

Efficient Extraction of QBF (Counter)models from Long-Distance Resolution Proofs [★]

Valeriy Balabanov¹, Jie-Hong R. Jiang¹, Mikoláš Janota², and Magdalena Widl³

¹GIEE, National Taiwan University, Taipei, Taiwan

²INESC-ID, Lisbon, Portugal

³Vienna University of Technology, Austria

1 Introduction

The broad range of applications for QBF [4, 8, 10, 13], and the tremendous success of solvers deciding the satisfiability problem of propositional logic (SAT) motivate the increasing effort being dedicated to QBF solving. Modern QBF solvers are mostly based on QCDCL, a variation of the resolution based *conflict driven clause learning* (CDCL) in SAT solving, and employ resolution calculi adapted to QBF: Q-resolution [9] and its extension long-distance Q-resolution (LQ-resolution) [1, 15]. In Q-resolution, the derivation of tautological clauses is not allowed. LQ-resolution specifies conditions under which the derivation of tautological clauses is sound and thus provides a more flexible proof system. There is strong evidence that LQ-resolution can produce shorter proofs than Q-resolution [6].

In addition to a syntactic proof to verify the result of the solver, the generation of a semantic (*counter*)*model* of the falsity (truth) of a QBF is crucial for many applications. Such a model, expressed as Herbrand (Skolem) function, can be extracted from a Q-resolution proof of a QBF in time linear in the proof size [1]; however, state-of-the-art algorithms to extract them from LQ-resolution proofs are exponential in the proof size [1, 2].

We present a linear-time algorithm to extract Herbrand (Skolem) functions from LQ-resolution proofs based on previous work on their extraction from Q-resolution proofs [1] and on strategy extraction from Q- and LQ-resolution proofs [6, 7]. Our experimental results clearly show the effectiveness of our algorithm compared to current approaches in the extraction time and the quality of the functions.

All results of this work have been submitted as technical paper to the International Conference on Principles and Practice of Constraint Programming 2014.

2 Preliminaries

We consider closed quantified Boolean formulas (QBF) in PCNF under their usual syntax and semantics and follow the notation used in [1]. In particular, we replace the appearance of both literals x and $\bar{x}$ in a clause by a *merged literal*, x^* .

[★] This work was supported in part by the National Science Council (NSC) of Taiwan under grants 101-2923-E-002-015-MY2 and 102-2221-E-002-232, by the Vienna Science and Technology Fund (WWTF) through project ICT10-018, and by the Austrian Science Fund (FWF) under grant S11409-N23.

PhaseFunctionBuild

input: a false QBF Φ and its LQ-resolution DAG $G_\Pi(V_\Pi, E_\Pi)$
output: an updated $G_\Pi(V_\Pi, E_\Pi)$ annotated with phase functions
begin
01 **foreach** vertex v of G_Π in topological order
02 **foreach** variable $x \in \text{vars}(v.\text{clause})$
03 **if** $x \in v.\text{clause}$ **then** $v.\text{phase}(x) = 1$;
04 **else if** $\bar{x} \in v.\text{clause}$ **then** $v.\text{phase}(x) = 0$;
05 **else** // $x^* \in v.\text{clause}$
06 **if** $v.\text{clause} = \text{reduce}(u.\text{clause})$
07 $v.\text{phase}(x) := u.\text{phase}(x)$;
08 **else if** $v.\text{clause} = \text{resolve}(u_1.\text{clause}, u_2.\text{clause})$ with pivot literal
 $p \in u_1.\text{clause}$ and $\bar{p} \in u_2.\text{clause}$
09 **if** $x \notin \text{vars}(u_1.\text{clause})$
10 $v.\text{phase}(x) := u_2.\text{phase}(x)$;
11 **else if** $x \notin \text{vars}(u_2.\text{clause})$
12 $v.\text{phase}(x) := u_1.\text{phase}(x)$;
13 **else**
14 $v.\text{phase}(x) := (\bar{p} \wedge u_1.\text{phase}(x)) \vee (p \wedge u_2.\text{phase}(x))$;
end

Fig. 1. Algorithm: Phase Function Derivation.

We use standard definitions of the sound and complete proof systems Q-resolution [9] and LQ-resolution [1] for false QBFs, and Q-consensus and LQ-consensus over cubes for true QBFs. We regard a proof Π as a directed acyclic graph (DAG) over a set V_Π of vertices. Each vertex is associated with a clause $v.\text{clause}$. We discuss only false QBFs, as true QBFs can be treated in a dual way.

We represent Herbrand functions in terms of *Right-First-And-Or (RFAO) formulas* [1]. An RFAO formula φ is recursively defined by $\varphi ::= \text{clause} \mid \text{cube} \mid \text{clause} \wedge \varphi \mid \text{cube} \vee \varphi$, which can be specified as an (ordered) sequence of nodes, $\text{node}_1, \text{node}_2, \dots, \text{node}_n$, where each node is either a clause or a cube.

3 (Counter)model Extraction from LQ-Resolution Proofs

In this section we present the algorithm *CountermodelExtractLQ* to extract a countermodel for a false QBF from its LQ-resolution proof in time linear with respect to the proof size. *CountermodelExtractLQ* extends the algorithm *CountermodelExtract* [1], which traverses a Q-resolution proof of a false QBF and builds the countermodel based on the universal reductions it encounters in the proof. Other than *CountermodelExtract*, *CountermodelExtractLQ* also deals with the tautological clauses in the proof, i.e., where a reduction of merged variables occurs.

Given a false QBF Φ , let G_Π be a DAG of its corresponding LQ-refutation. For any vertex $v \in G_\Pi$ and a literal $l \in v.\text{clause}$, procedure *PhaseFunctionBuild* (Figure 1) annotates the variable $x = \text{var}(l)$ in $v.\text{clause}$ with the *phase function* $v.\text{phase}(x)$, which intuitively represents the phase of l in $v.\text{clause}$ induced by an arbitrary assignment to existential variables of lower quantifier level. Note that, as

CountermodelExtractLQ

input: a false QBF Φ and its LQ-resolution DAG $G_\Pi(V_\Pi, E_\Pi)$

output: a countermodel in RFAO formulas

begin

```
01 PhaseFunctionBuild( $\Phi, G_\Pi(V_\Pi, E_\Pi)$ );
02 foreach universal variable  $x$  of  $\Phi$ 
03   RFAO_node_array[ $x$ ] :=  $\emptyset$ ;
04 foreach vertex  $v \in V_\Pi$  in topological order
05   if  $v.clause = \text{reduce}(u.clause)$ 
06      $v.shadcube := \overline{v.shadclause}$ ;
07     foreach universal variable  $x$  reduced from  $u.clause$  to get  $v.clause$ 
08       if  $x \in u.clause$ 
09         push  $v.shadclause$  to RFAO_node_array[ $x$ ];
10       else if  $\bar{x} \in u.clause$ 
11         push  $v.shadcube$  to RFAO_node_array[ $x$ ];
12       else if  $x^* \in u.clause$ 
13         push  $(v.shadclause \vee \overline{v.phase(x)})$  to RFAO_node_array[ $x$ ];
14         push  $(v.shadcube \wedge \overline{v.phase(x)})$  to RFAO_node_array[ $x$ ];
15   if  $v.clause$  is the empty clause
16     return RFAO_node_array;
end
```

Fig. 2. Algorithm: Countermodel Extraction from LQ-resolution Proof.

procedure *PhaseFunctionBuild* processes G_Π in a topological order, the phase function of each variable is well defined by the preceding phase functions. The procedure has both time and space complexities linear in the size of G_Π . No explicit substitution of the phase functions is needed in Line 14 as each phase function is represented by its defining variable.

Further, based on the definition of the phase function, the *effective literal*, denoted $v.elit(l)$, of $l \in v.clause$ is a literal that satisfies $v.elit(l) \leftrightarrow (x \leftrightarrow v.phase(x))$, where $x = var(l)$, and the *shadow clause*, denoted $v.shadclause$, is the clause $v.shadclause = \bigcup_{l \in v.clause} (v.elit(l))$. Intuitively, the effective literal $v.elit(l)$, represents the induced value of l in $v.clause$. Observe that Lines 3 and 4 of *PhaseFunctionBuild* imply that $v.elit(l) \leftrightarrow l$ whenever l is not a merged literal.

The algorithm *CountermodelExtractLQ* is outlined in Figure 2. It has a linear time complexity, since both *CountermodelExtract* and *PhaseFunctionBuild* have linear time complexities with respect to the proof size. We show the correctness of the algorithm and give an example for its application in the Appendix.

4 Experimental Results

We extended the tool RESQU [1] by *CountermodelExtractLQ* to RESQU-LPOLY and conducted experiments on the KBKF family of formulas [9] and the benchmark formulas of QBFEVAL'10 [12]. Q-resolution proofs were obtained with DEPQBF [11] and LQ-resolution proofs with DEPQBF-LQ [6]. We compared the (counter)model extraction by RESQU [1] from Q-resolution proofs, and by RESQU-LQU [1, 2] (ex-

ponential time complexity) and RESQU-LPOLY (polynomial time complexity) from LQ-resolution proofs. For both solving and extraction we set a time limit of 1,000s and a memory limit of 25GB. We validated the certificates with ABC [3] and compared the time and memory consumption and the certificate quality in terms of number of nodes in their And-Inverter-Graph (AIG).

For the KBKF family with $t \in [10..100]$ the proofs returned by DEPQBF were of exponential size, and those returned by DEPQBF-LQ of linear size in the parameter t , as was expected from [6]. Both RESQU and RESQU-LQU extracted countermodels from all the proofs for $t \leq 20$, but the cases with $t \geq 16$, resp. $t \geq 18$, could not be validated within the time limit. The time and memory consumption was between 0.1s, 7MB for $t = 10$ and 175.5s, 22GB for $t = 100$. RESQU-LQU required around half the time and a third of the memory of RESQU. RESQU-LPOLY extracted countermodels from all proofs of $t \in [10..100]$ in less than 0.4s, 6MB. All countermodels were validated within 0.1s. As for the certificate quality, RESQU and RESQU-LQ returned certificates between 3.1k nodes for $t = 10$ and 786.4k nodes for $t = 18$, which was clearly outperformed by RESQU-LPOLY with 279 nodes for $t = 10$ and 29.8k nodes for $t = 100$.

Out of the 569 formulas from QBFEVAL'10 [12], DEPQBF-LQ generated resolution proofs for 177 false QBFs and 98 true QBFs within the limits of 1,000s, 2GB RAM, and 1GB proof size. Among the 177 proofs of falsity, 144 were Q-resolution proofs, and 33 contained long-distance derivations; all the 98 proofs of truth were Q-resolution proofs. We focused on the 33 instances where LQ-resolution proofs were available. RESQU-LPOLY clearly outperformed RESQU and RESQU-LQU in terms of certificate quality. As for extraction time, RESQU-LPOLY clearly outperformed RESQU-LQ, but could not outperform RESQU on all instances. This different behavior can probably be attributed to the fact that RESQU processes Q-resolution proofs rather than LQ-resolution proofs. It is interesting to note that for three formulas DEPQBF was not able to produce Q-resolution proofs, but could obtain LQ-resolution proofs. Hence extracting certificates directly from LQ-resolution proofs can be beneficial for certain applications.

5 Conclusions

In this work we have shown that extracting a Herbrand-function countermodel (Skolem-function model) from any given LQ-resolution proof can be done in time linear with respect to the proof size. Apart from the theoretical advancement, we demonstrated the superiority of the new algorithm through experimental evaluation. Substantial savings on time and space resources were observed not only in crafted QBF instances, but also in application instances. More importantly, we have shown experimentally that the new procedure can yield certificates of high quality suitable for synthesis and other applications. The algorithm presented in this work can be further extended to extract (counter)models from QU-resolution proofs [14], and from LQU-resolution proofs [2].

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. V. Balabanov, M. Widl, and J.-H. R. Jiang. QBF resolution systems and their proof complexities. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, Lecture Notes in Computer Science. Springer, 2014. to appear.
3. Berkeley Logic Synthesis and Verification Group. ABC: A system for sequential synthesis and verification. <http://www.eecs.berkeley.edu/~alanmi/abc/>.
4. N. Dershowitz, Z. Hanna, and J. Katz. Bounded model checking with QBF. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 3569 of *LNCS*, pages 408–414. Springer, 2005.
5. N. Eén and N. Sörensson. An extensible SAT-solver. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.
6. U. Egly, F. Lonsing, and M. Widl. Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In *International Conference on Logic Programming and Automated Reasoning (LPAR)*, pages 291–308. Springer, 2013.
7. A. Goultiaeva, A. Van Gelder, and F. Bacchus. A uniform approach for generating proofs and strategies for both true and false QBF formulas. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 546–553. AAAI Press, 2011.
8. J.-H. R. Jiang, H.-P. Lin, and W.-L. Hung. Interpolating functions from large Boolean relations. In *International Conference on Computer-Aided Design (ICCAD)*, pages 779–784. IEEE/ACM, 2009.
9. H. Kleine Büning, M. Karpinski, and A. Flögel. Resolution for quantified Boolean formulas. *Information and Computation*, 117(1):12–18, Feb. 1995.
10. C.-F. Lai, J.-H. R. Jiang, and K.-H. Wang. BooM: A decision procedure for Boolean matching with abstraction and dynamic learning. In *Design Automation Conference (DAC)*, pages 499–504. ACM/IEEE, 2010.
11. F. Lonsing and A. Biere. DepQBF: A dependency-aware QBF solver (system description). *Journal on Satisfiability, Boolean Modeling and Computation*, 7:71–76, 2010.
12. MIND Laboratory (University of Genoa). QBF Solver Evaluation Portal. <http://www.qbflib.org/qbfeval/>.
13. S. Staber and R. Bloem. Fault localization and correction with QBF. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 355–368, 2007.
14. A. Van Gelder. Contributions to the theory of practical quantified Boolean formula solving. In *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 *International Conference on Computer-Aided Design (ICCAD)*, pages 442–449. ACM, 2002.

6 Appendix

We follow the notation used in [1]. In particular, We write $\Phi = \Phi_{\text{pfx}}\Phi_{\text{mtx}}$ for a QBF where Φ_{pfx} is the prefix and Φ_{mtx} is the matrix, $lvl(x)$ for the quantification level of a variable or literal x (the number of alternations between the quantifiers $\exists$ and $\forall$ from left to right), and $var(l)$ to refer to the variable of literal l . We denote the evaluation of a propositional formula ϕ under an assignment α by $\phi|_{\alpha}$ and consider α as a conjunction of literals.

We represent a semantic certificate for a false QBF by a Herbrand function $F[u](e_1, \dots, e_n)$ for each universal variable u such that $lvl(e_i) < lvl(u)$ for $i \in [1..n]$ and such that replacing all universal variables by their Herbrand functions renders the matrix unsatisfiable. The dual definition applies to a true QBF with Skolem functions for existential variables. We call a set of Herbrand (Skolem) functions a *countermodel* (*model*) for a false (true) QBF.

6.1 Correctness proof

To establish the correctness of the algorithm *CountermodelExtractLQ*, we need to recall the following two properties of the RFAO arrays [1]:

1. If $node_i$ under some (partial) assignment of variables becomes a validated clause (i.e. a clause that evaluates to 1) or a falsified cube (i.e. a cube that evaluates to 0), then we can remove $node_i$ (unless it is the last node) from the formula.
2. If $node_i$ becomes a falsified clause (i.e. a clause that evaluates to 0) or validated cube (i.e. a cube that evaluates to 1), then the value of the formula is determined by $node_i$, and we can remove any other node with index greater than i .

Theorem 1 below proves the correctness of the algorithm *CountermodelExtractLQ*.

Theorem 1. *Given a false QBF Φ and its LQ-resolution proof Π , the algorithm *CountermodelExtractLQ*(Φ, G_{Π}) produces a countermodel of Herbrand functions for the universal variables of Φ .*

To prove Theorem 1, we need to show that 1) the Herbrand functions returned by *CountermodelExtractLQ* obey the prefix order dependency (i.e., the Herbrand function $F[x]$ of universal variable x only refers to the variables with quantification levels less than that of x), and 2) their substitution for corresponding universal variables indeed makes the matrix Φ_{mtx} unsatisfiable. Proposition 1 establishes the first part.

Proposition 1. *Given a false QBF Φ and its LQ-resolution proof Π , let literal $l \in v.\text{clause}$ with $v \in V_{\Pi}$. If the truth or falsity of $v.\text{elit}(l)$ refers (through recursive substitution of phase functions) to some variable x , then $lvl(x) \leq lvl(l)$.*

Proof. Observe that the proposition holds, by the definition of effective literals, for any literal l that is not a merged literal. Since the clauses in Φ_{mtx} do not involve any merged literals, the proposition holds for all the literals in the clauses of Φ_{mtx} . On the other hand, for a merged literal $l \in v.\text{clause}$, if $v.\text{clause} = \text{reduce}(u.\text{clause})$ in Π , then $v.\text{elit}(l) = u.\text{elit}(l)$ according to Line 7 of the procedure *PhaseFunctionBuild*.

Moreover, if $v.\text{clause} = \text{resolve}(u_1.\text{clause}, u_2.\text{clause})$ and $\text{var}(l) \notin \text{vars}(u_1.\text{clause})$ (resp. $\text{var}(l) \notin \text{vars}(u_2.\text{clause})$), then $v.\text{elit}(l) = u_2.\text{elit}(l)$ (resp. $v.\text{elit}(l) = u_1.\text{elit}(l)$) according to Line 10 (resp. Line 12) of the procedure *PhaseFunctionBuild*. On the other hand, if $v.\text{clause} = \text{resolve}(u_1.\text{clause}, u_2.\text{clause})$ under pivot variable p and $\text{var}(l) \in \text{vars}(u_1.\text{clause})$ and $\text{var}(l) \in \text{vars}(u_2.\text{clause})$, then this is an LQ-resolution step (if it is not, then $l \in v.\text{clause}$ cannot be a merged literal). By the rule of LQ-resolution, $\text{lvl}(p) < \text{lvl}(l)$. Therefore, regardless of the origin of v from either universal reduction or resolution, if the proposition holds for any literal in every parent of $v \in V_\Pi$, then it must also hold for the literals in $v.\text{clause}$. By induction, the proposition holds for any $l \in v.\text{clause}$ for any $v \in V_\Pi$. ■

Similarly to Proposition 1, we can prove that for any $x \in \text{vars}(v.\text{clause})$, function $v.\text{phase}(x)$ only refers to the variables with quantification level less than $\text{lvl}(x)$. Taking into account the construction of procedure *CountermodelExtractLQ*, the following corollary follows.

Corollary 1. *Herbrand functions returned by the algorithm CountermodelExtractLQ obey the prefix order dependency.*

Proof. Given a universal reduction step $v.\text{clause} = \text{reduce}(u.\text{clause})$, and a literal $l \in u.\text{clause}$ such that $l \notin v.\text{clause}$ (i.e. l is a reduced literal), it holds that $\text{lvl}(l) > \text{lvl}(l')$ for any $l' \in v.\text{clause}$, by definition of universal reduction. Therefore, by Proposition 1, the truth or falsity of $v.\text{shadclause}$ only refers to the variables with quantification level less than $\text{lvl}(l)$. Similarly function $v.\text{phase}(x)$ only refers to the variables with quantification level less than $\text{lvl}(l)$, where $x = \text{var}(l)$. Hence for each universal variable x , its corresponding RFAO node array only refers to the variables with quantification level less than $\text{lvl}(x)$. ■

To prove that Herbrand functions returned by *CountermodelExtractLQ* form a countermodel, we follow a similar line of reasoning as in [1]. The algorithm *CountermodelExtractLQ* stores shadow clauses (cubes) in RFAO arrays rather than regular clauses (cubes) and it considers universal reduction on merged literals. Notice that, if no LQ-resolution step is present in a proof (i.e. no merged literal appears), then *CountermodelExtractLQ* returns exactly the same Herbrand functions as *Countermodel_construct*.

Regardless of the change from regular to shadow clauses (cubes), the two RFAO formula properties listed in Section 2 remain intact. In the following, when we say that some shadow clause of vertex v is α -implied, we mean it is α -implied by the shadow clause (the conjunction of the shadow clauses) corresponding to the parent vertex (parent vertices) of v . Lemma 1 shows the properties of α -implication among shadow clauses.

Lemma 1. *Given a false QBF Φ and its LQ-resolution proof Π , let $v \in V_\Pi$ and $v.\text{clause} = \text{resolve}(u_1.\text{clause}, u_2.\text{clause})$ in Π with pivot literals $p \in u_1.\text{clause}$ and $\bar{p} \in u_2.\text{clause}$. Then $(u_1.\text{shadclause}|_\alpha \wedge u_2.\text{shadclause}|_\alpha) \rightarrow v.\text{shadclause}|_\alpha$ under any assignment α to the variables in Φ .*

Proof. Let $\text{vars}(u_1.\text{clause}) = \{p\} \cup L_1 \cup M$ and $\text{vars}(u_2.\text{clause}) = \{\bar{p}\} \cup L_2 \cup M$, where L_1 and L_2 are the sets of variables local to $u_1.\text{clause}$ and $u_2.\text{clause}$, respectively, and

M is the set of their *common* variables, excluding the pivot variable p . If $v.elit(l)|_\alpha = 1$ for some $l \in v.clause$, then by the definition of shadow clause $v.shadclause|_\alpha = 1$. Therefore $(u_1.shadclause|_\alpha \wedge u_2.shadclause|_\alpha) \rightarrow v.shadclause|_\alpha$.

Consider the other case that $v.elit(l)|_\alpha = 0$ for each $l \in v.clause$. Without loss of generality, assuming $\bar{p} \in \alpha$, we prove $u_1.shadclause|_\alpha = 0$ in the following. (Assuming $p \in \alpha$, $u_2.shadclause|_\alpha = 0$ can be proved similarly).

By the definition of effective literal, $v.elit(l_1) = u_1.elit(l_1)$ holds for each l_1 with $var(l_1) \in L_1$. Hence if $v.elit(l_1)|_\alpha = 0$, then $u_1.elit(l_1)|_\alpha = 0$ for each l_1 with $var(l_1) \in L_1$. Further, for each variable x with $x \in M$, we have $v.phase(x)|_\alpha = ((\bar{p} \wedge u_1.phase(x)) \vee (p \wedge u_2.phase(x)))|_\alpha = u_1.phase(x)|_\alpha$. Therefore

$$v.elit(l)|_\alpha = (x \leftrightarrow v.phase(x))|_\alpha = (x \leftrightarrow u_1.phase(x))|_\alpha = u_1.elit(l')|_\alpha = 0,$$

where l and l' stand for the literals of variable x in $v.clause$ and $u_1.clause$, respectively. Consequently, $u_1.elit(l)|_\alpha = 0$ for each l with $var(l) \in \{p\} \cup L_1 \cup M$, and thus $u_1.shadclause|_\alpha = 0$.

Thereby $(u_1.shadclause|_\alpha \wedge u_2.shadclause|_\alpha) \rightarrow v.shadclause|_\alpha$ under any assignment α , and the lemma follows. $\blacksquare$

The following lemma shows that the substitution of all universal variables by their corresponding Herbrand functions, returned by *CountermodelExtractLQ*(Φ, G_Π), indeed makes Φ_{mtx} unsatisfiable, and thus completes the proof of Theorem 1.

Lemma 2. *Given a false QBF Φ and its LQ-resolution proof Π , the algorithm *CountermodelExtractLQ*(Φ, G_Π) returns Herbrand functions whose substitution for the corresponding universal variables makes the matrix Φ_{mtx} unsatisfiable.*

Proof. Given an assignment $\alpha_\exists$ to the existential variables of Φ , we show below that the constructed Herbrand functions induce an assignment $\alpha_\forall$ to the universal variables of Φ such that $\Phi_{\text{mtx}}|_\alpha = 0$ for $\alpha = \alpha_\exists \cup \alpha_\forall$.

Let V_D be the set of all vertices $v \in V_\Pi$ whose clauses were obtained by universal reduction in Π (i.e., $v.clause = \text{reduce}(u.clause)$ for some $u \in V_\Pi$). Notice that algorithm *CountermodelExtractLQ* processes G_Π in a topological order, meaning that a clause in Π is processed only after all of its ancestor clauses are processed. Therefore we consider all shadow clauses $v.shadclause$ with $v \in V_D$ in the topological order under the assignment α . First, assume that for each $v \in V_D$ its corresponding shadow clause $v.shadclause$ satisfies $v.shadclause|_\alpha = 1$, and therefore is α -implied. By Lemma 1, we conclude that in this case $v.shadclause$ is α -implied for any $v \in V_\Pi$. Hence the empty shadow clause (that corresponds to the empty clause) is α -inherited, and thus $\Phi_{\text{mtx}}|_\alpha = 0$.

Second, assume that for some vertex $v \in V_D$, the corresponding shadow clause $v.shadclause|_\alpha = 0$. Let $v'.shadclause$ be the first such encountered shadow clause. Denote u' as the parent of v' . Note that $u'.shadclause$ and all its ancestors must be α -inherited (as all the ancestors of $u'.shadclause$ are α -implied). Let $C_{u' \setminus v'}$ be the set of all the universal literals being reduced from $u'.clause$ to get $v'.clause$. By the definition of shadow clause, we have $u'.shadclause = v'.shadclause \bigcup_{l \in C_{u' \setminus v'}} u'.elit(l)$. Therefore we have

$$u'.shadclause|_\alpha = v'.shadclause|_\alpha \vee \bigvee_{l \in C_{u' \setminus v'}} u'.elit(l)|_\alpha = \bigvee_{l \in C_{u' \setminus v'}} u'.elit(l)|_\alpha.$$

Next we show that our construction yields $u'.elit(l)|_\alpha = 0$ for any $l \in C_{u' \setminus v'}$, therefore leading to $u'.shadclause|_\alpha = 0$. For each variable $x = var(l)$ with $l \in C_{u' \setminus v'}$, we examine its corresponding `RFAO_node_array[x]`. Since $w.shadclause|_\alpha = 1$ for any ancestor $w \in V_D$ of v' , the value of Herbrand function $F[x]$ under α is not determined by any of the RFAO nodes that were added to the RFAO array before the reduction of x happens in u' (by Property 1 of RFAO array mentioned in Section 2). We now analyze the following three cases for the reduction of x in $u'.clause$:

1. For x being reduced as a positive literal l , the corresponding RFAO clause node evaluates to 0 (since $v'.shadclause|_\alpha = 0$), and hence $F[x]|_\alpha = 0$ (by Property 2 of RFAO arrays mentioned in Section 2).
2. Similarly, for x being reduced as a negative literal l , the corresponding RFAO cube node evaluates to 1 (since $(v'.shadcube)|_\alpha = 1$), and hence $F[x]|_\alpha = 1$.
3. For x being reduced as a merged literal l , by algorithm *CountermodelExtractLQ* two RFAO nodes, namely, clause node $Node_1 = (v'.shadclause \vee v'.phase(x))$ and cube node $Node_2 = (v'.shadcube \wedge v'.phase(x))$, are added to the RFAO array. On the one hand, if $u'.phase(x)|_\alpha = v'.phase(x)|_\alpha = 0$, then $Node_1|_\alpha = Node_2|_\alpha = 1$. Therefore $F[x]|_\alpha = 1$ (determined by the RFAO cube $Node_2|_\alpha = 1$) and $u'.elit(l)|_\alpha = ((F[x] \wedge u'.phase(x)) \vee (F[x] \wedge u'.phase(x)))|_\alpha = 0$. On the other hand, if $u'.phase(l)|_\alpha = v'.phase(l)|_\alpha = 1$, then $Node_1|_\alpha = Node_2|_\alpha = 0$. Therefore $F[x]|_\alpha = 0$ (determined by the RFAO clause $Node_1|_\alpha = 0$) and $u'.elit(l)|_\alpha = 0$ similarly.

By the above analysis, it holds that $u'.elit(l)|_\alpha = 0$ for any $l \in C_{u' \setminus v'}$. Therefore $u'.shadclause|_\alpha = 0$, and taking into account that $u'.shadclause$ is α -inherited we establish $\Phi_{mtx}|_\alpha = 0$. ■

6.2 Example

Consider the false QBF Φ with its prefix and matrix defined as follows.

$$\begin{aligned}\Phi_{\text{pfx}} &= \exists ab \forall x \exists cd \forall y \exists e \\ \Phi_{\text{mtx}} &= (a, x, d)_1 (\bar{a}, b, \bar{x}, d)_2 (\bar{b}, x, c, y, e)_3 (\bar{c}, d, \bar{y}, e)_4 (d, \bar{e})_5 (\bar{d})_6\end{aligned}$$

Its falsity can be established by the LQ-resolution proof Π visualized in Figure ??.

By procedure *PhaseFunctionBuild*, the phase functions f_i and g_i corresponding to the literals of variables x and y , respectively, present in the clause C_i can be derived as shown in Figure ??. Note that, if $lit(x) \notin C_i$ ($lit(y) \notin C_i$), then $f_i = \emptyset$ ($g_i = \emptyset$).

Figure ?? shows the RFAO node-array contents for variables x and y after each $\vee$ -reduction step in the proof of Figure ?? after applying *CountermodelExtractLQ* algorithm in Figure 2. Note that $f_{10}^e = (x \wedge f_{10}) \vee (\bar{x} \wedge f_{10})$ stands for the effective literal of x^* in C_{10} . After re-expressing the RFAO formulas in terms of the existential variables, we get the following Herbrand functions for universal variables x and y .

$$\begin{aligned}F[x] &= (\overline{f_{12}}) \wedge (\overline{f_{12}}) & F[y] &= (f_{10}^e + d + \overline{g_{10}}) \wedge (\overline{f_{10}^e} \bar{d} \overline{g_{10}}) \\ &= (\overline{b}f_7 + bf_8) & &= (\overline{f_{10}^e} \bar{d} \overline{g_{10}}) = (xf_{10}) + (\bar{x}f_{10})\bar{d}\bar{g}_8 \\ &= (\overline{b}(\bar{a}f_1 + af_2) + b) & &= (x(\bar{b}\bar{a} + b)) + (\bar{x}(\bar{b}\bar{a} + b))\bar{d}\bar{c} \\ &= a\bar{b} & &= \bar{d}c\end{aligned}$$

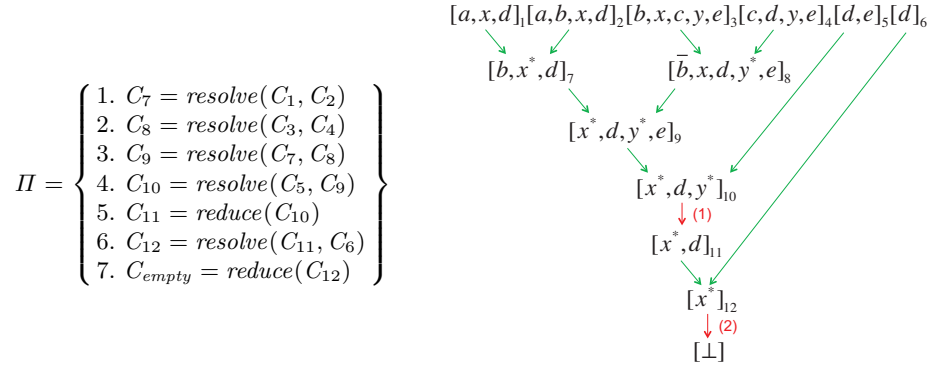


Fig. 3. LQ-resolution proof Π and its corresponding DAG.

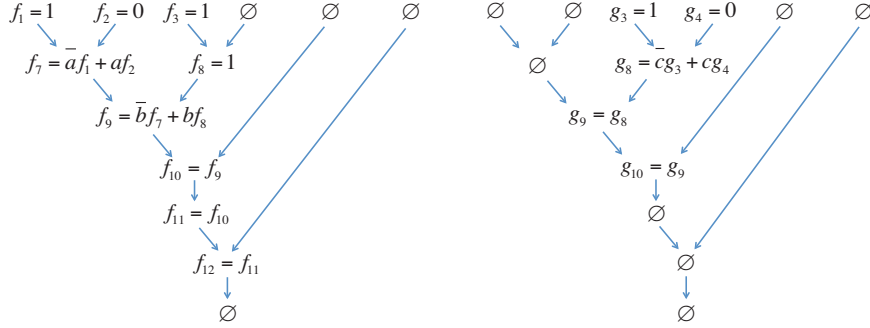


Fig. 4. Phase functions f and g for literals of variables x and y .

It can be verified that, for any assignment to the existential variables a, b, c, d , and e , formula Φ_{mtx} with variables x and y substituted with $F[x]$ and $F[y]$, respectively, is false.

6.3 Detailed experimental results

We applied the QBF solver DEPQBF [11] without and with long-distance resolution (command line parameter “`--long-dist-res`” [6]) to produce Q- and LQ-resolution proofs, respectively. Then we compared the model and countermodel extraction in three settings: RESQU [1] for Q-resolution proofs, RESQU-LQU [1, 2] for LQ-resolution proofs (without a polynomial bound on time complexity), and RESQU-LPOLY for LQ-resolution proofs (with polynomial time complexity). Under the three settings, before the extraction of models (resp. countermodels) the proofs generated by DEPQBF were reduced to their cores by removing those resolution steps irrelevant to the empty cube (resp. clause) derivation. Further, to validate the constructed models and countermodels, the SAT solver MINISAT [5] embedded in ABC [3] was applied.

Table ?? shows the runtime and memory statistics for QBF solving, countermodel extraction, and countermodel validation for the KBKF family of QBF in-

0.	$x : []$	$y : []$
1.	$x : []$	$y : \left[\begin{array}{l} \text{clause}(f_{10}^e, d, \overline{g_{10}}) \\ \text{cube}(f_{10}^e, \overline{d}, \overline{g_{10}}) \end{array} \right]$
2.	$x : \left[\begin{array}{l} \text{clause}(f_{12}) \\ \text{cube}(f_{12}) \end{array} \right]$	$y : \left[\begin{array}{l} \text{clause}(f_{10}^e, d, \overline{g_{10}}) \\ \text{cube}(f_{10}^e, \overline{d}, \overline{g_{10}}) \end{array} \right]$

Fig. 5. RFAO node arrays after processing universal reduction steps in topological order.

Table 1. Comparison on time and memory usage for KBKF instances.

t	DEPQBF		RESQU			DEPQBF-LQ		RESQU-LQU			RESQU-LPOLY		
	T-slv (s)	$ II $	T-ext (s)	Mem (MB)	T-vld (s)	T-slv (s)	$ II $	T-ext (s)	Mem (MB)	T-vld (s)	T-ext (s)	Mem (MB)	T-vld (s)
10	0.1	11.8k	0.1	7	0.1	0.0	155	0	4	0.1	0.0	1	0.1
11	0.2	25.6k	0.1	14	0.3	0.0	171	0.1	8	0.1	0.0	1	0.1
12	0.5	55.3k	0.3	58	0.7	0.0	187	0.1	18	0.1	0.0	1	0.1
13	1.2	118.8k	0.6	123	2.3	0.0	203	0.3	37	0.1	0.0	1	0.1
14	2.8	254.0k	1.4	261	7.6	0.0	219	0.7	79	0.1	0.0	1	0.1
15	6.8	540.7k	3.0	550	30.5	0.0	235	1.8	169	0.1	0.0	1	0.1
16	16.6	1.15M	6.7	1.2k	-1	0.0	251	3.9	360	0.8	0.0	1	0.1
17	41.0	2.42M	15.1	2.4k	-1	0.0	267	9.4	767	5.4	0.0	1	0.1
18	102.8	5.11M	33.6	5.1k	-1	0.0	283	20.5	1.6k	40.4	0.0	1	0.1
19	261.5	10.75M	74.1	10.7k	-1	0.0	299	48.8	3.4k	-1	0.0	1	0.1
20	674.2	22.54M	175.7	22.5k	-1	0.0	315	95.1	7.2k	-1	0.0	1	0.1
30	-1	-	-	-	-	0.0	155	-1	-	-	0.0	1	0.1
40	-1	-	-	-	-	0.0	171	-1	-	-	0.1	2	0.1
50	-1	-	-	-	-	0.0	187	-1	-	-	0.1	2	0.1
60	-1	-	-	-	-	0.0	203	-1	-	-	0.1	3	0.1
70	-1	-	-	-	-	0.0	219	-1	-	-	0.2	5	0.1
80	-1	-	-	-	-	0.0	235	-1	-	-	0.3	5	0.1
90	-1	-	-	-	-	0.0	251	-1	-	-	0.3	5	0.1
100	-1	-	-	-	-	0.0	267	-1	-	-	0.4	6	0.1

stances, which are all false and hard for Q-resolution, but easy for LQ-resolution, to refute [6, 9]. Column 1 lists the instances indexed by parameter t , which reflects the formula size (there are $3t + 2$ variables and $4t + 1$ clauses in the t^{th} member of the KBKF family). Columns denoted by RESQU, RESQU-LQU and RESQU-LPOLY stand for the three settings of countermodel extraction; Columns denoted by DEPQBF and DEPQBF-LQ indicate QBF solver settings without and with long distance resolution, respectively. The columns “T-slv,” “T-ext,” and “T-vld” report the CPU runtime in seconds for QBF solving, certificate extraction, and certificate validation, respectively; the columns “ $|II|$ ” refer to the proof size (in terms of the number of resolution steps); the columns “Mem” refer to the peak memory consumption in MB. An entry containing “-1” indicates that the computation was either out of the time limit of 1,000 seconds, or out of the memory limit of 25 GB. An entry containing “-” indicates that the data is not available.

As evident from Table ??, DEPQBF required time and yielded proof size exponential in t , whereas DEPQBF-LQ required time and yielded proof size linear in t . RESQU was able to extract countermodels from all the proofs produced by DEPQBF for $t \leq 20$, but the cases with $t \geq 16$ could not be validated within the time

Table 2. Comparison on certificate quality for KBKF instances.

t	RESQU				RESQU-LQU				RESQU-POLY			
	raw		opt		raw		opt		raw		opt	
	#AIG	#LVL	#AIG	#LVL	#AIG	#LVL	#AIG	#LVL	#AIG	#LVL	#AIG	#LVL
10	4.6k	2.0k	3.1k	1.6k	3.1k	267	93	10	279	28	10	2
11	9.2k	4.1k	6.1k	3.1k	6.1k	524	231	13	340	31	11	2
12	17.9k	8.2k	11.9k	6.1k	12.3k	1.0k	532	16	407	34	12	2
13	34.7k	16.4k	24.6k	12.3k	24.6k	2.1k	840	17	480	37	13	2
14	68.0k	32.8k	47.5k	24.5k	49.2k	4.1k	1.5k	19	559	40	14	2
15	134.1k	65.5k	95.1k	49.1k	98.3k	8.2k	2.7k	20	644	43	15	2
16	265.6k	131.1k	253.9k	82.0k	196.6k	16.4k	16.4k	29	735	46	16	2
17	-	-	-	-	393.2k	32.8k	61.6k	32	832	49	17	2
18	-	-	-	-	786.4k	65.5k	132.8k	33	935	52	18	2
19	-	-	-	-	-	-	-	-	1.0k	55	19	2
20	-	-	-	-	-	-	-	-	1.1k	58	20	2
30	-	-	-	-	-	-	-	-	2.6k	88	30	2
40	-	-	-	-	-	-	-	-	4.7k	118	40	2
50	-	-	-	-	-	-	-	-	7.4k	148	50	2
60	-	-	-	-	-	-	-	-	10.7k	178	60	2
70	-	-	-	-	-	-	-	-	14.6k	208	70	2
80	-	-	-	-	-	-	-	-	19.0k	238	80	2
90	-	-	-	-	-	-	-	-	24.1k	268	90	2
100	-	-	-	-	-	-	-	-	29.8k	298	100	2

limit. On the other hand, RESQU-LQU was only able to extract countermodels for $t \leq 20$ from the proofs produced by DEPQBF-LQ within the time limit, while the cases with $t \geq 18$ could not be validated within the time limit. RESQU-LQU used much less memory than RESQU because of the reduced proof size of LQ-resolution. In comparison, RESQU-LPOLY easily accomplished every extraction task within 0.4 seconds under 6 MB memory consumption; its produced countermodels were all validated within 0.1 seconds.

Continuing the above experiments, Table ?? shows the circuit sizes in terms of the number of and-inverter graph (AIG) nodes, denoted “#AIG”, and circuit depths, denoted “#LVL”, to evaluate the quality of the extracted Herbrand functions. The columns denoted with “raw” and “opt” refer to the AIGs before and after light-weight logic optimization by ABC [3] (under script `balance`, `dc2`), respectively. An entry containing “-” indicates either that the data is unavailable, or that ABC failed to read in the certificate due to its excessive size. It is worth noting that the simplified AIGs of the Herbrand functions of KBKF instance t produced by RESQU-LPOLY have $t + 1$ AIG nodes and are significantly smaller than the corresponding functions produced by other methods.

Figure ?? plots the certificate extraction time and validation time statistics for the different methods under comparison, for QBFEVAL’10 benchmarks. The x axis in the figure corresponds to the results obtained by RESQU-LPOLY, and the y axis corresponds to those obtained by RESQU and RESQU-LQU. Both axes are presented in the $\log_{10}$ scale. If a method failed to extract a certificate, then the corresponding result was set to the upper bound of the figure. As can be seen, RESQU-LPOLY constantly outperforms RESQU-LQU in terms of certificate extraction and validation time. On the other hand, since RESQU works on different proofs (produced by DEPQBF rather than DEPQBF-LQ), the results are not directly comparable

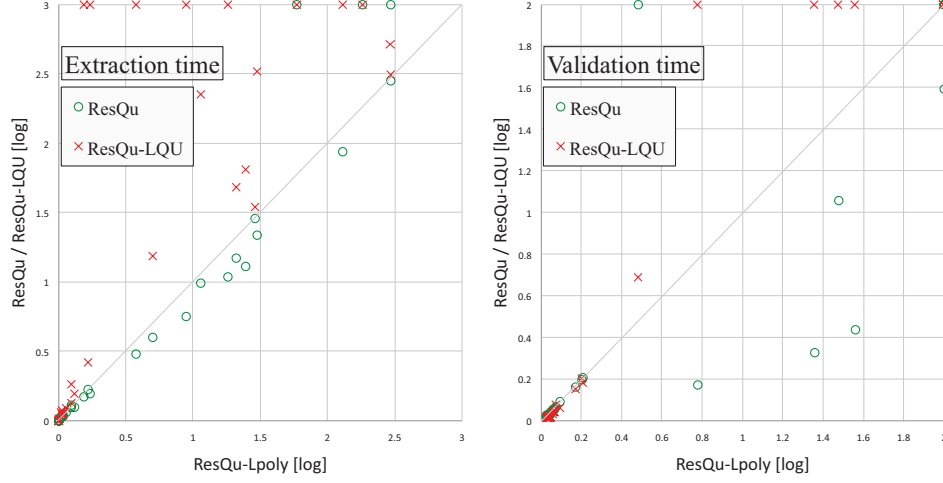


Fig. 6. Comparison on runtime for application benchmarks.

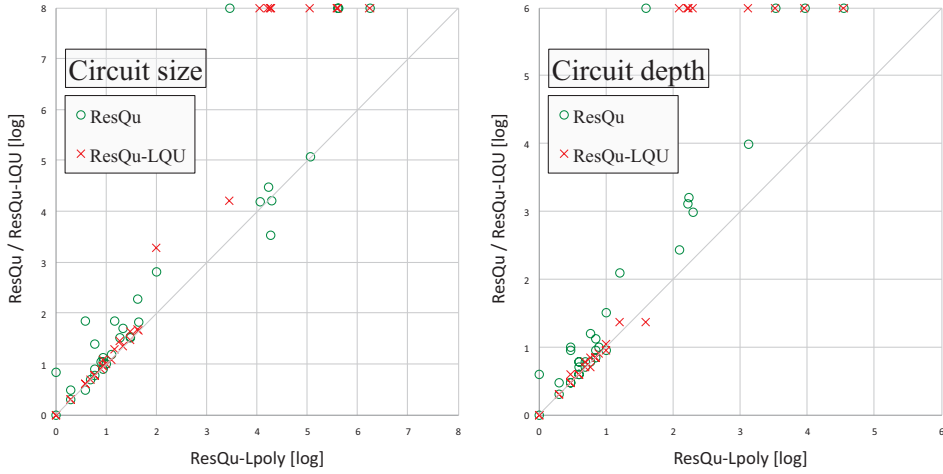


Fig. 7. Comparison on certificate quality for application benchmarks.

with those of RESQU-LPOLY. Also note that DEPQBF was not able to produce Q-resolution refutations for three formulas whose LQ-resolution proofs were obtainable. Hence, extracting certificates from LQ-resolution proofs is beneficial for certain applications.

To compare the quality of the extracted certificates, Figure ?? plots the results in terms of AIG size and circuit depth. As shown in Figure ??, certificates produced by RESQU-LPOLY are consistently smaller than those produced by RESQU-LQU. Furthermore RESQU-LQU was not able to produce certificates for eight instances, whereas RESQU-LPOLY handled them without difficulty. The certificates extracted by RESQU are in 90% of the cases larger in terms of both AIG size and circuit depth than those produced by RESQU-LPOLY, although the improvement is not as significant as that for RESQU-LQU.