neg u \in Cl(\mathcal{P})$. Since input clauses do not contain both polarities of any literal, there must be a resolution step inside the derivation, which merges u and $\neg u$ into one clause. Let $\mathcal{P}' = \mathcal{P}_1 \odot_x \mathcal{P}_2$ be such a step. Then, without loss of generality, $x, u \in Cl(\mathcal{P}_1)$ and $\neg x, \neg u \in Cl(\mathcal{P}_2)$ and by Corollary 4, there is a resolution path from u to x and from $\neg u$ to $\neg x$, i.e. $(u, x) \in \mathrm{D}^{\mathrm{rrs}}_{\Phi}$. However, if x depends on u, opposite polarities of u cannot be merged in a resolution step with the pivot x, a contradiction. ☐

Using Lemma 8, we can proceed to finish the proof of Theorem 2.

Proof (of Theorem 2) Towards a contradiction, let $\mathcal{P}$ be an LDQ($\mathrm{D}^{\mathrm{rrs}}$)-derivation from a formula $\Phi = \forall X \mathcal{Q}.\varphi$ and let $u \in X$ be such that both polarities of u occur in the derivation $\mathcal{P}$. We will make several assumptions on the structure of $\mathcal{P}$, which are general enough so that an arbitrary derivation can be transformed into this form. First of all, let us assume that $\mathcal{P}$ is tree-like, i.e. its underlying DAG is a tree. In order to transform a derivation into tree-like form, one just makes copies of those nodes that are used more than once.

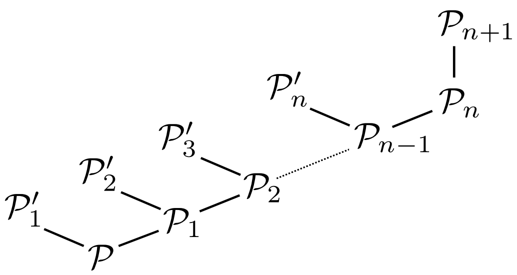

Fig. 3: Shape of the derivation constructed in the proof of Theorem 2.

Let us then note that if we have an LDQ($\mathrm{D}^{\mathrm{rrs}}$)-derivation from a formula Φ, we can delete (an arbitrary number of) universal variables from Φ and from all clauses of the derivation, omit reduction steps on those variables, and we will still have a valid LDQ($\mathrm{D}^{\mathrm{rrs}}$)-derivation (from a different formula). Moreover, all occurences of other literals, particularly u and $\neg u$, will be preserved. That allows us to make the assumption that Φ is of the form $\forall u \exists Y.\varphi$.

Let us further assume that $\mathcal{P}$ is minimal in the sense that it ends in a resolution step (we omit reduction steps at the end if necessary), i.e. $\mathcal{P} = \mathcal{P}_1 \odot_x \mathcal{P}'_1$ and in neither of the subderivations $\mathcal{P}_1, \mathcal{P}'_1$ both u and $\neg u$ occur. Next, let us omit all reduction steps on one polarity of u (introducing occurrences of that polarity to clauses below the reduction steps omitted). This must be possible, because the only way it can invalidate an inference rule is if we introduced the other polarity of u to a clause which already contained a literal on u. However, by our assumption, the only clause where this can happen is the root clause and it cannot contain both polarities of u by Lemma 8. Therefore, for omission we select that polarity of u which is contained in the root clause if there is any, and an arbitrary one if there is none.

Without loss of generality, let the polarity chosen for omission be $\neg u$. Since u is present in the derivation $\mathcal{P}$, but not in the root clause $Cl(\mathcal{P})$, there must be a reduction step on u somewhere in $\mathcal{P}$. As u is the only universal variable and we omitted all reduction steps on $\neg u$, all reduction steps in $\mathcal{P}$ are on u and $\mathcal{P}$ must have the from depicted in Figure 3, where $\mathcal{P}_n = \mathcal{P}_{n+1} - u$ is a lowermost reduction step on u and the subsequent resolutions are on pivots $x_n, \ldots, x_1$. Let $C_0 = Cl(\mathcal{P}), C_i = Cl(\mathcal{P}_i), C'_i = Cl(\mathcal{P}'_i)$. The clauses $C'_1, \ldots, C'_n, C_{n+1}$ are derived by LDQ($\mathrm{D}^{\mathrm{rrs}}$)-resolution and by Lemma 7 we know that we can use them to show resolution-path connections as if they were input clauses. By the transformations we considered we know that starting from an arbitrary LDQ($\mathrm{D}^{\mathrm{rrs}}$)-derivation we can obtain a valid LDQ($\mathrm{D}^{\mathrm{rrs}}$)-derivation in this form, so any contradiction we derive from here means a contradiction with the assumption that an LDQ($\mathrm{D}^{\mathrm{rrs}}$)-derivation contains both polarities of a universal variable from the outermost block, thus proving Theorem 2. With that, we are ready to finish the proof.

We will prove that there is a resolution path from $\neg u$ to u going through an existential literal in C_{n+1}, which is in contradiction with the soundness of reduction of u from C_{n+1}. Let us consider *open* resolution paths, i.e. resolution paths without their final literal. If an open resolution path ends in a literal ℓ of clause C, we say that the path *leads* to the clause C. By induction on n, we will prove that there is an open resolution path from $\neg u$ which leads to the clause C_n. If $n = 1$, we have the path $\neg u, \neg x_1, x_1$. For $n > 1$, let π be the open path leading to C_{n-1} and let ℓ be its last literal. Then either $\ell \in C_n$, in which case we have an open path leading to C_n, or $\ell \in C'_n$, in which case we have the open path $\pi, \neg x_n, x_n$ leading to C_n. An open path that leads to C_n also leads to C_{n+1}, because those two clauses only differ in the presence of u and therefore can be closed by the literal u to obtain the required resolution path. $\square$

5 Experiments

To gauge the potential of clause learning based on LDQ($\mathrm{D}^{\mathrm{std}}$), we ran experiments with the search-based solver DepQBF.[6] By default, DepQBF supports proof generation only in combination with the trivial dependency scheme—in that case, it generates Q-resolution or long-distance Q-resolution proofs (depending on whether long-distance resolution is enabled). However, by uncommenting a few lines in the source code, proof generation can also be enabled with the standard dependency scheme, and this option can even be combined with long-distance resolution. For false formulas, the resulting proofs are Q($\mathrm{D}^{\mathrm{std}}$)-resolution or LDQ($\mathrm{D}^{\mathrm{std}}$)-resolution refutations.

We compared the performance of DepQBF in four configurations,[7] each using a different proof system for constraint learning:

1. Long-distance Q-resolution with $\forall/\exists$-reduction according to $\mathrm{D}^{\mathrm{std}}$ (LDQD).
2. Long-distance Q-resolution with ordinary $\forall/\exists$-reduction (LDQ).
3. Q-resolution with $\forall/\exists$-reduction according to $\mathrm{D}^{\mathrm{std}}$ (QD).
4. Ordinary Q-resolution (Q).

These experiments were performed on a cluster with Intel Xeon E5649 processors at 2.53 GHz running 64-bit Linux. We set time and memory limits of 900 seconds and 4 GB, respectively. Instances were taken from two tracks of the QBF Gallery 2014: the *applications* track consisting of 6 instance families and a total of 735 formulas, and the *QBFLib* track consisting of 276 formulas.

For our first set of experiments, we disabled dynamic QBCE (Quantified Blocked Clause Elimination), a technique introduced with version 5.0 of DepQBF [30]. We further used bloqqer[8] with default settings as a preprocessor. Since LDQ($\mathrm{D}^{\mathrm{std}}$) generalizes both long-distance Q-resolution and Q($\mathrm{D}^{\mathrm{std}}$)-resolution, we were expecting a performance increase with LDQ($\mathrm{D}^{\mathrm{std}}$)-learning compared to learning based on the other proof systems. However, all four configurations showed virtually identical performance on both the application and QBFlib benchmark sets in terms of total runtime and instances solved within the time limit (see Table 1).

To get a more detailed picture, we broke down the results for the application track by instance family, limiting ourselves to instances that were solved by at least one configuration. The barplot in Figure 4 shows that there are considerable differences in performance between solver configurations for individual instances families, with each solver configuration being outperformed by another configuration on at least one family.

[6] `http://lonsing.github.io/depqbf/`

[7] As a sanity check, we verified that all configurations that were able to solve a particular instance returned the same result.

[8] `http://fmv.jku.at/bloqqer/`

Application track				
Configuration	Solved	True	False	Time
LDQD	377	186	191	343455
LDQ	377	186	191	345459
QD	377	183	194	343928
Q	376	182	194	345914

QBFLib track				
Configuration	Solved	True	False	Time
LDQD	130	69	61	140743
LDQ	131	69	62	141646
QD	129	67	62	140975
Q	127	65	62	142679

Table 1: Solved instances, solved true instances, solved false instances, and total runtime in seconds (including timeouts) with preprocessing (but without QBCE).

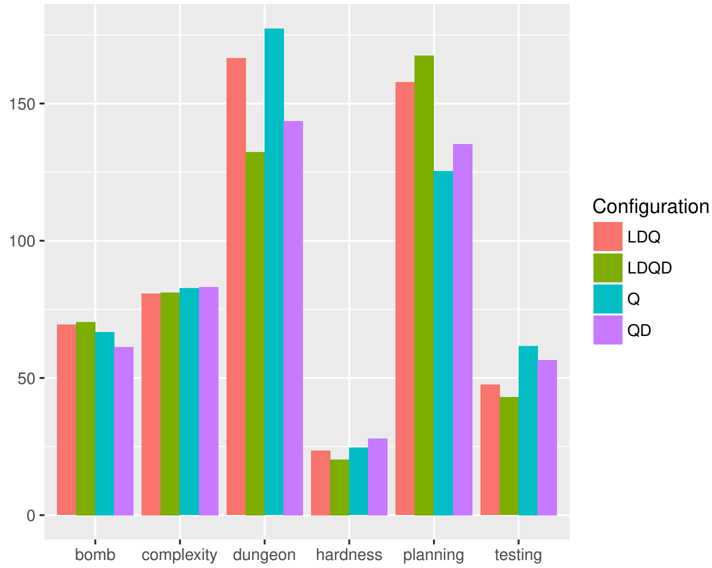

Fig. 4: Average runtime in seconds (y-axis) for instances from the application track for each instance family (x-axis), by solver configuration (with preprocessing, but without dynamic QBCE). Here, we only considered instances that were solved by at least one configuration.

For our second set of experiments, we turned on dynamic QBCE. This led to a significant performance increase both in terms of number of instances solved within the time limit and total runtime for both benchmark sets, a result that is consistent with the findings in [30]. However, as far as the performance of LDQ($\mathrm{D}^{\mathrm{std}}$)-learning is concerned, the application and QBFlib tracks differed significantly for this experiment. While LDQ($\mathrm{D}^{\mathrm{std}}$)-learning fared *worst* among the configurations both with respect to instances solved and total runtime on the application track, it was the *best*

Application track

Configuration	Solved	True	False	Time
LDQD	385	195	190	339143
LDQ	388	201	187	336739
QD	392	201	191	334965
Q	389	198	191	337141

QBFLib track

Configuration	Solved	True	False	Time
LDQD	145	75	70	132567
LDQ	133	64	69	141682
QD	137	70	67	134150
Q	129	62	67	142399

Table 2: Results with preprocessing and dynamic QBCE.

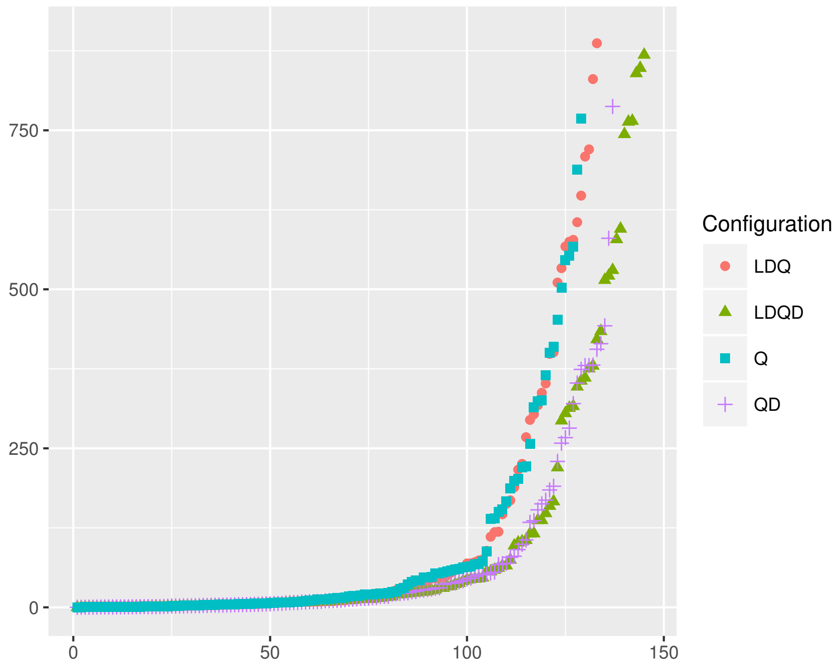

Fig. 5: Solved instances from the QBFLib track (x-axis) sorted by runtime (y-axis), by solver configuration (with preprocessing and dynamic QBCE).

configuration for the QBFlib track in both respects (see Table 2). Figure 5 shows that using the standard dependency scheme was beneficial both with and without long-distance resolution for the QBFlib instances.

For our final set of experiments, we left dynamic QBCE enabled but *disabled* preprocessing for the application track, as this was shown to lead to a performance *increase* in the case of learning with ordinary Q-resolution [30]. Indeed, this resulted

Configuration	Solved	True	False	Time
LDQD	440	223	217	287012
LDQ	435	223	212	291574
QD	440	225	215	291661
Q	437	221	216	337141

Table 3: Results for the application track with QBCE (but without preprocessing).

in a performance increase across the board (see Table 3). Moreover, LDQ(D^{std})-learning was the best configuration in terms of instances solved (on par with Q(D^{std})-resolution) as well as in terms of overall runtime. Moreover, LDQ(D^{std})-learning was the best configuration in terms of instances solved (on par with Q(D^{std})-resolution) as well as in terms of overall runtime.

6 Related Work

Recently and independently of this work, Beyersdorff and Blinkhorn investigated the soundness of Q-resolution proof systems parameterized by dependency schemes [6]. They define a property of dependency schemes D—*full exhibition*—which ensures that a certain version of long-distance Q(D)-resolution is sound, and show that the reflexive resolution-path dependency scheme has that property.

In a nutshell, a dependency scheme D is fully exhibited if every true QBF Φ has a model $\{f_e\}_{e \in var_\exists(\Phi)}$ such that f_e may only depend on a universal variable u if $(u, e) \in D_\Phi$ (such models have elsewhere been referred to as *D-models* [37]). It is fairly straightforward to show that Q(D)-resolution is sound if D has this property, but generalizing this result to proof systems with long-distance resolution presents a challenge. Beyersdorff and Blinkhorn show that full exhibition is sufficient for soundness of a restricted version of LDQ(D)-resolution, where complementary universal literals that are "merged" by resolution must be annotated with the (existential) pivot variable, and universal reduction can be applied only if every existential variable occurring in the premise or the annotation of a universal variable is independent of the universal variable to be reduced. However, it is uncertain whether proofs generated by DepQBF with LDQ(D)-learning satisfy this additional restriction.

How full exhibition relates our normality property is not entirely clear. Beyersdorff and Blinkhorn prove that full exhibition is *not* sufficient for soundness of LDQ(D)-resolution as defined here. In combination with Theorem 1, this shows that dependency schemes that are fully exhibited need not be normal. Whether there are normal dependency schemes that are not fully exhibited, on the other hand, remains open. Indeed, there is some evidence to the effect that normality entails full exhibition: consider a dependency scheme D that is not fully exhibited, and let $\Phi = \forall u \mathcal{Q}.\varphi$ be a true QBF that does not have a D-model. If we could show that there is an LDQ(D)-derivation $\mathcal{P}$ from Φ in which both u and $\neg u$ appear, we would be able to conclude that D is not normal, proving that normality implies full exhibition. We now sketch an argument for how this can be done under the assumption that u is the only universal variable of Φ. In this restricted case, the (non-)existence of a D-model can be expressed as a QBF Ψ by simply shifting existentials independent of u to the left. Because Φ does not have a D-model, Ψ must be false and admit a Q-resolution refutation $\mathcal{P}$, which is also an LDQ(D)-refutation of Φ. This refutation must contain both u and $\neg u$—otherwise, it could be turned into a Q-resolution refutation of Φ by simply postponing universal reduction. Obviously, the assumption that u is the only universal variable of Φ is very restrictive, but since we can suppose that D is monotone (recall that a dependency scheme is normal if it is both simple and monotone), there is hope that the argument for an arbitrary QBF can be reduced to this case by instantiating with a suitable variable assignment.

7 Discussion

The experiments in Section 5 show that DepQBF can benefit from learning based on LDQ(D^{std}). This benefit is essentially "for free", in that it does not require any changes to the implementation, but soundness of the resulting solver configuration is not immediate. The results of Section 4 contribute to a soundness proof, but they remain partial in two respects: first, soundness of LDQ(D^{std}) only implies that we can trust the solver when it outputs "false". To prove that "true" answers can be trusted as well, one has to show soundness of quantified term resolution when combined with the standard dependency scheme and long-distance resolution. Alternatively, one could use LDQ(D^{std}) for clause learning only, in combination with ordinary long-distance Q-resolution for term learning. Second, we observed synergies only when dynamic QBCE was activated, and it remains to show that clause learning based on LDQ(D^{std}) is sound in combination with this technique.

We take Theorem 1 as proof that, in principle, efficient certificate extraction from LDQ(D^{rrs})-refutations is possible. For practical purposes, the time bound of $O(|\mathcal{P}| \cdot n)$ is not good enough. For LDQ(D^{std}), a modified extraction algorithm achieves a runtime of $O(|\mathcal{P}| \cdot k)$, where k is the number of quantifier alternations of the input formula. A proof-of-concept implementation currently does not scale to proofs larger than a few megabytes, but we are confident that further improvements will lead to an efficient enough algorithm for practical use.

References

1. Arora, S., Barak, B.: Computational Complexity - A Modern Approach. Cambridge University Press (2009)
2. Balabanov, V., Jiang, J.R.: Unified QBF certification and its applications. Formal Methods in System Design **41**(1), 45–65 (2012)
3. Balabanov, V., Jiang, J.R., Janota, M., Widl, M.: Efficient extraction of QBF (counter)models from long-distance resolution proofs. In: B. Bonet, S. Koenig (eds.) Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI 2015, pp. 3694–3701. AAAI Press (2015)
4. Balabanov, V., Widl, M., Jiang, J.R.: QBF resolution systems and their proof complexities. In: C. Sinz, U. Egly (eds.) Theory and Applications of Satisfiability Testing - SAT 2014, *Lecture Notes in Computer Science*, vol. 8561, pp. 154–169. Springer Verlag (2014)
5. Benedetti, M., Mangassarian, H.: QBF-based formal verification: Experience and perspectives. J on Satisfiability, Boolean Modeling and Computation **5**(1-4), 133–191 (2008)
6. Beyersdorff, O., Blinkhorn, J.: Dependency schemes in QBF calculi: Semantics and soundness. In: M. Rueher (ed.) Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, *Lecture Notes in Computer Science*, vol. 9892, pp. 96–112. Springer Verlag (2016)

7. Beyersdorff, O., Chew, L., Janota, M.: On unification of QBF resolution-based calculi. In: E. Csuhaj-Varjú, M. Dietzfelbinger, Z. Ésik (eds.) Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, *Lecture Notes in Computer Science*, vol. 8635, pp. 81–93. Springer Verlag (2014)
8. Beyersdorff, O., Chew, L., Janota, M.: Proof complexity of resolution-based QBF calculi. In: E.W. Mayr, N. Ollinger (eds.) 32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, *LIPIcs*, vol. 30, pp. 76–89. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2015)
9. Beyersdorff, O., Chew, L., Mahajan, M., Shukla, A.: Feasible interpolation for QBF resolution calculi. In: M.M. Halldórsson, K. Iwama, N. Kobayashi, B. Speckmann (eds.) Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, *Lecture Notes in Computer Science*, vol. 9134, pp. 180–192. Springer Verlag (2015)
10. Biere, A., Lonsing, F.: Integrating dependency schemes in search-based QBF solvers. In: O. Strichman, S. Szeider (eds.) Theory and Applications of Satisfiability Testing - SAT 2010, *Lecture Notes in Computer Science*, vol. 6175, pp. 158–171. Springer Verlag (2010)
11. Biere, A., Lonsing, F., Seidl, M.: Blocked clause elimination for QBF. In: N. Bjørner, V. Sofronie-Stokkermans (eds.) International Conference on Automated Deduction - CADE 23, *Lecture Notes in Computer Science*, vol. 6803, pp. 101–115. Springer Verlag (2011)
12. Bloem, R., Könighofer, R., Seidl, M.: SAT-based synthesis methods for safety specs. In: K.L. McMillan, X. Rival (eds.) Verification, Model Checking, and Abstract Interpretation - VMCAI 2014, *Lecture Notes in Computer Science*, vol. 8318, pp. 1–20. Springer Verlag (2014)
13. Bubeck, U.: Model-based transformations for quantified Boolean formulas. Ph.D. thesis, University of Paderborn (2010)
14. Cadoli, M., Schaerf, M., Giovanardi, A., Giovanardi, M.: An algorithm to evaluate Quantified Boolean Formulae and its experimental evaluation. Journal of Automated Reasoning **28**(2) (2002)
15. Cashmore, M., Fox, M., Giunchiglia, E.: Partially grounded planning as Quantified Boolean Formula. In: D. Borrajo, S. Kambhampati, A. Oddi, S. Fratini (eds.) 23rd International Conference on Automated Planning and Scheduling, ICAPS 2013. AAAI (2013)
16. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM **5**, 394–397 (1962)
17. Egly, U.: On sequent systems and resolution for QBFs. In: A. Cimatti, R. Sebastiani (eds.) Theory and Applications of Satisfiability Testing - SAT 2012, *Lecture Notes in Computer Science*, vol. 7317, pp. 100–113. Springer Verlag (2012)
18. Egly, U., Lonsing, F., Widl, M.: Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In: K.L. McMillan, A. Middeldorp, A. Voronkov (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - LPAR 2013, *Lecture Notes in Computer Science*, vol. 8312, pp. 291–308. Springer Verlag (2013)
19. Giunchiglia, E., Narizzano, M., Tacchella, A.: Clause/term resolution and learning in the evaluation of Quantified Boolean Formulas. J. Artif. Intell. Res. **26**, 371–416 (2006)
20. Goultiaeva, A., Bacchus, F.: Exploiting QBF duality on a circuit representation. In: M. Fox, D. Poole (eds.) Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010. AAAI Press (2010)
21. Goultiaeva, A., Seidl, M., Biere, A.: Bridging the gap between dual propagation and CNF-based QBF solving. In: E. Macii (ed.) Design, Automation and Test in Europe, DATE 13, pp. 811–814. EDA Consortium San Jose, CA, USA / ACM DL (2013)
22. Goultiaeva, A., Van Gelder, A., Bacchus, F.: A uniform approach for generating proofs and strategies for both true and false QBF formulas. In: T. Walsh (ed.) Proceedings of IJCAI 2011, pp. 546–553. IJCAI/AAAI (2011)
23. Heule, M., Seidl, M., Biere, A.: A unified proof system for QBF preprocessing. In: S. Demri, D. Kapur, C. Weidenbach (eds.) Automated Reasoning - 7th International Joint Conference, IJCAR 2014, *Lecture Notes in Computer Science*, vol. 8562, pp. 91–106. Springer Verlag (2014)
24. Janota, M., Klieber, W., Marques-Silva, J., Clarke, E.M.: Solving QBF with counterexample guided refinement. In: A. Cimatti, R. Sebastiani (eds.) Theory and Applications of Satisfiability Testing - SAT 2012, *Lecture Notes in Computer Science*, vol. 7317, pp. 114–128. Springer Verlag (2012)
25. Janota, M., Marques-Silva, J.: On propositional QBF expansions and Q-resolution. In: M. Järvisalo, A. Van Gelder (eds.) Theory and Applications of Satisfiability Testing - SAT 2013, *Lecture Notes in Computer Science*, vol. 7962, pp. 67–82. Springer Verlag (2013)

26. Kleine Büning, H., Karpinski, M., Flögel, A.: Resolution for quantified Boolean formulas. Information and Computation **117**(1), 12–18 (1995)
27. Klieber, W., Sapra, S., Gao, S., Clarke, E.M.: A non-prenex, non-clausal QBF solver with game-state learning. In: O. Strichman, S. Szeider (eds.) Theory and Applications of Satisfiability Testing - SAT 2010, *Lecture Notes in Computer Science*, vol. 6175, pp. 128–142. Springer Verlag (2010)
28. Kronegger, M., Pfandler, A., Pichler, R.: Conformant planning as benchmark for QBF-solvers. In: International Workshop on Quantified Boolean Formulas - QBF 2013 (2013). `http://fmv.jku.at/qbf2013/`
29. Lonsing, F.: Dependency schemes and search-based QBF solving: Theory and practice. Ph.D. thesis, Johannes Kepler University, Linz, Austria (2012)
30. Lonsing, F., Bacchus, F., Biere, A., Egly, U., Seidl, M.: Enhancing search-based QBF solving by dynamic blocked clause elimination. In: M. Davis, A. Fehnker, A. McIver, A. Voronkov (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, *Lecture Notes in Computer Science*, vol. 9450, pp. 418–433. Springer Verlag (2015)
31. Lonsing, F., Egly, U., Van Gelder, A.: Efficient clause learning for quantified Boolean formulas via QBF pseudo unit propagation. In: M. Järvisalo, A. Van Gelder (eds.) Theory and Applications of Satisfiability Testing - SAT 2013, *Lecture Notes in Computer Science*, vol. 7962, pp. 100–115. Springer Verlag (2013)
32. Marques-Silva, J.P.: The impact of branching heuristics in propositional satisfiability algorithms. In: P. Barahona, J.J. Alferes (eds.) Progress in Artificial Intelligence, 9th Portuguese Conference on Artificial Intelligence, EPIA '99, *Lecture Notes in Computer Science*, vol. 1695, pp. 62–74. Springer Verlag (1999)
33. Niemetz, A., Preiner, M., Lonsing, F., Seidl, M., Biere, A.: Resolution-based certificate extraction for QBF. In: A. Cimatti, R. Sebastiani (eds.) Theory and Applications of Satisfiability Testing - SAT 2012, *Lecture Notes in Computer Science*, vol. 7317, pp. 430–435. Springer Verlag (2012)
34. Peitl, T., Slivovsky, F., Szeider, S.: Long distance Q-resolution with dependency schemes. In: N. Creignou, D.L. Berre (eds.) Theory and Applications of Satisfiability Testing - SAT 2016, *Lecture Notes in Computer Science*, vol. 9710, pp. 500–518. Springer Verlag (2016)
35. Rintanen, J.: Asymptotically optimal encodings of conformant planning in QBF. In: 22nd AAAI Conference on Artificial Intelligence, AAAI 2007, pp. 1045–1050. AAAI (2007)
36. Samer, M., Szeider, S.: Backdoor sets of quantified Boolean formulas. Journal of Automated Reasoning **42**(1), 77–97 (2009)
37. Slivovsky, F.: Structure in #SAT and QBF. Ph.D. thesis, TU Wien (2015)
38. Slivovsky, F., Szeider, S.: Computing resolution-path dependencies in linear time. In: A. Cimatti, R. Sebastiani (eds.) Theory and Applications of Satisfiability Testing - SAT 2012, *Lecture Notes in Computer Science*, vol. 7317, pp. 58–71. Springer Verlag (2012)
39. Slivovsky, F., Szeider, S.: Soundness of Q-resolution with dependency schemes. Theoretical Computer Science **612**, 83–101 (2016)
40. Staber, S., Bloem, R.: Fault localization and correction with QBF. In: J. Marques-Silva, K.A. Sakallah (eds.) Theory and Applications of Satisfiability Testing - SAT 2007, *Lecture Notes in Computer Science*, vol. 4501, pp. 355–368. Springer Verlag (2007)
41. Van Gelder, A.: Variable independence and resolution paths for quantified Boolean formulas. In: J. Lee (ed.) Principles and Practice of Constraint Programming - CP 2011, *Lecture Notes in Computer Science*, vol. 6876, pp. 789–803. Springer Verlag (2011)
42. Zhang, L., Malik, S.: Conflict driven learning in a quantified Boolean satisfiability solver. In: L.T. Pileggi, A. Kuehlmann (eds.) Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design, ICCAD 2002, pp. 442–449. ACM / IEEE Computer Society (2002)
43. Zhang, L., Malik, S.: Towards a symmetric treatment of satisfaction and conflicts in quantified Boolean formula evaluation. In: P.V. Hentenryck (ed.) Principles and Practice of Constraint Programming - 8th International Conference, CP 2002, *Lecture Notes in Computer Science*, vol. 2470, pp. 200–215. Springer Verlag (2002)