Quantified CDCL with Universal Resolution

Friedrich Slivovsky
TU Wien, Austria

Abstract

Quantified Conflict-Driven Clause Learning (QCDCL) solvers for QBF generate Q-resolution proofs. Pivot variables in Q-resolution must be existentially quantified. Allowing resolution on universally quantified variables leads to a more powerful proof system called QU-resolution, but so far, QBF solvers have used QU-resolution only in very limited ways. We present a new version of QCDCL that generates proofs in QU-resolution by leveraging propositional unit propagation. We detail how conflict analysis must be adapted to handle universal variables assigned by propagation, and show that the procedure is still sound and terminating. We further describe how dependency learning can be incorporated in the algorithm to increase the flexibility of decision heuristics. Experiments with crafted instances and benchmarks from recent QBF evaluations demonstrate the viability of the resulting version of QCDCL.

2012 ACM Subject Classification Theory of computation → Automated reasoning

Keywords and phrases QBF, Q-Resolution, QU-Resolution, CDCL

Digital Object Identifier 10.4230/LIPIcs.SAT.2022.24

Supplementary Material *Software (Source Code)*: `https://github.com/fslivovsky/miniQU`

Funding *Friedrich Slivovsky*: Vienna Science and Technology Fund (WWTF) Grant ICT19-060.

Acknowledgements The author would like to thank the anonymous reviewers for their helpful comments towards improving this paper.

1 Introduction

Sustained improvements in the performance of propositional satisfiability (SAT) solvers [16] is enabling a growing number of applications in formal verification and other areas [5, 10, 44]. In some of these applications, SAT oracles are used to solve problems that are hard for complexity classes beyond NP. Such problems presumably do not to have polynomial SAT encodings, which can result in prohibitive space requirements. A potential solution lies in the development of decision procedures for more succinct target logics such as Quantified Boolean Formulas (QBF). Satisfiability testing of QBF's is PSPACE-complete [40], and many problems have natural QBF encodings [38]. While there has been progress in the performance of QBF solvers, they have not reached the level of maturity seen in SAT solvers. Where conflict-driven clause learning (CDCL) is clearly the dominant paradigm in SAT solving, there are different approaches to QBF solving that appear to be orthogonal [29]. At a high level, these fall into two classes. One class consists of solvers that use SAT oracles for propositional reasoning and reduce QBF solving to a sequence of SAT calls on a propositional *abstraction* [12, 21–23, 35, 42]. The second class is comprised of solvers that seek to lift CDCL from propositional reasoning to *quantified* CDCL (QCDCL) [27, 33, 46, 47].[1] This work reexamines a core component of QCDCL.

A key factor in the success of CDCL SAT solvers is an efficient implementation of unit propagation (UP) [32]. The traditional analogue of unit propagation in QCDCL solvers is *quantified unit propagation* (QUP), which combines UP and *universal reduction*, a proof

[1] This classification is not exhaustive, and there are solvers that combine both aspects [31]. Moreover, preprocessing is crucial for the performance of abstraction-based QBF solvers, and many QBF preprocessing techniques are generalized from SAT [20].

rule that allows for an innermost universal variable to be removed from a clause [25]. QUP leads to more propagation than UP, but it has certain drawbacks. First and foremost, it complicates clause learning: applying a natural conflict analysis procedure results in clauses that are (syntactically) tautological [46]. The fact that such clauses can be useful in proof search is surprising, and it remained mysterious until the underlying proof system was studied under the name of *long-distance Q-resolution* [2]. By now it is well understood that tautological literals are syntactic placeholders for partial strategy functions [3, 6, 11, 41]. Since long-distance Q-resolution is a stronger proof system than Q-resolution, QCDCL with tautological learned clauses can solve certain classes of formulas that are otherwise intractable, but experiments suggest that tautological clauses are less useful in common benchmarks [15]. Tautologies can be eliminated from learned clauses by recursively resolving out existential literals and applying universal reduction [19], but this can result in clause learning taking exponential time [17]. So-called QPUP learning avoids this pathological case, but still requires additional resolution steps to remove tautologies [30]. Another, albeit minor, disadvantage of QUP is that it is more difficult to implement. In particular, in a two-watched-literal-scheme [32] the watched literals cannot be treated independently if the underlying variables have different quantifier types.

In this paper, we propose a modified version of QCDCL that uses propositional unit propagation instead of QUP. It not only avoids the disadvantages of QUP described above, but, by propagating variables regardless of their associated quantifier, is able to generate resolution proofs in which both existentially and universally quantified variables appear as pivots. That is, its underlying proof system is *QU-resolution*, which is known to be exponentially separated from both Q-resolution [17] and long-distance Q-resolution [4]. Conflict analysis must be adjusted to deal with existentially quantified variables propagated by terms, but is simplified in that tautological clauses are no longer a concern. We implemented our new version of QCDCL on top of MiniSat [14] and performed an experimental analysis. Our prototype demonstrated non-trivial use of QU-resolution by quickly solving formulas from the KBKF family [25], which is known to be hard for Q-resolution [8, 25], and the KBKF-LD family, which is hard for long-distance Q-resolution [4]. Further experiments on benchmark instances from recent QBF evaluations [34] demonstrate the potential of using propositional unit propagation with QCDCL.

The remainder of this paper is structured as follows. Section 2 covers standard concepts and definitions used in the rest of the paper. In Section 3, we review the QCDCL algorithm. Section 4 details how it can be modified to use propositional unit propagation and generate QU-resolution proofs. In Section 5, we present an experimental evaluation of a prototype implementation. We conclude in Section 6 with a discussion of our findings.

Related Work

We briefly discuss prior applications of QU-resolution and propositional unit propagation in search-based QBF solving. If the matrix of a QBF is propositionally unsatisfiable, then so is the QBF. This kind of *existential abstraction* was used to improve backtracking search algorithms [37]. If a SAT solver finds that the set of clauses at a node of the search tree is unsatisfiable, the node can be pruned. Otherwise, a satisfying assignment can guide decisions in the subtree. Existential abstraction was similarly integrated into QCDCL as a proof rule for deriving new clauses [31]. In this context, it was observed that the resolution derivation generated by the SAT solver is a QU-resolution derivation. It was even shown that the addition of existential abstraction to Q-resolution results in a proof system that simulates QU-resolution [31]. In both these approaches, the existential abstraction changes at every node in the search tree, and calling the SAT solver each time may result in a significant overhead. QU-resolution can also be used to simulate failed literal detection for QBF [17,18,28]. A version of QCDCL with propositional unit propagation was recently shown to simulate Q-resolution [7]. That work relies on a model of QCDCL as a (non-deterministic) proof system, and unit propagation is restricted to existentially quantified variables.

2 Preliminaries

We assume a countably infinite set of propositional *variables*. A *literal* is a variable x or a negated variable $\neg x$. A *clause* is a finite disjunction ($\lor$) of literals, and a *term* is a finite conjunction ($\land$) of literals. Clause are *tautological* and terms *contradictory* if they contain a variable and its negation. We sometimes consider clauses and terms as sets of literals. A *CNF (conjunctive normal form)* formula is a finite conjunction of non-tautological clauses, and a *DNF (disjunctive normal form)* formula is a finite disjunction of non-contradictory terms. An *(truth) assignment* of a set X of variables is a mapping $\tau : X \to \{0, 1\}$ of variables to truth values. We extend assignments to mappings of literals to truth values by setting $\tau(\neg x) := 1 - \tau(x)$. A literal ℓ is *satisfied* by the assignment τ if $\tau(\ell) = 1$, and *falsified* if $\tau(\ell) = 0$. A clause is satisfied if one of its literals is satisfied, and falsified if all of its literals are falsified. Similarly, a term is falsified if one of its literals is falsified, and satisfied if each of its literals is satisfied. A CNF formula is satisfied if all of its clauses are satisfied, and falsified if one of its clauses is falsified. A DNF formula is satisfied if one of its terms is satisfied, and falsified if all of its terms are falsified.

A *quantified Boolean formula (QBF)* in *prenex CNF* form is a pair $\Phi = \mathcal{Q}.\varphi$, consisting of a *(quantifier) prefix* $\mathcal{Q}$ and a CNF formula φ, called the *matrix* of Φ. The quantifier prefix is a sequence $Q_1x_1 \dots Q_nx_n$, where the x_i are pairwise different variables and $Q_i \in \{\forall, \exists\}$. A variable x_i is *existential* if $Q_i = \exists$, and *universal* if $Q_i = \forall$. We assume that QBF's are *closed*, that is, each variable occurring in φ also occurs in the quantifier prefix. The prefix induces an ordering on its variables as $x_i < x_j$ if $i < j$. When $x_i < x_j$ we say that x_i is *outer* to x_j and x_j is *inner* to x_i. The quantifiers induce a sequence $X_1, \dots, X_l$ of sets of variables, called *quantifier blocks*, appearing in maximal, contiguous subsequences $Q_ix_i \dots Q_jx_j$ such that $Q_k = Q_i$ for each $i \leq k \leq j$. The *truth value* $\mathcal{V}(\Phi)$ of a QBF $\Phi = \mathcal{Q}.\varphi$ can be inductively defined as follows. If Φ contains no variables then $\mathcal{V}(\Phi) = 0$ if φ contains the empty clause, and $\mathcal{V}(\Phi) = 1$ otherwise. When $\mathcal{Q} = Q_1x_1 \dots Q_nx_n$ with $n \geq 1$, we distinguish two cases. If $Q_1 = \forall$, then $\mathcal{V}(\Phi) = 1$ if $\mathcal{V}(\Phi[\neg x_1]) = 1$ and $\mathcal{V}(\Phi[x_1]) = 1$, and $\mathcal{V}(\Phi) = 0$ otherwise. Here, $\Phi[\ell]$ denotes the QBF $\mathcal{Q}'.\varphi[\ell]$ where $\mathcal{Q}' = Q_2x_2 \dots Q_nx_n$ and $\varphi[\ell]$ is obtained from φ by removing each clause that contains ℓ and removing x_1 from all remaining clauses. If $Q_1 = \exists$, we let $\mathcal{V}(\Phi) = 1$ if $\mathcal{V}(\Phi[\neg x_1]) = 1$ or $\mathcal{V}(\Phi[x_1]) = 1$, and $\mathcal{V}(\Phi) = 0$ otherwise. A QBF Φ is *true* (or *satisfiable*) if $\mathcal{V}(\Phi) = 1$, otherwise it is *false* (or *unsatisfiable*).

Q-Resolution

We consider several variants of the Q-resolution proof system [25]. *Derivations* in these systems are sequences of clauses where each clause is derived by an axiom or from clauses appearing earlier in the sequence using one of the proof rules. A *refutation* is derivation of the empty clause. The proof rules of *QU-resolution* [17] are shown in Figure 1. The original *Q-resolution* proof system is obtained by restricting pivot variables x in the resolution rule to existential variables. Q-resolution is sound and (refutationally) complete, that is, a QBF has a Q-resolution refutation if, and only if, it is false [25], and the same holds for QU-resolution [17].

$$\frac{}{C}\ \text{(Axiom)} \qquad\qquad \frac{C \vee u/\neg u}{C}\ \text{($\forall$-Reduction)}$$

In the axiom rule, C is a clause from the matrix. Universal reduction ($\forall$-reduction) allows the removal of a universal variable u or its negation $\neg u$ if u is inner to each variable occurring in C.

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \cup C_2}\ \text{(Resolution)}$$

The *resolvent* clause $C_1 \cup C_2$ must be non-tautological.

Figure 1 The rules of QU-resolution [17].

In Q-resolution and QU-resolution, resolvents must be non-tautological. This requirement can be slightly relaxed so that tautological clauses can be derived in certain cases. This leads to so-called *long-distance Q-resolution* (LDQ-resolution) [2].

3 Quantified CDCL

In this section, we describe a generic version of quantified CDCL following the original presentation [46, 47]. Pseudocode is shown in Listing 1. QCDCL is a backtracking search

Listing 1 Schematic QCDCL algorithm.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      # Propagation falsified a clause or satisfied a term.
      learnt, backtrack_level = analyze(conflict)
      if learnt.empty():
        return learnt.isTerm()
      else:
        backtrack(backtrack_level)
        attach(learnt)
    else:
      # No conflict, branch on an unassigned variable.
      decide()
```

algorithm that maintains a sequence of literals, called the *trail*, which induces a partial assignment of the input variables. The trail is empty initially and extended during search by unit propagation and decisions. *Unit propagation* identifies a clause that simplifies to a unit clause (ℓ) under the current trail assignment and appends ℓ to the trail. By default, this includes the application of universal reduction [13, 46]. We refer to this combination as *quantified unit propagation* (QUP) to distinguish it from propositional unit propagation (UP). Propagating a unit clause may lead to further unit clauses that must be propagated, and this process is repeated until no unit clauses remain or a clause simplifies to the empty clause. In the former case, the algorithm proceeds with a *decision*, which takes an unassigned variable x

Listing 2 QCDCL conflict analysis with long-distance Q-resolution.

```
def analyze(C):
  while not C.empty() and not isAsserting(C):
    l = lastPropagatedExistential(C)
    D = reason(l)
    C = resolveAndReduce(C, D, l)
  backtrack_level = secondHighestDecisionLevel(C)
  return C, backtrack_level
```

and appends one of the literals $x, \neg x$ to the trail. In propositional CDCL, this can be any unassigned variable. In QCDCL, decision variables are typically chosen from the leftmost quantifier block of the prefix (after applying the trail assignment), but there are techniques that allow for more flexibility [27, 33]. The alternation of decisions and unit propagation partitions the trail into *decision levels*. Decision level 0 consists of literals assigned by unit propagation before any decision has been made, and each decision increases the decision level by one.

A *conflict* occurs when unit propagation finds a falsified clause. *Conflict analysis*, which we will describe in more detail below, takes a conflict as a starting point to derive a new *learnt clause* using Q-resolution or long-distance Q-resolution (or, as we will later see, QU-resolution). If the learnt clause is empty, QCDCL has found a refutation and the input QBF is false. Otherwise, all literals with decision level greater than a certain *backtrack level* returned by conflict analysis are removed from the trail, and the learnt clause is added to the formula.

So far, we have only considered clauses from (or derived from) a CNF representation of the matrix. QCDCL simultaneously operates on terms from a DNF representation of the matrix in a completely dual manner. Unit propagation includes the (falsifying) assignment of universal literals occurring in unit terms, and if a term is satisfied, conflict analysis derives a new learnt term. When the matrix of a QBF is given in CNF, the terms in the DNF representation may be derived by *model generation* [19] or simply by Tseitin transformation of the negated matrix [24]. We will ignore these details and simply assume that the matrix is given as both CNF and DNF such that every complete assignment either falsifies a clause or satisfies a term. Further, to simplify the presentation, we will typically only describe algorithms for the clausal representation of the matrix (for terms, the roles of existential and universal variables are switched).

Listing 2 shows pseudocode for the conflict analysis routine in QCDCL with QUP and long-distance Q-resolution (cf. [15]). Starting from a clause C that was falsified by propagation, it derives a clause that is either empty or *asserting*. In the case of QUP, a clause is asserting if it contains a unique existential variable (the *asserting variable*) at the maximum decision level among existential variables in the clause, and any universal variable appearing in the clause that is outer to the asserting variable belongs to a lower decision level. Furthermore, the decision level of the asserting variable must be greater than 0 (if it is 0 the empty clause can be derived). An asserting clause becomes a unit clause (with respect to QUP) after backtracking to the second highest decision level present in the clause (or 0 if there is no such decision level) and propagates the asserting variable.

The conflict clause C itself cannot be asserting, as otherwise the asserting variable would have been propagated at a lower decision level. Until it arrives at an asserting clause (or the empty clause), conflict analysis repeatedly derives a new clause as follows. It finds the existential literal $\ell \in C$ that was last falsified by propagation, then resolves C with the clause D that was the *reason* for propagating $\neg\ell$, and applies universal reduction to the resolvent. The reason clause is such that it simplified to the unit clause $(\neg\ell)$ under a partial trail assignment.

This always leads to an empty or asserting clause, because eventually, any existential variables remaining in the derived clause must have been assigned by a decision. In particular, the innermost variable must be an existential decision variable e. If the clause would contain a universal variable u that is unassigned or assigned at a higher decision level, then the variable u would have been unassigned at the time when variable e was picked as a decision variable. But this is impossible if the solver follows the decision policy described earlier.

4 QCDCL with Propositional Unit Propagation

In this section, we describe how to modify QCDCL to use propositional unit propagation, starting with a version that uses Q-resolution as its underlying proof system (Section 4.1). Propositional unit propagation is weaker than the combination of propositional unit propagation and universal reduction commonly used in QCDCL solvers, but it allows for the propagation of universal variables, which leads to a version of QCDCL that generates QU-resolution proofs (Section 4.2).

4.1 Propagation and Learning with Q-Resolution

A simple way of integrating propositional unit propagation with QCDCL is to stop propagating when detecting a unit clause (u) that contains a single universal literal u. Instead of assigning $u := 1$, propagation assigns $u := 0$ (at the current decision level) and reports a conflict. We can still use the algorithm shown in Listing 2 for clause learning, but the definition of an asserting clause must be adapted as follows. The asserting variable is now the unique variable at maximum decision level among *all* variables in the clause, and it must be existential and at a decision level greater than 0.

Clauses derived during conflict analysis cannot be tautological, since the universal literals occurring in both premises of a resolution step are falsified by the trail assignment. Again, conflict analysis always terminates with a clause that is empty or asserting, since the innermost existential variable will eventually be a decision, so that outer universal variables must have been assigned at a lower decision level.

Note that, even though tautological clauses are no longer a concern, conflict analysis may still have to proceed beyond the last decision variable (even if it is existential), since the derived clause may contain universal variables that were propagated by terms. These cannot be removed by resolution and must instead be taken care of by universal reduction, which in turn requires that "blocking" existential variables inner to these universal variables be resolved out. Such blocking variables can be assigned at any decision level due to propagation, and may lead to conflict analysis visiting earlier decision levels.

4.2 Propagation and Learning with QU-Resolution

To get QU-resolution as the underlying proof system, we modify the algorithm described in the previous subsection by allowing the propagation of universal unit clauses (respectively, existential unit terms). That is, upon detecting a unit clause (u) where u is a universal literal, the algorithm proceeds by assigning $u := 1$, and a conflict arises only when unit propagation falsifies a clause.

Allowing unit propagation of variables regardless of their associated quantifier requires several modifications to conflict analysis. First, since unit propagation of clauses can now assign universal variables, the asserting variable is no longer required to be existential. That is, a clause is considered asserting if there is a unique variable at maximum decision level greater than 0. While the clause is not empty or asserting, it has to contain a falsified existential literal (otherwise, universal reduction would have derived the empty clause). More generally, there has to be a literal ℓ in the current clause that was last falsified by propagation, not including universal literals that were propagated by terms (which are to be removed by universal reduction). If literal ℓ was propagated by a clause D, a new clause is derived as the resolvent of C and D, followed by universal reduction. Note that the pivot may be a universal variable.

Otherwise, literal ℓ is an existential literal propagated by a term D. Such a literal can be removed neither by universal reduction nor by resolution, so conflict analysis cannot derive an asserting clause. Noting that an existential variable propagated by a term is the dual of a universal variable propagated by a clause, we proceed by reverting to the strategy from the previous subsection: the term D is considered satisfied, and conflict analysis restarts with the term D as the conflict term. Pseudocode for the new conflict analysis routine is shown as Listing 3. A detailed example is presented at the end of this section.

4.3 Soundness and Termination

Soundness of the above variants of QCDCL follows from soundness of QU-resolution: the algorithms return *false* if the empty clause was derived by QU-resolution, or *true* if the empty term was derived by the dual proof system for terms. For termination, first observe that decisions and propagation always leads to a conflict. Second, conflict analysis always terminates. This is because variables are visited and resolved in reverse trail order, so that no variable is resolved twice, and the trail index of pivot variables decreases in each step.[2] If conflict analysis does not terminate with an empty clause or term (in which case the algorithms terminate immediately), it derives an asserting clause (or term). Since this clause is a unit clause at an earlier decision level than the one at which the conflict occurred, and it propagates an assignment of the asserting variable that differs from its assignment on the current trail, it cannot occur among the original or previously learnt clauses. So each clause derived by learning is new, and since there are at most 3^n (non-tautological) clauses that can be derived from a QBF with n variables, the algorithms must terminate eventually.

4.4 Adding Dependency Learning

The above arguments for why QCDCL conflict analysis arrives at an asserting clause rely on a particular policy for choosing decision variables. Specifically, it was assumed that only variables from the outermost quantifier block (with assigned variables removed from the prefix) are considered as decision variables. It has frequently been observed that this policy is needlessly restrictive, and many techniques have been proposed to enable more liberal decision heuristics (see [27, 33] and the references therein). So-called dependency learning is a lazy approach that assumes variables can be assigned in any order until conflict analysis fails to derive an asserting clause [33]. When that happens, a pair (x, y) of variables is added

[2] Note that when conflict analysis is reset due to an existential literal ℓ propagated by a term D, literal ℓ (which was assigned last among variables in D) is *not* selected as the next pivot, since only satisfied literals in D are considered, and ℓ was set to false by unit propagation.

Listing 3 QCDCL conflict analysis with QU-resolution.

```
def analyze(C):
  while not C.empty() and not isAsserting(C):
    l = getPivot(C)
    D = reason(l)
    if not C.isClause() == D.isClause():
      # Variable at maximum DL cannot be removed.
      # Restart analysis with conflict clause/term D.
      C = D
      continue
    C = resolveAndReduce(C, D, l)
  backtrack_level = secondHighestDecisionLevel(C)
  return C, backtrack_level

def getPivot(C):
  # Pivot literals must be false if C is a clause or
  # true if C is a term.
  pivot_value = C.isTerm()
  candidates = [l for l in C if value(l) == pivot_value]
  # Moreover, pivots l must have been assigned by propagation
  # of a unit clause or term, which is stored as reason(l).
  candidates = [l for l in candidates if reason(l) is not None]
  # Universal variables propagated by terms must be removed
  # by universal reduction, so we exclude them.
  irreducible = existential if C.isClause() else universal
  pivots = [l for l in C if varType(var(l)) == irreducible or
                           C.isClause() == reason(l).isClause()]
  # Choose the variable that was assigned last.
  return max(pivots, key = trailIndex)
```

to a dependency relation, and variable y is only considered eligible for decision once x has been assigned. This technique was originally introduced for QCDCL with long-distance Q-resolution, but it can be easily adapted to the versions of QCDCL with propositional unit propagation presented here. Instead of assuming that an existential decision variable e cannot block reduction of a universal variable u due to the decision policy, we detect such cases, introduce a dependency (u, e), and backtrack to undo the decision level corresponding to variable e. Unlike in QCDCL with long-distance Q-resolution, propagation does not need to take dependencies into account, and no further changes are necessary. Termination is still ensured since a dependency is added whenever no asserting clause can be derived, and the decision policy will eventually revert to the strict policy based on the prefix order.

We conclude this section with an example illustrating a run of the new QCDCL algorithm with dependency learning and QU-resolution as a proof system.

▶ **Example 1.** Let $\Phi = \exists e_2, e_3 \forall a_1 \exists e_4 . \varphi$ be the KBKF-LD [4] formula for $n = 1$, where

$$\varphi = \underbrace{(e_2 \vee a_1 \vee \neg e_4)}_{C_1} \wedge \underbrace{(e_3 \vee \neg a_1 \vee \neg e_4)}_{C_2} \wedge \underbrace{(a_1 \vee e_4)}_{C_3} \wedge \underbrace{(\neg a_1 \vee e_4)}_{C_4} \wedge \underbrace{(\neg e_2 \vee \neg e_3 \vee \neg e_4)}_{C_5} .$$

- Initially, there are no dependencies and the trail $e_2@1, \neg e_4@2$ is obtained by decisions.[3] Clause C_3 then propagates a_1, and the resulting trail $e_2@1, \neg e_4@2, a_1@2(C_3)$ falsifies C_4. Conflict analysis finds a_1 as pivot variable with C_3 as a reason. Resolving C_4 with C_3 derives the unit clause $C_6 = (e_4)$, which is asserting and leads to the algorithm backtracking to decision level 0 and propagating e_4.
- Starting from $e_4@0$, the decision a_1 and subsequent propagations result in the trail $e_4@0(C_6), a_1@1, e_3@1(C_2), \neg e_2@1(C_5)$. This trail satisfies the matrix, and the corresponding term $(\neg e_2 \wedge e_3 \wedge a_1 \wedge e_4)$ serves as a starting point for conflict analysis. Existential reduction derives the term $(\neg e_2 \wedge e_3 \wedge a_1)$, which is not asserting since all three variables are assigned at decision level 1. The existential variables e_2, e_3 were propagated by clauses and cannot be removed by term resolution (also known as *consensus*), so existential reduction must be used instead. In particular, the outermost variable e_2 must be removed. However, reduction cannot be applied due to the blocking universal decision variable a_1. The algorithm learns the dependency (e_2, a_1), and backtracks to decision level 0.
- Next, the decision $\neg e_2$ leads to the trail $e_4@0(C_6), \neg e_2@1, a_1@1(C_1), e_3@1(C_2)$, so that conflict analysis again starts at the term $(\neg e_2 \wedge e_3 \wedge a_1)$. As in the previous conflict, variable e_2 must be removed by existential reduction, but in this case, the blocking universal variable a_1 was propagated by clause C_1. Conflict analysis cannot derive a term and instead restarts with C_1 as a conflict clause. Literal a_1 is not considered as a potential pivot since it is satisfied by the trail assignment, and variable e_2 is a decision variable, so e_4 is chosen as a pivot. Resolving C_1 with $C_6 = (e_4)$ and applying universal reduction, the algorithm derives the unit clause $C_7 = (e_2)$.
- The algorithm arrives at the trail $e_4@0(C_6), e_2@0(C_7), \neg e_3@0(C_5), \neg a_1@0(C_2)$ by propagation, which satisfies the matrix and leads to the initial term $(e_2 \wedge \neg e_3 \wedge \neg a_1)$ by existential reduction. Again, conflict analysis cannot derive a term because the blocking universal variable a_1 was propagated by a clause. Resuming conflict analysis at clause C_2, variable e_3 is chosen as a pivot with reason clause C_5. Resolving C_2 and C_5 results in the clause $(\neg e_2 \vee \neg a_1 \vee \neg e_4)$. Further resolving with the unit clauses $C_7 = (e_2)$ and $C_6 = (e_4)$ and applying universal reduction derives the empty clause, so the algorithm reports that the QBF Φ is false.

5 Implementation and Experiments

We implemented QCDCL with propositional unit propagation on top of MiniSat [14] in a system named MiniQU.[4] The performance of MiniQU was evaluated in two sets of experiments. First, we ran the solver on crafted instances that are provably hard to solve for Q-resolution and even LDQ-resolution [4], to determine whether MiniQU can find short proofs by leveraging QU-resolution. In the second experiment, we compared the system with the best publicly available solvers using instances from recent QBF evaluations [34]. All experiments were run on a cluster with Intel Xeon E5649 processors at 2.53 GHz running 64-bit Linux, with a memory limit of 8 GB.

[3] We write trails as sequences $\ell_1@dl_1(C_1), \ldots, \ell_i@dl_j(C_i)$ of literals with their associated decision level and, if assigned by propagation, their reason clause/term.

[4] `https://github.com/fslivovsky/miniQU`

5.1 Implementation

MiniQU supports the following propagation modes and underlying proof systems: UP with Q-resolution (Q), UP with QU-resolution (QU), and QUP with long-distance Q-resolution (LDQ). Being based on MiniSat, the solver inherits many of its characteristics, such as the restart policy, the decision heuristic (restricted by learned dependencies or prefix order), and an aggressive clause (and term) cleaning strategy. We briefly list a few relevant features and design choices:

- MiniQU accepts QDIMACS (prenex CNF) and QCIR (prenex non-CNF) formulas.
- For QDIMACS, initial terms are obtained by model generation [19] when all variables are assigned without conflict.
- Clause and term cleaning is performed based on the *literal blocks distance* (LBD) [1] and activity, rather than activity alone. As in MiniSat, half of the learned clauses and terms are removed. Clauses and terms with LBD at most 2 are kept indefinitely.
- Dependency learning is supported in Q-mode and QU-mode, and dependencies are reset every 20 restarts. For technical reasons, dependency learning is not supported in LDQ-mode.

5.2 Experiments with Crafted Instances

To determine whether MiniQU in QU-mode takes advantage of QU-resolution, we performed experiments with classes of formulas that have short QU-resolution refutations but no short proofs in Q-resolution or long-distance Q-resolution. For this purpose, we used two variants of the KBKF family of formulas introduced by and named after Kleine Büning, Karpinski, and Flögel [25]. Instances from these families were generated with the QBFFam[5] tool [9]. In both cases, we used parameter values between $n = 5$ and $n = 60$, and ran MiniQU in five different configurations: Q-mode with (Q-dl) and without (Q) dependency learning, QU-mode with (QU-dl) and without (QU) dependency learning, as well as LDQ-mode. The time limit for each run was set to 600 seconds.

We first considered the original KBKF class, which has short QU-resolution [17] and LDQ-resolution [15] refutations, but requires Q-resolution proofs of exponential size [8, 25]. The left plot in Figure 2 clearly shows that the running time of MiniQU in Q-mode grew exponentially with the parameter n. By contrast, the solver was able to find short refutations in LDQ-mode and QU-mode. Notably, in QU-mode this was the case only when dependency learning was active. There was also a clear gap between LDQ-mode and QU-mode, with LDQ-mode honing in on short proofs more reliably.

Second, we ran a similar experiment on the KBKF-LD family, which has short proofs in QU-resolution but requires exponential LDQ-resolution proofs (and therefore also exponential Q-resolution proofs) [4]. The right plot of Figure 2 shows an exponential growth in running time exhibited by MiniQU in Q-mode and LDQ-mode. In QU-mode, the solver found short refutations of these instances. Unlike in the first experiment, this was the case regardless of whether dependency learning was active. In fact, for larger values of n, the running times were lower and showed less variance without dependency learning.

[5] `https://github.com/marseidl/qbffam`

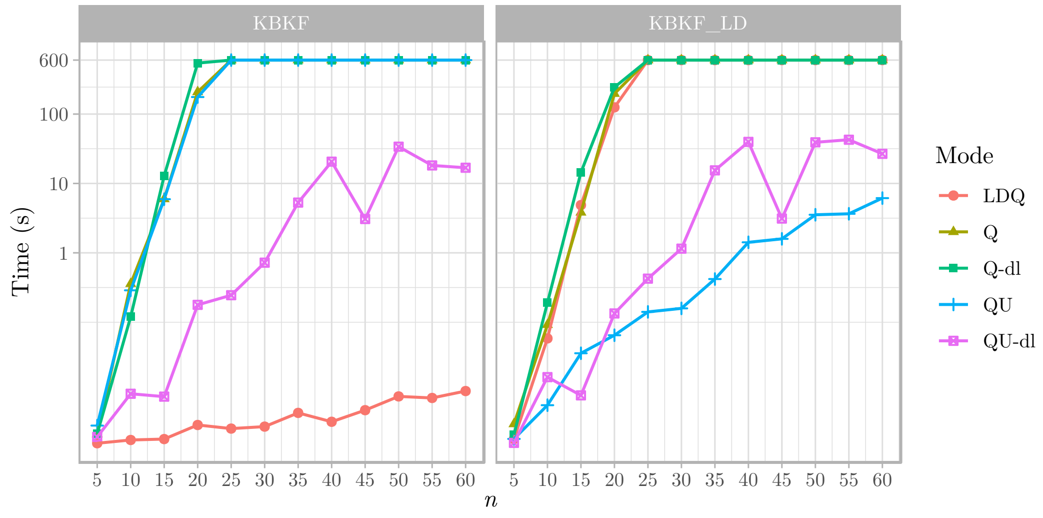

Figure 2 Running time (y-axis) of MINIQU in different modes on instances from the KBKF (left) and KBKF-LD (right) families, for different values of n (x-axis).

5.3 Experiments with QBF Evaluation Benchmarks

To assess whether QCDCL with propositional unit propagation is viable on standard benchmarks, we ran MINIQU alongside publicly available solvers on families from the 2019 and 2020 QBF evaluations [34]. For each run, we set a time limit of 900 seconds and capped memory at 8 GB using RUNSOLVER [36].

We first considered the union of the QBF evaluation 2019 and 2020 prenex non-CNF (QCIR) benchmark sets comprising 504 instances. Out of these, 40 instances had to be removed due to incorrect encodings that caused parsing errors, leaving 464 instances. Again, we ran MINIQU in five different configurations arising from the available propagation modes and dependency learning switched on and off.

Table 1 (left) shows a comparison with the solvers QUABS [42], QFUN [21], GHOSTQ [26], and QUTE [33]. Figure 3 displays these results as a cumulative solved instances plot. MINIQU solved the most instances in LDQ-mode, closely followed by Q-mode with dependency learning. In these configurations, the solver's results were on a par with QUABS and QFUN. Fewer instances were solved in QU-mode. Dependency learning improved the number of solved formulas in both Q-mode and QU-mode.

Next, we considered the union of the QBF evaluation 2019 and 2020 prenex CNF (QDIMACS) sets totaling 716 instances. These were first preprocessed by HQSPRE [45] using default settings and a timeout of 600 seconds. We subsequently removed instances for which preprocessing timed out or that were solved by preprocessing, leaving 455 instances. On these remaining (preprocessed) formulas, we ran MINIQU alongside the solvers CAQE [35], DEPQBF [27], and RAREQS [22]. Results are shown in Table 1 (right) and Figure 4.

MINIQU solved notably fewer instances than the two abstraction solvers. However, it was able to solve more instances than DEPQBF in all but one configuration. Again, the configurations using QU-mode solved the fewest instances. Curiously, unlike in the previous experiment, dependency learning reduced the number of solved instances.

We initially suspected that the lower number of solved instances in QU-mode might be due to an overhead caused by frequent resets of conflict analysis (Line 5 in Listing 3), resulting in fewer learned clauses and terms. To get a sense of raw performance, we compared the number

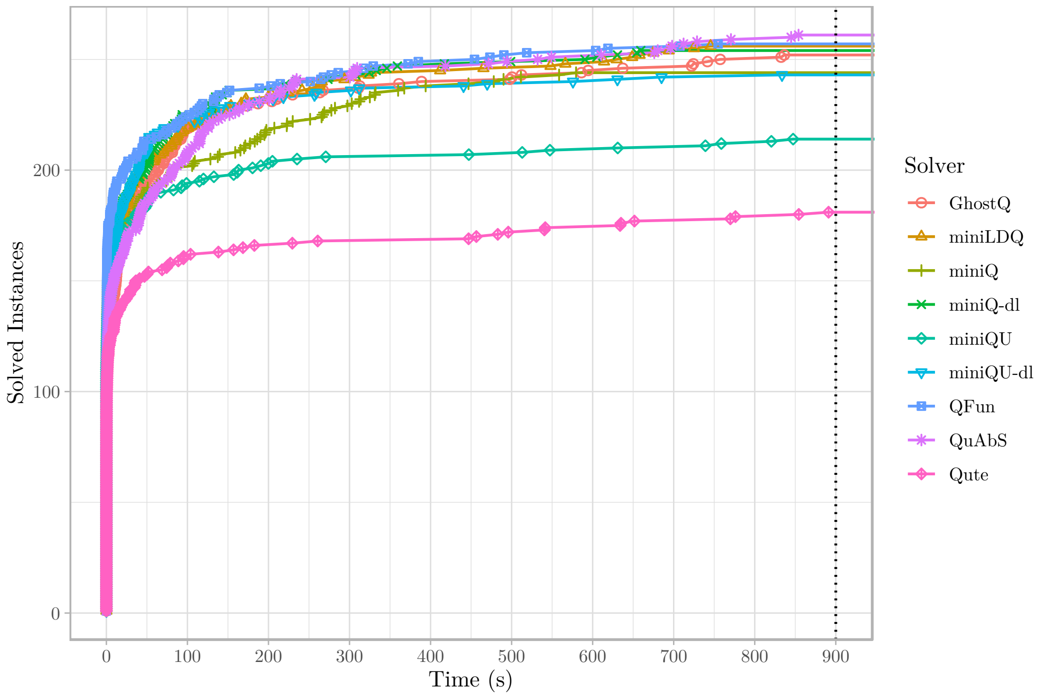

Figure 3 Solved instances (y-axis) by time (x-axis) for prenex non-CNF (QCIR) instances.

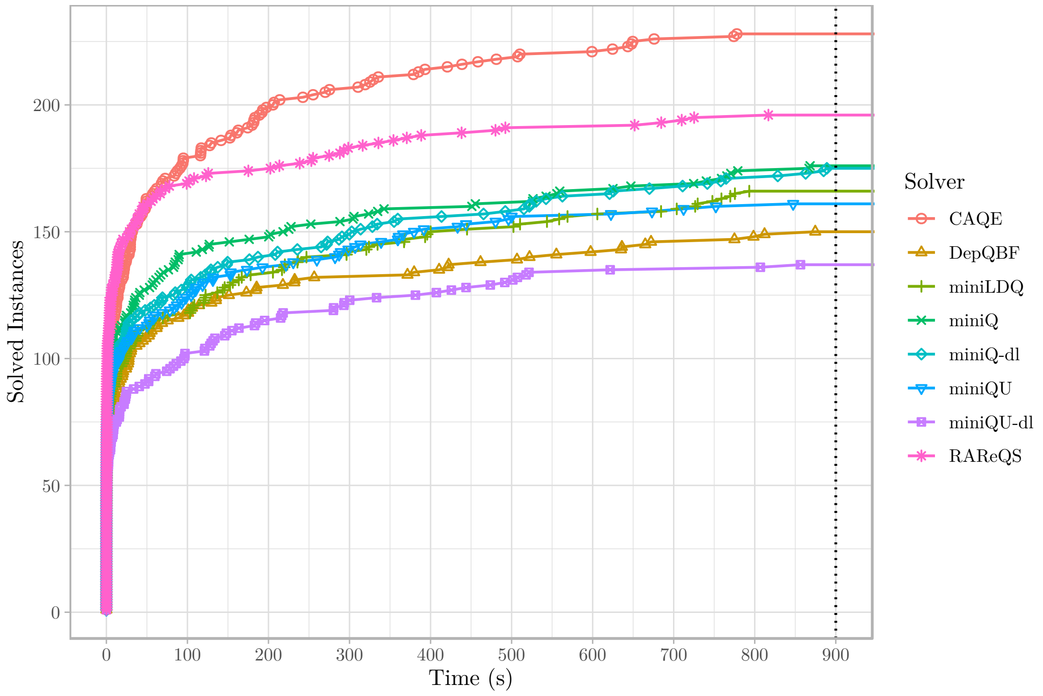

Figure 4 Solved instances (y-axis) by time (x-axis) for prenex non-CNF (PCNF) instances.

Prenex non-CNF (QCIR)			
Solver	#	SAT	UNSAT
QuAbS	261	150	111
QFun	257	143	114
miniLDQ	256	150	106
miniQ-dl	254	152	102
GhostQ	252	137	115
miniQ	244	149	95
miniQU-dl	243	145	98
miniQU	214	136	78
Qute	181	101	80

Prenex CNF (PCNF)			
Solver	#	SAT	UNSAT
CAQE	228	92	136
RAReQS	196	75	121
miniQ	176	65	111
miniQ-dl	175	62	113
miniLDQ	166	63	103
miniQU	161	57	104
DepQBF	150	60	90
miniQU-dl	137	45	92

Table 1 Results for prenex non-CNF (left) and prenex CNF (right) benchmarks. For each solver, the number of solved instances (#), the number of solved true instances (SAT), and the number of solved false instances (UNSAT) are reported.

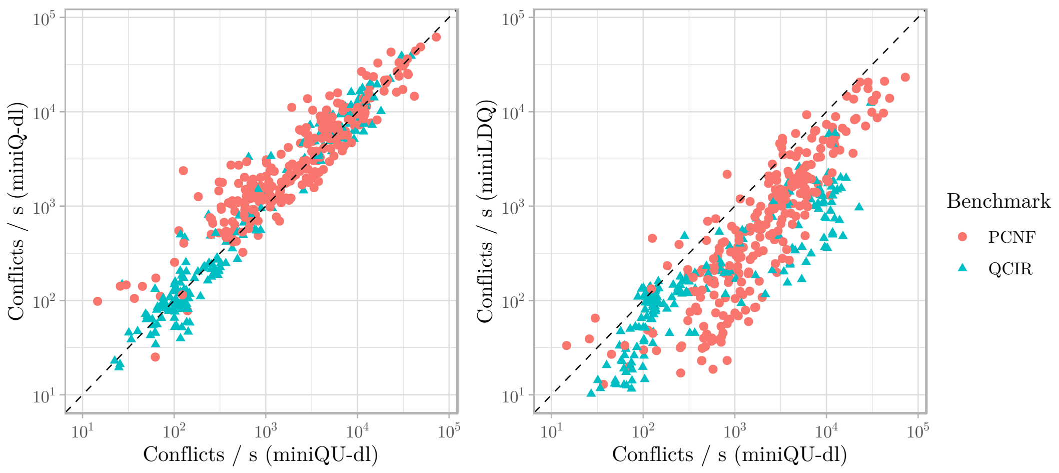

Figure 5 Comparison of conflicts per second on instances with timeouts between different modes.

of conflicts per second on instances that caused a timeout in Q-mode and QU-mode, as well as in LDQ-mode and QU-mode. Figure 5 (left side) shows that Q-mode and QU-mode are fairly evenly matched, contradicting our hypothesis that QU-mode slows down learning. At the same time, Figure 5 shows (right side) a substantial difference between LDQ-mode and QU-mode, with the number of conflicts in QU-mode exceeding those in LDQ-mode by about one order of magnitude on many instances.

6 Discussion

This paper introduced new versions of QCDCL that leverage propositional unit propagation. The experimental results presented in the previous section showcase the potential of these algorithms. QCDCL with QU-resolution as the underlying proof system was able to find short proofs of crafted formulas that are unavailable to other versions of QCDCL. Unfortunately, this advantage did not translate to improved performance on standard benchmark sets. However, a simpler version of the algorithm with Q-resolution as its underlying proof system performed very well, which suggests that the main challenge is proof search, rather than the strength of the underlying proof system. This is consistent with the excellent performance of the solver CAQE, whose traces correspond to a restricted version of Q-resolution [43].

Aside from performance, simplicity is a reason to adopt propositional unit propagation in QCDCL. For instance, it was possible to port clause minimization as implemented in MiniSat [39] to our system with minimal changes (with the caveat that universal reduction is currently not taken into account). We hope that a simplified QCDCL algorithm will allow for SAT techniques to be more easily integrated with solvers, and encourage experimentation.

Finally, there is a good theoretical argument in favor of these new QCDCL algorithms: it was recently shown that, as a non-deterministic proof system, QCDCL with propositional unit propagation is able to p-simulate Q-resolution, provided that the decision heuristic can ignore the variable order in the quantifier prefix [7]. In principle, such liberal decision policies are also supported by our new versions of QCDCL, through a fortuitous interaction of propositional unit propagation with dependency learning. Unlike in the original version [33], learned dependencies play no role in unit propagation, and can only result from out-of-order decisions where conflict analysis was unable to derive an asserting clause. Dependencies are added to ensure that the decision heuristic does not keep making the same "mistake" indefinitely, but this could possibly be achieved by other means, such as randomization.

References

1 Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In Craig Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 399–404, 2009.

2 Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF certification and its applications. *Formal Methods Syst. Des.*, 41(1):45–65, 2012.

3 Valeriy Balabanov, Jie-Hong Roland Jiang, Mikolas Janota, and Magdalena Widl. Efficient extraction of QBF (counter)models from long-distance resolution proofs. In *AAAI*, pages 3694–3701. AAAI Press, 2015.

4 Valeriy Balabanov, Magdalena Widl, and Jie-Hong R. Jiang. QBF resolution systems and their proof complexities. In *SAT*, volume 8561 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2014.

5 Roberto Baldoni, Emilio Coppa, Daniele Cono D'Elia, Camil Demetrescu, and Irene Finocchi. A survey of symbolic execution techniques. *ACM Comput. Surv.*, 51(3):50:1–50:39, 2018.

6 Olaf Beyersdorff, Joshua Blinkhorn, and Meena Mahajan. Building strategies into QBF proofs. *J. Autom. Reason.*, 65(1):125–154, 2021.

7 Olaf Beyersdorff and Benjamin Böhm. Understanding the relative strength of QBF CDCL solvers and QBF resolution. In *ITCS*, volume 185 of *LIPIcs*, pages 12:1–12:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

8 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.

9 Olaf Beyersdorff, Luca Pulina, Martina Seidl, and Ankit Shukla. Qbffam: A tool for generating QBF families from proof complexity. In *SAT*, volume 12831 of *Lecture Notes in Computer Science*, pages 21–29. Springer, 2021.

10 Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In Rance Cleaveland, editor, *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS '99, Proceedings*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 1999.

11 Nikolaj Bjørner, Mikolás Janota, and William Klieber. On conflicts and strategies in QBF. In *LPAR (short papers)*, volume 35 of *EPiC Series in Computing*, pages 28–41. EasyChair, 2015.

12 Roderick Bloem, Nicolas Braud-Santoni, Vedad Hadzic, Uwe Egly, Florian Lonsing, and Martina Seidl. Two SAT solvers for solving quantified boolean formulas with an arbitrary number of quantifier alternations. *Formal Methods Syst. Des.*, 57(2):157–177, 2021.

13 Marco Cadoli, Marco Schaerf, Andrea Giovanardi, and Massimo Giovanardi. An algorithm to evaluate quantified boolean formulae and its experimental evaluation. *J. Autom. Reason.*, 28(2):101–142, 2002.

14 Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT 2003*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

15 Uwe Egly, Florian Lonsing, and Magdalena Widl. Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In *LPAR*, volume 8312 of *Lecture Notes in Computer Science*, pages 291–308. Springer, 2013.

16 Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda. SAT competition 2020. *Artif. Intell.*, 301:103572, 2021.

17 Allen Van Gelder. Contributions to the theory of practical quantified boolean formula solving. In *CP*, volume 7514 of *Lecture Notes in Computer Science*, pages 647–663. Springer, 2012.

18 Allen Van Gelder, Samuel B. Wood, and Florian Lonsing. Extended failed-literal preprocessing for quantified boolean formulas. In *SAT*, volume 7317 of *Lecture Notes in Computer Science*, pages 86–99. Springer, 2012.

19 Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Clause/term resolution and learning in the evaluation of quantified boolean formulas. *J. Artif. Intell. Res.*, 26:371–416, 2006.

20 Marijn Heule, Matti Järvisalo, Florian Lonsing, Martina Seidl, and Armin Biere. Clause elimination for SAT and QSAT. *J. Artif. Intell. Res.*, 53:127–168, 2015.

21 Mikolás Janota. Towards generalization in QBF solving via machine learning. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18)*, pages 6607–6614. AAAI Press, 2018.

22 Mikolás Janota, William Klieber, João Marques-Silva, and Edmund M. Clarke. Solving QBF with counterexample guided refinement. *Artif. Intell.*, 234:1–25, 2016.

23 Mikolás Janota and João Marques-Silva. Solving QBF by clause selection. In *IJCAI*, pages 325–331. AAAI Press, 2015.

24 Mikolás Janota and João Marques-Silva. An achilles' heel of term-resolution. In *EPIA*, volume 10423 of *Lecture Notes in Computer Science*, pages 670–680. Springer, 2017.

25 Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified boolean formulas. *Inf. Comput.*, 117(1):12–18, 1995.

26 William Klieber. Ghostq. *J. Satisf. Boolean Model. Comput.*, 11(1):65–72, 2019.

27 Florian Lonsing and Armin Biere. Integrating dependency schemes in search-based QBF solvers. In *SAT*, volume 6175 of *Lecture Notes in Computer Science*, pages 158–171. Springer, 2010.

28 Florian Lonsing and Armin Biere. Failed literal detection for QBF. In *SAT*, volume 6695 of *Lecture Notes in Computer Science*, pages 259–272. Springer, 2011.

29 Florian Lonsing and Uwe Egly. Evaluating QBF solvers: Quantifier alternations matter. In *CP*, volume 11008 of *Lecture Notes in Computer Science*, pages 276–294. Springer, 2018.

30 Florian Lonsing, Uwe Egly, and Allen Van Gelder. Efficient clause learning for quantified boolean formulas via QBF pseudo unit propagation. In *SAT*, volume 7962 of *Lecture Notes in Computer Science*, pages 100–115. Springer, 2013.

31 Florian Lonsing, Uwe Egly, and Martina Seidl. Q-resolution with generalized axioms. In *SAT*, volume 9710 of *Lecture Notes in Computer Science*, pages 435–452. Springer, 2016.

32 Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *DAC*, pages 530–535. ACM, 2001.

33 Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Dependency learning for QBF. *J. Artif. Intell. Res.*, 65:180–208, 2019.

34 Luca Pulina and Martina Seidl. The 2016 and 2017 QBF solvers evaluations (qbfeval'16 and qbfeval'17). *Artif. Intell.*, 274:224–248, 2019.

35 Markus N. Rabe and Leander Tentrup. CAQE: A certifying QBF solver. In *FMCAD*, pages 136–143. IEEE, 2015.

36 Olivier Roussel. Controlling a solver execution with the runsolver tool. *J. Satisf. Boolean Model. Comput.*, 7(4):139–144, 2011.

37 Horst Samulowitz and Fahiem Bacchus. Using SAT in QBF. In *CP*, volume 3709 of *Lecture Notes in Computer Science*, pages 578–592. Springer, 2005.

38 Ankit Shukla, Armin Biere, Luca Pulina, and Martina Seidl. A survey on applications of quantified boolean formulas. In *ICTAI*, pages 78–84. IEEE, 2019.

39 Niklas Sörensson and Armin Biere. Minimizing learned clauses. In *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 237–243. Springer, 2009.

40 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 – May 2, 1973, Austin, Texas, USA*, pages 1–9. ACM, 1973.

41 Martin Suda and Bernhard Gleiss. Local soundness for QBF calculi. In *SAT*, volume 10929 of *Lecture Notes in Computer Science*, pages 217–234. Springer, 2018.

42 Leander Tentrup. Non-prenex QBF solving using abstraction. In *SAT*, volume 9710 of *Lecture Notes in Computer Science*, pages 393–401. Springer, 2016.

43 Leander Tentrup. On expansion and resolution in CEGAR based QBF solving. In *CAV (2)*, volume 10427 of *Lecture Notes in Computer Science*, pages 475–494. Springer, 2017.

44 Yakir Vizel, Georg Weissenbacher, and Sharad Malik. Boolean satisfiability solvers and their applications in model checking. *Proc. IEEE*, 103(11):2021–2035, 2015.

45 Ralf Wimmer, Christoph Scholl, and Bernd Becker. The (D)QBF preprocessor HQSpre – Underlying theory and its implementation. *J. Satisf. Boolean Model. Comput.*, 11(1):3–52, 2019.

46 Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified boolean satisfiability solver. In *ICCAD*, pages 442–449. ACM / IEEE Computer Society, 2002.

47 Lintao Zhang and Sharad Malik. Towards a symmetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. In *CP*, volume 2470 of *Lecture Notes in Computer Science*, pages 200–215. Springer, 2002.