

On the Interplay of Cube Learning and Dependency Schemes in QCDCL Proof Systems

Abhimanyu Choudhury ✉ 

The Institute of Mathematical Sciences, Chennai, India

Homi Bhabha National Institute, Training School Complex, Anushaktinagar, Mumbai, India

Meena Mahajan ✉ 

The Institute of Mathematical Sciences, Chennai, India

Homi Bhabha National Institute, Training School Complex, Anushaktinagar, Mumbai, India

Abstract

Quantified Conflict Driven Clause Learning (QCDCL) is one of the main approaches to solving Quantified Boolean Formulas (QBF). Cube-learning is employed in this approach to ensure that true formulas can be verified. Dependency Schemes help to detect spurious dependencies that are implied by the variable ordering in the quantifier prefix of QBFs but are not essential for constructing (counter)models. This detection can provably shorten refutations in specific proof systems, and is expected to speed up runs of QBF solvers.

The simplest underlying proof system [BeyersdorffBöhm-LMCS2023], formalises the reasoning in the QCDCL approach on false formulas, when neither cube-learning nor dependency schemes is used. The work of [BöhmPeitlBeyersdorff-AI2024] further incorporates cube-learning. The work of [ChoudhuryMahajan-JAR2024] incorporates a limited use of dependency schemes, but without cube-learning.

In this work, proof systems underlying the reasoning of QCDCL solvers which use cube learning, and which use dependency schemes at all stages, are formalised. Sufficient conditions for soundness and completeness are presented, and it is shown that using the standard and reflexive resolution path dependency schemes (D^{std} and D^{rrs}) to relax the decision order provably shortens refutations.

When the decisions are restricted to follow quantification order, but dependency schemes are used in propagation and learning, in conjunction with cube-learning, the resulting proof systems using the dependency schemes D^{std} and D^{rrs} are investigated in detail and their relative strengths are analysed.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases QBF, CDCL, Resolution, Dependency schemes

Funding *Meena Mahajan*: Partially supported by J C Bose National Fellowship.

1 Introduction

Despite the NP-hardness of the satisfiability problem, in the last three decades SAT solvers have been phenomenally successful in solving instances of humongous size, and have become the go-to tool in many practical industrial applications (see e.g. [32, 24]). This success has spurred ambitious programs to develop solvers for computationally even more hard problems. In particular, the PSPACE-complete problem of determining the truth of Quantified Boolean Formulas QBFs has many more applications (see e.g. [29]), and over the last twenty years QBF solvers have rapidly approached the state of industrial applicability.

The paradigm that revolutionized SAT solving is Conflict Driven Clause Learning CDCL ([30]), and this is also one of the principal approaches (but not the only one) in QBF solving. The CDCL technique was lifted to QBFs in the form of QCDCL ([33], see also [19]; in [21], the term QDPLL is used), and implemented in state-of-the-art solvers DepQBF [22, 23] and Qute [25] with further augmentations to enhance performance.

For both SAT and QBF, solving techniques are intricately connected with proof systems. The runtime trace of a solver on a formula can be thought of as a proof of the final outcome (sat/unsat, true/false). Proof systems abstract out the reasoning employed in the solvers, and allow representing these traces-as-proofs as formal proofs. The CDCL paradigm in SAT solvers corresponds to resolution, a very well-studied proof system. There are multiple ways in which resolution can be lifted to QBFs, see [5] for an overview. As shown in [4], resolution proofs can be efficiently extracted from traces of CDCL-based SAT solvers. For QBFs, QCDCL traces yield proofs in the proof system long-distance Q-resolution LDQ-Res [33, 3]. However, the converse direction, going from resolution proofs to CDCL runs, famously shown for SAT in [27, 1], seemingly breaks down for QBF and QCDCL as currently implemented; the reasoning employed in basic QCDCL solvers was abstracted in [7, 15] as the proof systems QCDCL and $\text{QCDCL}^{\text{cube}}$, and shown to be exponentially weaker than LDQ-Res .

The proof system QCDCL is a refutational proof system; it was formulated in [7] to explain the reasoning of basic QCDCL-style algorithms on false QBFs. The proof system $\text{QCDCL}^{\text{cube}}$, defined in [15], incorporates cube-learning as well, and can thus certify both falsity and truth of QBFs. Intriguingly, it was shown in [15] that even for false QBFs, where cube-learning is not necessary for completeness, it can still significantly shorten refutations. Very recently, it was shown in [8] that even when short refutations are actually found, it may take an exponentially long time to find them. Many other variants (different policies for decision order, propagations, reductions) have been studied extensively in [14].

One heuristic that has been used in some QCDCL solvers is the use of dependency schemes. These schemes involve performing some basic analysis of the formula structure and identifying spurious dependencies amongst variables, dependencies that are implied by the quantification order of variables but are not necessary for constructing (counter)models; see for instance [31]. Eliminating such dependencies would transform a QBF to a Dependency QBF, DQBF, for which the computational problem of deciding truth/falsity is even harder; it is NEXP-complete ([2, 12]). However, retaining the formulas as a QBF, and using information about spurious dependencies while propagating and learning, is still a feasible approach, that has been implemented in the solver DepQBF [22, 23] using the *standard dependency scheme* D^{std} . In resolution-based QBF proof systems, employing reduction rules based on the *reflexive resolution path dependency-scheme* D^{rrs} , is known to exponentially shorten refutations ([10]), and the expectation is that a similar advantage also manifests in QCDCL solvers.

This work makes progress towards formally understanding the strengths/limitations of using the dependency-scheme heuristic. The first steps in this direction were initiated in a recent work in [17]. It considered the simplest setting, in which the QCDCL proof system uses the LEV-ORD decision policy (deciding variables according to the quantification order), and does not learn cubes. A dependency scheme D is used in propagation by, and learning of, clauses. Additionally, a dependency scheme D' may be used to preprocess the formula, reducing all clauses according to D' before beginning the QCDCL trails. In this setting, when D and D' are “normal” schemes (as defined in [26]), the resulting proof systems were shown to be sound and refutationally complete. In the same setting, the four systems arising from using D^{rrs} in preprocessing, in propagation/learning, in both, and in neither, were shown to be incomparable with each other. In the underlying proof system LDQ-Res , using dependency information can never lengthen proofs. The handicap in QCDCL arises because QCDCL algorithms also need to search for the proof.

In this work, we consider more general settings. Our contributions are as follows:

Formalising intensive use of dependency schemes in QCDCL: We formalise the definitions of QCDCL and $\text{QCDCL}^{\text{cube}}$ proof systems that use dependency schemes more intensively:

in the decision policy, which determines which variables can be "decided" at a particular stage, as well as in propagation and learning, with and without cube learning. Using a dependency scheme D_1 in the decision policy means that a variable can be decided if all variables on which it depends, according to D_1 , are already assigned; this is the policy D_1 -ORD. Using a dependency scheme D_2 in propagation and learning means that reductions enabled by D_2 are performed whenever possible. For two dependency schemes D_1 and D_2 (which may be the same) we consider $\text{QCDCL}^{\text{cube}}$ proof systems that use D_1 in the decision policy and D_2 in propagation through and learning of clauses. We consider three scenarios with respect to cube-learning: (1) cube-learning is switched off completely; (2) cube propagation and learning is done without using any dependency schemes; or (3) cube propagation and learning use the scheme D_2 but disallow long-distance term-resolution. The reason for this difference between clause and cube learning is that long-distance term resolution is not (yet?) known to be sound if used in conjunction with dependency schemes. We show that for normal D_1 , D_2 , the resulting systems are sound and refutationally complete; Theorem 3.13.

Provable advantage of D-ORD: We show that other parameters remaining the same, using either D^{rrs} or D^{std} as D_1 is strictly better than using LEV-ORD; Theorem 3.14.

Using cube-learning, and dependency schemes only in propagation/learning: When the decision policy is restricted to LEV-ORD, we generalise the results from [17] to settings with cube-learning switched on, and also to settings where D^{std} rather than D^{rrs} is used. Specifically, we show that

1. Using D^{std} in pre-processing is useless; Proposition 4.2.
2. Switching on cube learning provably adds power even in the presence of D^{std} or D^{rrs} ; Theorem 4.16.
3. Adding D^{rrs} in various non-trivial ways to $\text{QCDCL}^{\text{cube}}$ results in proof systems that are not only pairwise incomparable, Theorem 4.18, but are also incomparable with both $\text{QCDCL}^{\text{cube}}$ and QCDCL ; Theorems 4.17 and 4.19.
4. Adding D^{std} to QCDCL is orthogonal to switching on cube learning; Theorem 4.20. In certain cases, adding D^{std} to both QCDCL and $\text{QCDCL}^{\text{cube}}$ offers a provable advantage.
5. Although D^{rrs} strictly refines D^{std} , in the context of QCDCL and $\text{QCDCL}^{\text{cube}}$, adding these schemes gives rise to incomparable systems; Theorem 4.21. Thus, the LEV-ORD policy can negate potential benefits of the strict refinement.

We use several known bounds on formulas from earlier works, and also show some new bounds for them. To obtain desired separations, we also introduce three carefully handcrafted new formulas. For easy reference, the known and new results (about previously defined and new formulas) are collated in Table 1 in Section 4. The known simulation order of the proof systems, incorporating prior known results as well as the new results proved here, are summarised in Figure 1, also in Section 4.

Organisation of the paper:

In Section 2, we give some basic definitions and describe the background about known proof systems and dependency schemes. In Section 3, we define the new proof systems, show soundness and completeness, and show that the decision policy D-ORD is strictly more powerful than LEV-ORD. In Section 4, we briefly discuss preprocessing, we define three new QBF families and show various lower and upper bounds for their proof sizes, and we describe the simulation order among various QCDCL systems. We conclude with some pointers for further directions of interest.

2 Preliminaries

2.1 Basic Notation

We follow notation from [7, 15]; see also [5]. Selected relevant items are included here.

A literal ℓ is a Boolean variable x or its negation $\bar{x}$, and $\text{var}(\ell)$ denotes the associated variable x . A clause is a disjunction of literals; a term or a cube is a conjunction of literals. For a clause or cube C , $\text{var}(C)$ denotes the set $\{\text{var}(\ell) \mid \ell \in C\}$. A propositional formula φ is built from variables using conjunction, disjunction, and negation; it is in conjunctive normal form (CNF) if it is a conjunction of clauses. For a formula φ , a variable x in φ , and a Boolean value a , $\varphi|_{x=a}$ refers to the formula obtained by substituting $x = a$ everywhere in φ . For a set S of clauses and a literal ℓ , we use shorthand $\ell \vee S$ to denote the set of clauses $\{\ell \vee C \mid C \in S\}$. The empty clause is denoted $\square$ and is unsatisfiable; the empty cube is denoted $\top$ and is always true. A clause (cube) is said to be tautological (contradictory) if for some variable x it contains both x and $\bar{x}$.

The resolution rule can be applied to clauses and to cubes. The resolvent of clauses $A' = A \vee x$ and $B' = B \vee \bar{x}$ is the clause $A \vee B$ denoted as $\text{res}(A', B', x)$ or $\text{res}(B', A', x)$. The resolvent of cubes $A' = A \wedge x$ and $B' = B \wedge \bar{x}$ is the cube $A \wedge B$, also denoted as $\text{res}(A', B', x)$ or $\text{res}(B', A', x)$.

A Quantified Boolean Formula (QBF) in prenex conjunction normal form (PCNF) is a prefix with a list of variables, each quantified either existentially or universally, and a matrix, which is a set (conjunction) of clauses over these variables. That is, it has the form

$$\Phi = Q\vec{x} \cdot \varphi = Q_1 x_1 Q_2 x_2 \dots Q_n x_n \varphi(x_1, x_2, \dots, x_n)$$

where φ is a propositional formula in CNF, and each Q_i is in $\{\exists, \forall\}$. We denote by $X_\exists$ ($X_\forall$ respectively) the set of all variables quantified existentially (resp. universally).

A QBF is true if for each existentially quantified variable x_i , there exists a (Skolem) function s_i , depending only on universally quantified variables x_j with $j < i$, such that substituting these s_i in φ yields a tautology. Similarly, the formula is false if for each universally quantified variable u_i , there is a (Herbrand) function h_i , depending only on existentially quantified variables x_j with $j < i$, such that substituting h_i in φ yields an unsatisfiable formula.

2.2 Some QBF proof systems, and the Dependency Scheme heuristic

The propositional proof system Resolution certifies unsatisfiability of a propositional formula by adding clauses obtaining through resolution until the empty clause is added. This can be lifted to QBFs in many ways. One of the simplest ways is to use the resolution rule along with a universal reduction rule, that allows removing a universal literal u or $\bar{u}$ from a clause if it is not ‘*blocked*’; that is, the clause has no existential literals quantified after u in the prefix. This gives rise to the system **QU-Res**; its restriction where resolution is allowed only on existential pivots is the system **Q-Res**. The long-distance resolution rule generalises resolution by permitting seemingly useless universal tautological clauses under certain side-conditions, and gives rise to the system **LDQ-Res** that generalises **Q-Res**. Informally, in this system, a resolution on x is permitted even if the resolvent ends up having u and $\bar{u}$ for some universal variable u , provided u is quantified to the right of x . The presence of u and $\bar{u}$ together, often referred to as a *merged literal* u^* , is to be interpreted not as a tautology but as a place-holder for a partial strategy for u depending on the value of the pivot x .

In direct analogy to **Q-Res** and **LDQ-Res** are the proof systems **Q-TermRes** and **LDQ-TermRes** for certifying truth. Here the resolution is performed on terms, or cubes, with universal pivots, and existential literals can be reduced from a term if not blocked by universal literals quantified after them. The goal is to derive the empty term. A key point of difference is at the start; since the QBF is in PCNF, there are no terms to begin with. The Axiom rule in these systems permits starting with any term that satisfies the matrix.

For formal definitions of these proof systems, see for instance, Figure 2 in [9] (for **Q-Res**, **QU-Res**, **LDQ-Res**), Figure 2, in [31] (for **Q-TermRes**). To help readability, we also include the definitions of the rules in the appendix.

The notion of blocking, used in the reduction rules, stems from the understanding that if variables x, y are quantified differently with y quantified after x , then the value of y in a (counter)model may non-trivially *depend* on x . If there is no literal blocking x , then the satisfaction of the clause (falsification of cube) should not rely on the unblocked universal (resp. existential) x . The dependency scheme heuristic refines this further. If a syntactic examination of the clause-variable structure can reveal that y does not really depend on x , *even though it is quantified later*, then x can be reduced even in the presence of y . This can drive the process towards the empty clause/term faster. Dependency schemes do precisely this. They identify pairs (x, y) where x and y are quantified differently and where y can be safely assumed to be independent of x . (Actually, they list pairs where y may depend on x ; the other pairs can be assumed to be independent.) The trivial dependency scheme associates with each QBF Φ the dependency set $D^{\text{trv}}(\Phi) = \{(x, y) \mid y \text{ is quantified after, and differently from, } x\}$. Other schemes can associate subsets of this set. The schemes relevant to this paper are the standard scheme D^{std} (Def 7 in [28] and Def 9 in [31]), and the reflexive resolution scheme D^{rrs} (Defs 3,4,6 in [31]); these definitions are reproduced below. For every Φ , $D^{\text{trv}}(\Phi) \supseteq D^{\text{std}}(\Phi) \supseteq D^{\text{rrs}}(\Phi)$.

► **Definition 2.1** (Standard Dependency Scheme, [31, Def 9]). *For a PCNF QBF $\Phi = Q\vec{x} \cdot \varphi$, the pair (x, y) is in $D^{\text{std}}(\Phi)$ if and only if $(x, y) \in D^{\text{trv}}(\Phi)$ and there exists a sequence of clauses $C_1, \dots, C_n \in \varphi$ and a sequence of existential literals $\ell_1, \dots, \ell_{n-1}$ such that:*

- $x \in C_1$ and $y \in C_n$, and
- for each $i \in [n-1]$, $(x, \text{var}(\ell_i)) \in D^{\text{trv}}(\Phi)$, $\text{var}(\ell_i) \in \text{var}(C_i)$, and $\text{var}(\ell_i) \in \text{var}(C_{i+1})$.

► **Definition 2.2** (Reflexive Resolution Path Dependency Scheme, [31, Defs 3,4,6]). *Fix any PCNF QBF $\Phi = Q\vec{x} \cdot \varphi$.*

An ordered pair of literals ℓ_1, ℓ_{2k} is connected (via a resolution path) if there is a sequence of clauses $C_1, \dots, C_k \in \varphi$ and a sequence of existential literals $\ell_2, \dots, \ell_{2k-1}$ such that:

- $\ell_1 \in C_1$ and $\ell_{2k} \in C_k$,
- For each $i \in [k-1]$, $\ell_{2i} = \neg \ell_{2i+1}$.
- For each $i \in [k-1]$, $(\text{var}(\ell_1), \text{var}(\ell_{2i})) \in D^{\text{trv}}$.
- For each $i \in [k]$, $\text{var}(\ell_{2i-1}) \neq \text{var}(\ell_{2i})$.
- For each $i \in [k]$, $\ell_{2i-1}, \ell_{2i} \in C_i$.

An ordered pair of variables (x, y) is a resolution-path dependency pair if both (x, y) and $(\neg x, \neg y)$ are connected, or if both $(x, \neg y)$ and $(\neg x, y)$ are connected.

$$D^{\text{rrs}}(\Phi) = \{(x, y) \mid (x, y) \in D^{\text{trv}}; (x, y) \text{ is a resolution-path dependency pair}\}.$$

Roughly, $(x, y) \in D^{\text{std}}(\Phi)$ if there is a sequence of clauses with the first containing x or $\bar{x}$, the last containing y or $\bar{y}$, and each pair of consecutive clauses containing an existential variable quantified to the right of x . For D^{rrs} scheme, $(x, y) \in D^{\text{rrs}}(\Phi)$ if (x, y) and $(\bar{x}, \bar{y})$

or $(x, \bar{y})$ and $(\bar{x}, y)$ are connected by a sequence of clauses, each pair of consecutive clauses containing an existential variable in opposite polarities quantified to the right of x .

When a dependency scheme D is incorporated into any of the preceding proof systems, the reduction rule becomes more generally applicable, and the side-conditions concerning merged literals also become more permissive. This gives rise to the proof systems $\mathbf{Q}(D)\text{-Res}$, $\mathbf{LDQ}(D)\text{-Res}$, $\mathbf{Q}(D)\text{-TermRes}$, $\mathbf{LDQ}(D)\text{-TermRes}$. See, for instance, Figure 3 and Section 3.3 in [31] (for $\mathbf{Q}\text{-TermRes}$ and D -reductions), and Figure 1 in [26] (for $\mathbf{LDQ}(D)\text{-Res}$). For a clause and a dependency scheme D , we denote by $\mathbf{red}\text{-}D(C)$ the clause obtained by removing all unblocked universal literals from C . Similarly, for a cube C , $\mathbf{red}\text{-}D_{\exists}(C)$ denotes the cube obtained by removing all unblocked existential literals from C . We denote by $\mathbf{red}\text{-}D(\Phi)$ the QBF Ψ obtained by replacing each clause C in the matrix of Φ with the clause $\mathbf{red}\text{-}D(C)$. When $D = D^{\text{trv}}$, we use the notation $\mathbf{red}(C)$ and $\mathbf{red}(\Phi)$.

An important subclass of dependency schemes are the so-called *normal* dependency schemes, which have the property of being "monotone" and "simple". See Def. 7 in [26] for the precise definition. (Though we will not need the precise definitions, for completeness, we include the definition of normal dependency schemes in the appendix.) The dependency schemes D^{trv} , D^{std} , D^{rrs} are all normal. This class of schemes is of interest to us because it is known that for normal dependency schemes, $\mathbf{Q}(D)\text{-Res}$ and $\mathbf{LDQ}(D)\text{-Res}$ are sound and refutationally complete [26]. The system $\mathbf{Q}(D)\text{-TermRes}$ is known to be sound and complete on true formulas for $D \in \{D^{\text{trv}}, D^{\text{rrs}}\}$ (in [31], soundness is shown for a stronger dependency scheme, D^{res} , implying soundness for D^{rrs} as well). However the soundness of $\mathbf{LDQ}(D)\text{-TermRes}$ is not known for dependency schemes other than D^{trv} .

We say proof system P_1 simulates a proof system P_2 if some computable function transforms proofs in P_2 into proofs in P_1 with at most polynomial blow-up in proof size. If this function is also computable in polynomial time (in the given proof size), we say that P_1 *p*-simulates P_2 . Two systems are said to be incomparable if neither simulates the other.

By definition $\mathbf{LDQ}(D)\text{-Res}$ *p*-simulates $\mathbf{LDQ}\text{-Res}$ and $\mathbf{Q}(D)\text{-Res}$, both of which *p*-simulate $\mathbf{Q}\text{-Res}$. Similarly, both $\mathbf{LDQ}\text{-TermRes}$ and $\mathbf{Q}(D)\text{-TermRes}$ *p*-simulate $\mathbf{Q}\text{-TermRes}$. It is also known that $\mathbf{Q}(D)\text{-Res}$ is exponentially stronger than $\mathbf{Q}\text{-Res}$ for $D = D^{\text{rrs}}$; [10, 11].

2.3 The proof systems QCDCL with and without cube learning

The proof system for QCDCL, as defined in [7], formalizes reasoning in QCDCL algorithms operating on false formulas without cube learning. These algorithms construct *trails* or partial assignments in a specific way – *decide* values of variables according to some policy, *propagate* values of existential variables that appear in clauses which become unit after restriction by the trail so far and by universal reduction (call such a clause the *antecedent* of the propagated literal) – trying to satisfy the matrix. If a conflict is reached, then the trail is inconsistent with any Skolem function. *Conflict analysis* is performed, and a new clause is *learned* and added to the matrix. If the empty clause is learned, the formula is deemed false. The corresponding refutation in the proof system QCDCL consists of the sequence of constructed trails, and for each trail the sequence of long-distance resolution steps performed in conflict analysis to learn a clause. The full definition can be found in [7] (Def. 3.5).

The above formulation of the QCDCL system only considers trails that end in a conflict. Trails ending in a satisfying assignment are ignored. This is enough to ensure refutational completeness – the ability to prove all false QBFs false. However, from satisfying assignments, solvers can and do learn cubes (or terms), and this is necessary to prove true QBFs true. In [15] it was shown that even while refuting false QBFs, allowing cube learning from satisfying assignments can be advantageous. This led to the definition of the proof system $\mathbf{QCDCL}^{\text{cube}}$,

and in [15], it was shown to be strictly stronger than the standard QCDCL system. The main idea in cube learning systems is to consider satisfying assignments also as conflicts, albeit of a different kind, and to learn cubes from these conflicts. (The algorithm learns that such a trail is inconsistent with any Herbrand function.) Learnt cubes are added disjunctively to the matrix, which thus at intermediate stages is not necessarily in CNF but is the disjunction of a CNF formula and some cubes. With the augmented CNF matrices, cubes that become unit after existential reduction can now propagate universal variables in a way that falsifies the cube. Also, with the augmented CNF matrices, a trail may end up satisfying a cube rather than the CNF; this too is now a conflict, and conflict analysis involving term-resolution can be performed to learn a new cube. For formal details, see Section 3 in [15].

Three factors affect the construction of a refutation or verification, and are relevant for our generalized definition in the following section: Three factors affect the construction of the refutation.

1. The decision policy: how to choose the next variable to branch on. In standard QCDCL as defined in [7], decisions must respect the quantifier prefix level order. (Variables x, y are at the same level if they are quantified the same way, and no variable with a different quantification appears between them in the prefix order.) This policy is called **LEV-ORD**. The most unrestricted policy is **ANY-ORD**; any variable can be decided at any point. Other policies such as **ANY-ORD**, **ASS-R-ORD**, **UNI-ANY**, are also possible; see [7, 16].
2. The unit propagation policy. Upon a partial assignment α to some variables, when does a clause C propagate a literal? In standard QCDCL the Reduction policy (used by most current QCDCL solvers [22, 25]) is used: a clause C propagates literal ℓ if after restricting C by α and applying all possible universal reductions, only ℓ remains. Also, propagations are made as soon as possible; see the description of natural trails (Def 3.4 in [7]).
3. The set of learnable clauses/cubes. These clauses/cubes explain the conflict at the end of a trail, and are derived using (possibly long-distance) clause/term resolution with propagated literals as pivots.

3 Adding dependency schemes to the QCDCL proof system

The work done in [17] was the first to formalise the addition of dependency schemes to the QCDCL proof system. It was done only for the setting where trails follow level-ordered decisions, and there is no cube learning. Here we relax both these restrictions; we allow dependency schemes to affect the decision policy of the trail, and we allow cube-learning.

3.1 Defining the D-ORD systems

Dependency schemes can be used in QCDCL algorithms in many ways:

(1) in specifying the decision order, (2) in specifying how reduction, propagation, and learning of clauses are performed, and (3) in specifying how these are performed for cubes, if at all. We set up unified notation to describe all such QCDCL-based proof systems.

► **Definition 3.1** (D-ORD). *For a dependency scheme D , the decision policy D-ORD permits a decision on a variable x at some point in a trail if all variables y on which x depends according to D (i.e. $(y, x) \in D$), have already been assigned.*

We define a new notation to describe QCDCL based proof systems introduced below with or without the option for cube learning.

► **Definition 3.2.** $\text{QCDCL}^{\text{ORD}}(\text{ClausePol}, \text{CubePol})$ is the QCDCL proof system where

1. **ORD** denotes the decision policy; e.g. **LEV-ORD**, **D'-ORD**, **ANY-ORD**.
In **D'-ORD**, **D'** denotes the dependency scheme, such as **D^{rrs}-ORD**, **D^{std}-ORD**.
2. **ClausePol** is the dependency scheme **D** used in reduction, propagation, and learning for clauses. Note that this scheme need not be the same as the **D'** in **D'-ORD**.
3. **CubePol** $\in \{\text{No-Cube}, \text{Cube-LD}, \text{Cube-D}\}$ denotes the type of usage of cube learning;
 - a. **No-Cube**: No cube learning.
 - b. **Cube-LD**: Cube Learning used, but no dependency scheme (only **D^{trv}**) in cube propagation, and cube learning using **LDQ-TermRes**.
 - c. **Cube-D**: Cube Learning used, dependency scheme **D** used in propagation, and cube learning is done using **Q(D)-TermRes**.

Note that in this notation, clause learning always uses **LDQ(D)-Res**, where **D** might well be **D^{trv}**. However, for cube learning, the propagation and learning can use either long-distance term resolution **LDQ(D)-TermRes**, or dependency schemes without long-distance **Q(D)-TermRes**, not both. We impose this condition because the soundness of **LDQ(D)-TermRes** is not known.

In the notation of Definition 3.2, the standard vanilla QCDCL system denoted **QCDCL_{RED}^{LEV-ORD}** in [7] would be **QCDCL^{LEV-ORD}(D^{trv}, No-Cube)**, whereas the **QCDCL^{cube}** system from [15] would be **QCDCL^{LEV-ORD}(D^{trv}, Cube-LD)**. Further, the dependency-based system **QCDCL(D^{rrs})** introduced in [17] would be **QCDCL^{LEV-ORD}(D^{rrs}, No-Cube)**.

To define what a derivation in these QCDCL systems must look like, we must first define trails and the learnable clauses and cubes from trails in this system. The following definitions are the natural generalisations of the corresponding ones from [7, 15]. We give a short illustration in Example 3.6 after Definition 3.5.

The trail is a sequence of literals (or $\square$, $\top$)

$$T = (p_{(0,1)}, \dots, p_{(0,g_0)}; \mathbf{d}_1, p_{(1,1)}, \dots, p_{(1,g_1)}; \mathbf{d}_2, \dots, \dots; \mathbf{d}_r, p_{(r,1)}, \dots, p_{(r,g_r)})$$

where the literals d_i (in boldface) are decision literals, the literals $p_{i,j}$ are propagated literals, and no opposing literals appear. We can also view it as a set of literals or an assignment, and the corresponding clause (cube) is the disjunction (conjunction) of all literals in it.

The learnable constraints from a trail are defined as follows:

► **Definition 3.3** (learning from conflict). *From a trail*

$$T = (p_{(0,1)}, \dots, p_{(0,g_0)}; \mathbf{d}_1, p_{(1,1)}, \dots, p_{(1,g_1)}; \mathbf{d}_2, \dots, \dots; \mathbf{d}_r, p_{(r,1)}, \dots, p_{(r,g_r)})$$

ending in a conflict $p_{(r,g_r)} \in \{\square, \top\}$, the set L_T of learnable constraints has a clause $C_{(i,j)}$ associated with each propagated literal $p_{(i,j)}$ propagated in the trail if $p_{(r,g_r)} = \square$, and a cube associated with each propagated literal in the trail if $p_{(r,g_r)} = \top$. These associated clauses/cubes are constructed by tracing the conflict backwards through the trail as follows. (**ante**(ℓ) denotes the clause/cube that causes literal ℓ to be propagated; i.e. the antecedent.) Starting with **ante**($\square$) (respectively **ante**($\top$)), we resolve in reverse over the antecedent clauses (cubes) that propagated the existential (universal) variables as described below. All such resulting clauses (cubes) are learnable constraints.

In particular, if $p_{(r,g_r)} = \square$, then

- $C_{(r,g_r)} = \text{red-D}(\text{ante}(p_{(r,g_r)}))$.
- For $i \in \{0, 1, \dots, r\}$ and $j \in [g_i - 1]$, if $\text{var}(p_{(i,j)}) \in X_\exists$ and $\bar{p}_{(i,j)} \in C_{(i,j+1)}$, then $C_{(i,j)}$ is the clause obtained by resolving the clause $C_{(i,j+1)}$ with the clause obtained from the antecedent of $p_{(i,j)}$ after reduction; such a resolution is possible on pivot $p_{(i,j)}$. Otherwise, $C_{(i,j)}$ is simply the same as $C_{(i,j+1)}$. Thus, $C_{(i,j)}$ equals $\text{red-D}[\text{res}(C_{(i,j+1)}, \text{red-D}(\text{ante}(p_{(i,j)})), p_{(i,j)})]$ if $\text{var}(p_{(i,j)}) \in X_\exists$ and $\bar{p}_{(i,j)} \in C_{(i,j+1)}$, and is $C_{(i,j+1)}$ otherwise.

- The learning process skips decision variables, so $C_{(i,g_i)}$ is defined using $p_{(i,g_i)}$ and $C_{(i+1,1)}$. For $i \in \{0, 1, \dots, r-1\}$, $C_{(i,g_i)}$ equals $\text{red-D}[\text{res}(C_{(i+1,1)}, \text{red-D}(\text{ante}(p_{(i,g_i)})), p_{(i,g_i)})]$ if $\text{var}(p_{(i,g_i)}) \in X_\exists$ and $\bar{p}_{(i,g_i)} \in C_{(i+1,1)}$, and is $C_{(i+1,1)}$ otherwise.

If $p_{(r,g_r)} = \top$, then non-trivial cube-resolution is performed when $p_{(i,j)}$ is universal, not existential. The set L_T depends on CubePol . If $\text{CubePol} = \text{Cube-LD}$, then red-D is replaced by $\text{red}_\exists$. If $\text{CubePol} = \text{Cube-D}$, then red-D is replaced by $\text{red-D}_\exists$, but the res step is performed only if it is a valid Q(D)-TermRes resolution step; otherwise we use the previously learnt cube, just as we do in clause learning when the resolution on $p_{(i,j)}$ is not defined.

► **Definition 3.4** (learning from satisfaction). From a trail T that assigns all variables, satisfies all clauses, and does not satisfy any cube, the set of learnable constraints is defined as follows: For any set L of literals, let t_L denote the cube that is the conjunction of all literals in L . Viewing the trail T as a set of literals,

$$L_T = \begin{cases} \{\text{red}_\exists(t_{T'}) \mid T' \subset T; T' \text{ satisfies all axioms and learnt clauses}\} & \text{if } \text{CubePol} = \text{Cube-LD} \\ \{\text{red-D}_\exists(t_{T'}) \mid T' \subset T; T' \text{ satisfies all axioms and learnt clauses}\} & \text{if } \text{CubePol} = \text{Cube-D}. \end{cases}$$

► **Definition 3.5.** For a specific choice of ORD , ClausePol , CubePol , let P be the proof system $\text{QCDCL}^{\text{ORD}}(\text{ClausePol}, \text{CubePol})$. A P -derivation ι from a PCNF QBF $\Phi = Q\vec{x} \cdot \varphi$ of a clause or cube C is a sequence ι of triples, $\iota = (T_1, C_1, \pi_1), \dots, (T_m, C_m, \pi_m)$, where each T_i is a trail, each C_i is a clause/cube, and $C_m = C$. The objects T_i, C_i, π_i are as defined below.

For each $i \in [m]$, φ_i is a propositional formula of the form $\varphi_i = (\bigwedge_{C \in \mathcal{C}_i} C) \vee (\bigvee_{T \in \mathcal{T}_i} T)$, where $\mathcal{C}_i$ is a set of clauses and $\mathcal{T}_i$ is a set of cubes. These formulas are defined iteratively; initially we have all the clauses of φ and no terms, and after each trail either a clause or a term is learnt and added. Formally,

$$\begin{aligned} \mathcal{C}_1 &= \{C \mid C \in \varphi\}, & \mathcal{T}_1 &= \emptyset \\ \text{If } C_i \text{ is a clause: } \mathcal{C}_{i+1} &= \mathcal{C}_i \cup \{C_i\}, & \mathcal{T}_{i+1} &= \mathcal{T}_i. \\ \text{If } C_i \text{ is a cube: } \mathcal{C}_{i+1} &= \mathcal{C}_i, & \mathcal{T}_{i+1} &= \mathcal{T}_i \cup \{C_i\}. \end{aligned}$$

For each $i \in [m]$, Φ_i is the QBF with the same quantifier prefix as Φ , and inner formula φ_i . For each $i \in [m]$, T_i is a trail from the formula Φ_i , C_i is a learnable clause or cube from T_i , and π_i is the derivation of C_i from Φ in the system as per ClausePol or CubePol .

A refutation in these systems is a derivation of the empty clause $\square$, and a verification in this system is a derivation of the empty term $\top$.

► **Example 3.6.** The TwoPHPandCT_n formulas, defined in [17](Section 4.5), have the prefix $Q = \forall u \exists x_1 \dots x_{s_n} \exists y_1 \dots y_{s_n} \forall v \exists z_1, z_2$ and the matrix

$$\begin{aligned} &u \vee \text{PHP}_n(x_1, \dots, x_{s_n}) \quad \bar{u} \vee \text{PHP}(y_1, \dots, y_{s_n}) \\ &v \vee z_1 \vee z_2, \quad v \vee \bar{z}_1 \vee z_2, \quad v \vee z_1 \vee \bar{z}_2, \quad v \vee \bar{z}_1 \vee \bar{z}_2 \end{aligned}$$

Here PHP_n refers to the propositional Pigeon-hole-principle formulas that assert the existence of a map from $n+1$ pigeons to n holes without collision; these formulas are known to be exponentially hard for resolution. Due to PHP_n , the matrix is unsatisfiable, and thus cube-learning makes no difference. So we consider **No-Cube**.

Consider refutations using D-ORD , with $\text{D} = \text{D}^{\text{rrs}}$ or D^{std} . For these formulas, it can be seen that $\text{D}^{\text{rrs}} = \emptyset$ and $\text{D}^{\text{std}} = \{(u, x_i), (u, y_i) : \text{for all } i\} \cup \{(v, z_1), (v, z_2)\}$. The v, z_1, z_2 variables are completely independent from the u, x, y variables; neither (x_i, v) nor (y_i, v) is in D for any i . Therefore using the D-ORD decision policy, we can decide v or z_i in the beginning. Consider the trail that decides to assign v to false and then z_1 to true. The clause $v \vee \bar{z}_1 \vee z_2$ becomes unit, so z_2 is propagated. Then the clause $v \vee \bar{z}_1 \vee \bar{z}_2$ becomes empty, and the empty clause is

propagated, leading to conflict. That is, in $T_1 = \bar{v}; z_1, z_2, \square$, we have $\text{ante}(\square) = v \vee \bar{z}_1 \vee \bar{z}_2$, and $\text{ante}(z_2) = v \vee \bar{z}_1 \vee z_2$. In conflict analysis, we first resolve (the reduced version of) $\text{ante}(\square)$ with $\text{ante}(z_2)$ to obtain $v \vee \bar{z}_1$.

If D^{rrs} is used in learning, then this can be reduced further to $\bar{z}_1$, which is learnt, i.e. included in C_2 and Φ_2 . The next trail then begins with the propagated literal $\bar{z}_1$, and propagates further; $T_2 = \bar{v}, \bar{z}_1, z_2, \square$, and $\text{ante}(\square) = v \vee z_1 \vee \bar{z}_2$, $\text{ante}(z_2) = v \vee z_1 \vee z_2$, $\text{ante}(\bar{z}_1) = \bar{z}_1$, allowing us to learn $\square$. This is a refutation in $\text{QCDCL}^{D^{\text{rrs}}}(\bar{D}^{\text{rrs}}, \text{No-Cube})$ or $\text{QCDCL}^{D^{\text{std}}}(\bar{D}^{\text{rrs}}, \text{No-Cube})$, but not in $\text{QCDCL}^{\text{LEV-ORD}}(\bar{D}^{\text{rrs}}, \text{No-Cube})$.

If D^{std} or D^{trv} is used in learning, then from trail T_1 the clause $v \vee \bar{z}_1$ is learnt (i.e. included in C_2 and Φ_2) since it cannot be further reduced. The next trail must again begin with a decision. With $T_2 = \bar{v}, \bar{z}_1, z_2, \square$, $\text{ante}(\square) = v \vee z_1 \vee \bar{z}_2$, $\text{ante}(z_2) = v \vee z_1 \vee z_2$, $\text{ante}(\bar{z}_1) = v \vee \bar{z}_1$, allowing us to learn $\square$. This is a refutation in $\text{QCDCL}^{D_1}(D_2, \text{No-Cube})$, where D_1 could be D^{rrs} or D^{std} but not D^{trv} , and D_2 could be D^{std} or D^{trv} . ◀

By definition, D-ORD generalises LEV-ORD, and ANY-ORD generalises D-ORD. Thus,

► **Observation 3.7.** *For a $\text{CubePol} \in \{\text{No-Cube}, \text{Cube-LD}, \text{Cube-D}\}$, and dependency schemes D, D' ,*

- *Every derivation in $\text{QCDCL}^{\text{LEV-ORD}}(D, \text{CubePol})$ is a derivation in $\text{QCDCL}^{D'-\text{ORD}}(D, \text{CubePol})$.*
- *Every derivation in $\text{QCDCL}^{D'-\text{ORD}}(D, \text{CubePol})$ is a derivation in $\text{QCDCL}^{\text{ANY-ORD}}(D, \text{CubePol})$.*

Trails in a system without cube learning are also trails in the corresponding system with cube learning. Hence:

► **Observation 3.8.** *For $\text{CubePol} \in \{\text{Cube-LD}, \text{Cube-D}\}$, and for any decision policy ORD, any derivation in $\text{QCDCL}^{\text{ORD}}(D, \text{No-Cube})$ is also a derivation in $\text{QCDCL}^{\text{ORD}}(D, \text{CubePol})$.*

If the matrix of the given PCNF formula is *unsatisfiable*, then no satisfying trail can ever be constructed, no matter what policy is used, so no "cube learning" can happen. Hence:

► **Observation 3.9.** *For $\text{CubePol} \in \{\text{Cube-LD}, \text{Cube-D}\}$, and for any decision policy ORD, if a PCNF formula Φ has an unsatisfiable matrix, then any derivation from Φ in the system $\text{QCDCL}^{\text{ORD}}(D, \text{CubePol})$ is also a derivation in $\text{QCDCL}^{\text{ORD}}(D, \text{No-Cube})$.*

The following result is shown in [17].

► **Proposition 3.10** (Theorem 1 in [17]). *For any normal dependency scheme D , the system $\text{QCDCL}^{\text{LEV-ORD}}(D, \text{No-Cube})$ is refutationally complete.*

A minor adaptation of the proof of Theorem 3.9 in [7] shows the following soundness:

► **Theorem 3.11.** *For any normal dependency scheme D , the proof systems $\text{QCDCL}^{\text{ANY-ORD}}(D, \text{Cube-LD})$ and $\text{QCDCL}^{\text{ANY-ORD}}(D, \text{Cube-D})$ are sound.*

Proof. To show that both these systems are sound, it is enough to show the following three statements: (1) The derivation of any learnt clause is a valid LDQ(D)-Res derivation. (2) If $\text{CubePol} = \text{Cube-LD}$, the derivation of any learnt cube is a valid LDQ-TermRes derivation, and the addition of cubes when learning from satisfaction is sound. (3) If $\text{CubePol} = \text{Cube-D}$, the derivation of any learnt cube is a valid Q(D)-TermRes derivation. From these three it follows that sticking together the derivations of the final learnt empty clause/term gives a proof in the corresponding system, and all these systems are known to be sound.

Statement (3) is true by definition: term resolution in learning is performed only if it is valid in Q(D)-TermRes. For Statement (2), cube learning is shown to be sound in [15,

Theorem 3.8]. For statement (1), we need to show that the resolution steps performed while learning respect the side-conditions of $\text{LDRes}(\mathcal{D})$. The analogous statement when $\mathcal{D} = \mathcal{D}^{\text{trv}}$ is proved in [7, Lemma 3.7, Proposition 3.8, Theorem 3.9], but the same proof works with any $\mathcal{D}$. It is formally shown in Lemma 3.12 below. $\blacktriangleleft$

► **Lemma 3.12.** *For any normal dependency scheme $\mathcal{D}$, the derivations of a clause learnt from a trail in the proof systems $\text{QCDCL}^{\text{ANY-ORD}}(\mathcal{D}, \text{Cube-LD})$ and $\text{QCDCL}^{\text{ANY-ORD}}(\mathcal{D}, \text{Cube-D})$ are valid $\text{LDQ}(\mathcal{D})$ -Res derivations.*

Proof. This proof essentially replicates the proofs of Lemma 3.7 and Proposition 3.8 from [7]. We need to show that every clause learnt from a trail :

$$\mathcal{T} = (p_{(0,1)}, \dots, p_{(0,g_0)}; \mathbf{d}_1, p_{(1,1)}, \dots, p_{(1,g_1)}; \mathbf{d}_2, \dots, \dots; \mathbf{d}_r, p_{(r,1)}, \dots, p_{(r,g_r)})$$

is a valid $\text{LDQ}(\mathcal{D})$ -Res derivation. Let $C_{i,j}$ denote the clause learnt corresponding to propagated literal $p_{i,j}$.

Step 1: No $C_{i,j}$ contains an existential tautology.

Suppose there exists a variable x such that $x \neq \text{var}(p_{i,j})$ and $x \in C_{i,j+1}$ and $\bar{x} \in \text{red-D}(\text{ante}(p_{i,j}))$. Let $A = \text{ante}(p_{i,j})$, then since x is existential variable, and $\bar{x} \in A$, therefore x must be assigned in the trail prior to the propagation of $p_{i,j}$.

On the other hand, we have $x \in C_{i,j+1}$, which is the learnable clause which is derived with the aid of antecedent clauses of literals occurring right of $p_{i,j}$ in the trail. In particular, we can find some $p_{k,m}$ right of $p_{i,j}$ in the trail with $x \in \text{ante}(p_{k,m})$. But because x appears in the trail to the left of $p_{i,j}$, this gives a contradiction since $\text{ante}(p_{k,m})$ must not become true before propagating $p_{k,m}$.

Step 2: Derivation of clauses with universal tautologies is sound.

Proof goes via contradiction. Suppose there exists a universal tautology derived which is unsound. Without loss of generality let that variable be u and the propagated literal over which this resolution happens be $p_{i,j}$. Since the resolution is unsound $(u, \text{var}(p_{i,j})) \in \mathcal{D}$ and on of the following conditions must hold:

1. $u \in C_{i,j+1}$ and $\bar{u} \in \text{ante}(p_{i,j})$
2. $u \vee \bar{u} \in C_{i,j+1}$ and $\bar{u} \in \text{ante}(p_{i,j})$
3. $u \in C_{i,j+1}$ and $u \vee \bar{u} \in \text{ante}(p_{i,j})$
4. $u \vee \bar{u} \in C_{i,j+1}$ and $u \vee \bar{u} \in \text{ante}(p_{i,j})$

Consider the first case: Since $u \in C_{i,j+1}$ there has to be a propagated literal $p_{k,m}$ right of $p_{i,j}$ in the trail such that $u \in \text{ante}(p_{k,m})$. In order to become unit, the u in $\text{ante}(p_{k,m})$ needs to vanish. We distinguish two cases:

Case (i): $\bar{u}$ was assigned before $p_{k,m}$ was propagated. Then $\bar{u}$ does not appear in the trail, then for $p_{i,j}$ to be propagated $\bar{u}$ must have been reduced in $\text{ante}(p_{i,j})$ which is possible only if $(u, \text{var}(p_{i,j})) \notin \mathcal{D}$ giving rise to a contradiction.

Case (ii): $u \in \text{ante}(p_{k,m})$ is removed via reduction. For propagations $p_{i,j}$, $p_{k,m}$ to both happen u , $\bar{u}$ could not be assigned in the trail prior to propagating $p_{i,j}$, therefore for $p_{i,j}$ to be propagated $\bar{u}$ must have been reduced in $\text{ante}(p_{i,j})$ which is possible only if $(u, \text{var}(p_{i,j})) \notin \mathcal{D}$ giving rise to a contradiction.

The same argument above works for all the remaining cases. $\blacktriangleleft$

Thus using dependency schemes in decision order gives sound and complete systems.

► **Theorem 3.13.** *For any dependency schemes $\mathcal{D}'$, $\mathcal{D}$ where $\mathcal{D}$ is normal, and for each $\text{CubePol} \in \{\text{No-Cube}, \text{Cube-LD}, \text{Cube-D}\}$, the proof system $\text{QCDCL}^{\mathcal{D}'\text{-ORD}}(\mathcal{D}, \text{CubePol})$ is sound and refutationally complete.*

Proof. By Proposition 3.10, $\text{QCDCL}^{\text{LEV-ORD}}(\mathcal{D}, \text{No-Cube})$ is refutationally complete. By Theorem 3.11, $\text{QCDCL}^{\text{ANY-ORD}}(\mathcal{D}, \text{CubePol})$ is sound. By Observation 3.7 and Observation 3.8, all the aforementioned systems are sound and refutationally complete. $\blacktriangleleft$

3.2 Strength of QCDCL based proof systems with D-ORD

In QCDCL based proof systems, incorporating dependency schemes into propagation and learning processes does not always yield benefits: as shown in [17], certain pathological formulas can render the addition of dependency schemes disadvantageous when decisions are constrained to the LEV-ORD decision policy.

If the dependency scheme is allowed to influence the decision order (i.e., the system adopts the D-ORD decision policy), we show below that the resulting systems are strictly more powerful than their counterparts using LEV-ORD. (For $\mathcal{D}_1 = \mathcal{D}^{\text{std}}$, an advantage over LEV-ORD was noted already in [21].) However, they remain strictly weaker than the LDQ(D)-Res systems.

► **Theorem 3.14.** *For dependency schemes $\mathcal{D}_1 \in \{\mathcal{D}^{\text{rrs}}, \mathcal{D}^{\text{std}}\}$ and $\mathcal{D}_2 \in \{\mathcal{D}^{\text{rrs}}, \mathcal{D}^{\text{std}}, \mathcal{D}^{\text{trv}}\}$, and for policy $\text{CubePol} \in \{\text{No-Cube}, \text{Cube-LD}, \text{Cube-D}\}$, the proof system $\text{QCDCL}^{\mathcal{D}_1\text{-ORD}}(\mathcal{D}_2, \text{CubePol})$ p -simulates $\text{QCDCL}^{\text{LEV-ORD}}(\mathcal{D}_2, \text{CubePol})$ and is not simulated by it.*

Proof. The p -simulation follows from Observation 3.7.

For $\mathcal{D}_1 = \mathcal{D}^{\text{std}}$, an advantage over LEV-ORD was noted already in [21].

For $\mathcal{D}_1 = \mathcal{D}^{\text{rrs}}$, to show that there is no reverse simulation, we consider the TwoPHPandCT formulae defined in [17], and described in Example 3.6. These have an unsatisfiable matrix, so by Observation 3.9, it suffices to show lower and upper bounds for $\text{CubePol} = \text{No-Cube}$.

For these formulas, $\mathcal{D}^{\text{rrs}} = \emptyset$ and $\mathcal{D}^{\text{std}} = \{(u, x_i), (u, y_i) : \text{for all } i\} \cup \{(v, z_1), (v, z_2)\}$.

It is shown in [17] (Lemma 5) that these formulas require exponential size refutations in $\text{QCDCL}^{\text{LEV-ORD}}(\mathcal{D}^{\text{rrs}}, \text{No-Cube})$ and $\text{QCDCL}^{\text{LEV-ORD}}(\mathcal{D}^{\text{trv}}, \text{No-Cube})$. Furthermore, the following observation shows that they also require exponential size refutations in $\text{QCDCL}^{\text{LEV-ORD}}(\mathcal{D}^{\text{std}}, \text{No-Cube})$; for these formulas, $\mathcal{D}^{\text{std}} = \{(u, x_i), (u, y_i) : \text{for all } i\} \cup \{(v, z_1), (v, z_2)\}$. With the LEV-ORD decision policy, the first decision must be on u , which causes no propagations, and subsequent decisions in LEV-ORD force refuting PHP (in either x or y), which is known to require exponential size.

On the other hand, we have already seen in Example 3.6 that they have short (constant-sized) refutations in $\text{QCDCL}^{\mathcal{D}_1\text{-ORD}}(\mathcal{D}_2, \text{No-Cube})$ if $\mathcal{D}_1 = \mathcal{D}^{\text{rrs}}$ or $\mathcal{D}^{\text{std}}$. $\blacktriangleleft$

► **Theorem 3.15.** *For $\text{CubePol} \in \{\text{No-Cube}, \text{Cube-LD}, \text{Cube-D}\}$ in $\mathcal{D} \in \{\mathcal{D}^{\text{trv}}, \mathcal{D}^{\text{std}}, \mathcal{D}^{\text{rrs}}\}$, the proof system LDQ(D)-Res p -simulates $\text{QCDCL}^{\mathcal{D}\text{-ORD}}(\mathcal{D}, \text{CubePol})$ and is not simulated by it.*

We defer the proof of this theorem to the next section, since it uses a new formula that we define there, the DoubleLongEq formulas.

4 Dependency-schemes-based QCDCL systems restricted to LEV-ORD

We now restrict our attention to proof systems utilizing only the LEV-ORD decision policy. Even with LEV-ORD, dependency schemes can affect (enhance or impair) the performance of QCDCL systems. These are the systems also considered in [7, 15, 17].

4.1 Preprocessing as a tool

In [17], another way of incorporating dependency schemes was also considered, through “preprocessing”. For a more complete comparison, we very briefly define such systems here as well. Preprocessing a formula using a dependency scheme simply means applying all reductions enabled by it on the input formula, and then proceeding with whatever version of QCDCL is of interest, on the reduced formula.

► **Definition 4.1.** For a QBF $\Phi = Q \cdot \phi$ and a normal dependency scheme D , a derivation of a clause C from Φ in $D + \text{QCDCL}^{\text{ORD}}(\text{ClausePol}, \text{CubePol})$ is a derivation of C from the QBF $\Psi = \text{red-}D(\Phi)$ in $\text{QCDCL}^{\text{ORD}}(\text{ClausePol}, \text{CubePol})$.

As shown in Theorem 4 of [17], preprocessing using D^{rrs} can significantly alter the system strength. In contrast, we observe below that preprocessing using schemes D^{trv} or D^{std} has no effect, no matter what version of QCDCL is the subsequent system.

► **Proposition 4.2.** For every decision policy ORD , dependency scheme $D \in \{D^{\text{trv}}, D^{\text{std}}, D^{\text{rrs}}\}$ and for $\text{CubePol} \in \{\text{No-Cube}, \text{Cube-LD}, \text{Cube-D}\}$, the proof systems $\text{QCDCL}^{\text{ORD}}(D, \text{CubePol})$, $D^{\text{trv}} + \text{QCDCL}^{\text{ORD}}(D, \text{CubePol})$, and $D^{\text{std}} + \text{QCDCL}^{\text{ORD}}(D, \text{CubePol})$ are equivalent to each other.

Proof. Let Φ be any given PCNF formula, and let $\Psi = \text{red}(\Phi)$. By definition, $D^{\text{std}}(\Phi) \subseteq D^{\text{trv}}(\Phi)$. So all reductions permitted by D^{trv} are also permitted by D^{std} . If a clause C of Φ has variables x, y with $(x, y) \in D^{\text{trv}}$, then by definition of D^{std} , (x, y) is also in $D^{\text{std}}(\Phi)$. So D^{std} does not enable any new reductions on initial clauses. Hence $\Psi = \text{red-}D^{\text{std}}(\Phi) = \text{red-}D^{\text{trv}}(\Phi)$, so the second and third proof systems are equivalent.

For any clause C , $\text{red}(C)$ can only remove universal variables from C . By the way Ψ is defined, if $(x, y) \in D(\Psi)$, then it is also in $D(\Phi)$. In the other direction, if $(x, y) \in D(\Phi)$, and if both x, y appear in the matrix of Ψ , then (x, y) is also in $D(\Psi)$ because the prefix of Φ and Ψ is the same, and the witnessing sequence (in the case of D^{std} or D^{rrs}) has only existential literals which are not removed, so the same sequence is a witness in Ψ too.

To show that the first and second proof systems are the same, note that for $D \in \{D^{\text{trv}}, D^{\text{std}}, D^{\text{rrs}}\}$, for any clause C , $\text{red-}D(C) = \text{red-}D(\text{red}(C))$. Therefore if one of D^{trv} , D^{std} or D^{rrs} are used in propagation and learning, then all the propagations in the first trail of either refutation are also enabled in the other. (To be pedantic, a derivation in Φ may have universal variables that have vanished from the matrix of Ψ but are still in the quantifier prefix of Ψ ; these can have no effect on any propagation since they vanished through applications of red .) Hence the first clause/cube learnt in any one system can also be learnt in the other. Continuing this argument on subsequent trails, the entire derivation can be replicated. ◀

4.2 Some New Formulae

We now introduce some new formulas which will be used to pinpoint the relative strengths of the proof systems.

The DoubleLongEq_n formulas.

The Equality formulas, first defined in [6], show that the proof system QCDCL with cube learning is stronger than QCDCL without [15]. We wish to show that cube-learning offers a similar advantage for systems with D^{rrs} . The Equality formulas cannot show this because $D^{\text{rrs}}(\text{Equality}) = \emptyset$; so using D^{rrs} in any way makes them easy to refute irrespective of cube-learning. To achieve the desired separation, we modify the Equality formula by adding

two clauses that make D^{rrs} and D^{trv} identical. These new formulas, called **DoubleLongEq**, maintain the separation without altering the hardness of **Equality**.

► **Formula 1.** *The **DoubleLongEq**_n formula has the prefix $\exists x_1 \dots x_n \forall u_1 \dots u_n \exists t_1 \dots t_n$ and the PCNF matrix*

$$\underbrace{(\bar{t}_1 \vee \dots \vee \bar{t}_n)}_{T_n} \wedge \bigwedge_{i=1}^n \left[\underbrace{(x_i \vee u_i \vee t_i)}_{A_i} \wedge \underbrace{(\bar{x}_i \vee \bar{u}_i \vee t_i)}_{B_i} \right] \wedge \underbrace{(\bar{u}_1 \vee \dots \vee \bar{u}_n \vee \bar{t}_1 \vee \dots \vee \bar{t}_n)}_{UT_n} \wedge \underbrace{(\bar{u}_1 \vee \dots \vee \bar{u}_n \vee t_1 \vee \dots \vee t_n)}_{UT'_n}$$

(Note: deleting the clauses UT_n and UT'_n gives the **Equality** formulas.)

► **Proposition 4.3.** *For $\Phi = \text{DoubleLongEq}$, $\text{red}(\Phi) = \Phi$, and $D^{rrs}(\Phi) = D^{\text{std}}(\Phi) = D^{\text{trv}}(\Phi)$.*

Proof. By definition, $D^{\text{trv}}(\text{DoubleLongEq}) = \{(u_i, t_j) : i, j \in \{1 \dots n\}\}$; that is, each t_j variable depends on each u_i variable. Since each occurrence of a u variable in the formula is blocked by some t variable, we have $\text{red}(\text{DoubleLongEq}) = \text{DoubleLongEq}$.

The next claim is that for this family of formulae Φ , $D^{rrs}(\Phi) = D^{\text{std}}(\Phi) = D^{\text{trv}}(\Phi)$. It suffices to show that $D^{\text{trv}}(\Phi) \subseteq D^{rrs}(\Phi)$.

We want to show that each t_j depending on each u_i is the case for D^{rrs} as well. We consider two cases.

Case 1: $i = j$. The clauses A_i and UT_n contain the resolution paths (u_i, t_i) and $(\bar{u}_i, \bar{t}_i)$ respectively. Therefore $(u_i, t_i) \in D^{rrs}$ for all $i \in \{1 \dots n\}$.

Case 2: $i \neq j$. The sequence of clauses A_i, UT_n contains the resolution path $(u_i, t_i), (\bar{t}_i, \bar{t}_j)$, while the clause UT'_n contains the resolution path $(\bar{u}_i, t_j)$. Therefore $(u_i, t_j) \in D^{rrs}$ for all $i \neq j \in \{1 \dots n\}$. ◀

This makes the usage of the dependency schemes D^{rrs} or D^{std} completely useless. We now show that the formulas are easy to refute with cube-learning, but hard if cube-learning is switched off. Both these results closely mirror the corresponding results for **Equality** shown in [7] and [15] respectively.

► **Lemma 4.4.** *For $D_1, D_2 \in \{D^{\text{trv}}, D^{\text{std}}, D^{rrs}\}$ and $\text{CubePol} \in \{\text{Cube-LD}, \text{Cube-D}_2\}$, the **DoubleLongEq** formulas have polynomial size refutations in $D_1 + \text{QCDCL}^{\text{LEV-ORD}}(D_2, \text{CubePol})$*

To prove this, we first show that these formulas are easy to refute with cube-learning.

► **Proposition 4.5.** *The **DoubleLongEq** formulas have polynomial size refutations in the proof system $\text{QCDCL}^{\text{LEV-ORD}}(D^{\text{trv}}, \text{Cube-LD})$.*

Proof. The polynomial size refutation for these formulas in $\text{QCDCL}^{\text{LEV-ORD}}(D^{\text{trv}}, \text{Cube-LD})$ is exactly the same as the refutation in $\text{QCDCL}^{\text{LEV-ORD}}(D^{\text{trv}}, \text{Cube-LD})$ for the **Equality** formulas, as described in [15]. By constructing trails in exactly the same manner, we first learn $2n - 2$ cubes of the form $(x_i \wedge \bar{u}_i)$ and $(\bar{x}_i \wedge u_i)$ for $i = 1 \dots n - 1$ and then start clause learning by constructing trails ending in a conflict. The two new clauses UT_n and UT'_n play no role whatsoever. For completeness, we reproduce the entire refutation below; a reader familiar with the construction from [15] can completely skip these details.

The proof goes in two stages. The first stage involves learning the cubes $x_i \wedge \bar{u}_i$ and $\bar{x}_i \wedge u_i$ for $i \in [n - 1]$. The first trail is the following.

$$\mathcal{T}_1 = \mathbf{x}_1; \dots; \mathbf{x}_n; \bar{\mathbf{u}}_1; \dots; \bar{\mathbf{u}}_n; \bar{\mathbf{t}}_1; \mathbf{t}_2; \dots; \mathbf{t}_n$$

It assigns all variables without conflict and satisfies the matrix. The partial assignment $x_1 \wedge \bar{u}_1 \wedge t_1 \wedge t_2 \wedge \dots \wedge t_n$ contained in it is also a satisfying assignment, and reducing it with $\text{red}_{\exists}$ we can learn the cube $x_1 \wedge \bar{u}_1$ from this trail.

Analogously, creating a complementary trail $\mathcal{T}'_1$ where each decision is the complement of the decision in $\mathcal{T}_1$, we can learn the cube $\bar{x}_1 \wedge u_1$.

Suppose we have learn $2i$ cubes in the same manner; $x_j \wedge \bar{u}_j$ and $\bar{x}_j \wedge u_j$ for $j = 1, \dots, i$. For $i + 1$, create the following trail.

$$\mathcal{T}_{i+1} = \mathbf{x}_1, u_1, t_1; \dots; \mathbf{x}_i, u_i, t_j; \mathbf{x}_{i+1}; \dots; \mathbf{x}_n; \bar{\mathbf{u}}_{i+1}; \dots; \bar{\mathbf{u}}_n; \bar{\mathbf{t}}_{i+1}; \mathbf{t}_{i+2}; \dots; \mathbf{t}_n$$

In this trail, for $j \leq i$, $\text{ante}(u_j) = x_j \wedge \bar{u}_j$ and $\text{ante}(t_j) = \bar{x}_j \vee \bar{u}_j \vee t_j$. As earlier, the trail satisfies all clauses without conflict. Extracting the partial assignment $x_{i+1} \wedge \bar{u}_{i+1} \wedge t_1 \wedge \dots \wedge t_i \wedge \bar{t}_{i+1} \wedge t_{i+2} \wedge \dots \wedge t_n$ which also satisfies the matrix, and reducing it, we can learn the cube $x_{i+1} \wedge u_{i+1}$. Analogously through a trail $\mathcal{T}'_{i+1}$ we learn $\bar{x}_{i+1} \wedge u_{i+1}$.

Having learnt the $2n - 2$ cubes in this manner, we start with clause learning, where we proceed by constructing the trails $\mathcal{U}_{n-1}, \mathcal{V}_{n-1}, \mathcal{U}_{n-2}, \mathcal{V}_{n-2}, \dots, \mathcal{U}_1, \mathcal{V}_1$ described below, and learn clauses $L_{n-1}, R_{n-1}, \dots, L_1, R_1$ corresponding to these trails. We use T_j to denote the subclause of T_n with literals t_i for $i \in j$.

The initial trail is

$$\mathcal{U}_{n-1} = (\mathbf{x}_1, u_1, t_1; \mathbf{x}_2, u_2, t_2; \dots; \mathbf{x}_{n-1}, u_{n-1}, t_{n-1}, \bar{t}_n, x_n, \square)$$

The antecedent clauses are as follows:

$$\begin{aligned} \text{ante}(u_j) &= x_j \wedge \bar{u}_j \\ \text{ante}(t_j) &= \bar{x}_j \vee \bar{u}_j \vee t_j \\ \text{ante}(\bar{t}_n) &= T_n \\ \text{ante}(x_n) &= x_n \vee u_n \vee t_n \\ \text{ante}(\square) &= \bar{x}_n \vee \bar{u}_n \vee t_n \end{aligned}$$

From these clauses we learn the clause $L_{n-1} = \bar{x}_{n-1} \vee \bar{u}_{n-1} \vee (u_n \vee \bar{u}_n) \vee T_{n-2}$.

Then we restart and create a symmetric trail to $\mathcal{U}_{n-1}$:

$$\mathcal{V}_{n-1} = (\bar{\mathbf{x}}_1, \bar{u}_1, t_1; \bar{\mathbf{x}}_2, \bar{u}_2, t_2; \dots; \bar{\mathbf{x}}_{n-1}, \bar{u}_{n-1}, t_{n-1}, \bar{t}_n, x_n, \square)$$

where the antecedent clauses are

$$\begin{aligned} \text{ante}(\bar{u}_j) &= \bar{x}_j \wedge u_j \\ \text{ante}(t_j) &= x_j \vee u_j \vee t_j \\ \text{ante}(\bar{t}_n) &= T_n \\ \text{ante}(x_n) &= x_n \vee u_n \vee t_n \\ \text{ante}(\square) &= \bar{x}_n \vee \bar{u}_n \vee t_n. \end{aligned}$$

From this trail we can learn the clause $R_{n-1} = x_{n-1} \vee u_{n-1} \vee (u_n \vee \bar{u}_n) \vee T_{n-2}$.

For i in the range of 2 to $n - 1$, we define the following clauses:

$$\begin{aligned} L_i &= \bar{x}_i \vee \bar{u}_i \vee \bigvee_{j=i+1}^n (u_j \vee \bar{u}_j) \vee T_{i-1} \\ R_i &= x_i \vee u_i \vee \bigvee_{j=i+1}^n (u_j \vee \bar{u}_j) \vee T_{i-1} \end{aligned}$$

We claim that from the trail $\mathcal{U}_i$ we learn the clause L_i and from the trail $\mathcal{V}_i$ we learn the clause R_i . We have already established this for $i = n - 1$. Suppose we have already learnt $L_{n-1}, R_{n-1}, \dots, L_{i+1}, R_{i+1}$ for $1 \leq i < n - 1$. Continuing, we consider the next trail,

$$\mathcal{U}_i = (\mathbf{x}_1, u_1, t_1; \mathbf{x}_2, u_2, t_2; \dots; \mathbf{x}_i, u_i, t_i, x_{i+1}, \square)$$

where the antecedent clauses are as follows.

$$\begin{aligned} \text{ante}(u_j) &= x_j \wedge \bar{u}_j \\ \text{ante}(t_j) &= \bar{x}_j \vee \bar{u}_j \vee t_j \\ \text{ante}(x_{i+1}) &= L_{i+1} \\ \text{ante}(\square) &= R_{i+1} \end{aligned}$$

From this we learn the clause $L_i = \bar{x}_i \bar{u}_i \vee \bigvee_{j=i+1}^n (u_j \vee \bar{u}_j) \vee T_{i-1}$. Next we create the symmetrical trail,

$$\mathcal{V}_i = (\bar{\mathbf{x}}_1, \bar{u}_1, t_1; \bar{\mathbf{x}}_2, \bar{u}_2, t_2; \dots; \bar{\mathbf{x}}_i, \bar{u}_i, t_i, x_{i+1}, \square)$$

and the antecedent clauses are as follows:

$$\begin{aligned} \text{ante}(\bar{u}_j) &= \bar{x}_j \wedge u_j \\ \text{ante}(t_j) &= x_j \vee u_j \vee t_j \\ \text{ante}(x_{i+1}) &= L_{i+1} \\ \text{ante}(\square) &= R_{i+1} \end{aligned}$$

From this we can learn the clause $R_i = x_i \vee u_i \vee \bigvee_{j=i+1}^n (u_j \vee \bar{u}_j) \vee T_{i-1}$.

The proof ends with the two trails

$$\mathcal{U}_1 = (\mathbf{x}_1, u_1, t_1, x_2, \square)$$

with antecedents clauses

$$\begin{aligned} \text{ante}(u_1) &= x_1 \wedge \bar{u}_1 \\ \text{ante}(t_1) &= \bar{x}_1 \vee t_1 \\ \text{ante}(x_2) &= L_2 \\ \text{ante}(\square) &= R_2 \end{aligned}$$

allowing us to learn the clause $L_1 = \bar{x}_1$, and finally the last trail

$$\mathcal{V}_1 = (\bar{x}_1, \bar{u}_1, t_1, x_2, \square)$$

with antecedent clauses

$$\begin{aligned} \text{ante}(\bar{x}_1) &= \bar{x}_1 \\ \text{ante}(\bar{u}_1) &= \bar{x}_1 \wedge u_1 \\ \text{ante}(t_1) &= \bar{x}_1 \vee u_1 \vee t_1 \\ \text{ante}(x_2) &= L_2 \\ \text{ante}(\square) &= R_2 \end{aligned}$$

Resolving over all propagations in this trail, we learn the empty clause, completing the refutation. $\blacktriangleleft$

Now proving Lemma 4.4 is straightforward:

Proof. (of Lemma 4.4.) It can be seen that the cubes learnt in the refutation described in Proposition 4.5 require no cube learning via resolution steps; they are all learnt from trails ending in satisfaction, using the term axiom rule and the $\text{red}_{\exists}$ rule. Therefore for this particular refutation, every cube learning step in the $\text{QCDCL}^{\text{LEV-ORD}}(\mathcal{D}^{\text{trv}}, \text{Cube-LD})$ refutation is also a valid step in a $\text{QCDCL}^{\text{LEV-ORD}}(\mathcal{D}^{\text{trv}}, \text{Cube-}\mathcal{D}^{\text{trv}})$ refutation.

By the discussion after the formula definitions, these are also valid refutations in $\mathcal{D}_1 + \text{QCDCL}^{\text{LEV-ORD}}(\mathcal{D}_2, \text{Cube-}\mathcal{D}_2)$ where $\mathcal{D}_1, \mathcal{D}_2 \in \{\mathcal{D}^{\text{trv}}, \mathcal{D}^{\text{std}}, \mathcal{D}^{\text{rrs}}\}$. $\blacktriangleleft$

We now turn to hardness.

► **Lemma 4.6.** *For $\mathcal{D}_1, \mathcal{D}_2 \in \{\mathcal{D}^{\text{trv}}, \mathcal{D}^{\text{std}}, \mathcal{D}^{\text{rrs}}\}$ the DoubleLongEq formulas require exponential size refutations in $\mathcal{D}_1 + \text{QCDCL}^{\text{LEV-ORD}}(\mathcal{D}_2, \text{No-Cube})$.*

Proof. By the discussion above, it suffices to show that the formulas require exponential size refutations in $\text{QCDCL}^{\text{LEV-ORD}}(\mathcal{D}^{\text{trv}}, \text{No-Cube})$.

In [13], the authors consider Σ^3 formulas with a specific structure, called *XUT*-formulas with the *XT*-property. They introduce a semantic measure called gauge for Σ^3 QBFs, and show that for an *XUT*-formula with the *XT*-property, the size of a refutation in $\text{QCDCL}^{\text{LEV-ORD}}(\mathcal{D}^{\text{trv}}, \text{No-Cube})$ is at least exponential in its gauge.

The DoubleLongEq formulas are easily seen to be *XUT*-formulas with the *XT*-property. We show now that they have linear gauge, implying that they are exponentially hard.

Recall the definition of *XUT* formulas and the gauge measure:

► **Definition 4.7** (*XT*-property, [13]). *Let Φ be a PCNF QBF of the form $\exists X \forall U \exists T \cdot \phi$, where X, U, T are non-empty sets of variables. Then Φ is an *XUT*-formula. We call a clause C an*

- *X-clause: if it is non-empty and contains only X variables,*
- *T-clause: if it is non-empty and contains only T variables,*
- *XT-clause: if it contains no U variable and at least one X and one T variable,*
- *XUT-clause: if it contains atleast one each of X, U , and T variables.*

Φ is said to fulfill the *XT*-property if ϕ contains no *XT*-clauses or unit *T* clause, and if no two *T* clauses in ϕ are resolvable (the resolvent of any two *T* clauses, if defined, is tautological).

► **Definition 4.8** (gauge, [13]). *Let Φ be an *XUT* formula. The gauge of Φ is the size of the narrowest *X*-clause derivable using only reductions and resolutions over variables in T .*

First observe that none of the axioms of DoubleLongEq are *X*-clauses. Therefore to derive an *X*-clause, there has to be some *T*-resolutions. A first *T*-resolution must involve either T_n or UT_n , since only these clauses have *T* variables negated. However, both these clauses have all n *T*-variables. Thus to eventually derive an *X*-clause, there must be a resolution on every t_i variable. Each such resolution introduces an x_i variable. Therefore by the time all *T* variables are removed, all the *X* variables are introduced. Therefore, the gauge of DoubleLongEq _{n} is n . $\blacktriangleleft$

With this hardness result about DoubleLongEq, we can now complete the proof of Theorem 3.15.

Proof. (of Theorem 3.15.) By definition, a valid LDQ(D)-Res refutation is contained within every $\text{QCDCL}^{\text{D-ORD}}(\mathcal{D}, \text{CubePol})$ refutation.

We now show that the simulation is strict. We first prove it for the case when `CubePol` is `No-Cube`, using the `DoubleLongEq` formulas. Then we tweak the formulas slightly to extend the result to other cube-learning policies.

We saw in Lemma 4.6 that these formulas are hard for $\text{QCDCL}^{\text{D-ORD}}(\text{D}, \text{No-Cube})$. To see that they are easy to refute in $\text{LDQ}(\text{D})\text{-Res}$, it is enough to construct a short refutation in LDQ-Res . By the rules of long-distance resolution, we can resolve each pair of A_i and B_i clauses on x_i to get the n clauses $C_i = u_i \vee \bar{u}_i \vee t_i$. Starting with the clause T_n and resolving sequentially with the C_i clauses, we obtain the purely universal clause $\bigvee_{i=1}^n u_i \vee \bar{u}_i$. This can be universally reduced to yield the empty clause, completing the refutation.

To extend the separation to systems that allow cube-learning, we slightly modify the `DoubleLongEq` formula. We add new existentially quantified variables at the end of the quantifier prefix, and we add to the matrix new clauses using these variables that encode PHP. This not only preserves that the formula is false, but also makes the formula matrix unsatisfiable. Therefore, cube-learning will never be able to help in any QCDCL refutation. The hardness for $\text{QCDCL}^{\text{D-ORD}}(\text{D}, \text{No-Cube})$ remains valid even after this modification, since any refutation of the modified formula must either refute the unmodified `DoubleLongEq` formula, or refute PHP, and PHP itself is propositionally hard for resolution.

The LDQ-Res refutation of the modified formula remains the same as for `DoubleLongEq` since the new clauses are completely disjoint. $\blacktriangleleft$

The PreRRSTrapdoor_n formulas.

The next formula is designed to explore how adding D^{rrs} in different ways affects the system. It sends QCDCL trails into a "trap" (of refuting the hard existential Pigeonhole Principle PHP; see Example 3.6) if D^{rrs} is not used in propagation, but allows a short refutation (a contradiction on two variables) when D^{rrs} is used. This leads to the definition of a formula inspired by the `Trapdoor` and `Dep-Trap` formulas from [7](Def. 4.5) and [17](Def. 4.4) respectively.

► **Formula 2.** *The PreRRSTrapdoor_n formula has the prefix*

$\exists a \forall p \exists y_1, \dots, y_{s_n} \forall w \forall v \exists t \exists x_1, \dots, x_{s_n} \forall u \exists b \exists q \exists r \exists s$, and the matrix is as given below.

$$\begin{aligned} & \text{PHP}_n^{n+1}(x_1, \dots, x_{s_n}) \\ \text{for } i \in [s_n] : & (\bar{y}_i \vee x_i \vee u \vee b), (y_i \vee \bar{x}_i \vee u \vee b) \\ \text{for } i \in [s_n] : & (y_i \vee w \vee v \vee t \vee b), (y_i \vee w \vee v \vee \bar{t} \vee b), (\bar{y}_i \vee w \vee v \vee t \vee b), (\bar{y}_i \vee w \vee v \vee \bar{t} \vee b) \\ & (\bar{u} \vee \bar{b}), (v \vee \bar{b} \vee \bar{r}), (\bar{v} \vee b \vee s), (a \vee \bar{b}), (\bar{a} \vee \bar{b}), (p \vee q), (\bar{p} \vee \bar{q}) \end{aligned}$$

This formula has an unsatisfiable matrix (due to the presence of PHP).

► **Observation 4.9.** *The variable "w" is not necessary for the lower or upper bounds proved for this formula. Initialising $\text{PreRRSTrapdoor}|_{w=0}$ or removing the variable "w" entirely affects neither the bounds nor their proofs.*

However we keep it in because the PreRRSTrapdoor formulas are defined to extend the `Trapdoor` formula (defined in [7]) which has the "w" variable. Also, it shows that even if the preprocessing step (by D^{rrs}) is non-trivial and changes the formula, addition of D^{rrs} in propagation can still make a difference.

► **Proposition 4.10.** $\text{D}^{\text{rrs}}(\text{PreRRSTrapdoor}) = \{(u, b), (v, b), (p, q)\}$.

Proof. We look at all universal variables individually.

First consider p . The only other variable it shares a clause with is q , and q does not share clauses with any other existential variable. Therefore, q is the only potential variable that can depend on p in D^{rrs} , and it indeed does so as witnessed by the path $((p, q), (\bar{q}, p))$.

Next consider w . Since it appears in only one polarity in the matrix, therefore by definition no variable can depend on it.

Third consider v . Consider any path starting with $\bar{v}$ and ending in v , and linked by existential variables (in opposing polarities) right of v . Such a path must begin with the clause $\bar{v} \vee b \vee s$. Since $\bar{s}$ does not even appear in the formula, the linking literal must be b . The next clause must contain $\bar{b}$, and also either v or an existential variable right of v . The only such clause is $v \vee \bar{b} \vee \bar{r}$, and r cannot be used to further extend the path since the positive literal r does not appear in the formula. Hence the only such path $((\bar{v}, b), (\bar{b}, v))$, and b is the only existential depending on v in D^{rrs} .

Finally consider u . r, s appear in only one polarity in the axioms and q is completely disjoint from any clause with u . So potentially only b can depend on u . it indeed does so, because there is a path $(u, b), (\bar{b}, \bar{u})$; therefore $(u, b) \in D^{rrs}$.

Thus $D^{rrs}(\text{PreRRSTrapdoor}) = \{(u, b), (v, b), (p, q)\}$. ◀

Hence $\text{red-}D^{rrs}(\text{PreRRSTrapdoor}) = \text{PreRRSTrapdoor}|_{w=0}$. That is, if we preprocess using D^{rrs} , everything stays the same except that the variable w "disappears".

However, just preprocessing by D^{rrs} is not enough to make this formula easy to refute. The following lemmas shows that the presence of D^{rrs} *during propagation* is crucial to achieving polynomial sized refutations; its absence forces exponential size.

► **Lemma 4.11.** *For $\text{CubePol} \in \{\text{No-Cube}, \text{Cube-LD}, \text{Cube-}D^{rrs}\}$ the PreRRSTrapdoor formulas have polynomial size refutations in $D^{rrs} + \text{QCDCL}^{\text{LEV-ORD}}(D^{rrs}, \text{CubePol})$*

Proof. By Observation 3.8, it suffices to show polynomial size refutations in the system $D^{rrs} + \text{QCDCL}^{\text{LEV-ORD}}(D^{rrs}, \text{No-Cube})$.

Any trail must start with a decision on a . A decision in either polarity propagates $\bar{b}$, and with D^{rrs} used in propagation, further propagates s since s does not depend on v . Next, the variable p must be decided; any polarity propagates a q literal. At this point y_1 must be decided. Since t also does not depend on v , this decision in either polarity propagates a t literal and then a conflict. An example trail is as follows: $\mathcal{T} = \mathbf{a}, \bar{b}, s; \mathbf{p}, \bar{q}; \mathbf{y}_1, t, \square$. The conflict reached is due to the negation of the complete tautology on y_1 and t . Thus in 4 such trails the empty clause can be learnt, completing the refutation.

The refutation is described in detail as follows: We construct a polynomial time $D^{rrs} + \text{QCDCL}^{\text{LEV-ORD}}(D^{rrs}, \text{No-Cube})$ refutation of PreRRSTrapdoor . Since D^{rrs} for the formula is $\{(u, b), (v, b), (p, q)\}$, preprocessing using D^{rrs} reduces the formula to

$$\begin{aligned} & \exists a \forall p \exists y_1, \dots, y_{s_n} \forall v \exists t \exists x_1, \dots, x_{s_n} \forall u \exists b \exists q \exists r \exists s \\ & \quad \text{PHP}_n^{n+1}(x_1, \dots, x_{s_n}) \\ \text{for } i \in [s_n] : & \quad \bar{y}_i \vee x_i \vee u \vee b, \quad y_i \vee \bar{x}_i \vee u \vee b \\ \text{for } i \in [s_n] : & \quad y_i \vee v \vee t \vee b, \quad y_i \vee v \vee \bar{t} \vee b, \quad \bar{y}_i \vee v \vee t \vee b, \quad \bar{y}_i \vee v \vee \bar{t} \vee b \\ & \quad \bar{u} \vee \bar{b}, \quad v \vee \bar{b} \vee \bar{r}, \quad \bar{v} \vee b \vee s \\ & \quad a \vee \bar{b}, \quad \bar{a} \vee \bar{b}, \quad p \vee q, \quad \bar{p} \vee \bar{q} \end{aligned}$$

Now consider the following trail. Due to D^{rrs} being used in propagation as well, the literal t will be propagated even before v is decided, producing a conflict in the clauses involving t .

$$T_1 = (\mathbf{a}, \bar{b}, s; \mathbf{p}, \bar{q}; \mathbf{y}_1, t, \square)$$

where the antecedent clauses are as follows:

$$\begin{aligned}
\text{ante}(\bar{b}) &= \bar{a} \vee \bar{b} \\
\text{ante}(s) &= \bar{v} \vee b \vee s \\
\text{ante}(\bar{q}) &= \bar{p} \vee \bar{q} \\
\text{ante}(t) &= \bar{y}_1 \vee v \vee t \vee b \\
\text{ante}(\square) &= \bar{y}_1 \vee v \vee \bar{t} \vee b
\end{aligned}$$

From this trail we learn the clause $L_1 = \bar{a} \vee \bar{y}_1$.

Next construct the trail:

$$T_2 = (\bar{\mathbf{a}}, \bar{b}, s; \mathbf{p}, \bar{q}; \bar{\mathbf{y}}_1, t, \square)$$

where the antecedent clauses are as follows:

$$\begin{aligned}
\text{ante}(\bar{b}) &= a \vee \bar{b} \\
\text{ante}(s) &= \bar{v} \vee b \vee s \\
\text{ante}(\bar{q}) &= \bar{p} \vee \bar{q} \\
\text{ante}(t) &= y_1 \vee v \vee t \vee b \\
\text{ante}(\square) &= y_1 \vee v \vee \bar{t} \vee b
\end{aligned}$$

From this trail, we learn the clause $L_2 = a \vee y_1$.

Now consider the following third trail:

$$T_3 = (\mathbf{a}, \bar{b}, \bar{y}_1, t, \square)$$

with antecedent clauses as follows:

$$\begin{aligned}
\text{ante}(\bar{b}) &= \bar{a} \vee \bar{b} \\
\text{ante}(\bar{y}_1) &= L_1 = \bar{a} \vee \bar{y}_1 \\
\text{ante}(t) &= y_1 \vee v \vee t \vee b \\
\text{ante}(\square) &= y_1 \vee v \vee \bar{t} \vee b
\end{aligned}$$

From here we learn the unit clause $L_3 = \bar{a}$.

Finally we have the fourth trail which is fully propagated and has no decisions.

$$T_4 = (\bar{a}, \bar{b}, y_1, t, \square)$$

where,

$$\begin{aligned}
\text{ante}(\bar{a}) &= \bar{a} \\
\text{ante}(\bar{b}) &= \bar{a} \vee \bar{b} \\
\text{ante}(y_1) &= L_1 = a \vee y_1 \\
\text{ante}(t) &= y_1 \vee v \vee t \vee b \\
\text{ante}(\square) &= y_1 \vee v \vee \bar{t} \vee b
\end{aligned}$$

From this trail we learn the empty clause ($\square$), thus completing the refutation. $\blacktriangleleft$

► **Lemma 4.12.** *For $\text{CubePol} \in \{\text{No-Cube}, \text{Cube-LD}, \text{Cube-D}^{\text{rrs}}\}$ the PreRRSTrapdoor formulas require exponential size refutations in $\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{CubePol})$*

Proof. Since the **PreRRSTrapdoor** formulas have an unsatisfiable matrix, therefore, by Observation 3.9 it suffices to show hardness in $D^{rrs} + QCDCL^{LEV-ORD}(D^{trv}, \text{No-Cube})$

Observe that any trail must start with a decision on a , propagating a b literal, followed by a decision on p , propagating a q literal: $\mathcal{T} = \mathbf{a}/\bar{\mathbf{a}}, \bar{\mathbf{b}}; \mathbf{p}/\bar{\mathbf{p}}, \bar{\mathbf{q}}/q$.

At this point in the trail, the formula matrix has reduced to

$$\begin{aligned} & \text{PHP}_n^{n+1}(x_1, \dots, x_{s_n}) \\ \text{for } i \in [s_n] : & \quad (\bar{y}_i \vee x_i \vee u), (y_i \vee \bar{x}_i \vee u) \\ \text{for } i \in [s_n] : & \quad (y_i \vee v \vee t), (y_i \vee v \vee \bar{t}), (\bar{y}_i \vee v \vee t), (\bar{y}_i \vee v \vee \bar{t}) \\ & \quad (\bar{v} \vee s) \end{aligned}$$

This is effectively the matrix of the **Trapdoor** formulas from [7], with one extra clause $\bar{v} \vee s$. After this point, all decisions are made on y variables propagating a corresponding x variable. By the time all y variables are decided, a conflict in PHP on the x variables is achieved. Thus, exactly like the **Trapdoor** formulas, refuting **PreRRSTrapdoor** boils down to refuting PHP, which is known to require exponential size. $\blacktriangleleft$

The StdDepTrap formulas.

The third new formula **StdDepTrap** we introduce shows the advantage of using D^{std} in propagation and learning. The goal is to create clauses that can be learnt easily with D^{std} , but are hard to learn without it. These learned clauses enable a quick refutation in QCDCL with D^{std} , but without them, one is stuck refuting something hard.

► **Formula 3.** *The StdDepTrap_n formula has the prefix*

$\exists b \forall w_1 \exists z_1, \dots, z_{s_n} \forall w_2 \exists a \exists d \exists c \forall u \exists x \exists y \exists p \exists e_1 \exists e_2$, and the matrix is as given below.

$$\begin{aligned} & \bar{b} \vee \text{PHP}_n^{n+1}(z_1, \dots, z_{s_n}) \\ & (y \vee p), (y \vee \bar{p}), (w_1 \vee e_1), (w_2 \vee e_2) \\ & (b \vee y), (a \vee \bar{y}), (\bar{a} \vee x), (\bar{c} \vee u \vee \bar{x}), (d \vee c \vee \bar{y}), (\bar{d} \vee c \vee \bar{y}) \end{aligned}$$

The variables w_1, w_2, u are essentially “separators”, putting existential variables into different levels. The variables e_1, e_2, x are blockers, ensuring that the separators do not get reduced too early in the trail.

► **Proposition 4.13.** $D^{\text{std}}(\text{StdDepTrap}) = \{(w_1, e_1), (w_2, e_2), (u, x)\}$.

Proof. We want to show that $D^{\text{std}}(\text{StdDepTrap}) = \{(w_1, e_1), (w_2, e_2), (u, x)\}$. For each universal variable (there are three, w_1, w_2, u), we consider its dependencies.

The variable w_1 appears in exactly one clause, and that clause has just one other variable e_1 . This variable e_1 is quantified to the right of w_1 and appears in no other clause. So e_1 , and no other variable, depends on w_1 in D^{std} . Similarly, e_2 , and no other variable, depends on w_2 .

The variable u appears only in the clause $(\bar{c} \vee u \vee \bar{x})$. Since c is left of u in the quantifier prefix, it does not depend on u or provide a path for other variables to depend either. Since x is quantified after u , it does depend on u . However, x appears only in this clause and no other clause, so it cannot provide further paths either. Therefore it is the only variable depending on u .

Thus, $D^{\text{std}}(\text{StdDepTrap}) = \{(w_1, e_1), (w_2, e_2), (u, x)\}$. $\blacktriangleleft$

When D^{std} is used in propagation, we show that these formulas have short refutations.

► **Lemma 4.14.** *For $\text{CubePol} \in \{\text{No-Cube}, \text{Cube-LD}, \text{Cube-D}^{\text{trs}}\}$ the StdDepTrap formulas have polynomial size refutations in $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{std}}, \text{CubePol})$*

Proof. By Observation 3.8, it suffices to show polynomial size refutations in the system $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{std}}, \text{No-Cube})$.

For this purpose, consider the trail $\mathcal{T}_1 = \bar{b}, y, a, x, \bar{c}, d, \square$. Since $(u, y) \notin \text{D}^{\text{std}}(\text{StdDepTrap})$, the learnable sequence for this trail is

$$L_{\mathcal{T}_1} = \{c \vee \bar{y}, u \vee \bar{x} \vee \bar{y}, \bar{a} \vee \bar{y}, \bar{y}, b\}.$$

We choose to learn $\bar{y}$, and then proceed to the next trail $\mathcal{T}_2 = \bar{y}, p, \square$. The learnable clauses for this trail are

$$L_{\mathcal{T}_2} = \{y \vee \bar{p}, y, \square\}.$$

Thus the empty clause $\square$ is learnt, completing the refutation. ◀

However, without D^{std} , the propagations force refuting the PHP clauses, which is hard.

► **Lemma 4.15.** *For $\text{CubePol} \in \{\text{No-Cube}, \text{Cube-LD}, \text{Cube-D}^{\text{trv}}\}$ the StdDepTrap formulas require exponential size refutations in $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{CubePol})$*

Proof. We first show that the StdDepTrap formulas require exponential size without cube learning i.e. in $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{No-Cube})$ and then extend the argument for the case of $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{Cube-LD})$ and $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{Cube-D}^{\text{trv}})$

Every $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{No-Cube})$ trail must start with a decision on the variable b , unless and until a literal on b is learnt; thenceforth a trail must begin by propagating that literal.

Suppose that the $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{No-Cube})$ trail starts with a decision b . In such a case there are no propagations possible. Next a decision must be made on w_1 , which may or may not propagate e_1 . At this point, no other propagations are possible, and decisions must be made on all the z variables. Since the z variables are only involved in the PHP clauses, this leads to a conflict in the PHP clauses.

Suppose that the $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{No-Cube})$ trail starts with the decision $\bar{b}$. Then there is a unique forced trail leading to a conflict, namely,

$$\mathcal{T}_1 = \bar{b}, y, a, x, \bar{c}, d, \square.$$

The learnable sequence for this trail under regular reduction is

$$L_{\mathcal{T}_1} = \{c \vee \bar{y}, u \vee \bar{x} \vee \bar{y}, \bar{a} \vee u \vee \bar{y}, u \vee \bar{y}, b\}.$$

It is easy to see that learning any clause other than b does not affect the decisions in the trail at all. Thus, with trails of this type, in a few stages the clause b will be learnt inevitably.

To summarise, any $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{No-Cube})$ trail must start with a decision on b . If the decision is b , it leads to a conflict in the PHP clauses. If the trails avoid decision b and start with $\bar{b}$, we will eventually be forced to learn the unit clause b , leading to trails starting with propagating b , and again reaching a conflict in the PHP clauses. Therefore, any $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{No-Cube})$ refutation of StdDepTrap reduces to refuting PHP, thus requiring exponential size.

In the case of $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{Cube-LD})$ and $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{Cube-D}^{\text{trv}})$, since the PHP clauses are unsatisfiable, $\bar{b}$ must be in any satisfying assignment of the matrix of StdDepTrap . Therefore for cube learning to ever play a role, the $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{Cube-LD})$ or $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{Cube-D}^{\text{trv}})$ trail must start with deciding b as $\bar{b}$. However, as discussed

above, trails starting with $\bar{b}$ are forced and rapidly hit a conflict; they cannot lead on to a satisfying assignment. Thus, even though the matrix of **StdDepTrap** is satisfiable (e.g. by the literals $\bar{b}, y, a, x, c, u, w_1, w_2$), such assignments can never be discovered through QCDCL trails. Hence, the $\text{QCDCL}^{\text{LEV-ORD}}(\mathcal{D}^{\text{trv}}, \text{No-Cube})$ hardness lifts as cube learning is useless, and the **StdDepTrap** formulas continue to be hard for $\text{QCDCL}^{\text{LEV-ORD}}(\mathcal{D}^{\text{trv}}, \text{Cube-LD})$ and $\text{QCDCL}^{\text{LEV-ORD}}(\mathcal{D}^{\text{trv}}, \text{Cube-}\mathcal{D}^{\text{trv}})$. ◀

4.3 Strength Relations between the Proof Systems

The hardness of formulas in QCDCL systems with or without cube-learning and dependency schemes are collected in Table 1; the table includes known bounds as well as those established in Section 4.2.

Using these bounds, we now establish various relations between the newly introduced LEV-ORD-based proof systems and also between them and other QCDCL-based proof systems. Figure 1 summarises these and earlier known relations in a visually clear way.

As discussed cube-learning in QCDCL proof systems is the concept of allowing "terms" to be learnt from satisfying trails, and intuitively more (optional) learning power would mean a more powerful proof system. The first theorem validates this claim. It states that for any QCDCL proof system with dependency schemes, adding cube-learning yields a more powerful system than the corresponding system without it.

► **Theorem 4.16.** *For $\mathcal{D}_1, \mathcal{D}_2 \in \{\mathcal{D}^{\text{trv}}, \mathcal{D}^{\text{std}}, \mathcal{D}^{\text{rrs}}\}$ and $\text{CubePol} \in \{\text{Cube-LD}, \text{Cube-}\mathcal{D}_2\}$ the proof system $\mathcal{D}_1 + \text{QCDCL}^{\text{LEV-ORD}}(\mathcal{D}_2, \text{CubePol})$ is strictly stronger than $\mathcal{D}_1 + \text{QCDCL}^{\text{LEV-ORD}}(\mathcal{D}_2, \text{No-Cube})$*

Proof. From Observation 3.8 the systems with cube-learning are at least as strong as the corresponding systems without cube learning. The **DoubleLongEq** formulas show that they are in fact strictly stronger; see the bounds in Lemmas 4.4 and 4.6. ◀

For the next three theorems we focus specifically on $\mathcal{D}^{\text{rrs}}$. The first of these shows that irrespective of the manner in which $\mathcal{D}^{\text{rrs}}$ is used in a QCDCL proof system, the resulting system is provably incomparable in strength to the QCDCL system that does not use dependency schemes. This was already known for the setting without cube-learning, Theorem 5 in [17]. We show here that cube learning makes no difference; the systems still remain incomparable. This highlights the fact that adding dependency schemes is not always beneficial.

► **Theorem 4.17.** *Any proof systems $\mathcal{P}_1, \mathcal{P}_2$ are incomparable, where*

$$\begin{aligned} \mathcal{P}_1 &\in \{\text{QCDCL}^{\text{LEV-ORD}}(\mathcal{D}^{\text{trv}}, \text{CubePol}) \mid \text{CubePol} \in \{\text{No-Cube}, \text{Cube-LD}, \text{Cube-}\mathcal{D}^{\text{rrs}}\}\} \text{ and} \\ \mathcal{P}_2 &\in \left\{ \mathcal{D}_1 + \text{QCDCL}^{\text{LEV-ORD}}(\mathcal{D}_2, \text{CubePol}) \mid \begin{array}{l} (\mathcal{D}_1, \mathcal{D}_2) \in \{(\mathcal{D}^{\text{trv}}, \mathcal{D}^{\text{rrs}}), (\mathcal{D}^{\text{rrs}}, \mathcal{D}^{\text{trv}}), (\mathcal{D}^{\text{rrs}}, \mathcal{D}^{\text{rrs}})\}, \\ \text{CubePol} \in \{\text{Cube-LD}, \text{Cube-}\mathcal{D}_2\} \end{array} \right\}. \end{aligned}$$

Proof. There are eighteen incomparability claims expressed so thirty-six separations are required! Fortunately, just two formulas establish all the desired separations.

The bounds from [17] along with Observation 3.8 and Observation 3.9 imply that the **Dep-Trap** formulas require exponential size refutations in every system in $\mathcal{P}_2$, but have polynomial size refutations in every system in $\mathcal{P}_1$. The bounds from [7] and [17] along with Observation 3.8 imply that the **TwinEq** formulas require exponential size refutations in every system in $\mathcal{P}_1$, but have polynomial size refutations in every system in $\mathcal{P}_2$. ◀

The next theorem shows that adding $\mathcal{D}^{\text{rrs}}$ in different ways to QCDCL yields systems incomparable in strength. (Again, this was already known for the setting without cube-learning, Theorem 4 in [17].) This highlights that adding dependency schemes in any one of preprocessing or propagation and learning is not inherently better than the other.

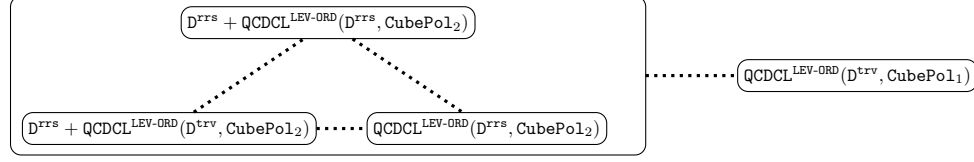
Formula	Matrix	Dependencies	Weakest System where Easy	Strongest System where Hard
Equality ([6]-Def. 3.1)	Sat	$D^{trs} = \emptyset$ $D^{std} = D^{trv}$	$QCDCL_{LEV-ORD}(D^{trv}, \text{Cube-LD})$ [15]-Prop 5.2 $QCDCL_{LEV-ORD}(D^{trv}, \text{Cube-D}^{trv})$ [15]-Prop 5.2 $QCDCL$ with D^{trs} [17]-Lemma 1 $QCDCL_{LEV-ORD}(D^{std}, \text{Cube-LD})$ $QCDCL_{LEV-ORD}(D^{trv}, \text{Cube-D}^{std})$	$QCDCL_{LEV-ORD}(D^{trv}, \text{No-Cube})$ [7]-Cor 5.8 $QCDCL_{LEV-ORD}(D^{std}, \text{No-Cube})$
TwinEq ([15]-Def 6.1)	Sat	$D^{trs} = \emptyset$ $D^{std} = D^{trv}$	$QCDCL$ with D^{trs} [17]-Sec 4.9	$QCDCL_{LEV-ORD}(D^{trv}, \text{Cube-LD})$ [15]-Prop 6.3
Trapdoor ([7]-Def 4.5)	Unsat	$D^{trs} = \emptyset$ $D^{std} = \{(w, t)\}$	$QCDCL$ with D^{trs} [17]-Lemma 2	$QCDCL_{LEV-ORD}(D^{trv}, \text{Cube-LD})$ (Obs. 3.9) $QCDCL_{LEV-ORD}(D^{trv}, \text{Cube-D}^{trv})$ (and [7]-Prop 4.6) $QCDCL_{LEV-ORD}(D^{std}, \text{Cube-LD})$ $QCDCL_{LEV-ORD}(D^{std}, \text{Cube-D}^{std})$
Dep-Trap ([17]-Sec 4.4)	Unsat	$D^{trs} = \{(w, t)\}$ [17] $D^{std} = \{(w, t)\} \cup \{(u, x_i)\}$	$QCDCL_{LEV-ORD}(D^{trv}, \text{No-Cube})$ [17]-Lemma 3 $QCDCL_{LEV-ORD}(D^{std}, \text{No-Cube})$	$QCDCL_{cube}$ with D^{trs} ([17], Obs. 3.9)
TwoPHPandCT ([17]-Sec 4.5)	Unsat	$D^{trs} = \emptyset$ [17]	$D^{trs} + QCDCL_{LEV-ORD}(D^{trv}, \text{No-Cube})$ [17]-Lemma 5 $QCDCL_{LEV-ORD}(D^{trs}, \text{No-Cube})$ [17]	$QCDCL_{LEV-ORD}(D^{trv}, \text{Cube-LD})$ ([17], Obs. 3.9) $QCDCL_{LEV-ORD}(D^{trv}, \text{Cube-D}^{trv})$ ([17], Obs. 3.9) $QCDCL_{LEV-ORD}(D^{trs}, \text{Cube-LD})$ ([17], Obs. 3.9) $QCDCL_{LEV-ORD}(D^{trs}, \text{Cube-D}^{trs})$ ([17], Obs. 3.9)
PreDepTrap ([17]-Sec 4.7)	Sat $\text{red-}D^{trs}; \text{Unsat}$	$D^{trs} = \{(w, t)\}$ [17]	$QCDCL$ [17] $QCDCL(D^{trs})$ [17]	$D^{trs} + QCDCL_{cube}([17], \text{Obs. 3.9})$ $D^{trs} + QCDCL_{cube}(D^{trs})([17], \text{Obs. 3.9})$
PropDep-Trap ([17]-Sec 4.8)	Unsat	$D^{trs} = \{(w, t), (b_1, z_1), (b_2, z_2)\}$ [17]	$QCDCL_{LEV-ORD}(D^{trv}, \text{No-Cube})$ ([17]) $D^{trs} + QCDCL_{LEV-ORD}(D^{trv}, \text{No-Cube})$ ([17])	$QCDCL_{LEV-ORD}(D^{trs}, \text{Cube-LD})$ ([17], Obs. 3.9) $QCDCL_{LEV-ORD}(D^{trs}, \text{Cube-D}^{trs})$ ([17], Obs. 3.9)
*DoubleLongEq	Sat	$D^{trs} = D^{trv}$ (Proposition 4.3)	$QCDCL_{LEV-ORD}(D^{trv}, \text{Cube-LD})$ (Lemma 4.4) $QCDCL_{cube}$ with D^{trs} (Lemma 4.4)	$QCDCL$ (Lemma 4.6)
*PreRRSTrapdoor	Unsat	$D^{trs} = \{(u, b), (v, b), (p, q)\}$ (Proposition 4.10)	$D^{trs} + QCDCL_{LEV-ORD}(D^{trv}, \text{No-Cube})$ ([17])	$QCDCL$ with D^{trs} (Lemma 4.6)
*StdDepTrap	Sat	$D^{std} = \{(u, x), (w_1, e_1), (w_2, e_2)\}$ (Proposition 4.13)	$QCDCL_{LEV-ORD}(D^{std}, \text{No-Cube})$ (Lemma 4.14)	$QCDCL_{LEV-ORD}(D^{trv}, \text{Cube-LD})$ (Lemma 4.15)

Table 1 Formulas, their dependencies, and ease/hardness of refutations.

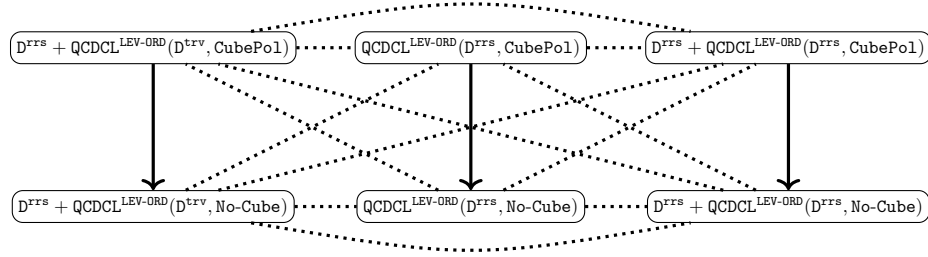
" $QCDCL$ with D^{trs} " includes $\{D^{trs} + QCDCL_{LEV-ORD}(D, \text{No-Cube})\}$ where $D \in \{D^{trv}, D^{trs}\}$, as well as $QCDCL_{LEV-ORD}(D^{trs}, \text{No-Cube})$.

" $QCDCL_{cube}$ with D^{trs} " includes $D^{trs} + QCDCL_{LEV-ORD}(D, \text{CubePo1})$ as well as $QCDCL_{LEV-ORD}(D^{trs}, \text{CubePo1})$,

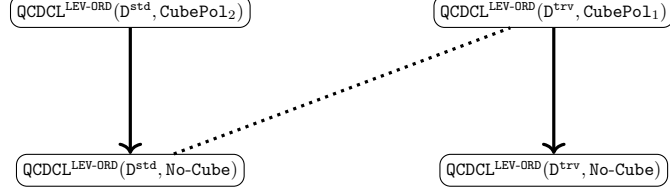
where $D \in \{D^{trv}, D^{trs}\}$ and $\text{CubePo1} \in \{\text{Cube-LD}, \text{Cube-D}^{trs}\}$.



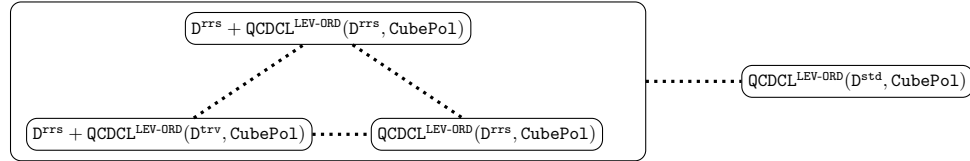
(a) For $\text{CubePol}_1 \in \{\text{No-Cube}, \text{Cube-LD}, \text{Cube-D}^{trv}\}$; $\text{CubePol}_2 \in \{\text{Cube-LD}, \text{Cube-D}^{rrs}\}$.
From Theorems 4.17 and 4.18.



(b) For $\text{CubePol}_1 \in \{\text{Cube-LD}, \text{Cube-D}^{rrs}\}$.
From Theorems 4.16 and 4.19.



(c) For $\text{CubePol}_1 \in \{\text{Cube-LD}, \text{Cube-D}^{trv}\}$, $\text{CubePol}_2 \in \{\text{Cube-LD}, \text{Cube-D}^{std}\}$.
From Theorems 4.16 and 4.20.



(d) For $\text{CubePol}_1 \in \{\text{No-Cube}, \text{Cube-LD}, \text{Cube-D}^{std}\}$. From Theorem 4.21.

■ **Figure 1** The simulation order of various QCDCL systems.
 $A \rightarrow B$ means A simulates B but B does not simulate A .
 $A \cdots B$ means neither A nor B simulates the other.

► **Theorem 4.18.** *For a fixed $\text{CubePol} \in \{\text{Cube-LD}, \text{Cube-D}^{\text{rrs}}\}$, the three systems $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{CubePol})$, $\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{CubePol})$, and $\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{CubePol})$ are pairwise incomparable.*

Proof. Refer to Table 1 and Observation 3.8 and Observation 3.9.

It can be seen that **TwoPHPandCT** formulas are easy to refute when D^{rrs} is used in preprocessing i.e. in $\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{CubePol})$ and $\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{CubePol})$, but hard otherwise i.e. $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{CubePol})$. On the other hand, the **PreDepTrap** formulas are hard to refute if preprocessed by D^{rrs} , but easy otherwise. Together, they witness that $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{CubePol})$ is incomparable with $\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{CubePol})$ and $\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{CubePol})$.

Further, the **PropDep-Trap** formulas are easy to refute if the propagations and clause learning do not use D^{rrs} , but become hard if D^{rrs} is used. This is independent of whether preprocessing is used and whether cube-learning is switched on. On the other hand, the (new) **PreRRSTrapdoor** formulas show that using D^{rrs} in propagations can be advantageous. Together, these two formulas witness $\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{CubePol})$ and $\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{CubePol})$ are incomparable. ◀

Earlier, we have seen that adding cube-learning to a QCDCL system with dependency always yields a stronger system, Theorem 4.16. The following theorem shows that even adding cube-learning to a QCDCL system using D^{rrs} is incomparable to a QCDCL system without cube-learning but using D^{rrs} in a different way.

► **Theorem 4.19.** *For any $\text{CubePol} \in \{\text{Cube-LD}, \text{Cube-D}^{\text{rrs}}\}$, adding D^{rrs} to the proof system $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{CubePol})$ in one way and to $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{No-Cube})$ in any different way yields incomparable proof systems. In particular,*

1. $\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{CubePol})$ is incomparable with $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{No-Cube})$ and $\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{No-Cube})$.
2. $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{CubePol})$ is incomparable with $\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{No-Cube})$ and $\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{No-Cube})$.
3. $\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{CubePol})$ is incomparable with $\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{No-Cube})$ and $\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{No-Cube})$.

Proof. Refer to Table 1 and Observation 3.8 and Observation 3.9.

The **TwoPHPandCT** formulas are easy to refute if and only if preprocessed by D^{rrs} , irrespective of whether or not cube-learning is used and whether or not D^{rrs} is used in propagation and learning. The situation is exactly reversed for the **PreDepTrap** formulas, which are easy to refute if and only if not preprocessed by D^{rrs} . Together, these show eight of the twelve claimed incomparability relations, namely

$$\begin{aligned} &\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{CubePol}) \text{ and } \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{No-Cube}), \\ &\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{CubePol}) \text{ and } \text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{No-Cube}), \\ &\text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{CubePol}) \text{ and } \text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{No-Cube}), \\ &\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{CubePol}) \text{ and } \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{No-Cube}). \end{aligned}$$

The **PropDep-Trap** formulas are easy to refute if and only if D^{rrs} is not used for propagation and learning, irrespective of its use in preprocessing, and irrespective of whether or not cube-learning is used. On the other hand, the **PreRRSTrapdoor** formulas are easy to refute if D^{rrs} is used for preprocessing and in propagation and learning, but not if it is used only for preprocessing. Together, these show the remaining four claimed incomparability relations, namely

$$\begin{aligned} &\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{CubePol}) \text{ and } \text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{No-Cube}) \\ &\text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{rrs}}, \text{CubePol}) \text{ and } \text{D}^{\text{rrs}} + \text{QCDCL}^{\text{LEV-ORD}}(\text{D}^{\text{trv}}, \text{No-Cube}). \end{aligned} \quad \blacktriangleleft$$

Now we come to D^{std} . First, we show that, as for D^{rrs} , a QCDCL system with D^{std} is incomparable in strength to a standard QCDCL system without any dependency schemes.

► **Theorem 4.20.** *For $\text{CubePol} \in \{\text{Cube-LD}, \text{Cube-}D^{\text{std}}\}$, the proof systems $\text{QCDCL}^{\text{LEV-ORD}}(D^{\text{std}}, \text{No-Cube})$ and $\text{QCDCL}^{\text{LEV-ORD}}(D^{\text{trv}}, \text{CubePol})$ are incomparable.*

Proof. For $\text{CubePol} \in \{\text{Cube-LD}, \text{Cube-}D^{\text{std}}\}$, the Equality formulas have polynomial size $\text{QCDCL}^{\text{LEV-ORD}}(D^{\text{trv}}, \text{CubePol})$ refutations (the refutations in [15] are of this type), but require exponential size $\text{QCDCL}^{\text{LEV-ORD}}(D^{\text{std}}, \text{No-Cube})$ refutations (the lower bound from [7] carries over, because $D^{\text{std}} = D^{\text{trv}}$). On the other hand, the newly defined **StdDepTrap** formulas have polynomial size $\text{QCDCL}^{\text{LEV-ORD}}(D^{\text{std}}, \text{No-Cube})$ refutations but require exponential size $\text{QCDCL}^{\text{LEV-ORD}}(D^{\text{trv}}, \text{CubePol})$ refutations (Lemmas 4.14 and 4.15). ◀

Finally, we compare the different dependency schemes D^{rrs} and D^{std} . The mere fact that D^{rrs} is a refinement of D^{std} (more general, eliminates more dependencies) does not make it better; for that matter, D^{rrs} is a refinement of D^{trv} , but using it can be a disadvantage for some formulas. Similarly, we prove below that neither of D^{rrs} and D^{std} has a proof-theoretic advantage over the other, irrespective of the presence or absence of cube-learning.

► **Theorem 4.21.** *For any $(D_1, D_2) \in \{(D^{\text{trv}}, D^{\text{rrs}}), (D^{\text{rrs}}, D^{\text{trv}}), (D^{\text{rrs}}, D^{\text{rrs}})\}$, and $\text{CubePol} \in \{\text{No-Cube}, \text{Cube-LD}, \text{Cube-}D^{\text{std}}\}$, the proof systems $P_1 \in \text{QCDCL}^{\text{LEV-ORD}}(D^{\text{std}}, \text{CubePol})$ and $P_2 \in D_1 + \text{QCDCL}^{\text{LEV-ORD}}(D_2, \text{CubePol})$ are incomparable.*

Proof. Using refutations and lower bound arguments from [7, 17] along with Observation 3.8 and Observation 3.9, the **Trapdoor** and **Dep-Trap** formulas bear witness; the former are easy in P_2 but hard in P_1 , whereas the situation is reversed for the latter. ◀

5 Conclusions

In the context of QBF proof systems, dependency schemes are expected to aid the process of refutation. Indeed, in the proof systems **Q-Res** and **QU-Res**, two of the earliest resolution-based QBF proof systems to be studied [20, 18], it is known that using D^{rrs} can shorten proofs exponentially – the Equality formulas require exponential refutation size in these but have polynomial-sized proofs in $Q(D^{\text{rrs}})\text{-Res}$. It was thus a surprise to see this advantage does not automatically translate to QCDCL algorithms; we see that when restricted to **LEV-ORD** decisions, even in the presence of cube learning, usage of dependency schemes for propagation and learning is not always advantageous. The hardness in these systems is primarily a consequence of the level-ordered nature of decisions – it can be shown that all formulas with lower bounds shown in Section 4 are easy when the decision policy for QCDCL is $D^{\text{std-ORD}}$ or $D^{\text{rrs-ORD}}$. In fact, proper lower bound techniques for a decision policy $D\text{-ORD}$ are unknown, and to us, this is the big open question arising from this work.

Even for the level-ordered policy, there are many unresolved questions. The comparison between D^{trv} and D^{rrs} already shows that using a more refined scheme is not necessarily an advantage. The relative strengths of D^{std} and D^{rrs} (which refines D^{std}) is not yet clear. How other dependency schemes would relate in this scenario is completely unexplored.

Since QCDCL-style reasoning is explained through long-distance clause/term resolution, it is worth highlighting three long-standing open questions about those systems. Firstly, does using dependency schemes confer any advantage with clausal long-distance; i.e. is the simulation of **LDQ-Res** by **LDQ(D)-Res** strict for D^{std} or D^{rrs} ? Secondly, is the use of dependency schemes in long-distance term resolution (the system **LDQ-TermRes**) sound? Thirdly, since D^{std} is the scheme actually used in **DepQBF**, separations specific to D^{std} would be quite

interesting. Can we even show that $Q(D^{\text{std}})\text{-Res}$ is strictly more powerful than $Q\text{-Res}$? This question has remained open since D^{std} was first implemented in DepQBF over 15 years ago.

References

- 1 Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *J. Artif. Intell. Res.*, 40:353–373, 2011. doi:10.1613/jair.3152.
- 2 Salman Azhar, Gary Peterson, and John Reif. Lower bounds for multiplayer non-cooperative games of incomplete information. *Journal of Computers and Mathematics with Applications*, 41:957 – 992, 2001.
- 3 Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF certification and its applications. *Formal Methods Syst. Des.*, 41(1):45–65, 2012. doi:10.1007/s10703-012-0152-6.
- 4 Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *J. Artif. Intell. Res. (JAIR)*, 22:319–351, 2004. doi:10.1613/jair.1410.
- 5 Olaf Beyersdorff. Proof complexity of quantified Boolean logic – a survey. In Marco Benini, Olaf Beyersdorff, Michael Rathjen, and Peter Schuster, editors, *Mathematics for Computation (M4C)*, pages 353–391. World Scientific, 2022.
- 6 Olaf Beyersdorff, Joshua Blinkhorn, and Luke Hinde. Size, cost, and capacity: A semantic technique for hard random QBFs. *Log. Methods Comput. Sci.*, 15(1), 2019. doi:10.23638/LMCS-15(1:13)2019.
- 7 Olaf Beyersdorff and Benjamin Böhm. Understanding the relative strength of QBF CDCL solvers and QBF resolution. *Logical Methods in Computer Science*, 19(2), 2023. preliminary version in ITCS 2021. doi:10.46298/lmcs-19(2:2)2023.
- 8 Olaf Beyersdorff, Benjamin Böhm, and Meena Mahajan. Runtime vs. extracted proof size: an exponential gap for CDCL on QBFs. In *Proceedings of 38th AAAI Conference on Artificial Intelligence, 2024*, 2024.
- 9 Olaf Beyersdorff, Leroy Chew, and Mikolás Janota. New resolution-based QBF calculi and their proof complexity. *ACM Trans. Comput. Theory*, 11(4):26:1–26:42, 2019. doi:10.1145/3352155.
- 10 Joshua Blinkhorn and Olaf Beyersdorff. Shortening QBF proofs with dependency schemes. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 263–280. Springer, 2017. doi:10.1007/978-3-319-66263-3_17.
- 11 Joshua Blinkhorn and Olaf Beyersdorff. Dynamic dependency awareness for QBF. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 5224–5228. ijcai.org, 2018. doi:10.24963/ijcai.2018/726.
- 12 Joshua Blinkhorn, Tomás Peitl, and Friedrich Slivovsky. Davis and Putnam meet Henkin: Solving DQBF with resolution. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, volume 12831 of *Lecture Notes in Computer Science*, pages 30–46. Springer, 2021. doi:10.1007/978-3-030-80223-3_4.
- 13 Benjamin Böhm and Olaf Beyersdorff. Lower bounds for QCDCL via formula gauge. *J. Autom. Reason.*, 67(4):35, 2023. preliminary version in SAT 2021. doi:10.1007/s10817-023-09683-1.
- 14 Benjamin Böhm and Olaf Beyersdorff. QCDCL vs QBF resolution: Further insights. *J. Artif. Intell. Res.*, 81:741–769, 2024. preliminary version in SAT 2023. doi:10.1613/jair.1.15522.
- 15 Benjamin Böhm, Tomás Peitl, and Olaf Beyersdorff. QCDCL with cube learning or pure literal elimination - what is best? *Artif. Intell.*, 336:104194, 2024. preliminary version on IJCAI 2022. doi:10.1016/j.artint.2024.104194.

- 16 Benjamin Böhm, Tomás Peitl, and Olaf Beyersdorff. Should decisions in QCDCL follow prefix order? *J. Autom. Reason.*, 68(1):5, 2024. doi:10.1007/s10817-024-09694-6.
- 17 Abhimanyu Choudhury and Meena Mahajan. Dependency schemes in CDCL-based QBF solving: A proof-theoretic study. *J. Autom. Reason.*, 68(3):16, 2024. preliminary version in FSTTCS 2023. doi:10.1007/s10817-024-09707-4.
- 18 Allen Van Gelder. Contributions to the theory of practical quantified boolean formula solving. In Michela Milano, editor, *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, pages 647–663. Springer, 2012. doi:10.1007/978-3-642-33558-7_47.
- 19 Enrico Giunchiglia, Paolo Marin, and Massimo Narizzano. Reasoning with quantified boolean formulas. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 761–780. IOS Press, 2009. doi:10.3233/978-1-58603-929-5-761.
- 20 Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified boolean formulas. *Inf. Comput.*, 117(1):12–18, 1995. doi:10.1006/inco.1995.1025.
- 21 Florian Lonsing. *Dependency schemes and search-based QBF solving: theory and practice*. PhD thesis, Johannes Kepler University, Linz, Austria, 2012.
- 22 Florian Lonsing and Armin Biere. DepQBF: A dependency-aware QBF solver. *J. Satisf. Boolean Model. Comput.*, 7(2-3):71–76, 2010. doi:10.3233/sat190077.
- 23 Florian Lonsing and Uwe Egly. DepQBF 6.0: A search-based QBF solver beyond traditional QCDCL. In *Proceedings of International Conference on Automated Deduction (CADE)*, pages 371–384, 2017. doi:10.1007/978-3-319-63046-5_23.
- 24 João P. Marques Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications. IOS Press, 2021. doi:10.3233/FAIA200987.
- 25 Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Dependency learning for QBF. *J. Artif. Intell. Res.*, 65:180–208, 2019. doi:10.1613/jair.1.11529.
- 26 Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Long-distance Q-resolution with dependency schemes. *J. Autom. Reasoning*, 63(1):127–155, 2019. doi:10.1007/s10817-018-9467-3.
- 27 Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175(2):512–525, 2011. doi:10.1016/j.artint.2010.10.002.
- 28 Marko Samer and Stefan Szeider. Backdoor sets of quantified boolean formulas. *J. Autom. Reasoning*, 42(1):77–97, 2009. doi:10.1007/s10817-008-9114-5.
- 29 Ankit Shukla, Armin Biere, Luca Pulina, and Martina Seidl. A survey on applications of quantified Boolean formulas. In *Proc. IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 78–84, 2019. doi:10.1109/ICTAI.2019.00020.
- 30 João P. Marques Silva and Kareem A. Sakallah. GRASP - a new search algorithm for satisfiability. In Rob A. Rutenbar and Ralph H. J. M. Otten, editors, *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, San Jose, CA, USA, November 10-14, 1996*, pages 220–227. IEEE Computer Society / ACM, 1996. doi:10.1109/ICCAD.1996.569607.
- 31 Friedrich Slivovsky and Stefan Szeider. Soundness of Q-resolution with dependency schemes. *Theor. Comput. Sci.*, 612:83–101, 2016. doi:10.1016/j.tcs.2015.10.020.
- 32 Moshe Y. Vardi. Boolean satisfiability: theory and engineering. *Communications of the ACM*, 57(3):5, 2014. doi:10.1145/2578043.
- 33 Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified boolean satisfiability solver. In Lawrence T. Pileggi and Andreas Kuehlmann, editors, *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design, ICCAD 2002, San Jose, California, USA, November 10-14, 2002*, pages 442–449. ACM / IEEE Computer Society, 2002. doi:10.1145/774572.774637.

A Normal Dependency Schemes.

► **Definition A.1.** A dependency scheme D is said to be a monotone dependency scheme if $D(\phi[\tau]) \subseteq D(\phi)$ for every PCNF formula ϕ and assignment τ to subset $\text{var}(\phi)$.

► **Definition A.2.** A dependency scheme D is said to be a simple dependency scheme if for every PCNF formula $\Phi = \forall XQ.\phi$, every LDQ(D) derivation P from Φ , for every $u \in X$ either u or $\bar{u}$ do not appear in P .

► **Definition A.3.** A dependency scheme D is said to be a normal dependency scheme [26] if:

- D is a Monotone dependency scheme
- D is a Simple dependency scheme

D^{trv} , D^{std} , D^{rrs} are all normal dependency schemes.

B Rules in various QBF Proof systems

For an initial PCNF and a fixed dependency scheme D , rules for various proofs systems are defined as follows.

For clauses:

- **Axiom** (clause axiom rule) $\frac{}{A}$ where A is any clause in the matrix.
- **red-D** (clause reduction with D): $\frac{A \vee \ell}{A}$
where $\text{var}(\ell) \in X_\forall$, and for each $x \in X_\exists \cap \text{var}(A)$, $(\text{var}(\ell), x) \notin D$.
- **Res** (Resolution): $\frac{A \vee \ell \quad B \vee \neg \ell}{A \vee B}$
where $\text{var}(\ell) \in X_\exists$, and $A \vee B$ is not tautological.
- **LDRes(D)** (Long-distance Resolution with D): $\frac{A \vee \ell \quad B \vee \neg \ell}{A \vee B}$
where $\text{var}(\ell) \in X_\exists$; for each $x \in X_\exists$, either x or $\neg x$ is not in $A \cup B$; for each $u \in X_\forall$, if $u \in A$ and $\neg u \in B$, then $(u, \text{var}(\ell)) \notin D$.
(Note that tautological clauses can be generated. In much of the literature, the symbol u^* is used to denote that both u and $\neg u$ are in a clause.)

The proof system Q(D)-Res uses the rules **Axiom**, **red-D**, and **Res**; the proof system LDQ(D)-Res uses the rules **Axiom**, **red-D**, and **LDRes(D)**. The proof systems LDQ-Res and Q-Res are the special cases of LDQ(D)-Res and Q(D)-Res where $D = D^{\text{trv}}$. A refutation of a false QBF is a derivation of the empty clause using the permitted rules.

For cubes/terms, the situation is essentially dual, exchanging the roles of $X_\exists$ and $X_\forall$, to give the rules **red-D $_\exists$** (term reduction with D), **TermRes** (Term Resolution), and **LDTRes(D)** (Long-distance Term Resolution with D). Also, the Axiom rule is modified to the term axiom rule: $\frac{}{A}$ where A is a non-contradictory cube whose literals satisfy the matrix.

The proof system Q(D)-TermRes uses the rules **Axiom**, **red-D $_\exists$** , and **TermRes**; the proof system LDQ(D)-TermRes uses the rules **Axiom**, **red-D $_\exists$** , and **LDTRes(D)**. The proof systems Q-TermRes and LDQ-TermRes are their respective special cases where $D = D^{\text{trv}}$. A verification of a true QBF is a derivation of the empty term using the permitted rules.