Long-Distance Resolution: Proof Generation and Strategy Extraction in Search-Based QBF Solving ⋆

Uwe Egly, Florian Lonsing, and Magdalena Widl

Institute of Information Systems, Vienna University of Technology, Austria
http://www.kr.tuwien.ac.at/staff/{egly,lonsing,widl}

Abstract. Strategies (and certificates) for quantified Boolean formulas (QBFs) are of high practical relevance as they facilitate the verification of results returned by QBF solvers and the generation of solutions to problems formulated as QBFs. State of the art approaches to obtain strategies require traversing a Q-resolution proof of a QBF, which for many real-life instances is too large to handle. In this work, we consider the long-distance Q-resolution (LDQ) calculus, which allows particular tautological resolvents. We show that for a family of QBFs using the LDQ-resolution allows for exponentially shorter proofs compared to Q-resolution. We further show that an approach to strategy extraction originally presented for Q-resolution proofs can also be applied to LDQ-resolution proofs. As a practical application, we consider search-based QBF solvers which are able to learn tautological clauses based on resolution and the conflict-driven clause learning method. We prove that the resolution proofs produced by these solvers correspond to proofs in the LDQ calculus and can therefore be used as input for strategy extraction algorithms. Experimental results illustrate the potential of the LDQ calculus in search-based QBF solving.

1 Introduction

The development of decision procedures for quantified Boolean formulas (QBFs) has recently resulted in considerable performance gains. A common approach is search-based QBF solving with conflict-driven clause learning (CDCL) [2,11], which is related to propositional logic (SAT solving) in that it extends the DPLL algorithm [3]. In addition to solving the QBF decision problem, QBF solvers with clause learning are able to produce Q-resolution proofs [7] to certify their answer. These proofs can be used to obtain solutions to problems encoded as QBFs. For instance, if the QBF describes a synthesis problem, then the system to be synthesized can be generated by inspecting the proof [12]. Such solutions can be represented as control strategies [6] expressed by an algorithm that computes assignments to universal variables rendering the QBF false, or as control circuits [1] expressed by Herbrand or Skolem functions. Both approaches can be used for true QBFs as well as false QBFs. In this work, we focus on false QBFs.

Strategies can be extracted from Q-resolution proofs of false QBFs based on a game-theoretic view [6]. A strategy is an assignment to each universal (∀) variable that depends on assignments to existential (∃) variables with a lower quantification level and maintains

⋆ This work was supported by the Austrian Science Fund (FWF) under grant S11409-N23 and by the Vienna Science and Technology Fund (WWTF) through project ICT10-018.

the falsity of the QBF. The extraction algorithm is executed for each quantifier block from the left to the right. Certificate extraction [1] constructs solutions in terms of a Herbrand function for each $\forall$ variable from a Q-resolution proof of a false QBF. Replacing each $\forall$ variable by its Herbrand function in the formula and removing the quantifiers yields an unsatisfiable propositional formula. The run time of both extraction algorithms is polynomially related to the size of the proof. It is therefore beneficial to have short proofs in practice. Calculi more powerful than traditional Q-resolution possibly allow for shorter proofs.

A way to strengthen Q-resolution is to admit tautological resolvents under certain conditions. In search-based QBF solving, this approach called *long-distance resolution* is applied to learn tautological clauses in CDCL [15,16]. The idea of generating tautological resolvents is formalized in the *long-distance Q-resolution calculus (LDQ)* [1].

We consider long-distance resolution in search-based QBF solving. First, we show that formulas of a certain family of QBFs [7] have LDQ-resolution proofs of polynomial size in the length of the formula. According to [7], any Q-resolution proof of these formulas is exponential.

Second, we prove that long-distance resolution for clause learning in QBF solvers as presented in [15,16] corresponds to proof steps in the formal LDQ-resolution calculus.This observation is complementary to the correctness proof of learning tautological clauses (i.e. the theorem in [15]) in that we embed this practical clause learning procedure in the formal framework of the LDQ-resolution calculus. Thereby we obtain a generalized view on the practical side and the theoretical side of long-distance resolution in terms of the clause learning procedure and the formal calculus, respectively.

Third, we prove that strategy extraction [6] is applicable to LDQ-resolution proofs in the same way as it is to Q-resolution proofs, which have been its original application.

Our results show that the complete workflow from generating proofs in search-based QBF solving to extracting strategies from these proofs can be based on the LDQ-resolution calculus. We modified the search-based QBF solver DepQBF [9] to learn tautological clauses in CDCL.We report preliminary experimental results which illustrate the potential of the LDQ-resolution calculus in terms of a lower effort in the search process. Since LDQ-resolution proofs can be significantly shorter than Q-resolution proofs, strategy extraction will also benefit from applying LDQ-resolution proofs in practice.

2 Preliminaries

Given a set $\mathcal{V}$ of Boolean variables, the set $L := \mathcal{V} \cup \{\overline{v} \mid v \in \mathcal{V}\}$ of literals contains each variable in its positive and negative polarity. We write $\overline{v}$ for the opposite polarity of v regardless of whether v is positive or negative. A quantified Boolean formula (QBF) in prenex conjunctive normal form (PCNF) over a set $\mathcal{V}$ of variables and the two quantifiers $\exists$ and $\forall$ is of the form $\mathcal{P}.\phi$ where (1) $\mathcal{P} := Q_1 v_1 ... Q_n v_n$ is the *prefix* with $Q_i \in \{\exists, \forall\}$, $v_i \in \mathcal{V}$ and all v_i are pairwise distinct for $1 \leq i \leq n$, and (2) $\phi \subseteq 2^L$ is a set of clauses called the *matrix*. We write $\square$ for the empty clause and occasionally represent a clause as disjunction of literals. A *quantifier block* of the form QV combines all subsequent variables with the same quantifier Q in set V. A prefix can be alternatively written as sequence $Q_1 V_1 ... Q_m V_m$ of quantifier blocks where $Q_i \neq Q_{i+1}$ for $1 \leq i \leq m-1$.

We use $\mathsf{var}(l)$ to refer to the variable of literal l and $\mathsf{vars}(\phi)$ to refer to the set of variables used in the matrix. A QBF is *closed* if $\mathsf{vars}(\phi) = \{v_i \mid 1 \leq i \leq n\}$, i.e., if all variables used in the matrix are quantified and vice versa. In the following, by QBF we refer to a closed QBF in PCNF. Function $\mathsf{lev} : L \to \{1,...,m\}$ refers to the quantification level of a literal and $\mathsf{quant} : L \to \{\exists,\forall\}$ refers to the quantifier type of a literal such that for $1 \leq i \leq m$ and any $l \in L$, if $\mathsf{var}(l) \in V_i$ then $\mathsf{lev}(l) := i$ and $\mathsf{quant}(l) := Q_i$. A literal l is *existential* if $\mathsf{quant}(l) = \exists$ and *universal* if $\mathsf{quant}(l) = \forall$. We use the same terms for the variable $\mathsf{var}(l)$ of l. We write e for an existential variable and x for a universal variable.

Given a QBF $\psi = \mathcal{P}.\phi$, an *assignment* is a mapping of variables in $\mathsf{vars}(\phi)$ to the truth values *true* ($\top$) and *false* ($\bot$). We denote an assignment as a set σ of literals such that if $\mathsf{var}(l)$ is assigned to $\top$ then $l \in \sigma$, and if $\mathsf{var}(l)$ is assigned to $\bot$ then $\overline{l} \in \sigma$. An assignment is total if for each $v \in \mathsf{vars}(\phi)$ either $v \in \sigma$ or $\overline{v} \in \sigma$, and partial otherwise.

A *clause C under an assignment σ* is denoted by $C_{\lceil\sigma}$ and is defined as follows: $C_{\lceil\sigma} = \top$ if $C \cap \sigma \neq \emptyset$, $C_{\lceil\sigma} = \bot$ if $C \setminus \{v \mid \overline{v} \in \sigma\} = \emptyset$, and $C_{\lceil\sigma} = C \setminus \{v \mid \overline{v} \in \sigma\}$ otherwise. A *QBF $\psi = \mathcal{P}.\phi$ under an assignment σ* is denoted by $\psi_{\lceil\sigma}$ and is obtained from ψ by replacing each $C \in \phi$ by $C_{\lceil\sigma}$, eliminating truth constants $\top$ and $\bot$ by standard rewrite rules from Boolean algebra, removing for each literal in σ its variable from the prefix, and removing each quantifier block that does no longer contain any variable.

The QBF $\forall v.\phi$ is true if and only if $\phi_{\lceil\{v\}} = \top$ and $\phi_{\lceil\{\overline{v}\}} = \top$. The QBF $\exists v.\phi$ is true if and only if $\phi_{\lceil\{v\}} = \top$ or $\phi_{\lceil\{\overline{v}\}} = \top$. The QBF $\forall v\mathcal{P}.\phi$ is true if and only if $\mathcal{P}.\phi_{\lceil\{v\}}$ and $\mathcal{P}.\phi_{\lceil\{\overline{v}\}}$ are true . The QBF $\exists v\mathcal{P}.\phi$ is true if and only if $\mathcal{P}.\phi_{\lceil\{v\}}$ or $\mathcal{P}.\phi_{\lceil\{\overline{v}\}}$ is true.

Given a QBF $\psi = \mathcal{P}.\phi$ and a clause $C \in \phi$, *universal reduction* [7] produces the clause $C' := \mathsf{reduce}(C) := C \setminus \{l \mid \mathsf{quant}(l) = \forall \text{ and } \forall e \in C : \mathsf{quant}(e) = \exists \to \mathsf{lev}(e) < \mathsf{lev}(l)\}$ by removing all universal literals from C which have a maximal quantification level. Universal reduction on ψ produces the QBF resulting from ψ by the application of universal reduction to each clause in ϕ.

A clause C is *satisfied* under an assignment σ if $C_{\lceil\sigma} = \top$ and *falsified* under σ and universal reduction if $\mathsf{reduce}(C_{\lceil\sigma}) = \Box$.

Clause learning (Section 4) is based on restricted variants of *resolution*, which is defined as follows. Given two clauses C^l and C^r and a *pivot variable* p with $p \in C^l$ and $\overline{p} \in C^r$, resolution produces the *resolvent* $C := \mathsf{resolve}(C^l, p, C^r) := (C^l \setminus \{p\} \cup C^r \setminus \{\overline{p}\})$.

Long-distance (LD) resolution [15] is an application of resolution where the resolvent $C = \mathsf{resolve}(C^l, p, C^r)$ is *tautological*, i.e. $\{v, \overline{v}\} \subseteq C$ for some variable v. In contrast to [15], our definition of LD-resolution allows for unrestricted LD-resolution steps. LD-resolution in the context of [15] is restricted and hence sound, whereas its unrestricted variant is unsound, as pointed out in the following example.

Example 1. For the true QBF $\forall x \exists e.(x \vee \overline{e}) \wedge (\overline{x} \vee e)$, an erroneous LD-refutation is given by $C_1 := \mathsf{resolve}((x \vee \overline{e}), e, (\overline{x} \vee e)) = (x \vee \overline{x})$ and $C_2 := \mathsf{reduce}(C_1) = \Box$.

Q-resolution [7] is a restriction of resolution. Given two *non-tautological* clauses C^l and C^r and an *existential* pivot variable p, the *Q-resolvent* is defined as follows. Let $C' := \mathsf{resolve}(\mathsf{reduce}(C^l), p, \mathsf{reduce}(C^r))$ be the resolvent of the two universally reduced clauses C^l and C^r. If C' is non-tautological then $C := \mathsf{reduce}(C')$ is the Q-resolvent of C^l and C^r. Otherwise no Q-resolvent exists.

The *long-distance Q-resolution (LDQ) calculus* [1] extends Q-resolution by allowing *certain* tautological resolvents. The rules of this calculus amount to a restricted application of LD-resolution. In the following we reproduce the formal rules (LDQ-rules) of the LDQ-calculus [1] using our notation and definitions. We write p for an existential pivot variable, x for a universal variable, and x^* as shorthand for $x \vee \overline{x}$. We call x^* a *merged literal*. Further, X^l and X^r are sets of universal literals (merged or unmerged), such that for each $x \in X^l$ it holds that if x is not a merged literal then either $\overline{x} \in X^r$ or $x^* \in X^r$, and otherwise either of $x \in X^r$ or $\overline{x} \in X^r$ or $x^* \in X^r$. X^r does not contain any additional literals. X^* contains the merged literal of each literal in X^l. Symmetric rules are omitted.

$$[p]\,\frac{C^l \vee p \qquad C^r \vee \overline{p}}{C^l \vee C^r} \quad \text{for all } v \in C^l \text{ it holds that } \overline{v} \notin C^r \tag{r_1}$$

$$[p]\,\frac{C^l \vee p \vee X^l \qquad C^r \vee \overline{p} \vee X^r}{C^l \vee C^r \vee X^*} \quad \begin{array}{l}\text{for all } x \in X^r \text{ it holds that } \mathsf{lev}(p) < \mathsf{lev}(x)\\ \text{for all } v \in C^l \text{ it holds that } \overline{v} \notin C^r\end{array} \tag{r_2}$$

$$[x]\,\frac{C \vee x'}{C} \quad \begin{array}{l}\text{for } x' \in \{x, \overline{x}, x^*\} \text{ and}\\ \text{for all existential } e \in C \text{ it holds that } \mathsf{lev}(e) < \mathsf{lev}(x')\end{array} \tag{u_1}$$

Rule r_2 is a restricted application of $\mathsf{resolve}(C^l, p, C^r)$ in that a tautological clause can be derived only if literals occurring in both polarities are universal and have a higher quantification level than p. Rule u_1 extends universal reduction by removing x^*.

Given a QBF ψ, a *derivation* of a clause C is a sequence of applications of resolution and universal reduction to the clauses in ψ and to derived clauses resulting in C. If either only Q-resolution, only LD-resolution or only the LDQ-calculus is applied, then the derivation is a *(Q,LD,LDQ)-derivation*. A (Q,LD,LDQ)-derivation of the empty clause $\Box$ is a *(Q,LD,LDQ)-refutation* or *(Q,LD,LDQ)-proof*. Both Q-resolution and the LDQ-calculus are sound and refutationally complete proof systems for QBFs that do not contain tautological clauses [1,7]. Figure 1 shows an LDQ-refutation.

3 Short LDQ-Proofs for Hard Formulas

We argue that LDQ-resolution has the potential to shorten proofs of false QBFs by showing that the application of LDQ-resolution on QBFs of a particular family [7] results in proofs of polynomial size.

A formula φ_t in this family $(\varphi_t)_{t \geq 1}$ of QBFs has the quantifier prefix

$$\exists d_0 d_1 e_1 \forall x_1 \exists d_2 e_2 \forall x_2 \exists d_3 e_3 \ldots \forall x_{t-1} \exists d_t e_t \forall x_t \exists f_1 \ldots f_t$$

and a matrix consisting of the following clauses:

$$\begin{array}{llll}
C_0 = \overline{d}_0 & C_1 = d_0 \vee \overline{d}_1 \vee \overline{e}_1 & \\
C_{2j} = d_j \vee \overline{x}_j \vee \overline{d}_{j+1} \vee \overline{e}_{j+1} & C_{2j+1} = e_j \vee x_j \vee \overline{d}_{j+1} \vee \overline{e}_{j+1} & \text{for } j = 1, \ldots, t-1\\
C_{2t} = d_t \vee \overline{x}_t \vee \overline{f}_1 \vee \ldots \vee \overline{f}_t & C_{2t+1} = e_t \vee x_t \vee \overline{f}_1 \vee \ldots \vee \overline{f}_t & \\
B_{2j-1} = x_j \vee f_j & B_{2j} = \overline{x}_j \vee f_j & \text{for } j = 1, \ldots, t
\end{array}$$

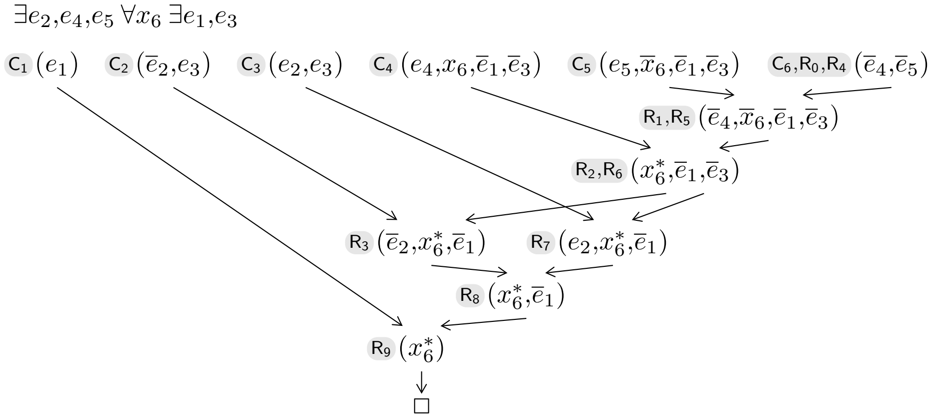

Fig. 1. The LDQ-refutation as a running example. Labels C_1 to C_6, R_1 to R_9 denote the original clauses and the resolvents, respectively. E.g. "R_2,R_6" is shorthand for "$R_2 = R_6$", meaning that the clauses R_2 and R_6 are equal. The derivation and the labels are explained in Examples 2 and 3.

By Theorem 3.2 in [7], any Q-refutation of φ_t for $t \geq 1$ is exponential in t. The formula φ_t has a polynomial size Q-resolution refutation if universal pivot variables are allowed [14]. In the following, we show how to obtain polynomial size LDQ-refutations in the form of a directed acyclic graph (DAG). A straightforward translation of this DAG to a tree results in an exponential blow-up.

Proposition 1. *Any φ_t has an LDQ-refutation of polynomial size in t for $t \geq 1$.*

Proof. An LDQ-refutation with $O(t)$ clauses for $(\varphi_t)_{t\geq 1}$ can be constructed as follows:

1. Derive $d_t \vee \overline{x}_t \vee \bigvee_{i=1}^{t-1} \overline{f}_i$ from B_{2t} and C_{2t}. Derive $e_t \vee x_t \vee \bigvee_{i=1}^{t-1} \overline{f}_i$ similarly.
2. Use both clauses from Step 1 together with $C_{2(t-1)}$ and derive the clause $d_{t-1} \vee \overline{x}_{t-1} \vee \bigvee_{i=1}^{t-1} \overline{f}_i \vee x_t^*$. Observe that the quantification level of d_t and e_t is smaller than the level of x_t. Use $B_{2(t-1)}$ to get $d_{t-1} \vee \overline{x}_{t-1} \vee \bigvee_{i=1}^{t-2} \overline{f}_i \vee x_t^*$. Derive the clause $e_{t-1} \vee x_{t-1} \vee \bigvee_{i=1}^{t-2} \overline{f}_i \vee x_t^*$ in a similar way.
3. Iterate the procedure to derive $d_2 \vee \overline{x}_2 \vee \bigvee_{i=1}^{1} \overline{f}_i \vee \bigvee_{i=3}^{t} x_i^*$ as well as $e_2 \vee x_2 \vee \bigvee_{i=1}^{1} \overline{f}_i \vee \bigvee_{i=3}^{t} x_i^*$.
4. With C_2, derive $d_1 \vee \overline{x}_1 \vee \overline{f}_1 \vee \bigvee_{i=2}^{t} x_i^*$. Use B_2 to obtain $d_1 \vee \overline{x}_1 \vee \bigvee_{i=2}^{t} x_i^*$. Derive $e_1 \vee x_1 \vee \bigvee_{i=2}^{t} x_i^*$ in a similar fashion.
5. Use the two derived clauses together with C_0 and C_1 to obtain $\bigvee_{i=1}^{t} x_i^*$, which can be reduced to the empty clause by universal reduction. $\square$

This result leads to the assumption that QBF solving algorithms can benefit from employing the LDQ-calculus. Next, we discuss how it is integrated in search-based QBF solvers.

4 LDQ-Proof Generation in Search-Based QBF Solving

As an application of LDQ-resolution, we consider *search-based QBF solving with conflict-driven clause learning (QCDCL)*. Search-based QBF solving is an extension of the DPLL

```
State ld-qcdcl()
  while (true)
    State s = qbcp();
    if (s == UNDET)
      assign_dec_var();
    else
      if (s == UNSAT)
        a = analyze_conflict();
      else if (s == SAT)
        a = analyze_solution();
      if (a == INVALID)
        return s;
      else
        backtrack(a);
```

```
Assignment analyze_conflict()
  i = 0;
  R_i = find_confl_clause();
  while (!stop_res(R_i))
    p_i = get_pivot(R_i);
    R'_i = get_antecedent(p_i);
    R_{i+1} = resolve(R_i,p_i,R'_i);
    R_{i+1} = reduce(R_{i+1});
    i++;
  add_to_formula(R_i);
  return get_retraction(R_i);
```

Fig. 2. Search-based QBF solving with LD-QCDCL [15,16] using long-distance resolution.

algorithm [2,3]. Given a QBF $\psi = \mathcal{P}.\phi$, the idea of QCDCL [4,8,15,16] is to dynamically generate and add derived clauses to the matrix ϕ. If ψ is false, then the empty clause $\Box$ will finally be generated. In this case, the sequence of clauses involved in the generation of all the learned clauses forms a Q-refutation of ψ.

We focus on the generation of tautological learned clauses in QCDCL based on long-distance (LD) resolution [15,16]. We call the application of this method in search-based QBF solving *LD-QCDCL*. The soundness proof of LD-QCDCL (Lemma 2 and the theorem in [15]) shows that the learned clauses have certain properties *in the context of LD-QCDCL*. Due to these properties of the learned clauses, LD-resolution is applied in a restricted fashion in LD-QCDCL, which ensures soundness. In general, unrestricted LD-resolution relying on the definition in Section 2 is unsound, as pointed out in Example 1.

We prove that the generation of a (tautological) learned clause by LD-resolution in LD-QCDCL corresponds to a derivation in the LDQ-resolution calculus [1] from Section 2. Hence learning tautological clauses in LD-QCDCL produces LDQ-refutations. With our observation we embed the LD-QCDCL procedure [15,16] in the formal framework of the LDQ-resolution calculus, the soundness of which was proved in [1].

In order to make the presentation of our results self-contained and to emphasize the relevance of long-distance resolution in search-based QBF solving, we describe LD-QCDCL in the following. Figure 2 shows a pseudo code.

In our presentation of LD-QCDCL we use the following terminology. Given a QBF $\psi = \mathcal{P}.\phi$, a clause $C \in \phi$ is *unit* if and only if $C = (l)$ and $\mathsf{quant}(l) = \exists$, where l is a *unit literal*. The operation of *unit literal detection* $UL(C) := \{l\}$ collects the assignment $\{l\}$ from the unit clause $C = (l)$. In this case, clause $C = \mathsf{ante}(l)$ is the *antecedent clause* of the assignment $\{l\}$. Otherwise, if C is not unit then $UL(C) := \{\}$ is the empty assignment. Unit literal detection is extended from clauses to sets of clauses in ψ: $UL(\psi) := \bigcup_{C \in \phi} UL(C)$. Resolution (function resolve) and universal reduction (function reduce) are defined as in Section 2.

The operation of *quantified boolean constraint propagation (QBCP)* extends an assignment σ to $\sigma' \supseteq \sigma$ by iterative applications of unit literal detection and universal reduction until fixpoint[1] and computes ψ under σ', such that for $\psi' := \mathsf{reduce}(\psi_{\lceil\sigma'})$, $QBCP(\psi_{\lceil\sigma}) := \psi'_{\lceil\sigma'}$.

LD-QCDCL successively generates partial assignments to the variables in a given QBF ψ. This process amounts to splitting the goal of proving falsity or truth of a QBF into subgoals by case distinction based on QBF semantics. Similar to [15,16], we assume that all clauses in the original ψ are non-tautological. Initially, the current assignment σ is empty. First, QBCP is applied to $\psi_{\lceil\sigma}$ (function `qbcp`). If $QBCP(\psi_{\lceil\sigma}) \neq \top$ and $QBCP(\psi_{\lceil\sigma}) \neq \bot$, then the QBF is undetermined under σ (`s == UNDET`). A variable from the leftmost quantifier block, called *decision variable* or *assumption*, is selected heuristically and assigned a value (function `assign_dec_var`). Assigning the decision variable extends σ to a new assignment σ' and QBCP is applied again to $\psi_{\lceil\sigma}$ with respect to $\sigma := \sigma'$.

If $QBCP(\psi_{\lceil\sigma}) = \bot$ ($QBCP(\psi_{\lceil\sigma}) = \top$), then the QBF is false (true) under the current assignment σ and the result of the subcase corresponding to σ has been determined (`s == SAT` or `s == UNSAT`). The case $QBCP(\psi_{\lceil\sigma}) = \bot$ is called a *conflict* because σ does not satisfy all the clauses in ϕ. Analogously, the case $QBCP(\psi_{\lceil\sigma}) = \top$ is called a *solution* because σ satisfies all clauses in ϕ. Depending on the cases, σ is analyzed. In the following, we focus on the generation of a learned clause from a conflict by function `analyze_conflict`. Dually to clause learning, LD-QCDCL learns *cubes*, i.e. conjunctions of literals, from solutions by function `analyze_solution`. We refer to related literature on cube learning [4,5,8,10,16].

Consider the case $QBCP(\psi_{\lceil\sigma}) = \bot$. Function `analyze_conflict` generates a learned clause as follows. Since $QBCP(\psi_{\lceil\sigma}) = \bot$, there is at least one clause $C \in \phi$ which is falsified, i.e. $\mathsf{reduce}(C_{\lceil\sigma}) = \Box$. Function `find_confl_clause` finds such a clause C and initially sets $R_i := C$ for $i = 0$, where R_i denotes the current resolvent in the derivation of the clause to be learned (`while`-loop).

In the derivation of the learned clause, the current resolvent R_i is resolved with the antecedent clause $R'_i := \mathsf{ante}(l)$ of an existential variable $p_i = \mathsf{var}(l)$, where $\bar{l} \in R_i$ (functions `get_antecedent` and resolve). Variable p_i has been assigned by unit literal detection during QBCP and it is the pivot variable of the current resolution step (function `get_pivot`). According to [16], function `get_pivot` selects the unique variable p_i as pivot which has been assigned most recently by unit literal detection among the variables in R_i. Hence in the derivation variables are resolved on in reverse assignment ordering. Universal reduction is applied to the resolvent (function reduce).

If the current resolvent R_i satisfies a particular stop criterion (`stop_res`) then the derivation terminates and R_i is the clause to be learned. The stop criterion according to [16] makes sure that R_i is an *asserting* clause, which amounts to the following property: R_i is unit under a new assignment $\sigma' \subset \sigma$ obtained by retracting certain assignments from the current assignment σ. Function `get_retraction` computes the assignments to be retracted from σ by backtracking (function `backtrack`). The learned clause R_i is added to ϕ. QBCP with respect to the new assignment $\sigma := \sigma'$ detects that R_i is unit.

LD-QCDCL determines that ψ is false if and only if the empty clause $\Box$ is derived by function `analyze_conflict`. This case (and similarly for true QBFs and cube learning) is indicated by `r == INVALID`, meaning that all subcases have been explored and the truth of ψ has been determined.

[1] For simplicity, we omit *monotone (pure) literal detection* [2], which is typically part of QBCP.

Example 2. We illustrate LD-QCDCL by the QBF from Fig. 1. In the following, C_i and R_i, respectively, denote clauses and resolvents as shown in Fig. 1. Equal, multiply derived resolvents are depicted as single resolvents with multiple labels, e.g. "R_2,R_6".

Given the empty assignment $\sigma := \{\}$, QBCP detects the unit clause C_1, records the antecedent clause $\mathsf{ante}(e_1) := C_1$, and collects the assignment $\{e_1\}$: $\sigma := \sigma \cup \{e_1\} = \{e_1\}$. No clause is unit under σ at this point. Assume that variable e_2 is selected as decision variable and assigned to *true*, i.e., $\sigma := \sigma \cup \{e_2\} = \{e_1, e_2\}$. Clause C_2 is unit under σ and $\sigma := \sigma \cup \{e_3\} = \{e_1, e_2, e_3\}$ with $\mathsf{ante}(e_3) := C_2$. Further, clauses C_4 and C_5 are unit under σ and universal reduction, and $\sigma := \sigma \cup \{e_4\} \cup \{e_5\} = \{e_1, e_2, e_3, e_4, e_5\}$ with $\mathsf{ante}(e_4) := C_4$ and $\mathsf{ante}(e_5) := C_5$. Now, clause C_6 is falsified under σ, which constitutes a conflict.

The derivation of the learned clause starts with $R_0 := C_6$. Variable e_5 has been assigned most recently among the variables in R_0 assigned by unit literal detection. Hence R_0 is resolved with $\mathsf{ante}(e_5) = C_5$, which gives R_1. The following pivot variables are selected in similar fashion. Further, R_1 is resolved with $\mathsf{ante}(e_4) = C_4$, which gives R_2. Finally, R_2 is resolved with $\mathsf{ante}(e_3) = C_2$, which gives R_3 to be learned and added to the clause set.

The clause R_3 is unit under $\sigma' \subset \sigma$ and universal reduction, where $\sigma = \{e_1, e_2, e_3, e_4, e_5\}$ and $\sigma' = \{e_1\}$. Hence the assignments in $\sigma \setminus \sigma' = \{e_2, e_3, e_4, e_5\}$ are retracted to obtain the new current assignment $\sigma := \sigma' = \{e_1\}$. Now, QBCP detects the unit clauses R_3 and C_3, and $\sigma := \sigma \cup \{\overline{e}_2, e_3\} = \{e_1, \overline{e}_2, e_3\}$. Like above, the clauses C_4 and C_5 are unit and C_6 is falsified. The assignment obtained finally is $\sigma = \{e_1, \overline{e}_2, e_3, e_4, e_5\}$.

At this point, the empty clause is derived as follows (for readability we continue the numbering of the resolvents R_i at the previously learned clause R_3): like above, starting from $R_4 := C_6 = R_0$, R_4 is resolved with $\mathsf{ante}(e_5) = C_5$ and $\mathsf{ante}(e_4) = C_4$, which gives $R_5 := R_1$ and $R_6 := R_2$, respectively. Further, R_6 is resolved with $\mathsf{ante}(e_3) = C_3$, which gives R_7. Two further resolution steps on $\mathsf{ante}(e_2) = R_3$ and $\mathsf{ante}(e_1) = C_1$ give R_8 and R_9, respectively. Finally $\Box$ is obtained from R_9 by universal reduction.

With Proposition 4 below, we prove that every application of universal reduction and resolution (functions $\mathsf{resolve}$ and reduce in Fig. 2) corresponds to a rule of the LDQ-resolution calculus [1] from Section 2. We use the following notation. Every resolution step S_i by function $\mathsf{resolve}$ in the derivation of a learned clause has the form of a quadruple $S_i = (R_i, p_i, R'_i, R_{i+1})$, where $i \geq 0$, R_i is the previous resolvent, p_i is the existential pivot variable, $R'_i = \mathsf{ante}(l)$ is the antecedent clause of a literal $\overline{l} \in R_i$ with $\mathsf{var}(l) = p_i$, and R_{i+1} is the resolvent of R_i and R'_i. Proposition 2 and Proposition 3 hold due to the definition of unit literal detection, because the derivation of a learned clause starts at a falsified clause, and because existential variables assigned as unit literals are selected as pivots.

Proposition 2. *Every clause R_i in function* `analyze_conflict` *in Fig. 2 is falsified under the current assignment σ and universal reduction.*

Proof. For resolvents returned by function $\mathsf{resolve}$, we argue by induction. Consider the first step S_0 and the clause R_0, which by definition of function `find_confl_clause` is falsified under σ and universal reduction. If R_0 is tautological by $x^* \in R_0$ then variable x must be unassigned. If it were assigned then either $x \in \sigma$ or $\overline{x} \in \sigma$ and hence R_0 would be satisfied but not falsified under σ and thus R_0 would not be returned by function `find_confl_clause`. Therefore, the property holds for R_0.

Consider an arbitrary step S_i with $i > 0$ and assume that the property holds for R_i. The clause R_i is resolved with an antecedent clause R'_i of a unit literal. That is, the clause R'_i has been unit under σ and universal reduction, and hence contains exactly one existential literal l such that $l \in \sigma$. If R'_i is tautological by $x^* \in R'_i$ then x must be unassigned by similar arguments as above. Otherwise, R_i would have been satisfied and not unit. The variable $p_i = \mathsf{var}(l)$ has been assigned by unit literal detection and it is selected as pivot of the resolution step S_i. Hence no literal of p_i occurs in the resolvent R_{i+1}. If R_{i+1} is tautological by $x^* \in R_{i+1}$ then x must be unassigned. Otherwise, either R_i or R'_i would be satisfied, which either contradicts the assumption that the property holds for R_i or the fact that R'_i was unit, respectively. Therefore, the property holds for the resolvent R_{i+1}.

The property also holds for clauses returned by function reduce since this function is applied to clauses which have the property and universal reduction only removes literals from clauses. □

Proposition 3. *A tautological clause R_i in function* `analyze_conflict` *in Fig. 2 is never due to an existential variable e with $e \in R_i$ and $\overline{e} \in R_i$.*

Proof. We argue by induction. Similar to [15,16], we assume that all clauses in the original QBF ψ are non-tautological. Consider the first step S_0 and the clause R_0, which by definition of function `find_confl_clause` is falsified under σ and universal reduction. By contradiction, assume that $e \in R_0$ and $\overline{e} \in R_0$, hence R_0 is tautological due to an existential variable e. Since R_0 is falsified, either $e \in \sigma$ or $\overline{e} \in \sigma$. In either case R_0 is satisfied but not falsified since both $e \in R_0$ and $\overline{e} \in R_0$. Hence, the property holds for R_0.

Consider an arbitrary step S_i with $i > 0$ and assume that the property holds for R_i. By contradiction, assume that the resolvent R_{i+1} of R_i and R'_i is tautological due to an existential variable e with $e \in R_{i+1}$ and $\overline{e} \in R_{i+1}$. We distinguish three cases how the literals e and $\overline{e}$ have been introduced in R_{i+1}: (1) $e \in R_i$ and $\overline{e} \in R_i$, (2) $e \in R'_i$ and $\overline{e} \in R'_i$, and (3) $e \in R_i$ and $\overline{e} \in R'_i$ (the symmetric case $\overline{e} \in R_i$ and $e \in R'_i$ can be handled similarly). By assumption that the property holds for R_i, case (1) cannot occur. In case (2), R'_i is the antecedent clause of a unit literal $l \in R'_i$. Therefore, either $e \notin R'_i$ or $\overline{e} \notin R'_i$ because otherwise R'_i would not have been found as unit: if e is assigned then R'_i would be satisfied and if e is unassigned then R'_i is not unit by definition of unit literal detection. Hence case (2) cannot occur. For case (3), R'_i is the antecedent clause of a unit literal. Since $\overline{e} \in R'_i$, variable e must be assigned with $e \in \sigma$ because R'_i has been unit. Then R_i is satisfied because $e \in R_i$, which contradicts Proposition 2. Since none of the three cases can occur, the property holds for the resolvent R_{i+1}. □

Proposition 4. *Every application of universal reduction and resolution in the derivation of a learned clause in function* `analyze_conflict` *in Fig. 2 corresponds to an application of a rule of the LDQ-resolution calculus [1] introduced in Section 2.*

Proof. The following facts about function `analyze_conflict` conform to the rules of the LDQ-resolution calculus. By assumption, all clauses in the original QBF ψ (i.e. not containing learned clauses) are non-tautological. The original LD-QCDCL procedure [15,16] relies on the same assumption. By Proposition 3, all tautological resolvents R_{i+1} by function resolve are due to universal variables in R_{i+1}. Only existential pivot variables are selected by function `get_antecedent` because universal literals cannot be unit in clauses.

The LDQ-rule u_1 of universal reduction is defined for tautological clauses as well. Therefore, universal reduction by function reduce corresponds to the LDQ-rule u_1.

Consider an arbitrary resolution step $S_i = (R_i, p_i, R'_i, R_{i+1})$ in the derivation of a learned clause. If R_{i+1} is non-tautological then S_i corresponds to the LDQ-rule r_1.

If R_{i+1} is tautological by $x^* \in R_{i+1}$ such that $x^* \in R_i$ or $x^* \in R'_i$ and (1) if $x^* \in R_i$ then $x \notin R'_i$ and $\overline{x} \notin R'_i$, and (2) if $x^* \in R'_i$ then $x \notin R_i$ and $\overline{x} \notin R_i$, then S_i corresponds to the LDQ-rule r_1.

If R_{i+1} is tautological by $x^* \in R_{i+1}$ with $\mathsf{lev}(p_i) < \mathsf{lev}(x)$ then S_i corresponds to the LDQ-rule r_2 because the condition on the levels of the pivot variable p_i and the variable x, which causes the tautology, holds.

In the following, we show that the problematic case where the resolvent R_{i+1} is tautological by $x^* \in R_{i+1}$ with $\mathsf{lev}(x) < \mathsf{lev}(p_i)$, thus violating the level condition, cannot occur.

By contradiction, assume that R_{i+1} is tautological by $x^* \in R_{i+1}$ with $\mathsf{lev}(x) < \mathsf{lev}(p_i)$. Assume that $x \in R_i$ and $\overline{x} \in R'_i$. By Proposition 2, R_i is falsified under the current assignment σ and universal reduction. Hence variable x is unassigned. If it were assigned then we would have $\overline{x} \in \sigma$ because $x \in R_i$, but then the antecedent clause R'_i would be satisfied since $\overline{x} \in R'_i$. Hence R'_i would not have been unit and would not be selected by function `get_antecedent`. Since $\mathsf{lev}(x) < \mathsf{lev}(p_i)$ and x is unassigned, the antecedent clause R'_i could not have been unit. In this case, a literal $l \in R'_i$ of the pivot variable $p_i = \mathsf{var}(l)$ would prevent universal reduction from reducing the literal $\overline{x} \in R'_i$, which is a contradiction. The same reasoning as above applies to the other cases where $\overline{x} \in R_i$ and $x \in R'_i$, $x^* \in R_i$ and $x \in R'_i$, $x \in R_i$ and $x^* \in R'_i$, and to $x^* \in R_i$ and $x^* \in R'_i$. Hence Proposition 4 holds. $\square$

In the following example, we illustrate Proposition 4 by relating the steps in the LDQ-refutation shown in Fig. 1 to rules in the LDQ-calculus.

Example 3. Referring to the resolvents R_i in Example 2 and to clause labels in Fig. 1, clause "R1,R5" is obtained by Rule r_1, clause "R2,R6" by r_2 where $x_6 \in X^l$ and $\overline{x}_6 \in X^r$, clause R7 by r_1, clause R3 by r_1, clause R8 by r_2 where $x_6^* \in X^l$ and $x_6^* \in X^r$, clause R9 by r_1, and clause $\square$ by u_1.

We have modified the search-based QBF solver DepQBF [9] to generate tautological learned clauses by LD-QCDCL as in Fig. 2. This is the variant DepQBF-LDQ implementing the LDQ-resolution calculus, which follows from Proposition 4. Instead of *dependency schemes*, both DepQBF and DepQBF-LDQ applied the variable ordering by the quantification levels in the prefix of a QBF. We considered the solver yQuaffle [15,16] as a reference implementation of LD-QCDCL[2]. The left part of Table 1 shows the number of instances solved in the benchmark set from the QBF evaluation 2012 (QBFEVAL'12-pre),[3] which was preprocessed by Bloqqer.[4] Compared to DepQBF-LDQ, yQuaffle in total solved fewer instances, among them five instances not solved by DepQBF-LDQ. DepQBF-LDQ solved three instances less than DepQBF and solved two instances not solved by DepQBF. A comparison of the 115 instances solved by both DepQBF-LDQ and DepQBF illustrates the

[2] `http://www.princeton.edu/~chaff/quaffle.html`, last accessed in July 2013.
[3] We refer to supplementary material like further experiments, binaries, log files, and an appendix: `http://www.kr.tuwien.ac.at/staff/lonsing/lpar13.tar.7z`
[4] `http://fmv.jku.at/bloqqer/`

QBFEVAL'12-pre (276 formulas)	
yQuaffle	61 (32 sat, 29 unsat)
DepQBF	120 (62 sat, 58 unsat)
DepQBF-LDQ	117 (62 sat, 55 unsat)

115 solved by both:	DepQBF-LDQ	DepQBF
Avg. assignments	13.7×10^6	14.4×10^6
Avg. backtracks	43,676	50,116
Avg. resolutions	573,245	899,931
Avg. learn.clauses	31,939 (taut: 5,571)	36,854
Avg. run time	51.77	57.78

Table 1. Search-based QBF solvers with (yQuaffle, DepQBF-LDQ) and without LD-resolution (DepQBF) in clause learning on preprocessed instances from QBFEVAL'12. Number of solved instances (left) with a timeout of 900s and detailed statistics (right).

Parameter t	13	14	15	16	17	18	19	20
yQuaffle	0.448	0.524	0.606	0.694	0.788	0.888	TO	TO
DepQBF	118	253	540	1,146	2,424	5,111	10,747	22,544
DepQBF-LDQ	0.287	0.330	0.376	0.425	0.477	0.532	0.590	0.651

Table 2. Number of resolution steps (in units of 1,000) in refutations of selected formulas in the family φ_t from Section 3. The solvers yQuaffle and DepQBF-LDQ implement the LDQ-resolution calculus, and DepQBF implements Q-resolution. The timeout (TO) was 900 seconds.

potential of the LDQ-resolution calculus in LD-QCDCL. For DepQBF-LDQ, the average numbers in the right part of Table 1 are smaller than for DepQBF, regarding assignments (-5%), backtracks (-13%), resolution steps (-37%), learned clauses (-14%), and run time (-11%). On average, 17% (5,571) of the learned clauses were tautological.

We computed detailed statistics to measure the effects of tautological learned clauses in DepQBF-LDQ. Thereby we focus on instances which were solved and where tautological clauses were learned. Tautological clauses were learned on 38 of the 117 instances solved by DepQBF-LDQ (32%). Among these 38 instances, 2,714,908 clauses were learned in total, 641,746 of which were tautological clauses (23%). A total of 22,324,295 learned clauses became unit by unit literal detection, among them 903,619 tautological clauses (4%). A total of 1,364,248 learned clauses became falsified, among them no tautological clauses (0%). Hence we did not observe a tautological clause to be falsified and used as a start point to derive a new learned clause (falsified clauses are returned by function `find_confl_clause` in Fig. 2).

On a different benchmark set from the QBF competition 2010,[3] DepQBF-LDQ solved three instances more than DepQBF and solved five instances not solved by DepQBF. On that set, we observed fewer resolutions (-11%) and smaller run time (-9%) with DepQBF-LDQ, compared to DepQBF. Further, tautological clauses were learned on 25% of the instances solved by DepQBF-LDQ in that set. On these instances, 35% of the learned clauses were tautological. Among the learned clauses which became unit, 8% were tautological. Like for the set QBFEVAL'12-pre, we did not observe a tautological clause to be falsified.

Additionally, we empirically confirmed Proposition 1. As expected, the refutation size for the family $(\varphi_t)_{t\geq1}$ produced by yQuaffle and DepQBF-LDQ scales linearly with t. In contrast to that, the refutation size scales exponentially with Q-resolution [7] in DepQBF. Table 2 illustrates the difference in the refutation sizes. Somewhat unexpectedly, yQuaffle times out on formulas of size $t\geq19$ (and DepQBF times out for $t\geq21$), whereas

DepQBF-LDQ solves formulas of size up to $t = 100$ in about one second of run time (we did not test with higher parameter values). As an explanation, we found that the number of *cubes* learned by yQuaffle (i.e. the number of times function `analyze_solution` in Fig. 2 is called) doubles with each increase of t. The learned cubes do not affect the refutation size but the time to generate the refutation. With DepQBF-LDQ, both the number of learned clauses and learned cubes scales linearly with t.

5 Extracting Strategies from LDQ-Proofs

We show that the method to extract strategies from Q-refutations [6] is also correct when applied to LDQ-refutations. This result enables a complete workflow including QBF solving and strategy extraction based on the LDQ-resolution calculus. A similar workflow could be implemented based on a translation of an LDQ-refutation into a Q-refutation as presented in [1]. However, this translation can cause an exponential blow-up in proof size. By applying strategy extraction directly on LDQ-refutations we avoid this blow-up.

Strategy extraction [6] is described as a game between a universal ($\forall$) player and an existential ($\exists$) player on a Q-refutation of a QBF . The game aims at an assignment to $\forall$ variables that renders the matrix unsatisfiable. It proceeds through the quantifier prefix from the left to the right alternating the two players according to the quantifier blocks. The $\exists$ player arbitrarily chooses an assignment $\sigma_\exists$ to the variables in the current block. Then the proof is modified according to $\sigma_\exists$ using sound derivation rules outside the Q-resolution calculus. This modification results in a smaller derivation of $\Box$ with all literals contained in $\sigma_\exists$ and their opposite polarities being removed. Based on this modified proof, an assignment $\sigma_\forall$ to the following quantifier block, a $\forall$ block, is calculated such that applying $\sigma_\forall$ to each clause of the proof and applying some extra derivation rules to the proof results in a derivation of $\Box$. In this section we show with an argument similar to [6], that (1) the modification of an LDQ-refutation according to any assignment to $\exists$ variables derives $\Box$, and (2) the modification of an LDQ-refutation according to a computed assignment to $\forall$ variables derives $\Box$.

The reason why this method works for LDQ-refutations in the same way as for Q-refutations is the following. Consider an LDQ-refutation under an assignment $\sigma_\exists$ to $\exists$ variables of some quantifier block of level ℓ. Then the applications of rule r_2 from Section 2 (LD-steps) on $\forall$ variables with quantification level $\ell+1$ are always removed. This is the case because an LD-step can result in a merged literal x^* only if the pivot variable p (an $\exists$ variable) has a lower quantification level than x. Thus before the $\forall$ player's turn, the pivot variable of each LD-step that results in merged literals of the respective quantifier block is contained in the partial assignment. Either of the parents in the LD-step is then set to $\top$, and by fixing the derivation, only one polarity of the $\forall$ variable is left in the derived clause.

The algorithms `play` and `assign` describe the algorithm presented in [6], where `play` implements the alternating turns of the $\forall$ and the $\exists$ player. Each player chooses an assignment to the variables in the current quantifier block (Lines 3 and 6 of `play`). The proof is modified after each assignment (Lines 7 and 8 of `play`) and results in an LDQ-refutation of the QBF under the partial assignment. The modification of the LDQ-refutation Π consists of two steps represented by `assign` and `transform`. The algorithm `assign` applies an assignment to Π. It changes each leaf clause according to the definition of a clause under an assignment in Section 2. Then it adjusts the successor

Algorithm 1: `play`

Input : QBF $\mathcal{P}.\psi$, LDQ-refutation Π

foreach *Quantifier block* Q *in* $\mathcal{P}$ *from left to right* **do**
  if Q *is existential* **then**
    $\sigma \leftarrow$ any assignment to each variable in Q;
  else Q is universal
    $C \leftarrow$ topologically first clause in Π with no existential literals;
    $\sigma \leftarrow \{\overline{x} \mid x \in C \wedge \mathsf{var}(x) \in Q\} \cup \{x \mid x \notin C \wedge \overline{x} \notin C \wedge \mathsf{var}(x) \in Q\}$;
  $\Pi^p \leftarrow$ `assign` (Π,σ) *(Π^p is not an LDQ-refutation)*;
  $\Pi \leftarrow$ `transform` (Π^p) *(Π is an LDQ-refutation)*;

Algorithm 2: `assign`

Input : LDQ-refutation Π, assignment σ to all variables of outermost block
Output: Refutation under assignment σ containing LD-rules and P-rules

foreach *leaf clause* C *in* Π **do**
  $C \leftarrow C_{\lceil\sigma}$;
foreach *inner clause* C *topologically in* Π **do**
  if C *is a resolution clause* **then**
    C^l,$C^r \leftarrow$ parents of C;
    $p \leftarrow$ pivot of C;
    $C \leftarrow$ p-resolve(C^l,p,C^r);
  else C is a clause derived by reduction
    $C^c \leftarrow$ parent of C;
    $x \leftarrow$ variable reduced from C^c;
    $C \leftarrow$ p-reduce(C^c,p);
return Π

clauses in topological order (from leaves to root) by either applying an LDQ-resolution rules or, in cases where the pivot variable or a reduced variable has been removed from at least one of the parents, by applying one of the additional rules presented in [6,13]. These additional rules (P-rules) are reproduced in the following. Symmetric rules are omitted.

$$[p]\ \frac{C^l \vee p \qquad \top}{\top} \qquad (r_3) \qquad\qquad [x]\ \frac{C}{C} \quad x \notin C \qquad (u_2)$$

$$[p]\ \frac{C^l \vee p \qquad C^r}{C^r} \quad \overline{p} \notin C^r \qquad (r_4) \qquad\qquad [x]\ \frac{\top}{\top} \qquad (u_3)$$

$$[p]\ \frac{C^l \qquad C^r}{\mathsf{narrower}(C^l,C^r)} \quad \overline{p},p \notin C^l \text{and}\ \overline{p},p \notin C^r \qquad (r_5)$$

In rule r_5, $\mathsf{narrower}(C^l,C^r)$ returns the clause containing fewer literals. If C^l and C^r contain the same number of literals, C^l is returned. The narrowest clause is $\Box$ and $\top$ is defined to contain all literals. In the remainder, we write p-resolve(C^l,p,C^r) for a resolution step

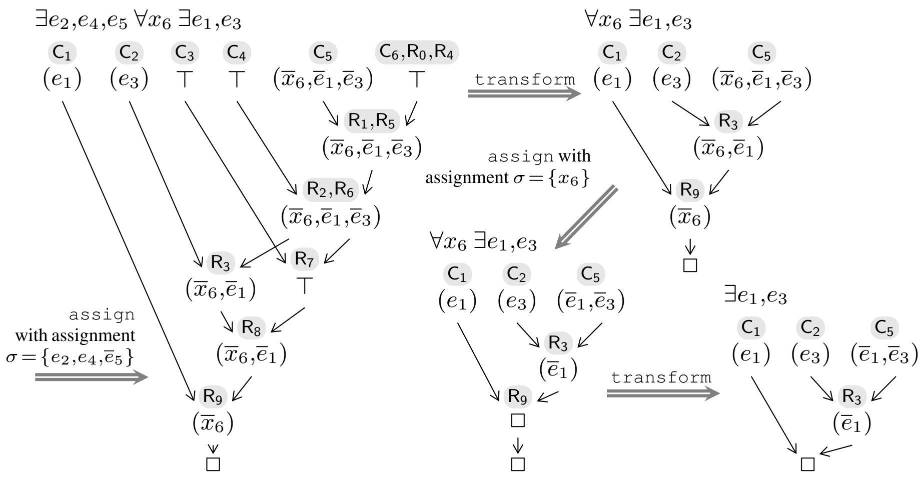

Fig. 3. Two possible iterations of the strategy extraction algorithm `play` on the example in Fig. 1

over pivot variable p according to rules r_1 to r_5, and p-reduce(C,x) for a reduction step reducing variable x according to rules u_1 to u_3.

After this procedure, the refutation contains applications of P-rules and thus is a proof outside the LDQ-resolution calculus. It is transformed back into an LDQ-refutation by the following procedure. Starting at the leaves of the proof, the algorithm `transform` (Π^p), where Π^p is a proof that contains clauses derived using LDQ-rules and P-rules, steps through the proof in topological order. Each clause derived by rule r_3, is merged with its parent $\top$. Each clause derived by rule r_4 is merged with its parent C^r. Each clause derived by rule r_5, is merged with its narrower parent. Each clause derived by rules u_1 to u_3 is merged with its parent. When an empty clause $C = \Box$ is encountered, the procedure stops and all clauses that are not involved in deriving C are removed. $\top$-clauses are eliminated by applying rule r_4 or by removing clauses when $\Box$ is found. The resulting refutation is an LDQ-refutation.

Example 4. Figure 3 depicts a possible execution of the `play` algorithm on the instance introduced in Fig. 1. First, an arbitrary assignment is chosen for the first existential quantifier block. The leftmost proof shows the result of executing `assign` on the original proof in Fig. 1. The leaf clauses are changed according to σ. P-rules are applied to the derived clauses "R_1,R_5", "R_2,R_6", R_8 (by rule r_4) and R_7 (by rule r_3). The merged literal x_6^* has disappeared in clause "R_2,R_6" because of the assignment to e_4, which is the pivot variable of the resolution step deriving "R_2,R_6". Before continuing with the $\forall$ player's move, the proof is transformed back to the LDQ-resolution calculus by deleting redundant clauses and edges as depicted in the proof in the right upper corner of Fig. 3. Next, an assignment is calculated for the variable x_6 in following universal quantifier block by inspecting the clause R_9 from which $\overline{x}_6$ is reduced. The proof is then modified according to the computed assignment, which sets R_9 to $\Box$ in the middle lower proof. If there were more than one variable in this quantifier block, reducing one after another would result in a subsequent application of universal reduction, eventually deriving $\Box$. In the next transformation, a list of redundant clauses containing $\Box$ is removed, resulting in the lower right proof. This remaining proof shows unsatisfiability of a propositional formula. The example can be executed similarly for any other assignment to the variables in the existential quantifier blocks.

This algorithm is correct when executed on a Q-resolution proof [6]. We show that it is also correct when executed on an LDQ-resolution proof. To this end, we prove that `assign`, when called in Line 7 of `play`, returns a derivation of $\Box$ using LDQ-rules and P-rules. Proposition 5 shows that this holds for an arbitrary assignment to all $\exists$ variables in the outermost quantifier block, and Proposition 6 shows the same for the computed assignment to $\forall$ variables.

We start by showing that any clause generated from parent(s) under a partial assignment by applying an LDQ-rule or a P-rule subsumes the clause generated from the original parent(s) under the partial assignment. The proof of the following lemma is based on a case distinction of C^l, C^r, and C containing none, at least one, or only literals also contained in σ. The subset relation is shown separately for each case.[5]

Lemma 1. *(cf. Lemma 2.6 in [13]) Given a QBF $\psi = \exists\mathcal{V}\mathcal{P}\phi$ with $\mathcal{V}$ the set of all variables of the outermost quantifier block, $\mathcal{P}$ the prefix of ψ without $\exists\mathcal{V}$, and ϕ the matrix of ψ, let C, C^l and C^r be clauses of ϕ, and σ an assignment to $\mathcal{V}$. Then it holds that* $\mathsf{p\text{-}resolve}(C^l_{\lceil\sigma}, p, C^r_{\lceil\sigma}) \subseteq \mathsf{p\text{-}resolve}(C^l, p, C^r)_{\lceil\sigma}$ *and* $\mathsf{p\text{-}reduce}(C_{\lceil\sigma}, x) \subseteq \mathsf{p\text{-}reduce}(C,x)_{\lceil\sigma}$.

With respect to the application of rule r_2 (LD-step), we observe the following from the `play` algorithm: Let ℓ be the level of an existential quantifier block, p be an existential variable with $\mathsf{lev}(p) = \ell$, x be a universal variable with $\mathsf{lev}(x) = \ell+1$, $\sigma_\exists$ be an assignment to the variables of the quantifier block with level ℓ, and C be a clause derived by rule r_2 with pivot variable p producing the merged literal x^*. Recall that by the conditions for rule r_2 it must hold that $\mathsf{lev}(p) < \mathsf{lev}(x^*)$ whenever any merged literal x^* is produced by resolving over a pivot p. The algorithm `play` iterates over the prefix from the lower to the higher quantification levels. Therefore, $\sigma_\exists$ must contain a literal of p. By modifying the proof according to $\sigma_\exists$, one of C's parents becomes $\top$ and with that, one polarity of the x disappears. By further modifying the proof, the P-rule r_4 must be applied to derive the modified C, which keeps only opposite polarity of x. Therefore, x^* is no longer contained in the proof when its quantifier block is processed.

Lemma 2. *Given a QBF in PCNF $\psi = \exists\mathcal{V}\mathcal{P}\phi$ with $\mathcal{V}$ the set of all variables of the outermost quantifier block, $\mathcal{P}$ the prefix of ψ without $\exists\mathcal{V}$, and ϕ the matrix of ψ, an LDQ-derivation Π of a clause C from ψ, and an assignment $\sigma_\exists$ to $\mathcal{V}$, it holds that $\Pi' = \mathtt{assign}(\Pi,\sigma_\exists)$ derives a clause C' from $\mathcal{P}\phi_{\lceil\sigma_\exists}$ such that $C' \subseteq C_{\lceil\sigma_\exists}$.*

Proof. By induction on the structure of Π using Lemma 1. $\Box$

Proposition 5. *Given a QBF in PCNF $\psi = \exists\mathcal{V}\mathcal{P}\phi$ with $\mathcal{V}$ the set of all variables of the outermost quantifier block, $\mathcal{P}$ the prefix of ψ without $\exists\mathcal{V}$, and ϕ the matrix of ψ, an LDQ-refutation Π of ψ, and an assignment $\sigma_\exists$ to $\mathcal{V}$, it holds that $\Pi^p = \mathtt{assign}(\Pi,\sigma_\exists)$ derives $\Box$ from $\mathcal{P}\phi_{\lceil\sigma_\exists}$.*

[5] We refer to Footnote 3 for an appendix containing a detailed proof of Lemma 1.

Proof. By Lemma 2, for any clause C derived in Π it holds that Π' derives a clause C' such that $C' \subseteq C_{\lceil \sigma_\exists}$. Therefore, if $C = \square$, then Π' must derive a clause $C' = \square$. $\square$

Proposition 6. *Given a QBF in PCNF $\psi = \forall \mathcal{V} \mathcal{P} \phi$ with $\mathcal{V}$ the set of all variables of the outermost quantifier block, $\mathcal{P}$ the prefix of ψ without $\forall \mathcal{V}$, and ϕ the matrix of ψ, an LDQ-refutation Π of ψ, and an assignment $\sigma_\forall$ to $\mathcal{V}$ as computed in Line 6 of Algorithm 1, $\Pi^p = \mathtt{assign}(\Pi, \sigma_\forall)$ derives $\square$ from $\mathcal{P}\phi_{\lceil \sigma_\forall}$.*

Proof. For any $l \in \sigma_\forall$ it holds that $\mathsf{var}(l)$ is either not reduced at all, or reduced exactly once in Π. If $\mathsf{var}(l)$ is not reduced at all, then it is not involved in Π and therefore its assignment does not alter the proof. Let $R \subseteq \sigma_\forall$ be the set of literals of opposite polarity of those that are reduced exactly once in the proof. Then there is a set $\mathcal{C}$ with $|\mathcal{C}| = |R|$ of clauses such that the clauses in $\mathcal{C}$ are directly following one another, each reducing exactly one literal r in R. The last reduced clause of $\mathcal{C}$ results in $\square$. This is the case because all literals of R are in the outermost quantifier block. The algorithm `assign` $(\Pi, \sigma_\forall)$ then applies rule u_2 to each clause C, setting each C in $\mathcal{C}$ to $\square$. $\square$

6 Conclusions and Future Work

We have shown that the LDQ-resolution calculus [1] allows for a complete workflow in search-based QBF solving, including the generation of LDQ-refutations in QBF solvers and the extraction of strategies [6] from these LDQ-refutations. The run time of strategy extraction is polynomial in the refutation size. Therefore, a speedup in strategy extraction can be obtained from having short LDQ-refutations, compared to Q-refutations [7].

It is unclear whether Herbrand functions can be efficiently constructed in certificate extraction [1] based on LDQ-refutations. It is possible to build Herbrand functions from truth tables generated by the strategy extraction method in [6]. However, since each possible assignment to the existential variables has to be considered, the run time of this naive method is exponential in the size of the quantifier prefix.

Regarding practice, learning tautological clauses by LD-QCDCL as used in QBF solvers is conceptually simpler than disallowing tautological resolvents. Tautological resolvents can entirely be avoided in clause learning [4]. However, this approach has an exponential worst case [14], in contrast to a more sophisticated polynomial-time procedure [10].

Experimental results for our implementation of LD-QCDCL illustrate the potential of the LDQ-calculus in search-based QBF solving. For instances solved by both methods, one learning only non-tautological clauses and the other learning also tautological clauses, we observed fewer backtracks, resolution steps, and learned clauses for the latter.

Long-distance resolution can also be applied to derive learned *cubes* or *terms*, i.e. conjunctions of literals (Proposition 6 in [16]). Dually to learned clauses, the learned cubes represent a *term-resolution proof* [4] of a true QBF. Our implementation of LD-QCDCL in DepQBF-LDQ includes cube learning as well.

In LD-QCDCL, a tautological clause is satisfied as soon as the variable causing the tautology is assigned either truth value. These clauses cannot become unit *under the current assignment* and hence cannot be used to derive a new learned clause in this context. Therefore, further experiments are necessary to assess the value of learning tautological clauses.

In general, it would be interesting to compare the different clause learning methods [4,10,15,16] in search-based QBF solving to identify their individual strengths.

References

1. V. Balabanov and J.-H. R. Jiang. Unified QBF Certification and Its Applications. *Formal Methods in System Design*, 41:45–65, 2012.
2. M. Cadoli, A. Giovanardi, and M. Schaerf. An Algorithm to Evaluate Quantified Boolean Formulae. In *AAAI/IAAI*, 1998.
3. M. Davis, G. Logemann, and D. W. Loveland. A Machine Program for Theorem-Proving. *Communications of the ACM*, 5(7):394–397, 1962.
4. E. Giunchiglia, M. Narizzano, and A. Tacchella. Clause/Term Resolution and Learning in the Evaluation of Quantified Boolean Formulas. *Journal of Artificial Intelligence Research*, 26:371–416, 2006.
5. A. Goultiaeva and F. Bacchus. Recovering and Utilizing Partial Duality in QBF. In *16h International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 7962 of *LNCS*. Springer, 2013.
6. A. Goultiaeva, A. Van Gelder, and F. Bacchus. A Uniform Approach for Generating Proofs and Strategies for Both True and False QBF Formulas. In *22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 546–553. AAAI Press, 2011.
7. H. Kleine Büning, M. Karpinski, and A. Flögel. Resolution for Quantified Boolean Formulas. *Information and Computation*, 117(1):12–18, Feb. 1995.
8. R. Letz. Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas. In *TABLEAUX*, pages 160–175, 2002.
9. F. Lonsing and A. Biere. DepQBF: A Dependency-Aware QBF Solver (System Description). *Journal on Satisfiability, Boolean Modeling and Computation*, 7:71–76, 2010.
10. F. Lonsing, U. Egly, and A. Van Gelder. Efficient Clause Learning for Quantified Boolean Formulas via QBF Pseudo Unit Propagation. In *16h International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 7962 of *LNCS*. Springer, 2013.
11. J. P. Marques Silva, I. Lynce, and S. Malik. Conflict-Driven Clause Learning SAT Solvers. In *Handbook of Satisfiability*, pages 131–153. IOS Press, 2009.
12. S. Staber and R. Bloem. Fault Localization and Correction with QBF. In *10th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 4501 of *LNCS*. Springer, 2007.
13. A. Van Gelder. Input Distance and Lower Bounds for Propositional Resolution Proof Length. In *8th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 3569 of *LNCS*. Springer, 2005.
14. A. Van Gelder. Contributions to the Theory of Practical Quantified Boolean Formula Solving. In *18th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 7514 of *LNCS*, pages 647–663. Springer, 2012.
15. L. Zhang and S. Malik. Conflict Driven Learning in a Quantified Boolean Satisfiability Solver. In *2002 IEEE/ACM International Conference on Computer-aided Design*, pages 442–449, 2002.
16. L. Zhang and S. Malik. Towards a Symmetric Treatment of Satisfaction and Conflicts in Quantified Boolean Formula Evaluation. In *18th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 2470 of *LNCS*, pages 200–215. Springer, 2006.

A Appendix

A.1 Proofs

Proof (Lemma 1).

We distinguish between each case of C containing or not containing literals of σ.

$C_0(\sigma,C)$: $\exists v \in \sigma$ such that $v \in C \wedge \overline{v} \in C$. This case never happens because no tautologies over $\exists$ variables are allowed.
$C_1(\sigma,C)$: $\forall v \in \sigma$ it holds that $v \notin C$ and $\overline{v} \notin C$. Then $C_{\lceil\sigma} = C$.
$C_2(\sigma,C)$: $\exists v \in \sigma$ such that $v \in C$. Then $C_{\lceil\sigma} = \top$.
$C_3(\sigma,C)$: $\forall v \in \sigma$ it holds that $v \notin C$ and $\exists v \in \sigma$ such that $\overline{v} \in C$. Then $C_{\lceil\sigma} = C \setminus \{\overline{v} \mid v \in \sigma\}$.

For resolution steps, either of the following cases applies (symmetric cases are omitted):
(1) $C_1(\sigma, C^l)$ and $C_1(\sigma, C^r)$: $C^l_{\lceil\sigma} = C^l$, $C^r_{\lceil\sigma} = C^r$, and $\mathsf{p\text{-}resolve}(C^l, p, C^r)_{\lceil\sigma} = \mathsf{p\text{-}resolve}(C^l, p, C^r)$. By applying rule r_1 or r_2, we obtain $\mathsf{p\text{-}resolve}(C^l_{\lceil\sigma}, p, C^r_{\lceil\sigma}) = \mathsf{p\text{-}resolve}(C^l,p,C^r)$. Therefore the subset relation holds.
(2) $C_2(\sigma,C^l)$ and $C_1(\sigma,C^r)$: $C^l_{\lceil\sigma} = \top$, $C^r_{\lceil\sigma} = C^r$, and $\mathsf{p\text{-}resolve}(C^l,p,C^r)_{\lceil\sigma} = \top$. By applying rule r_3, we obtain $\mathsf{p\text{-}resolve}(C^l_{\lceil\sigma},p,C^r_{\lceil\sigma}) = \top$. Therefore the subset relation holds.
(3) $C_3(\sigma,C^l)$ and $C_1(\sigma,C^r)$: Since $C_1(\sigma,C^r)$, $\forall v \in \sigma$ it holds that $p \neq v$ and $p \neq \overline{v}$. Then $C^l_{\lceil\sigma} = C^l \setminus \{\overline{v} \mid v \in \sigma\}$, $C^r_{\lceil\sigma} = C^r$, and $\mathsf{p\text{-}resolve}(C^l,p,C^r)_{\lceil\sigma} = C^l \cup C^r \setminus \{\overline{v} \mid v \in \sigma\}$. By applying rule r_1 or r_2, we obtain $\mathsf{p\text{-}resolve}(C^l_{\lceil\sigma},p,C^r_{\lceil\sigma}) = C^l \cup C^r \setminus \{\overline{v} \mid v \in \sigma\}$. Therefore the subset relation holds.
(4) $C_2(\sigma,C^l)$ and $C_2(\sigma,C^r)$: $C^l_{\lceil\sigma} = \top$, $C^r_{\lceil\sigma} = \top$, and $\mathsf{p\text{-}resolve}(C^l,p,C^r)_{\lceil\sigma} = \top$. By applying rule r_3, we obtain $\mathsf{p\text{-}resolve}(C^l_{\lceil\sigma},p,C^r_{\lceil\sigma}) = \top$. Therefore the subset relation holds.
(5) $C_2(\sigma,C^l)$ and $C_3(\sigma,C^r)$: Then $C^l_{\lceil\sigma} = \top$, and $C^r_{\lceil\sigma} = C^r \setminus \{\overline{v} \mid v \in \sigma\}$. We have to distinguish two cases:
(a) $\exists v \in \sigma$ such that $\mathsf{var}(v) = p$. Then $\mathsf{p\text{-}resolve}(C^l,p,C^r)_{\lceil\sigma} = (C^l \cup C^r) \setminus \{\overline{v} \mid v \in \sigma\}$. By applying rule r_4, we obtain $\mathsf{p\text{-}resolve}(C^l_{\lceil\sigma},p,C^r_{\lceil\sigma}) = C^r \setminus \{\overline{v} \mid \overline{v} \in \sigma\}$. Therefore the subset relation holds.
(b) Otherwise (σ does not contain the pivot p). Then $\mathsf{p\text{-}resolve}(C^l,p,C^r)_{\lceil\sigma} = \top$. By applying rule r_3, we obtain $\mathsf{p\text{-}resolve}(C^l_{\lceil\sigma},p,C^r_{\lceil\sigma}) = \top$. Therefore the subset relation holds.
(6) $C_3(\sigma, C^l)$ and $C_3(\sigma, C^r)$ $C^l_{\lceil\sigma} = C^l \setminus \{\overline{v} \mid v \in \sigma\}$, $C^r_{\lceil\sigma} = C^r \setminus \{\overline{v} \mid v \in \sigma\}$, and $\mathsf{p\text{-}resolve}(C^l, p, C^r)_{\lceil\sigma} = C^l \cup C^r \setminus \{\overline{v} \mid v \in \sigma\}$. By applying rule r_1 or r_2, we obtain $\mathsf{p\text{-}resolve}(C^l_{\lceil\sigma},p,C^r_{\lceil\sigma}) = C^l \cup C^r \setminus \{\overline{v} \mid v \in \sigma\}$. Therefore the subset relation holds.

For reduction steps, either of the following cases applies:
(7) $C_1(\sigma,C)$: $C_{\lceil\sigma} = C$ and $\mathsf{p\text{-}reduce}(C,x)_{\lceil\sigma} = \mathsf{p\text{-}reduce}(C,x)$. Therefore the subset relation holds.
(8) $C_2(\sigma,C)$: $C_{\lceil\sigma} = \top$ and $\mathsf{p\text{-}reduce}(C,x)_{\lceil\sigma} = \top$. By applying rule u_3 we obtain $\mathsf{p\text{-}reduce}(C_{\lceil\sigma},x) = \top$. Therefore the subset relation holds.
(9) $C_3(\sigma,C)$: $C_{\lceil\sigma} = C \setminus \{\overline{v} \mid v \in \sigma\}$ and $\mathsf{p\text{-}reduce}(C,x)_{\lceil\sigma} = (C \setminus \{x\}) \setminus \{\overline{v} \mid v \in \sigma\}$. By applying rule u_1, we obtain $\mathsf{p\text{-}reduce}(C_{\lceil\sigma},x) = (C \setminus \{x\}) \setminus \{\overline{v} \mid v \in \sigma\}$. Therefore the subset relation holds.

$\square$

A.2 Data Related to Experimental Results

The two benchmark sets *QBFEVAL'10* and *QBFEVAL'12-pre* considered in Sections 4 and A.4 are available through the following links, respectively:

- `http://www.kr.tuwien.ac.at/events/qbfgallery2013/benchmarks/eval2010.tar.7z`
- `http://www.kr.tuwien.ac.at/events/qbfgallery2013/benchmarks/eval12r2-bloqqer.tar`

We provide a data package containing binaries, log files and additional material: `http://www.kr.tuwien.ac.at/staff/lonsing/lpar13.tar.7z`

This data package contains binaries of DepQBF and DepQBF-LDQ, log files of experiments, formulas from the family φ_t for $t = 1,\ldots 100$, and selected proofs for formulas from φ_t produced by DepQBF and DepQBF-LDQ.

A.3 Experimental Results with the Family φ_t

All reported experiments were run on AMD Opteron 6238, 2.6 GHz, 64-bit Linux with limits of 7 GB memory and 900 seconds wall clock time.

The solvers DepQBF-LDQ (which is a modification of DepQBF [9]) and yQuaffle [15,16] are based on LD-QCDCL as shown in Fig. 2 and hence implement the LDQ-resolution calculus, which follows from Proposition 4.

Parameter t	13	14	15	16	17	18	19	20
yQuaffle	0.448	0.524	0.606	0.694	0.788	0.888	TO	TO
DepQBF	118	253	540	1,146	2,424	5,111	10,747	22,544
DepQBF-LDQ	0.287	0.330	0.376	0.425	0.477	0.532	0.590	0.651

Table 3. Number of resolution steps (in units of 1,000) in proofs of selected formulas in the family φ_t from Section 3. The solvers yQuaffle and DepQBF-LDQ implement the LDQ-resolution calculus, and DepQBF implements Q-resolution. The timeout (TO) was 900 seconds.

Table 3 shows the proof sizes for selected formulas in the family φ_t. As expected from the theoretical result in Section 3, the size of the proofs produced by yQuaffle and DepQBF-LDQ, both implementing the LDQ-resolution calculus, scales linearly with respect to the parameter t. In contrast to that, the proof size scales exponentially with Q-resolution [7] in DepQBF, where tautological resolvents are disallowed.[6]

Somewhat unexpectedly, yQuaffle times out on formulas of size $t \geq 19$ (and DepQBF times out for $t \geq 21$), whereas DepQBF-LDQ solves formulas of size up to $t = 100$ in about one second of run time (we did not test with higher parameter values). Table 4 shows the run times on selected instances. As an explanation, we found that the number of *cubes* learned by yQuaffle (i.e. the number of times function `analyze_solution` in Fig. 2 is called) doubles with each increase of t. With DepQBF-LDQ, the number of learned clauses and learned cubes scales linearly with t. Tables 5 and 6 show the numbers of learned clauses and cubes, respectively.

[6] DepQBF produces the same proofs for the family φ_t if we apply a more sophisticated implementation which avoids an exponential worst case in the clause learning procedure [10,14].

Parameter t	13	14	15	16	17	18	19	20
yQuaffle	<1	1	4	18	115	617	TO	TO
DepQBF	1	2	5	12	31	81	219	675
DepQBF-LDQ	<1	<1	<1	<1	<1	<1	<1	<1

Table 4. Wall clock time in seconds spent by the solvers on selected formulas in the family φ_t from Section 3. The timeout (TO) was 900 seconds.

Parameter t	13	14	15	16	17	18	19	20
yQuaffle	25	27	29	31	33	35	TO	TO
DepQBF	4,097	8,193	16,385	32,769	65,537	131,073	262,145	524,289
DepQBF-LDQ	14	15	16	17	18	19	20	21

Table 5. Number of learned clauses on selected formulas in the family φ_t from Section 3.

Parameter t	13	14	15	16	17	18	19	20
yQuaffle	4,096	8,192	16,384	32,768	65,536	131,072	TO	TO
DepQBF	10,242	20,489	40,984	81,975	163,958	327,925	655,860	1,311,731
DepQBF-LDQ	91	105	120	136	153	171	190	210

Table 6. Number of learned cubes on selected formulas in the family φ_t from Section 3.

A.4 Experimental Results with Competition Benchmarks

All reported experiments were run on AMD Opteron 6238, 2.6 GHz, 64-bit Linux with limits of 7 GB memory and 900 seconds wall clock time.

Table 7 shows the number of instances solved by DepQBF, DepQBF-LDQ, and yQuaffle with respect to the benchmarks sets from the QBF competitions 2010 (QBFEVAL'10) and 2012 (QBFEVAL'12-pre), where the latter set was preprocessed using Bloqqer.[7] On

QBFEVAL'10 (568 formulas, no preprocessing)	
yQuaffle	174 (75 sat, 99 unsat)
DepQBF	365 (154 sat, 211 unsat)
DepQBF-LDQ	368 (156 sat, 212 unsat)
QBFEVAL'12-pre (276 formulas, preprocessed	
yQuaffle	61 (32 sat, 29 unsat)
DepQBF	120 (62 sat, 58 unsat)
DepQBF-LDQ	117 (62 sat, 55 unsat)

Table 7. Number of instances solved in the benchmark sets from the QBF comptetitions 2010 (without preprocessing) and 2012 (preprocessed by Bloqqer).

the set QBFEVAL'12-pre, DepQBF-LDQ solved two instances not solved by DepQBF

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

and DepQBF solved five instances not solved by DepQBF-LDQ. On the set QBFEVAL'10, DepQBF-LDQ solved five instances not solved by DepQBF and DepQBF solved two instances not solved by DepQBF-LDQ.

Similar to the right part of Table 1, Table 8 shows detailed statistics for instances from QBFEVAL'10 and QBFEVAL'12-pre which were solved by both DepQBF and DepQBF-LDQ.

QBFEVAL'10 (363 instances solved by both)		
	DepQBF-LDQ	DepQBF
Avg. assignments	10.51×10^6	10.37×10^6
Avg. backtracks	56,973	57,628
Avg. resolutions	642,043	721,569
Avg. learn.clauses	17,232 (taut: 2,180)	16,673
Avg. run time	49.45	54.11
QBFEVAL'12-pre (115 instances solved by both)		
	DepQBF-LDQ	DepQBF
Avg. assignments	13.7×10^6	14.4×10^6
Avg. backtracks	43,676	50,116
Avg. resolutions	573,245	899,931
Avg. learn.clauses	31,939 (taut: 5,571)	36,854
Avg. run time	51.77	57.78

Table 8. Detailed statistics (average values) with respect to instances solved by both DepQBF-LDQ and DepQBF.

We computed statistics to measure the effects of tautological learned clauses in DepQBF-LDQ on the two benchmark sets.

In the set QBFEVAL'10, DepQBF-LDQ solved 368 instances. For 93 of these 368 instances, tautological clauses were learned (25%). Among these 93 instances, 3,995,202 clauses were learned in total, 1,430,562 of which were tautological clauses (35%). A total of 44,399,956 learned clauses became unit by unit literal detection, among them 3,635,451 tautological clauses (8%). A total of 1,938,967 learned clauses became falsified, among them 0 tautological clauses (0%).