International Workshop on Quantified Boolean Formulas 2013[1]

Informal Workshop Report[2]

Florian Lonsing, Martina Seidl

florian.lonsing@tuwien.ac.at, martina.seidl@jku.at

July 10, 2013

[1]This work was partially funded by the Vienna Science and Technology Fund (WWTF) under grant ICT10-018 and by the Austrian Science Fund under grants S11408-N23 and S11409-N23.

Preface

After a break of more than one decade, the International Workshop on Quantified Boolean Formulas 2013 (QBF 2013), collocated with the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT 2013), is the first event dedicated to research on quantified Boolean formulas (QBF).

Recently, there has been a lot of progress in QBF research on a variety of topics ranging from solving, to preprocessing, to certifications, and to encodings. All of these results were presented at many different scientific events. The purpose of the QBF workshop is to provide a dedicated forum for QBF researchers and users to meet, share ideas, and discuss the status quo. Almost 40 registered participants clearly state the general interest in an event like the QBF Workshop 2013.

During the workshop seven presentations will be given based on the peer-reviewed extended abstracts collected in this informal report, which is available from the workshop's website:

http://fmv.jku.at/qbf2013/

Due to the informal nature of the QBF Workshop, the authors of accepted papers are free to submit their work to other scientific events or journals without any restrictions.

The topics of the contributions are manifold, ranging from theoretical results to practical encodings. At this point we would like to highlight that all contributions contain novel content, to a certain extent, which has not been presented in this form before. Moreover, certain contributions share insights which are entirely new and have never been published before. This is particularly remarkable because in the call for contributions we put emphasis on works which trigger interesting discussions during the workshop. Novelty was not a strict requirement. Therefore, we would like to thank the contributors for their papers as well as the program committee members and reviewers for their support and their high quality feedback.

As preparation for the QBF Workshop, we organized the QBF Gallery 2013 which is a non-competitive evaluation of state-of-the-art tools. Like the QBF Workshop, the QBF Gallery is affiliated to the SAT conference 2013. Details are available at the event's website:

http://www.kr.tuwien.ac.at/events/qbfgallery2013/

Results will be presented during the QBF Workshop and at the SAT conference. We hope that with the seven accepted presentations and with the report on the QBF Gallery, the QBF Workshop will offer an interesting meeting for the QBF research community.

Florian Lonsing and Martina Seidl, Wien/Linz, July 2013

Program Committee

- Fahiem Bacchus, University of Toronto, Canada
- Armin Biere, University of Linz, Austria
- Nikolaj Björner, Microsoft Research, USA
- Uwe Bubeck, University of Paderborn, Germany
- Hans Kleine Büning, University of Paderborn, Germany
- Hubie Chen, Universidad del País Vasco and Ikerbasque, Spain
- Nadia Creignou, Université de la Méditerranée Aix-Marseille, France
- Uwe Egly, Vienna University of Technology, Austria
- Allen Van Gelder, University of California at Santa Cruz, USA
- Enrico Giunchiglia, Università di Genova, Italy
- Mikoláš Janota, INESC-ID Lisboa, Portugal
- Massimo Narizzano, Università di Genova, Italy
- Stefan Szeider, Vienna University of Technology, Austria

Contents

Conformant Planning as a Benchmark for QBF-Solvers*

Martin Kronegger, Andreas Pfandler, and Reinhard Pichler
Vienna University of Technology, Austria
{kronegger, pfandler, pichler}@dbai.tuwien.ac.at

1 Introduction

Planning has a long history as an important field of artificial intelligence. The task of domain-independent planning is to find a sequence of actions (called a plan) to achieve a desired goal, given some initial state.

Concerning the computational complexity, even the most basic case, i. e., STRIPS-like planning (also called classical planning), is NP-complete when restricted to plans with length polynomial in the input. To cope with this high complexity, powerful heuristics such as Fast Forward [9] and Fast Downward [7] have been developed. In addition, also exact methods have been proposed. For instance, a planner called SATPLAN [11] reduces a classical planning problem to SAT. Another promising SAT-based approach is the planner Mp by Rintanen [18].

In practice one often has to deal with uncertainty, i. e., incomplete knowledge. Thus, it is of high importance to study this form of planning (also called conformant planning). Here we consider the case of deterministic actions with incomplete knowledge. Baral et al. [2] have shown that the complexity raises to Σ_2P-completeness for this form of planning. Following the successful history of SAT in the area of classical planning it is a natural step to encode conformant planning problems as quantified Boolean formulas (QBFs). Several encodings have been proposed in the literature, such as the encodings by Rintanen [17].

In this paper we present a framework for transforming conformant planning instances to QBFs in order to analyze how well QBF-solvers can handle planning problems. Since the planning instances are given in a non-ground modelling language several refinements are necessary to obtain QBFs of reasonable size. A key feature of our approach is that we aim at *optimal* plans, i. e., plans of minimum length. The goal of this work is to gather some experience with this approach and highlight desiderata that could improve the performance and applicability of QBF-solvers in the area of planning.

Our *main contributions* are as follows: (1) We have implemented a prototype of our framework that encodes planning instances given in a fragment of the widely used planning domain definition language (PDDL) with uncertainty in the initial state as QBFs. (2) We introduce a new kind of benchmarks that has a strong focus on uncertainty and hence gives challenging new instances. (3) Based on these instances we perform a detailed experimental evaluation – comparing specialized planning tools with the performance of several QBF-solvers when applied to QBFs generated by our framework.

* This research was supported by the Austrian Science Fund (FWF): P25518-N23.

2 Solver

We consider planning instances given in PDDL restricted to the STRIPS-subset and extended by uncertainty in the initial state. Our framework (depicted in Figure 1) is capable of computing the minimal plan length of planning instances. After parsing the input, the grounding algorithm analyzes the given planning instance and calculates a lower bound ℓ on the plan length. Starting with a plan length of ℓ, the grounder then grounds only "important" parts of the instances. The QBF-encoder takes the ground representation as input, transforms it to a quantified Boolean formula (QBF) that is satisfiable if and only if the planning problem has a plan of length ℓ. Then, the QBF-encoder invokes an external QBF-solver. In case the generated QBF is satisfiable, our framework extracts the plan (with optimal length) from the assignment of the leftmost $\exists$-block. If the QBF is unsatisfiable, ℓ is incremented, additional "important" parts of the problem may need grounding, and the subsequent QBF is passed to the solver. The grounding step is the bottleneck of the approach, since the planning instances are given in a non-ground modelling language and a naive implementation can cause an exponential blow-up in the number of variables. Clearly, a smaller grounding directly yields smaller QBFs. Therefore, we use information from the specification to guide the grounding process and apply techniques for computing lower bounds to avoid the generation of unnecessary QBFs.

The QBF-encoder essentially employs the $\exists\forall\exists$-encoding for conformant planning by Rintanen [17] using explanatory framing axioms. The resulting QBF expresses "there exists a sequence of actions from the initial state to a goal state such that for all unknown variables in the initial state there exist a truth assignment for the remaining variables". During the generation of the formula, the encoder also applies mild forms of preprocessing such as unit-propagation and (in some cases) labeling of sub-formulas. By using these simple enhancements and the refinements for grounding, we are able to drastically reduce the size of the generated QBFs by a factor of approximately 17 in the new type of benchmark which we introduce in Section 3.

3 Evaluation

We have evaluated a prototype of our framework. Below, we show how the runtime behavior depends on the used QBF-solver and compare it with other planning tools (T0 [14, 20], ConformantFF [8, 4]). To allow for a better comparison, we have created a new type of benchmark that focuses on uncertainty in the initial state. Due to space limitations, we only report on this new benchmark type.

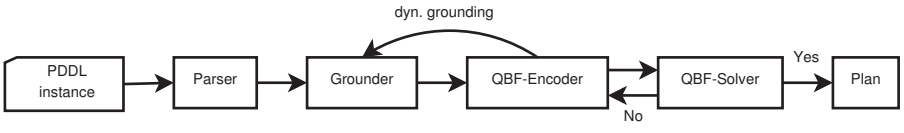


Fig. 1. Architecture of our solver.

The new benchmark we introduce is called "Dungeon". In this setting a player wants to defeat monsters living in a dungeon. Each monster requires different items to be defeated. In the beginning, the player picks at most one item from each pool of items. In addition, the player can exchange several items for one more powerful item if he holds all necessary "ingredients". Eventually, the player enters the dungeon. When entering the dungeon, the player is forced to pick additional items. The dilemma is that the player does not know which items he will get, i. e., the additional items are *unknown*. The goal is to pick items such that irrespective of the additional items he defeats at least one monster.

We have run our tests on a server with two Intel Xeon E5345 processors, 48GB RAM, and openSUSE 11.4. We ran the single threaded solvers with a 15 minutes timeout and a memory limit of 16GB. The upper bound for the plan length of our solver was set to 200. We used AQME [15, 1], DepQBF [12, 5], QuBE[6, 16] and sKizzo [3, 19] in conjunction with our solver. For a better comparison we also evaluated ConformantFF and T0 on the same instances. If a QBF-solver supported the generation of certificates or returning an assignment of the leftmost $\exists$-block (currently, from the considered solvers only sKizzo and DepQBF), we enabled this option and included the additional amount of time needed in the runtime measurement.

Our results indicate that randomly generated "Dungeon" instances are hard for both, specialized planning tools and QBF-solvers. Considering only the number of solved instances, T0 is the winner with 102 out of 144 solved instances. The other solvers were able to solve the following amount of instances: 96 (AQME), 81 (QuBE), 80 (DepQBF), 76 (sKizzo), 64 (ConformantFF). It turned out that the strength of our framework (with different QBF-solvers) is that it sometimes finds considerably shorter plans than heuristics do. Compared to T0 our solver found a shorter plan in 33 instances and compared to ConformantFF in 15 instances.

In Figure 2 we depict the minimum plan lengths obtained for an instance by a QBF-solver/planning tool. Table 1 shows the performance of the different solvers on selected interesting cases of the "Dungeon"-instances using the abbreviation "unsol" for "unsolvable", "to" for "timeout" and "ex250" for "exit code 250".

Due to space limitations, we mention only a few observations made in our evaluation. In the benchmark results we see that AQME explicitly shows that there is no plan of length at most 200, while other solvers run into a timeout. Unfortunately, AQME does not return certificates nor assignments, which makes it inapplicable in practice, as no plans can be extracted. Clearly, building a portfolio solver that returns certificates is a special challenge, since all solvers in the portfolio should handle certificates in a uniform way. Interestingly, AQME performs quite well compared to current state-of-the-art tools. Furthermore, it seems that a large number of monsters influences the performance of the planning tools much more than the QBF-based approach. In contrast, a large number of unknowns seems to be more critical for the QBF-approach. Finally, consider an instance that has an optimal solution with plan length ℓ. It is interesting to see that the runtime of the QBF-solvers increases quite dramatically with the plan length, but drops on satisfiable formulas with plan length ℓ. Due to the incremental generation of QBFs for each plan length an incremental QBF-solver (see, e. g., [13]) could give a notable performance boost.

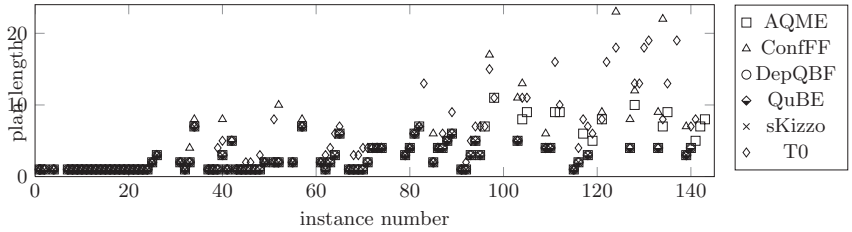


Fig. 2. An overview of all solutions found by the considered solvers.

#	#i	#m	#u	AQME t/pl	AQME cl/vars	ConfFF t/pl	DepQBF t/pl	DepQBF cl/vars	QuBE t/pl	QuBE cl/vars	sKizzo t/pl	sKizzo cl/vars	T0 t/pl
30	10	5	10	603.53	111887	0.04	to	27478	to	9031	to	11826	0.43
				>200	41377	unsol	≥49	10271	≥16	3473	≥21	4503	unsol?
40	10	20	4	2.24	1733	437.39	0.25	1733	0.29	1733	0.32	1733	0.09
				3	648	8	3	648	3	648	3	648	5
72	15	75	10	428.26	7195	to	1.06	7195	0.92	7195	-	5398	565.85
				4	2093	-	4	2093	4	2093	ex250	1651	4
111	25	50	3	41.31	63922	to	to	35374	to	42511	-	49648	0.62
				9	7321	-	≥5	4329	≥6	5077	ex250	5825	16
143	30	150	5	690.02	133543	to	to	65899	to	82810	-	48988	to
				8	10616	-	≥4	5756	≥5	6971	ex250	4541	-

Table 1. Detailed results for some interesting instances showing the instance number (#), the number of: items (#i), monsters (#m), unknown variables (#u), clauses (cl), variables (vars), the time in seconds (t) and the plan length (pl).

4 Conclusion

In this work we have presented a framework for transforming planning instances with uncertainty in the initial state to QBFs. Big collections of conformant planning instances that fit the input language of our framework (i. e., including the plethora of instances provided by the International Planning Competition [10]), thus become accessible as interesting, hard benchmarks for QBF-solvers.

Our experiments have revealed that on some instances the QBF-approach outperforms conventional planning tools. Still, we see several interesting points to be addressed by the QBF-community in order to make the QBF-based approach yet more efficient and beneficial. Returning assignments is necessary to be able to extract the actual plan. Obtaining full certificates is desirable as it also allows to verify the solution. Furthermore, as for each plan length a new QBF has to be generated, incremental QBF-solvers could give especially in this setting a drastic boost in performance. Since the QBF-solvers need most time on QBF-instances for plan lengths that are close to a solution, also QBF-solvers biased towards unsatisfiability could be a promising approach. Clearly any progress made in the world of QBF-solvers will directly carry over to our approach for planning.

We see further experiments as well as creating solver-specific QBF transformations as the most important tasks. Also developing additional hard benchmark-types and improving the dynamic grounder remains future work.

References

1. Website of AQME. http://www.mind-lab.it/aqme/, [accessed: April 25, 2013]
2. Baral, C., Kreinovich, V., Trejo, R.: Computational complexity of planning and approximate planning in the presence of incompleteness. Artificial Intelligence 122(1-2), 241–267 (2000)
3. Benedetti, M.: Evaluating QBFs via symbolic skolemization. In: Proc. LPAR. pp. 285–300 (2004)
4. Website of ConformantFF. http://fai.cs.uni-saarland.de/hoffmann/cff.html, [accessed: April 25, 2013]
5. Website of DepQBF. http://fmv.jku.at/depqbf/, [accessed: April 25, 2013]
6. Giunchiglia, E., Marin, P., Narizzano, M.: QuBE7.0, system description. Journal of Satisfiability. 7(8), 83–88 (2010)
7. Helmert, M.: The fast downward planning system. Journal of Artificial Intelligence Research (JAIR) 26, 191–246 (2006)
8. Hoffmann, J., Brafman, R.I.: Conformant planning via heuristic forward search: A new approach. Artificial Intelligence 170(6-7), 507–541 (2006)
9. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research (JAIR) 14, 253–302 (2001)
10. International conference on automated planning and scheduling (ICAPS). http://ipc.icaps-conference.org, [accessed: April 25, 2013]
11. Kautz, H., Selman, B., Hoffmann, J.: SATPlan: Planning as satisfiability. In: Abstract booklet of the 5th International Planning Competition, event in the context of ICAPS 2006 (2006)
12. Lonsing, F., Biere, A.: Integrating dependency schemes in search-based QBF solvers. In: Proc. SAT. pp. 158–171 (2010)
13. Marin, P., Miller, C., Lewis, M.D.T., Becker, B.: Verification of partial designs using incremental QBF solving. In: Proc. DATE. pp. 623–628 (2012)
14. Palacios, H., Geffner, H.: Compiling uncertainty away in conformant planning problems with bounded width. Journal of Artificial Intelligence Research (JAIR) 35, 623–675 (2009)
15. Pulina, L., Tacchella, A.: A self-adaptive multi-engine solver for quantified Boolean formulas. Constraints 14(1), 80–116 (2009)
16. Website of QuBE. http://www.star-lab.it/~qube/, [accessed: April 25, 2013]
17. Rintanen, J.: Asymptotically optimal encodings of conformant planning in QBF. In: Proc. AAAI. pp. 1045–1050 (2007)
18. Rintanen, J.: Madagascar: Efficient planning with SAT. In: The 2011 International Planning Competition – Description of Participating Planners – Deterministic Track. pp. 61–64 (2011)
19. Website of sKizzo. http://skizzo.info, [accessed: April 25, 2013]
20. Website of T0. http://ldc.usb.ve/~hlp/, [accessed: April 25, 2013]

Solution extraction from long-distance resolution proofs *

Uwe Egly and Magdalena Widl

Institute of Information Systems, Vienna University of Technology, Austria
{uwe,widl}@kr.tuwien.ac.at

1 Introduction

Much effort has been devoted in the past decade to the development of decision procedures for quantified Boolean formulas (QBFs) in order to improve their performance. Besides solving the decision problem of whether a closed QBF is true and returning yes or no, modern QBF solvers can produce clause or cube proofs to certify their answer. These proofs can be used to derive solutions to problems formulated as QBFs. For instance, if the QBF describes a synthesis problem, the solution to this problem can be generated from the proof. Examples for representations of such solutions are control circuits (manifested in Herbrand or Skolem functions) or control strategies. We review two approaches for solution generation below. We focus on false QBFs and clause Q-refutations. For true QBFs, both approaches work in a symmetric fashion.

Balabanov and Jiang [1] propose to extract a Herbrand function for each universal ($\forall$) variable from a Q-refutation of a false QBF. These functions are constructed in such a way that replacing each $\forall$ variable by its Herbrand function in the matrix and removing the prefix yields an unsatisfiable Boolean formula. Unsatisfiability can be checked by a common SAT solver if required. Their algorithm traverses the refutation and constructs a formula for each $\forall$ variable from the nodes of the refutation where the variable is reduced by universal reduction. The run-time of their extraction algorithm is polynomial in *the size of the refutation*.

Goultiaeva et al. [3] employ a game-theoretic view to QBFs. They present an algorithm which executes a winning strategy for the universal player from the refutation. The existential ($\exists$) player chooses a move, i.e., an assignment of truth values to all $\exists$ variables in the outermost quantifier block. The algorithm modifies the Q-refutation according to this move and removes the current quantifier block from the prefix. From the resulting refutation, the $\forall$ player computes her move, i.e., an assignment of truth values to all $\forall$ variables in the outermost quantifier block which is now universal. Modifications take place as above and the $\exists$ player continues. The run-time of this algorithm is also polynomial in *the size of the refutation*.

For both methods, their runtime is directly related to the refutation size. It is therefore desirable to have short refutations. Since there are families of QBFs (like the one from Kleine Büning et al. [4] discussed below), for which any Q-refutation is exponential,

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

the use of more powerful calculi to allow more succinct refutations is essential for fast extraction of Herbrand functions or strategies.

A way to strengthen the Q-resolution calculus is to add the additional inference rules presented in [1]. The resulting calculus is called *long-distance resolution* (LDQ-resolution) [9]. Its inference rules are reproduced in Appendix A as rules r_1 to r_4 and u_1. LDQ-resolution allows to resolve two clauses upon an $\exists$ pivot p resulting in a resolvent with complementary pairs of $\forall$ literals provided each pair's variable has a higher quantifier level than p. The derived clause is thus tautological.

This paper is intended to advertise the use of LDQ-resolution. First, LDQ-resolution may greatly simplify the clause learning component [2,6] in current QBF solvers. In this component, additional effort (in terms of Q-resolution steps) is required in order to avoid the generation of tautological clauses over $\forall$ literals. This additional effort often results in an increase of refutation size. Second, we show that LDQ-resolution has the potential to produce shorter refutations. More precisely, we show an exponential speed-up in refutation size compared to Q-resolution using the family $(\varphi_t)_{t\geq 1}$ of false QBFs introduced by Kleine Büning et al. in [4]. Third, we show that the strategy extraction algorithm in [3] can handle LDQ-refutations. Therefore, the performance of this algorithm directly benefits from exponentially shorter LDQ-refutations. Finally, we point out open questions with respect to the certificate extraction procedure in [1].

2 Short proofs for hard formulas

Let $(\varphi_t)_{t\geq 1}$ be a family of QBFs in PCNF. The formula φ_t has prefix

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

and the following clauses in the matrix:

K_0	$\overline{d_0}$	K_1	$d_0 \vee \overline{d_1} \vee \overline{e_1}$	
K_{2j}	$d_j \vee \overline{x_j} \vee \overline{d_{j+1}} \vee \overline{e_{j+1}}$	K_{2j+1}	$e_j \vee x_j \vee \overline{d_{j+1}} \vee \overline{e_{j+1}}$	$j = 1, \ldots, t-1$
K_{2t}	$d_t \vee \overline{x_t} \vee \overline{f_1} \vee \cdots \vee \overline{f_t}$	K_{2t+1}	$e_t \vee x_t \vee \overline{f_1} \vee \cdots \vee \overline{f_t}$	
B_{2j-1}	$x_j \vee f_j$	B_{2j}	$\overline{x_j} \vee f_j$	$j = 1, \ldots, t$

Kleine Büning et al. prove in [4, Theorem 3.2] that any Q-refutation for φ_t is exponential in t. Van Gelder [8] shows that there exists a short Q-resolution refutation for formulas of this class if resolution over universal variables is allowed. We show that φ_t has polynomial size LDQ-refutations. They have the form of a directed acyclic graph (DAG). Following the notation in [1], we will use x^* as a shorthand for $x \vee \overline{x}$.

1. Derive $d_t \vee \overline{x_t} \vee \bigvee_{i=1}^{t-1} \overline{f_i}$ from B_{2t} and K_{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 $K_{2(t-1)}$ 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 less 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 K_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 K_0 and K_1 to obtain $\bigvee_{i=1}^{t} x_i^*$, which can be reduced to $\square$.

The number of clauses in this refutation is in $O(t)$.

We have identified formulas which have short LDQ-refutations, but any Q-refutation of the same formula is exponential. Hence the processing of the short refutations by extraction algorithms can be much faster provided the algorithms can handle such a refutation without too much additional complication.

3 Strategy extraction from long-distance resolution

We show that the strategy extraction method from Q-refutations presented by Goultiaeva et al. in [3] can be applied in the same manner to LDQ-refutations.

As sketched in the introduction, the method is described as a game between a $\forall$ player and an $\exists$ player. The game 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 to the variables in the current block. The assignment for the $\forall$ player is then computed according to previous assignments as follows.

First, all leaf nodes of the refutation, the clauses of the input formula, are modified according to the existential assignment as defined below similarly to [3,7].

Definition 1. *A (partial)* assignment *to a set V of variables of a QBF in PCNF is a set σ of literals of V such that if $l \in \sigma$ then $\bar{l} \notin \sigma$. A clause K under an assignment σ is denoted $K_{\lceil\sigma}$ and defined as follows: $K_{\lceil\sigma} = \top$ if $K \cap \sigma \neq \emptyset$ and $K_{\lceil\sigma} = K \setminus \{q \mid q \in K \wedge \overline{q} \in \sigma\}$ otherwise.*

Second, the inner nodes and $\square$ are derived by applying LDQ-resolution rules or, in cases where the pivot element has been removed by the assignment, by applying additional sound derivation rules (P-rules) presented by Goultiaeva et al. [3] and reproduced in Appendix A. Then the refutation, now containing rules outside the LDQ-resolution calculus, is transformed back into an LDQ-refutation. This transformation results in each $\forall$ variable of the next quantifier block to be reduced at most once. At the $\forall$ player's turn, the assignment to each of these variables is determined by either choosing the opposite polarity of the variable's literal removed in the remaining universal reduction or any polarity in case the variable is not reduced at all. It is shown in [3] that for a Q-refutation, after each restriction of either the $\exists$ or the $\forall$ player, $\square$ is derived.

We show that this method also works for LDQ-refutations. First, we show in a fashion similar to [3,7] that in a refutation applying LDQ-rules and P-rules, each node generated from the restricted leaves subsumes the restriction of the node by the partial assignment. The proof of the following lemma can be found in Appendix A.

Lemma 1. *(cf. Lemma 2.6 in [7]) Given a QBF in PCNF $\psi = \exists V \mathcal{P} \phi$ with V the set of all variables of the outermost quantifier block, $\mathcal{P}$ the prefix of ψ without $\exists V$, and ϕ the matrix of ψ, let K, K^l and K^r be clauses of ϕ, and σ an assignment to V. Then it holds that $\mathsf{res}(K^l_{\lceil\sigma}, K^r_{\lceil\sigma}, p) \subseteq \mathsf{res}(K^l, K^r, p)_{\lceil\sigma}$ and $\mathsf{red}(K_{\lceil\sigma}, x) \subseteq \mathsf{red}(K, x)_{\lceil\sigma}$.*

Using this lemma, we show with an argument similar to Goultiaeva et al. [3], that (1) by applying the restriction rules with respect to an assignment to the variables of an $\exists$ quantifier block to an LDQ-refutation, the resulting proof derives $\square$, and (2) that further restricting the LDQ-refutation by the computed assignment to the $\forall$ variables of the next turn also derives $\square$. (More details can be found in Appendix A.)

The key reason why this works for LDQ-refutations in the same fashion as for Q-refutations is the following. In a restricted refutation, the applications of rules deriving tautologies of some $\forall$ variables (LD-steps, r_2 to r_4 in Appendix A) are always removed by the partial assignment to $\exists$ variables of the previous quantifier blocks. This is the case because an LD-step can result in a tautology $x \vee \overline{x}$ of some $\forall$ variable x only if the pivot element p (an $\exists$ variable) has a lower quantifier level than x. Thus before the $\forall$ player's turn, the pivot element of each LD-step that results in tautologies involving $\forall$ variables of the respective quantifier block is contained in the partial assignment. Therefore, either of the parents of the LD-step is set to $\top$, and by applying the corresponding derivation rule, only one polarity of the $\forall$ variable is left in the derived clause.

4 Conclusion and future work

We have shown an exponential speed-up in refutation size if LDQ-resolution instead of Q-resolution is employed to any QBF φ_t belonging to the family introduced in [4]. Given the fact that state of the art methods for the extraction of Herbrand functions or strategies require traversing the refutation, it is desirable to retrieve LDQ-refutations instead of Q-refutations from QBF solvers in order to speed up such extraction methods. We have further shown that strategies can be retrieved from LDQ-refutations by applying the state of the art algorithm described in [3]. This implies an exponential speed-up for the strategy extraction from any QBF in the family $(\varphi_t)_{t\geq 1}$.

It remains an open question how Herbrand functions can be extracted efficiently from LDQ-refutations. It is possible to build Boolean formulas from truth tables generated by the strategy extraction method in [3]. However, since each possible assignment to the existential variables has to be considered, this naive method is exponential in the quantifier prefix.

Further, to the best of our knowledge, a workflow including output and verifiation of LDQ-resolution proofs is currently not supported by state of the art QBF solvers. (According to Van Gelder [8], QuBE-cert produces tautological clauses, but it is unclear whether the LDQ calculus is used.) In QBF solvers based on clause learning such as DepQBF [5], the generation of tautological clauses is avoided by additional resolution steps which remove existential variables with a higher quantifier level in order to enable a separate universal reduction for each problematic universal variable. By allowing long distance resolution steps, the learning component could be simplified and the obtained refutations could be shorter.

References

1. V. Balabanov and J.-H. R. Jiang. Unified QBF certification and its applications. *Formal Methods in System Design*, 41(1):45–65, Apr. 2012.

2. E. Giunchiglia, M. Narizzano, and A. Tacchella. Clause/Term Resolution and Learning in the Evaluation of Quantified Boolean Formulas. *Journal of Artificial Intelligence Research (JAIR)*, 26:371–416, Sept. 2006.
3. 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)*, IJCAI'11, pages 546–553. AAAI Press, 2011.
4. H. Kleine Büning, M. Karpinski, and A. Flögel. Resolution for Quantified Boolean Formulas. *Information and Computation*, 117(1):12–18, Feb. 1995.
5. F. Lonsing and A. Biere. DepQBF: A Dependency-Aware QBF Solver (System Description). *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 7:71–76, 2010.
6. F. Lonsing, U. Egly, and A. Van Gelder. Efficient clause learning for quantified boolean formulas via QBF pseudo unit propagation. In *Theory and Application of Satisfiability Testing (SAT)*, 2013.
7. A. Van Gelder. Input Distance and Lower Bounds for Propositional Resolution Proof Length. In *Theory and Application of Satisfiability Testing (SAT)*, 2005.
8. A. Van Gelder. Contributions to the Theory of Practical Quantified Boolean Formula Solving. In M. Milano, editor, *Principles and Practice of Constraint Programming (CP)*, Lecture Notes in Computer Science, pages 647–663. Springer Berlin Heidelberg, 2012.
9. L. Zhang and S. Malik. Towards a Symmetric Treatment of Satisfaction and Conflicts in Quantified Boolean Formula Evaluation. In P. Hentenryck, editor, *Principles and Practice of Constraint Programming (CP)*, volume 2470 of *Lecture Notes in Computer Science*, pages 200–215. Springer Berlin Heidelberg, 2006.

A Strategy extraction from LDQ-resolution proofs

We write e for an existential variable, x (with or without subscript) for a universal variable, x^* for $x \vee \overline{x}$, $\mathsf{var}(q')$ for the variable of a literal $q' \in \{q, \overline{q}\}$, $\mathsf{lev}(q')$ for the quantifier level of the variable $\mathsf{var}(q')$ counting from the left to the right of the prefix, K^l, K^r, and K for leaf clauses or derived clauses, p for the existential pivot variable, and $\square$ for the empty clause. By QBF we refer to a QBF in PCNF. An LDQ-refutation is an LDQ-resolution derivation of $\square$.

The following reproduces the rules of the LDQ-resolution calculus (LDQ-rules) presented by Balabanov and Jiang [1] with the differences that universal reduction is not integrated in the resolution rules but represented by a separate rule and that only one universal literal is reduced in one step.

$$[p]\ \frac{K^l \vee p \qquad K^r \vee \overline{p}}{K^l \vee K^r} \tag{r_1}$$

$$[p]\ \frac{K^l \vee p \vee x \qquad K^r \vee \overline{p} \vee \overline{x}}{K^l \vee K^r \vee x^*} \quad \mathsf{lev}(p) < \mathsf{lev}(x) \tag{r_2}$$

$$[p]\ \frac{K^l \vee p \vee x \qquad K^r \vee \overline{p} \vee x^*}{K^l \vee K^r \vee x^*} \quad \mathsf{lev}(p) < \mathsf{lev}(x) \tag{r_3}$$

$$[p]\ \frac{K^l \vee p \vee x^* \qquad K^r \vee \overline{p} \vee x^*}{K^l \vee K^r \vee x^*} \quad \mathsf{lev}(p) < \mathsf{lev}(x) \tag{r_4}$$

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

As in [1], we extend universal reduction to delete also x^* and call x^* the merged variable in r_2, r_3, r_4, because $x, \overline{x}$ occur in both parent clauses, but they are merged into one occurrence of x^* in the resolvent. We require the restriction on the quantifier level in these rules to also apply to other merged variables in K^l and K^r. In r_1, K^l and K^r may contain $x, \overline{x}$ or x^* but *no merging is allowed*.

We define a (partial) assignment and a clause under a (partial) assignment similarly to [3,7]:

Definition 1. *A (partial)* assignment *to a set V of variables of a QBF in PCNF is a set σ of literals of V such that if $l \in \sigma$ then $\overline{l} \notin \sigma$. A clause K under an assignment σ is denoted $K_{\lceil\sigma}$ and defined as follows: $K_{\lceil\sigma} = \top$ if $K \cap \sigma \neq \emptyset$ and $K_{\lceil\sigma} = K \setminus \{q \mid q \in K \wedge \overline{q} \in \sigma\}$ otherwise.*

The algorithms `play` and `restrict` describe the algorithm presented by Goultiaeva et al. [3]. `play` implements the alternating turns of the universal ($\forall$) and the existential ($\exists$) player moving from the left to the right in the quantifier prefix. Each player chooses an assignment to the variables in the current quantifier block. The proof is modified after each assignment and results in an LDQ-refutation of the QBF under

the partial assignment. The $\exists$ player chooses an arbitrary assignment and the $\forall$ player chooses an assignment that depends on the current modified proof.

The modification of the LDQ-resolution consists of two steps represented by the procedures `restrict` and `transform`. First, `restrict` restricts the proof according to the chosen assignment. It modifies the leaves of the proof by applying the assignment according to Definition 1 and then modifies the successor nodes by either applying one of the LDQ-resolution rules described above or, if the pivot variable has been removed from at least one of the parents, by applying one of the additional rules presented in [3,7]. The additional rules (P-rules) are reproduced in the following (symmetric rules are omitted):

$$[p]\ \frac{K^l \vee p \qquad \top}{\top} \tag{r_5}$$

$$[p]\ \frac{K^l \vee p \qquad K^r}{K^r} \quad \overline{p} \notin K^r \tag{r_6}$$

$$[p]\ \frac{K^l \qquad K^r}{\mathsf{narrower}(K^l,K^r)} \quad \overline{p}, p \notin K^l \text{and } \overline{p}, p \notin K^r \tag{r_7}$$

$$[x]\ \frac{K}{K} \quad x \notin K \tag{u_2}$$

$$[x]\ \frac{\top}{\top} \tag{u_3}$$

where $\mathsf{narrower}(K^l, K^r)$ returns the clause containing less literals. If K^l and K^r contain the same number of literals, K^l is returned. Observe that $\square$ is the narrowest clause and $\top$ contains all literals.

Note that, even though each P-rule represents a sound derivation step, applying P-rules results in a proof outside the LDQ-calculus. Starting at the leaves of the proof, the function `transform` (Π^p), where Π^p is a proof containing clauses derived using LDQ-rules and P-rules, removes parts of the proof that have become redundant due to the application of P-rules. Thereby also all applications of P-rules are removed. More specifically, for node $K = \top$ derived by rule r_5, its parent $K^l \vee p$ is removed with all its ancestors, and K is merged with its other parent $\top$. A similar procedure is applied to nodes derived by rules r_6 and r_7. Each node derived by rules u_1 to u_3 is merged with its parent. When $\square$ is derived, the procedure stops and all nodes that are not ancestors of $\square$ are removed. $\top$-clauses are eliminated by eventually applying rule r_6 or by removing nodes when $\square$ is found. The resulting refutation is an LDQ-refutation.

After restricting the LDQ-refutation by a computed assignment $\sigma_\forall$ to $\forall$ variables, only one redundant clause, namely $\square$, has to be removed because after the previous restriction by $\sigma_\exists$ and the following repair, the refutation contains at most one universal reduction for any universal variable x of the current quantifier block. This is the case because the condition for x to be reduced from any clause K is that K does not contain any $\exists$ variables with a quantifier level higher than x. Any $\exists$ variable in K is therefore in $\sigma_\exists$, and after restricting the refutation, K is either (1) set to $\top$ or (2) all $\exists$ variables are removed. The topologically first clause (furthest to $\square$) where (2) happens derives $\square$ after reducing each of its $\forall$ variables. An assignment $\sigma_\forall$ to a $\forall$ variable x as computed in Line 8 of Algorithm 1 chooses the opposite polarity of x when it is reduced in node K. This way, according to Definition 1, K becomes $\square$ and only the $\square$ derived from K has to be removed.

In the following we present a proof that `restrict`, when called in Line 4 or Line 9 of `play`, returns a derivation of $\square$ using LDQ-rules and P-rules. Proposition 1 shows that this holds for an arbitrary assignment to all $\exists$ variables in the outermost quantifier block, and Proposition 2 shows the same for the computed assignment to $\forall$ variables.

We write $\mathsf{res}(K^l,K^r,p)$ for a resolution step over pivot element p according to rules r_1 to r_7, and $\mathsf{red}(K,x)$ for a reduction step reducing variable x according to rules u_1 to u_3.

We start by showing that any node generated by applying an LDQ-rule or a P-rule from the restricted leaves subsumes the restriction of the node by the partial assignment.

Lemma 1. *(cf. Lemma 2.6 in [7]) Given a QBF in PCNF $\psi = \exists V \mathcal{P}\phi$ with V the set of all variables of the outermost quantifier block, $\mathcal{P}$ the prefix of ψ without $\exists V$, and ϕ the matrix of ψ, let K, K^l and K^r be clauses of ϕ, and σ an assignment to V. Then it holds that $\mathsf{res}(K^l_{\lceil\sigma}, K^r_{\lceil\sigma}, p) \subseteq \mathsf{res}(K^l, K^r, p)_{\lceil\sigma}$ and $\mathsf{red}(K_{\lceil\sigma}, x) \subseteq \mathsf{red}(K, x)_{\lceil\sigma}$.*

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

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

For resolution steps, either of the following cases applies (symmetric cases are omitted):
(1) $C_1(\sigma, K^l)$ and $C_1(\sigma, K^r)$: $K^l_{\lceil\sigma} = K^l$, $K^r_{\lceil\sigma} = K^r$, and $\mathsf{res}(K^l, K^r, p)_{\lceil\sigma} = \mathsf{res}(K^l, K^r, p)$. By applying rule r_1, r_2, r_3, or r_4, we obtain $\mathsf{res}(K^l_{\lceil\sigma}, K^r_{\lceil\sigma}, p) = \mathsf{res}(K^l, K^r, p)$. Therefore the subset relation holds.
(2) $C_2(\sigma, K^l)$ and $C_1(\sigma, K^r)$: $K^l_{\lceil\sigma} = \top$, $K^r_{\lceil\sigma} = K^r$, and $\mathsf{res}(K^l, K^r, p)_{\lceil\sigma} = \top$. By applying rule r_5, we obtain $\mathsf{res}(K^l_{\lceil\sigma}, K^r_{\lceil\sigma}, p) = \top$. Therefore the subset relation holds.
(3) $C_3(\sigma, K^l)$ and $C_1(\sigma, K^r)$: Since $C_1(\sigma, K^r)$, $\forall q \in \sigma$ it holds that $p \neq q$ and $p \neq \overline{q}$. Then $K^l_{\lceil\sigma} = K^l \setminus \{\overline{q} \mid q \in \sigma\}$, $K^r_{\lceil\sigma} = K^r$, and $\mathsf{res}(K^l, K^r, p)_{\lceil\sigma} = K^l \cup K^r \setminus \{\overline{q} \mid q \in \sigma\}$. By applying rule r_1, r_2, r_3, or r_4, we obtain $\mathsf{res}(K^l_{\lceil\sigma}, K^r_{\lceil\sigma}, p) = K^l \cup K^r \{\overline{q} \mid q \in \sigma\}$. Therefore the subset relation holds.
(4) $C_2(\sigma, K^l)$ and $C_2(\sigma, K^r)$: $K^l_{\lceil\sigma} = \top$, $K^r_{\lceil\sigma} = \top$, and $\mathsf{res}(K^l, K^r, p)_{\lceil\sigma} = \top$. By applying rule r_5, we obtain $\mathsf{res}(K^l_{\lceil\sigma}, K^r_{\lceil\sigma}, p) = \top$. Therefore the subset relation holds.
(5) $C_2(\sigma, K^l)$ and $C_3(\sigma, K^r)$: Then $K^l_{\lceil\sigma} = \top$, and $K^r_{\lceil\sigma} = K^r \setminus \{\overline{q} \mid q \in \sigma\}$. We have to distinguish two cases:
(a) $\exists q \in \sigma$ such that $\mathsf{var}(q) = p$. Then $\mathsf{res}(K^l, K^r, p)_{\lceil\sigma} = (K^l \cup K^r) \setminus \{\overline{q} \mid q \in \sigma\}$. By applying rule r_6, we obtain $\mathsf{res}(K^l_{\lceil\sigma}, K^r_{\lceil\sigma}, p) = K^r \setminus \{\overline{q} \mid \overline{q} \in \sigma\}$. Therefore the subset relation holds.
(b) Otherwise (σ does not contain the pivot p). Then $\mathsf{res}(K^l, K^r, p)_{\lceil\sigma} = \top$. By applying rule r_5, we obtain $\mathsf{res}(K^l_{\lceil\sigma}, K^r_{\lceil\sigma}, p) = \top$. Therefore the subset relation holds.
(6) $C_3(\sigma, K^l)$ and $C_3(\sigma, K^r)$ $K^l_{\lceil\sigma} = K^l \setminus \{\overline{q} \mid q \in \sigma\}$, $K^r_{\lceil\sigma} = K^r \setminus \{\overline{q} \mid q \in \sigma\}$, and $\mathsf{res}(K^l, K^r, p)_{\lceil\sigma} = K^l \cup K^r \setminus \{\overline{q} \mid q \in \sigma\}$. By applying rule r_1, r_2, r_3, or r_4, we obtain $\mathsf{res}(K^l_{\lceil\sigma}, K^r_{\lceil\sigma}, p) = K^l \cup K^r \setminus \{\overline{q} \mid q \in \sigma\}$. Therefore the subset relation holds.

For reduction steps, either of the following cases applies:
(7) $C_1(\sigma, K)$: $K_{\lceil\sigma} = K$ and $\mathsf{red}(K, x)_{\lceil\sigma} = \mathsf{red}(K, x)$. Therefore the subset relation holds.
(8) $C_2(\sigma, K)$: $K_{\lceil\sigma} = \top$ and $\mathsf{red}(K, x)_{\lceil\sigma} = \top$. By applying rule u_3 we obtain $\mathsf{red}(K_{\lceil\sigma}, x) = \top$. Therefore the subset relation holds.
(9) $C_3(\sigma, K)$: $K_{\lceil\sigma} = K \setminus \{\overline{q} \mid q \in \sigma\}$ and $\mathsf{red}(K, x)_{\lceil\sigma} = (K \setminus \{x\}) \setminus \{\overline{q} \mid q \in \sigma\}$. By applying rule u_1, we obtain $\mathsf{red}(K_{\lceil\sigma}, x) = (K \setminus \{x\}) \setminus \{\overline{q} \mid q \in \sigma\}$. Therefore the subset relation holds. □

With respect to the application of rules r_2 to r_4 (LD-steps), we observe the following from the `play` algorithm and cases (1) to (6) in the above lemma: `play` iterates over the quantifier prefix from the left to the right. The condition for a clause to be derived by an LD-step is that the quantifier level of the pivot variable p is lower than the level of the merging variable x^*. If the algorithm is executing the existential quantifier level ℓ, any of the cases (1) to (6) except (5a), i.e. all cases where the pivot remains unassigned, can only happen on an LD-step if $\mathsf{lev}(x^*) - \ell > 1$. That is, $\mathsf{var}(x)$ is not in the next quantifier block. Otherwise (if x^* is on the next quantifier level) p must be assigned in the current turn and case (5a) applies. Case (5a), where p is assigned, can happen on any LD-step. In this case, the LD-step is modified by rule r_6, which removes the merged variable. It is thus always the case that after restricting the refutation according to an existential quantifier block on the ℓ-th level, the refutation does not contain any LD-step generating a variable x^* with $\mathsf{lev}(x^*) = \ell + 1$. Therefore, in the following $\forall$ player's turn, none of the respective $\forall$ variables occurs as tautology.

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

Proof. By induction on the structure of Π using Lemma 1. □

Proposition 1. *Given a QBF in PCNF $\psi = \exists V \mathcal{P} \phi$ with V the set of all variables of the outermost quantifier block, $\mathcal{P}$ the prefix of ψ without $\exists V$, and ϕ the matrix of ψ, an LDQ-resolution Π deriving $\square$ from ψ, and an assignment $\sigma_\exists$ to V, it holds that $\Pi^p = \texttt{restrict}(\Pi, \sigma_\exists)$ derives $\square$ from $\mathcal{P}\phi_{\lceil\sigma_\exists}$.*

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

Proposition 2. *Given a QBF in PCNF $\psi = \forall V \mathcal{P} \phi$ with V the set of all variables of the outermost quantifier block, $\mathcal{P}$ the prefix of ψ without $\forall V$, and ϕ the matrix of ψ, an LDQ-resolution Π deriving $\square$ from ψ, and an assignment $\sigma_\forall$ to V as computed in Line 8 of Algorithm 1, $\Pi^p = \texttt{restrict}(\Pi, \sigma_\forall)$ derives $\square$ from $\mathcal{P}\phi_{\lceil\sigma_\forall}$.*

Proof. For any $q \in \sigma_\forall$ it holds that q is either not reduced at all, or reduced exactly once in Π. If q 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 dual literals that are reduced exactly once in the proof. Then there is a set $\mathcal{K}$ with $|\mathcal{K}| = |R|$ of nodes such that the nodes

in $\mathcal{K}$ are directly following one another, each reducing exactly one literal r in R. The last reduced node of $\mathcal{K}$ results in $\square$. This is the case because all literals of R are in the outermost quantifier block. `restrict` (Π,$\sigma_\forall$) then applies rule u_2 to each clause K, setting each K in $\mathcal{K}$ to $\square$.

$\square$

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**
    $\sigma_\exists \leftarrow$ any assignment to variables in Q
    $\Pi^p \leftarrow$ `restrict` (Π,$\sigma_\exists$) *(Π^p is not an LDQ-resolution)*
    $\Pi \leftarrow$ `transform` (Π^p) *(Π is an LDQ-resolution)*
  else Q is universal
    $K \leftarrow$ topologically first node with no existential literals
    $\sigma_\forall \leftarrow \{\overline{x} \mid x \in K \wedge \mathsf{var}(x) \in Q\} \cup \{x \mid x \notin K \wedge \overline{x} \notin K \wedge \mathsf{var}(x) \in Q\}$
    $\Pi^p \leftarrow$ `restrict` (Π,$\sigma_\forall$)
    $\Pi \leftarrow$ `transform` (Π^p)

Algorithm 2: `restrict`

Input : LDQ-refutation Π, assignment σ for variables in outermost block
Output: Restricted refutation containing LD-rules and P-rules

```
1  foreach leaf node K in Π do
2      K ← K⌈σ;
3  foreach inner node K topologically in Π do
4      if K is a resolution node then
5          K^l,K^r ← parents of K
6          p ← pivot of K
7          K ← res(K^l,K^r,p)
8      else K is a reduction node
9          K^c ← parent of K
10         x ← variable reduced from K^c
11         K ← red(K^c,p)
12
13 return Π
```

∀Exp+Res Does not P-Simulate Q-resolution

Mikoláš Janota[1] and Joao Marques-Silva[1,2]

[1] IST/INESC-ID, Lisbon, Portugal
[2] University College Dublin, Ireland

1 Introduction

In our recent work [9], we formalized a proof system ∀Exp+Res underlying expansion–based QBF solvers, namely sKizzo [1], RAReQS [8], and to some extend Quantor [2].

In the same work, we conjectured that ∀Exp+Res and Q-resolution [4] cannot p-simulate one another. In this work we confirm one half of this conjecture, i.e. that ∀Exp+Res cannot p-simulate Q-resolution. This observation furthers our understanding of the differences between the different underlying methods for solving QBFs as well as the differences between the underlying proof systems.

2 Preliminaries

A *literal* is a Boolean variable or its negation. For a literal l, we write $\bar{l}$ to denote the literal *complementary* to l, i.e. $\bar{x} = \neg x$, $\overline{\neg x} = x$. A *clause* is a disjunction of finitely many literals. A formula in *conjunctive normal form* (CNF) is a conjunction of finitely many clauses. As usual, whenever convenient, a clause is treated as a set of literals and a CNF formula as a set of sets of literals.

For a literal $l = x$ or $l = \bar{x}$, we write $\mathsf{var}(l)$ for x. Analogously, for a clause C, $\mathsf{var}(C)$ denotes $\{\mathsf{var}(l) \mid l \in C\}$ and for a CNF ψ, $\mathsf{var}(\psi)$ denotes $\{l \mid l \in \mathsf{var}(C), C \in \psi\}$.

Substitutions are denoted as $x_1/\psi_1, \ldots, x_n/\psi_n$, with $x_i \neq x_j$ for $i \neq j$. The set of variables $x_1, \ldots, x_n$ is called the *domain* of the substitution. An application of a substitution is denoted as $\phi[x_1/\psi_1, \ldots, x_n/\psi_n]$ meaning that variables x_i are simultaneously substituted with corresponding formulas ψ_i in ϕ. A substitution is called an *assignment* iff each ψ_i is one of the constants 0, 1. An assignment is called *total*, or *complete*, for a set of variables X if each $x \in X$ is in the domain of the assignment. For substitutions $\tau_1 = x_1/\psi_1, \ldots, x_n/\psi_n$ and $\tau_2 = y_1/\xi_1, \ldots, y_m/\xi_m$ with distinct domains we write $\tau_1 \cup \tau_2$ for the substitution $x_1/\psi_1, \ldots, x_n/\psi_n, y_1/\xi_1, \ldots, y_m/\xi_m$.

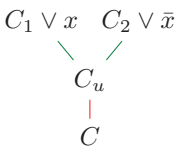
Quantified Boolean Formulas (QBFs) [3] are an extension of propositional logic with quantifiers with the standard semantics that $\forall x. \Psi$ is satisfied by the same truth assignments as $\Psi[x/0] \wedge \Psi[x/1]$ and $\exists x. \Psi$ as $\Psi[x/0] \vee \Psi[x/1]$. Unless specified otherwise, QBFs are in *closed prenex* form with a CNF *matrix*, i.e. $\mathcal{Q}_1 X_1 \ldots \mathcal{Q}_k X_k.\ \phi$, where X_i are pairwise disjoint sets of variables; $\mathcal{Q}_i \in \{\exists, \forall\}$ and $\mathcal{Q}_i \neq \mathcal{Q}_{i+1}$. The formula ϕ is in CNF and is defined only on variables $X_1 \cup \ldots \cup X_k$. The propositional part ϕ is called the *matrix* and the rest the *prefix*. We write QCNF to talk about formulas in this form. If a variable x is in the set X_i, we say that x is at *level* i and write $\mathsf{lv}(x) = i$; we write $\mathsf{lv}(l)$ for $\mathsf{lv}(\mathsf{var}(l))$. A closed QBF is *false* (resp. *true*), iff it is semantically equivalent to the constant 0 (resp. 1).

If a variable is universally quantified, we say that the variable is universal. For a literal l and a universal variable x such that $\mathsf{var}(l) = x$, we say that l is universal. The notions of existential variable and literal are defined analogously.

2.1 Q-resolution

Q-resolution [4] is an extension of propositional resolution for showing that a QCNF is false. For a clause C, a universal literal $l \in C$ is *blocked* by an existential literal $k \in C$ iff $\mathsf{lv}(l) < \mathsf{lv}(k)$. $\forall$*-reduction* is the operation of removing from a clause C all universal literals that are *not* blocked by some existential literal. For two $\forall$-reduced clauses $x \vee C_1$ and $\bar{x} \vee C_2$, where x is an existential variable, a *Q-resolvent* [4] is obtained in two steps. (1) Compute $C_u = C_1 \cup C_2 \setminus \{x, \bar{x}\}$. If C_u contains complementary literals, the Q-resolvent is undefined. (2) Compute C_r by $\forall$-reducing C_u; C_r is called the resolvent of C_1 and C_2. For a QCNF $\mathcal{P}.\phi$, a *Q-resolution proof* of a clause C is a finite sequence of clauses $C_1, \ldots, C_n$ where $C_n = C$ and any C_i in the sequence is part of the given matrix ϕ or it is a Q-resolvent for some pair of the preceding clauses. A Q-resolution proof is called a *refutation* iff C is the empty clause, denoted by $\bot$.

In this paper Q-resolution proofs are treated as connected directed acyclic graphs, i.e. each clause in the proof corresponds to some node labeled with that clause. We assume that the input clauses are already $\forall$-reduced. Q-resolution steps are depicted as on the right. Note that the $\forall$-reduction step is depicted separately.



2.2 Proof complexity

A proof system P is a relation $P(\Phi, \pi)$ computable in polynomial time such that a formula Φ is true iff there exists a proof π for which $P(\Phi, \pi)$. A proof system P_1 *p-simulates* a proof system P_2 iff any proof in P_2 of a formula Φ can be translated into a proof in P_1 of Φ in polynomial time (c.f. [5,10]).

We count the sizes of Q-resolution proofs as the number of resolution steps plus the number of $\forall$-reductions where each reduced literal is counted separately.

2.3 Expansions

Based on the equivalence $\forall y.\Phi = \Phi[y/0] \wedge \Phi[y/1]$, *expansion* of universal quantifiers enables decreasing the number of quantifiers and maintaining prenex normal form at the cost of introducing fresh variables. Once all universal variables are expanded, a SAT solver can be invoked. *Partial expansions* enable us to mitigate the size of the formula. For instance, for the formula $\forall y \exists x.\, (y \vee x) \wedge (y \vee \bar{x})$ it is sufficient to consider an expansion with $y/0$ to show the formula false.

For a general QBF $\Phi = \forall \mathcal{U}_1 \exists \mathcal{E}_2 \ldots \forall \mathcal{U}_{2N-1} \exists \mathcal{E}_{2N} \,.\, \phi$ (WLOG we start with a universal quantifier) our recent work [9] defines an expansion as follows.

Definition 1 (**$\forall$-expansion tree**). *A* $\forall$-expansion tree *is a rooted tree $\mathcal{T}$ such that each path $p_0 \xrightarrow{\tau_1} p_1 \ldots \xrightarrow{\tau_N} p_N$ in $\mathcal{T}$ from the root node p_0 to some leaf node p_N has exactly N edges and each edge $p_{i-1} \xrightarrow{\tau_i} p_i$ is labeled with a total assignment τ_i to the variables $\mathcal{U}_{2i-1}$, for $i \in 1..N$. Each path in $\mathcal{T}$ is uniquely determined by its labeling.*

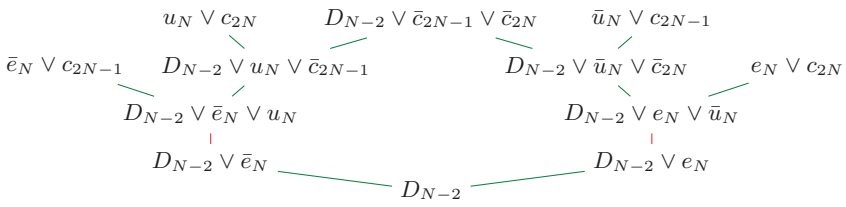


Fig. 1. Q-resolution proof sketch (where $D_{N-2} = \bar{c}_1 \vee \cdots \vee \bar{c}_{2N-3} \vee \bar{c}_{2N-2}$)

Definition 2 (∀-expansion). *Let $\mathcal{T}$ be a ∀-expansion tree. For a root-to-leaf path P in $\mathcal{T}$ and a clause C, the following rules define the ∀-expansion of C by P, ∀-expansion of ϕ by P, and ∀-expansion of Φ by $\mathcal{T}$. These expansions are denoted as $\mathscr{E}(P, C)$, $\mathscr{E}(P, \psi)$, and $\mathscr{E}(\mathcal{T}, \Phi)$, respectively.*

1. *For each path P_k in $\mathcal{T}$ from the root, labeled by assignments $\tau_1, \ldots, \tau_k$, and an existential variable x with $\mathsf{lv}(x) = 2k$ define a fresh variable $x^{\tau_1,\ldots,\tau_k}$.*
2. *For each path P in $\mathcal{T}$ from the root to some leaf labeled by $\tau_1, \ldots, \tau_N$, and a clause $C \in \phi$ define $\mathscr{E}(P, C)$ as $C[\tau_1 \cup \ldots \tau_N \cup \tau_R]$ where*

$$\tau_R = \{x/x^{\tau_1,\ldots,\tau_k} \mid 1 \leq k \leq N, x \text{ an existential variable s.t. } \mathsf{lv}(x) = 2k\}$$

3. *For each path P in $\mathcal{T}$ from the root to some leaf define $\mathscr{E}(P, \phi)$ as a union of $\mathscr{E}(P, C)$ for $C \in \phi$.*
4. *Define $\mathscr{E}(\mathcal{T}, \Phi)$ as the union of all $\mathscr{E}(P, \phi)$ for each root-to-leaf path P in $\mathcal{T}$.*

Definition 3. *A proof in ∀Exp+Res of a formula Φ is a pair $(\mathcal{T}, \pi)$ where π is a resolution proof of $\mathscr{E}(\mathcal{T}, \Phi)$.*

3 Separation

For a natural number N, construct the following formula[3].

$$\begin{array}{l} \exists e_1 \forall u_1 \exists c_1 c_2 \ldots \exists e_N \forall u_N \exists c_{2N-1} c_{2N}. \\ \bigwedge_{i \in 1..N} (\bar{e}_i \vee c_{2i-1}) \wedge (\bar{u}_i \vee c_{2i-1}) \wedge (e_i \vee c_{2i}) \wedge (u_i \vee c_{2i}) \wedge \\ \bigvee_{i \in 1..2N} \bar{c}_i \end{array} \tag{1}$$

The formula has a Q-resolution proof comprising linear number of resolution steps. Starting from the clause $\bigvee_{i \in 1..2N} \bar{c}_i$, the literals $\bar{c}_i$ are gradually resolved away from the highest to the lowest level. Figure 1 shows how the variables c_{2N-1}, c_{2N} are resolved away. The other pairs of variables are resolved away in the same fashion resulting in the empty clause in the end.

Now we show that any ∀Exp+Res proof of the formula is exponential. Let us look at the expansion of the first universal variable, i.e. the variable u_1. We observe that both

[3] See http://sat.inesc-id.pt/~mikolas/qbfw13 for a generator.

of the values are needed in the expansion. Let us expand u_1, yielding the two following formulas, one for $u_1/1$ and one for $u_1/0$ (for now other variables are left unexpanded).

$$\begin{array}{l}\exists e_2 \forall u_2 \exists c_3 c_4 \ldots \exists e_N \forall u_N \exists c_{2N-1} c_{2N}. \\ (\bar{e}_1 \vee c_1^{u_1/1}) \wedge (c_1^{u_1/1}) \wedge (e_1 \vee c_2^{u_1/1}) \wedge \\ \bigwedge_{i \in 2..N} (\bar{e}_i \vee c_{2i-1}) \wedge (\bar{u}_i \vee c_{2i-1}) \wedge (e_i \vee c_{2i}) \wedge (u_i \vee c_{2i}) \wedge \\ \bar{c}_1^{u_1/1} \vee \bar{c}_2^{u_1/1} \vee \bigvee_{i \in 3..2N} \bar{c}_i \end{array} \tag{2}$$

$$\begin{array}{l}\exists e_2 \forall u_2 \exists c_3 c_4 \ldots \exists e_N \forall u_N \exists c_{2N-1} c_{2N}. \\ (\bar{e}_1 \vee c_1^{u_1/0}) \wedge (c_2^{u_1/0}) \wedge (e_1 \vee c_2^{u_1/0}) \wedge \\ \bigwedge_{i \in 2..N} (\bar{e}_i \vee c_{2i-1}) \wedge (\bar{u}_i \vee c_{2i-1}) \wedge (e_i \vee c_{2i}) \wedge (u_i \vee c_{2i}) \wedge \\ \bar{c}_1^{u_1/0} \vee \bar{c}_2^{u_1/0} \vee \bigvee_{i \in 3..2N} \bar{c}_i \end{array} \tag{3}$$

The formula (2) is satisfiable because e_1 can be set to 1, $c_2^{u_1/1}$ to 0 and the rest of the existential variables to 1 and similarly for the formula (3). Since both formulas (2) and (3) are satisfiable, both of them are needed to show unsatisfiability.

Regardless of how the rest of variables in (2) and (3) are expanded, the only variable they share is e_1 because universal variables are expanded away and all existential variables are labeled with $u_1/1$ and $u_1/0$, respectively.

From Craig's interpolation theorem [6,7], there is an interpolant I using only variables common to (2) and (3), i.e. e_1, and it holds that (2) $\Rightarrow I$ and (3) $\Rightarrow \neg I$. Due to the structure of the formulas, I must be e_1. Equivalently (2) $\wedge\, \bar{e}_1$ and (3) $\wedge\, e_1$ are false (unsatisfiable). The formula (2) $\wedge\, \bar{e}_1$ is equivalent to the following.

$$\begin{array}{l}(c_1^{u_1/1}) \wedge (c_2^{u_1/1}) \wedge \\ \exists e_2 \forall u_2 \exists c_3 c_4 \ldots \exists e_N \forall u_N \exists c_{2N-1} c_{2N}. \\ \bigwedge_{i \in 2..N} (\bar{e}_i \vee c_{2N-1}) \wedge (\bar{u}_i \vee c_{2N-1}) \wedge (e_i \vee c_{2N}) \wedge (u_i \vee c_{2N}) \wedge \\ \bigvee_{i \in 3..2N} \bar{c}_i \end{array}$$

Since $c_1^{u_1/1}, c_2^{u_1/1}$ appear in only the first part of the formula, the second part of the formula has to be false (i.e. unsatisfiable after full expansion). The second part of the formula is in the same form as the initial formula just with three less variables. Repeating the same arguments, this formula has to be fully expanded. Following the same argument for (3) $\wedge\, e_0$ shows that the proof needs to expand each variable in both polarities at all depths. Thus yielding an exponential expansion of the given formula.

4 Conclusion and Future Work

This paper investigates a recently-made conjecture [9] that states that Q-resolution [4] and expansion-based solving (∀Exp+Res) are incomparable proof systems. We investigated half of the conjecture, namely that ∀Exp+Res does *not* p-simulate Q-resolution. This observation further complements the result of [9], where it was shown that ∀Exp+Res *can* p-simulate *tree* Q-resolution. The result shown here confirms that this p-simulation result is "tight", i.e. non-tree Q-resolution cannot be simulated. To show that that Q-resolution cannot simulate ∀Exp+Res is the subject of future work.

References

1. Benedetti, M.: Evaluating QBFs via symbolic Skolemization. In: Baader, F., Voronkov, A. (eds.) LPAR. pp. 285–300 (2004)
2. Biere, A.: Resolve and expand. In: SAT (2004)
3. Büning, H.K., Bubeck, U.: Theory of quantified Boolean formulas. In: Handbook of Satisfiability. IOS Press (2009)
4. Büning, H.K., Karpinski, M., Flögel, A.: Resolution for quantified Boolean formulas. Inf. Comput. 117(1) (1995)
5. Cook, S.A., Reckhow, R.A.: The relative efficiency of propositional proof systems. J. Symb. Log. 44(1), 36–50 (1979)
6. Craig, W.: Linear reasoning. a new form of the Herbrand-Gentzen theorem. J. Symb. Log. 22(3), 250–268 (1957)
7. Harrison, J.: Handbook of Practical Logic and Automated Reasoning. Cambridge University Press (2009)
8. Janota, M., Klieber, W., Marques-Silva, J., Clarke, E.M.: Solving QBF with counterexample guided refinement. In: Cimatti, A., Sebastiani, R. (eds.) SAT. Lecture Notes in Computer Science, vol. 7317, pp. 114–128. Springer (2012)
9. Janota, M., Marques-Silva, J.: On propositional QBF expansions and Q-resolution. In: SAT (2013), a preprint available at http://sat.inesc-id.pt/~mikolas/.
10. Urquhart, A.: The complexity of propositional proofs. Bulletin of the EATCS 64 (1998)

Variable Dependencies and Q-Resolution*

Friedrich Slivovsky and Stefan Szeider

Institute of Information Systems, Vienna University of Technology, Vienna, Austria
fs@kr.tuwien.ac.at,stefan@szeider.net

Abstract. We propose $Q(D)$-*resolution*, a proof system for Quantified Boolean Formulas. $Q(D)$-resolution is a generalization of Q-resolution parameterized by a dependency scheme D. This system is closely related to search-based QCNF solvers that use dependency schemes to generalize the QDPLL algorithm, such as DepQBF. We prove that $Q(D^{\text{std}})$-resolution is sound by presenting a transformation of $Q(D^{\text{std}})$-resolution refutations into ordinary Q-resolution refutations, where D^{std} denotes the so-called Standard Dependency Scheme.

1 Introduction

Most QBF solvers generate certificates for the (un)satisfiability of input formulas. These certificates can help verify the solver's correctness or provide additional information to an application that uses the solver as a subroutine. Search-based QBF (QDPLL) solvers typically use Q-resolution [2] refutations for certificates [5]. QDPLL solvers normally assign variables in the order induced by the quantifier prefix of an input QBF (in prenex normal form). In this case, the trace generated by the solver can be used to construct a Q-resolution refutation or a cube resolution proof of this formula [3]. Since assigning variables in the order given by the quantifier prefix can impair solver performance, it was suggested to use *dependency schemes* [6, 7] to give solvers more freedom in choosing the order of variable assignments [1] (informally, a dependency scheme is a mapping D that associates each formula $\mathcal{F}$ with a binary relation $D_{\mathcal{F}}$ on its variables that represents potential variable dependencies). This idea was successfully implemented in the state-of-the-art solver DepQBF [4, 1]. However, DepQBF's use of dependency schemes breaks the immediate correspondence between solver traces and Q-resolution proofs.

In this article, we propose $Q(D)$-*resolution*, a proof system for Quantified Boolean Formulas in prenex CNF normal form (QCNF). $Q(D)$-resolution is a generalization of Q-resolution, parameterized by a dependency scheme D. $Q(D)$-resolution uses a more powerful version of the *universal reduction* rule found in Q-resolution: a universally quantified variable x can be removed from a clause as long as that clause contains no variables that may depend on x, according to D [1]. $Q(D)$-resolution is closely related to search-based QCNF solvers that use dependency schemes to generalize the QDPLL algorithm in the spirit of DepQBF.

Our main contribution is an algorithm that transforms a $Q(D^{\text{std}})$-resolution refutation of a formula into an ordinary Q-resolution refutation of the same formula, where

* This research was supported by the ERC (COMPLEX REASON, 239962).

D^{std} is the so-called Standard Dependency Scheme [7]. Due to space constraints all proofs are placed in the appendix.

2 Preliminaries

We consider quantified boolean formulas in *quantified conjunctive normal form* (QCNF). A QCNF formula consists of a (quantifier) *prefix* and a CNF formula, called the *matrix*. A CNF formula is a finite set of *clauses*, where each clause is a finite set of *literals*. Literals are negated or unnegated propositional *variables*. If x is a variable, we put $\overline{x} = \neg x$ and $\overline{\neg x} = x$, and let $var(x) = var(\neg x) = x$. If X is a set of literals, we write $\overline{X}$ for the set $\{\, \overline{x} : x \in X \,\}$. For a clause C, we let $var(C)$ be the set of variables occuring (negated or unnegated) in C. For a QCNF formula $\mathcal{F}$ with matrix F, we put $var(\mathcal{F}) = var(F) = \cup_{C \in F} var(C)$, and $lit(\mathcal{F}) = var(\mathcal{F}) \cup \overline{var(\mathcal{F})}$. We call a clause *tautological* if it contains the same variable negated as well as unnegated. Unless otherwise stated, we assume that the matrix of a formula contains only non-tautological clauses. The prefix of a QCNF formula $\mathcal{F}$ is a sequence $\mathsf{Q}_1 x_1 \ldots \mathsf{Q}_n x_n$ of *quantifications* $\mathsf{Q}_i x_i$, where $x_1, \ldots, x_n$ is a permutation of $var(\mathcal{F})$ and $\mathsf{Q}_i \in \{\forall, \exists\}$ for $1 \leq i \leq n$. We define the *depth* of variable x_p as $\delta_{\mathcal{F}}(x_p) = p$, and let $q_{\mathcal{F}}(x_p) = \mathsf{Q}_p$. The sets of *existential* and *universal* variables occurring in $\mathcal{F}$ are given by $var_\exists(\mathcal{F}) = \{\, x \in var(\mathcal{F}) : q_{\mathcal{F}}(x) = \exists \,\}$ and $var_\forall(\mathcal{F}) = \{\, x \in var(\mathcal{F}) : q_{\mathcal{F}}(x) = \forall \,\}$, respectively. We define $R_{\mathcal{F}} = \{\, (x, y) : \delta_{\mathcal{F}}(x) < \delta_{\mathcal{F}}(y) \,\}$ and write $R_{\mathcal{F}}(x) = \{\, y \in var(\mathcal{F}) : (x, y) \in R_{\mathcal{F}} \,\}$ and $L_{\mathcal{F}}(x) = \{\, y \in var(\mathcal{F}) : (y, x) \in R_{\mathcal{F}} \,\}$, where $x \in var(\mathcal{F})$. A *proto-dependency scheme* is a mapping D that associates each QCNF formula $\mathcal{F}$ with a binary relation $D_{\mathcal{F}} \subseteq R_{\mathcal{F}}$. The *trivial dependency scheme* D^{trv} maps each QCNF formula $\mathcal{F}$ to the relation $D^{\text{trv}}_{\mathcal{F}} = \{\, (x, y) \in R_{\mathcal{F}} : q_{\mathcal{F}}(x) \neq q_{\mathcal{F}}(y) \,\}$.

3 Q(D)-Resolution

Definition 1 (**$Q(D)$-resolution derivation).** *Let $\mathcal{F}$ be a QCNF formula with matrix F and let D be a proto-dependency scheme. A (tree-like)* $Q(D)$-resolution derivation *from $\mathcal{F}$ is a pair (T, λ), where $T = (V(T), E(T))$ is a rooted binary tree and λ is a mapping that associates each $t \in V(T)$ with a non-tautological clause C_t and satisfies the following conditions.*

1. *If t is a leaf of T then $C_t \in F$.*
2. *If t has distinct children t_1 and t_2, there is an existential variable $e \in var_\exists(\mathcal{F})$ such that $C_t = (C_{t_1} \cup C_{t_2}) \setminus \{e, \neg e\}$ and $\{e, \neg e\} \subseteq C_{t_1} \cup C_{t_2}$.*
3. *If t has a single child t', there is a literal ℓ with $var(\ell) \in var_\forall(\mathcal{F})$ such that $\ell \in C_{t'}$ and $C_t = C_{t'} \setminus \{\ell\}$. Moreover, there is no existential variable $e \in var(C_t)$ such that $(var(\ell), e) \in D_{\mathcal{F}}$.*

We say π derives *C_r, where r denotes the root of T. The derivation π is a* $Q(D)$-resolution refutation *of $\mathcal{F}$ if it derives the empty clause.*

Ordinary (tree-like) Q-resolution [2] corresponds to $Q(D^{\text{trv}})$-resolution (more precisely, these systems polynomially simulate each other).

Let D be a proto-dependency scheme, and let $\pi = (T, \lambda)$ be a $Q(D)$-resolution derivation from a QCNF formula $\mathcal{F}$. By the *height* of π we mean the height of T, while the *size* of π is the number of vertices in T. For $t \in T$, let T_t denote the subtree of T rooted at t, and λ_t the restriction of λ to T_t. Then $\pi_t = (T_t, \lambda_t)$ is a $Q(D)$-resolution derivation. We call π_t a *subderivation* of π.

We implicitly associate with π a function $\rho_\pi : E(T) \to \mathit{lit}(\mathcal{F})$ that maps an edge (t, t') of T to the (unique) literal in $\lambda(t') \setminus \lambda(t)$, where t is a child of t'. Let $e \in E(T)$ and $v = \mathit{var}(\rho_\pi(e))$. If v is an existential variable of $\mathcal{F}$ then e is a *resolution step*, otherwise e is a *universal reduction step*. In either case we say e is a *derivation step* and that e *reduces* v. The set of variables reduced by some resolution step of π is denoted $\mathit{resolved}(\pi)$. We say π is *ordered* if the sequence of variables reduced on any branch of T is in the order of $R_\mathcal{F}$. That is, for any pair $(e_1, e_2) \in E(T) \times E(T)$ such that there is a path from the root of T to an endpoint of e_2 passing through e_1 we have $(x, y) \in R_\mathcal{F}$, where x is the variable reduced by e_1 and y is the variable reduced by e_2.

Lemma 2. *Let D be a proto-dependency scheme and $\mathcal{F}$ a QCNF formula. Any ordered $Q(D)$-resolution refutation of $\mathcal{F}$ is a $Q(D^{\text{trv}})$-resolution refutation of $\mathcal{F}$.*

We now describe two operations on $Q(D)$-resolution derivations (detailed definitions are given in the appendix). Let D be a proto-dependency scheme and let $\mathcal{F}$ be a QCNF formula. Let $\pi = (T, \lambda)$ and $\pi' = (T', \lambda')$ be $Q(D)$-resolution derivations from $\mathcal{F}$ deriving the clauses C and C', respectively. If u is a universal variable and $u \in \mathit{var}(C)$ then $\mathit{red}(\pi, u)$ is obtained by attaching a new universal reduction step to the root of π that reduces u. The resulting derivation is a $Q(D)$-resolution derivation that derives $C \setminus \{u, \neg u\}$ provided there is no existential variable $e \in \mathit{var}(C)$ such that $(u, e) \in D_\mathcal{F}$.

If $\{e, \neg e\} \subseteq C \cup C'$ for some $e \in \mathit{var}_\exists(\mathcal{F})$, and $(C \cup C') \setminus \{e, \neg e\}$ is non-tautological then $\mathit{res}(\pi, \pi', e)$ is constructed by attaching a resolution step to the roots of π and π' that reduces e. The result is a $Q(D)$-resolution derivation of the clause $(C \cup C') \setminus \{e, \neg e\}$.

We say π and π' are *equivalent* and write $\pi \equiv \pi'$ if there is a bijection $f : V(T) \to V(T')$ such that f is an isomorphism of T and T', and $\lambda(t) = \lambda'(f(t))$ for each $t \in T$.

4 Standard Dependencies

Definition 3 (Connected). *Let $\mathcal{F}$ be a QCNF formula with matrix F and $X \subseteq \mathit{var}(\mathcal{F})$. An X-*path *between clauses $C, C' \in F$ is a sequence $C_1, \ldots, C_k$ of clauses in F with $C_1 = C$ and $C' = C_k$ such that $\mathit{var}(C_i) \cap \mathit{var}(C_{i+1}) \cap X \neq \emptyset$ for all $1 \leq i < k$. The clauses $C, C' \in F$ are* connected *with respect to X if there is an X-path between them. We extend this to variables and say $x, y \in \mathit{var}(\mathcal{F})$ are* connected *with respect to X if there are clauses $C, C' \in F$ such that C and C' are connected with respect to X and $x \in \mathit{var}(C)$, $y \in \mathit{var}(C')$.*

Definition 4 (Standard Dependency Scheme [7]). *Let $\mathcal{F}$ be a QCNF formula with matrix F and let $x, y \in \mathit{var}(\mathcal{F})$ such that $q_\mathcal{F}(x) \neq q_\mathcal{F}(y)$. We call $\{x, y\}$ a dependency pair with respect to $X \subseteq \mathit{var}(\mathcal{F})$ if and x and y are connected with respect to X. The*

standard dependency scheme D^{std} *associates each QCNF formula* $\mathcal{F}$ *with the relation* $D^{\text{std}}_{\mathcal{F}} = \{\, (x, y) \in R_{\mathcal{F}} : \{x, y\}$ *is a dependency pair with respect to* $R_{\mathcal{F}}(x) \cap var_{\exists}(\mathcal{F}) \,\}$.

Observe that the mapping D^{std} is a proto-dependency scheme. The next statement is a slightly modified version of a result from [9].

Lemma 5. *Let* D *be a proto-dependency scheme and let* $\mathcal{F}$ *be a QCNF formula with matrix* F*. Let* $x, y \in var(\mathcal{F})$ *be distinct variables. Let* $\pi = (T, \lambda)$ *be a* $Q(D)$*-resolution derivation from* $\mathcal{F}$*, and let* C *be the clause derived by* π*. If* $x, y \in var(C)$ *then* x *and* y *are connected with respect to* $resolved(\pi)$.

Lemma 5 allows us to establish the following corollary, which is used in the proof of Lemma 8 below to rule out the existence of universal reduction steps of a certain form.

Corollary 6. *Let* D *be a proto-dependency scheme. Let* $\mathcal{F}$ *be a QCNF formula and let* $x, y \in var(\mathcal{F})$ *such that* $(x, y) \in R_{\mathcal{F}}$ *and* $q_{\mathcal{F}}(x) \neq q_{\mathcal{F}}(y)$*. Let* $\pi = (T, \lambda)$ *be a* $Q(D)$*-resolution derivation from* $\mathcal{F}$ *with* $x, y \in var(C)$*, where* C *is the clause derived by* π*. If* $resolved(\pi) \subseteq R_{\mathcal{F}}(x)$ *then* $(x, y) \in D^{\text{std}}_{\mathcal{F}}$.

5 Reordering $Q(D^{\text{std}})$-resolution refutations

The next result shows that universal reduction steps can be inserted into an ordered $Q(D)$-resolution derivation to obtain another ordered $Q(D)$-resolution derivation. This statement holds for arbitrary proto-dependency schemes D.

Proposition 7. *Let* D *be a proto-dependency scheme. Let* $\mathcal{F}$ *be a QCNF formula with* $u \in var_{\forall}(\mathcal{F})$*, and let* $\pi \equiv red(\pi', u)$ *be a* $Q(D)$*-resolution derivation from* $\mathcal{F}$ *such that* π' *is ordered. Given* π*, one can compute an ordered* $Q(D)$*-resolution derivation from* $\mathcal{F}$ *that derives the same clause as* π.

Lemma 8. *Let* $\mathcal{F}$ *be a QCNF formula and let* $e \in var_{\exists}(\mathcal{F})$*. Let* $\pi \equiv res(\pi_1, \pi_2, e)$ *be a* $Q(D^{\text{std}})$*-resolution derivation of a clause* C *from* $\mathcal{F}$ *such that* π_1 *and* π_2 *are ordered. Given* π*, one can compute an ordered* $Q(D^{\text{std}})$*-resolution derivation from* $\mathcal{F}$ *that derives a clause* $C' \subseteq C$.

Proposition 7 and Lemma 8 are instrumental in proving our main result, as stated below. Recall that $Q(D^{\text{trv}})$-resolution is equivalent to ordinary Q-resolution.

Theorem 9. *Given a* $Q(D^{\text{std}})$*-resolution refutation of a QCNF formula* $\mathcal{F}$ *one can compute a* $Q(D^{\text{trv}})$*-resolution refutation of* $\mathcal{F}$.

6 Conclusion

A $Q(D^{\text{trv}})$-resolution refutation computed from a $Q(D^{\text{std}})$-resolution refutation using the algorithm of Theorem 9 can be of size exponential in the size of the input refutation. It is an open question whether this is an artifact of our approach or whether there is in fact an exponential separation between $Q(D^{\text{std}})$-resolution and $Q(D^{\text{trv}})$-resolution. The resolution-path dependency scheme [8, 9] is a strict generalization of D^{std}. We believe that an analogue of Theorem 9 can be proved for a slightly restricted version of the resolution-path dependency scheme.

References

1. Armin Biere and Florian Lonsing. Integrating dependency schemes in search-based QBF solvers. In Ofer Strichman and Stefan Szeider, editors, *Theory and Applications of Satisfiability Testing - SAT 2010*, volume 6175 of *LNCS*, pages 158–171. Springer, 2010.
2. H. Kleine Büning, M. Karpinski, and A. Flögel. Resolution for quantified Boolean formulas. *Information and Computation*, 117(1):12–18, 1995.
3. Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Clause/term resolution and learning in the evaluation of quantified Boolean formulas. *J. Artif. Intell. Res. (JAIR)*, 26:371–416, 2006.
4. Florian Lonsing. *Dependency Schemes and Search-Based QBF Solving: Theory and Practice*. PhD thesis, Johannes Kepler University, Linz, Austria, April 2012.
5. Aina Niemetz, Mathias Preiner, Florian Lonsing, Martina Seidl, and Armin Biere. Resolution-based certificate extraction for QBF. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing SAT 2012*, volume 7317 of *Lecture Notes in Computer Science*, pages 430–435. Springer Berlin Heidelberg, 2012.
6. Marko Samer. Variable dependencies of quantified CSPs. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *Proceedings of LPAR 2008, 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, November 22-27, 2008 Doha, Qatar*, volume 5330 of *LNCS*, pages 512–527. Springer, 2008.
7. Marko Samer and Stefan Szeider. Backdoor sets of quantified Boolean formulas. *Journal of Automated Reasoning*, 42(1):77–97, 2009.
8. Friedrich Slivovsky and Stefan Szeider. Computing resolution-path dependencies in linear time. In *Theory and Applications of Satisfiability Testing - SAT 2012*, volume 7317 of *Lecture Notes in Computer Science*, pages 58–71. Springer, 2012.
9. Allen Van Gelder. Variable independence and resolution paths for quantified boolean formulas. In Jimmy Lee, editor, *Principles and Practice of Constraint Programming - CP 2011*, volume 6876 of *Lecture Notes in Computer Science*, pages 789–803. Springer Verlag, 2011.

Appendix

Proof of Lemma 2

Proof. Let $\pi = (T, \lambda)$ be an ordered $Q(D)$-refutation of $\mathcal{F}$. Towards a contradiction assume that there is a universal reduction step $(t, t') \in E(T)$ reducing the variable $u \in var_\forall(\mathcal{F})$ such that $var(\lambda(t')) \cap var_\exists(\mathcal{F}) \cap R_\mathcal{F}(u) \neq \emptyset$. Let $e \in var(\lambda(t')) \cap var_\exists(\mathcal{F}) \cap R_\mathcal{F}(u)$. Since π is a refutation, on the path from the root of T to t' there has to be a resolution step $(s, s') \in E(T)$ that reduces e. But $(\rho_\pi((s, s')), \rho_\pi((t, t'))) = (e, u)$ and $(e, u) \notin R_\mathcal{F}$, so π is not ordered.

Definitions of $res(\pi, \pi', e)$ and $red(\pi, u)$

Let D be a proto-dependency scheme and let $\mathcal{F}$ be a QCNF formula. Let $\pi = (T, \lambda)$ and $\pi' = (T', \lambda')$ be $Q(D)$-resolution derivations from $\mathcal{F}$ deriving the clauses C and C', respectively. If u is a universal variable and $u \in var(C)$ then $red(\pi, u)$ is defined as follows. We let $red(\pi, u) = (T'', \lambda'')$, where $V(T'') = V(T) \cup \{r''\}$ for some $r'' \notin V(T)$, $E(T'') = E(T) \cup \{(r'', r)\}$, and r denotes the root of T. The mapping $\lambda'' : V(T'') \to 2^{lit(\mathcal{F})}$ is given by

$$\lambda''(t) = \begin{cases} C \setminus \{u, \neg u\} & \text{if } t = r''; \\ \lambda(t) & \text{otherwise.} \end{cases}$$

If $\{e, \neg e\} \subseteq C \cup C'$ for some $e \in var_\exists(\mathcal{F})$, and $(C \cup C') \setminus \{e, \neg e\}$ is non-tautological then $res(\pi, \pi', e) = (T'', \lambda'')$ is defined as follows. We let $V(T'') = V(T) \cup V(T') \cup \{r''\}$ and $E(T'') = E(T) \cup E(T') \cup \{(r'', r), (r'', r')\}$, where $r'' \notin V(T) \cup V(T')$ and r, r' denote the roots of T, T', respectively. (Without loss of generality, we will assume that T and T' are vertex-disjoint.) The mapping $\lambda : V(T'') \to 2^{lit(\mathcal{F})}$ is given by

$$\lambda(t) = \begin{cases} (C_1 \cup C_2) \setminus \{e, \neg e\} & \text{if } t = r''; \\ \lambda_1(t) & \text{if } t \in V(T); \\ \lambda_2(t) & \text{otherwise.} \end{cases}$$

Proof of Lemma 5

Proof. By induction on the height h of π. If $h = 0$ then $C \in F$ and $\{x, y\}$ is a dependency pair with respect to the empty set. Suppose the statement of the lemma holds for derivations of height up to k and let $h = k + 1$. Suppose $\pi \equiv red(\pi', u)$ for a suitable derivation π' and a universal variable u. Then $x, y \in var(C')$, where C' is the clause derived by π'. Hence $\{x, y\}$ is a dependency pair with respect to $resolved(\pi') = resolved(\pi)$ by induction hypothesis. Suppose $\pi \equiv res(\pi_1, \pi_2, e)$ for suitable derivations π_1, π_2 and an existential variable e. Let C_1 and C_2 be the clauses

derived by π_1 and π_2, respectively. If $x, y \in var(C_1)$ or $x, y \in var(C_2)$ the statement again follows by induction hypothesis. So assume, without loss of generality, that $x \in var(C_1)$ and $y \in var(C_2)$. Note that $e \in var(C_1) \cap var(C_2)$ and that both π_1 and π_2 have height at most k. By induction hypothesis $\{x, e\}$ is a dependency pair with respect to $resolved(\pi_1)$, and $\{y, e\}$ is a dependency pair with respect to $resolved(\pi_2)$. Let $C'_1, \ldots, C'_{k_1}$ be a $resolved(\pi_1)$-path with $x \in C'_1$ and $e \in C'_{k_1}$, and let $C''_1, \ldots, C''_{k_2}$ be a $resolved(\pi_2)$-path with $e \in C''_1$ and $y \in C''_{k_2}$. The sequence $C'_1, \ldots, C_{k'_1}, C''_1, \ldots, C''_{k_2}$ is an X-path where $X = resolved(\pi_1) \cup resolved(\pi_2) \cup \{e\} = resolved(\pi)$, and so x and y are connected with respect to $resolved(\pi)$.

Proof of Proposition 7

Proof. We prove a slightly stronger statement saying that the derivation to be computed has height bounded by the height of π. The proof is by induction on the height h of π. The base case $h = 1$ is immediate. Assume the statement holds for derivations up to height k and let $h = k + 1$. Suppose $\pi' \equiv red(\pi_1, v)$ for suitable π_1 and $v \in var_\forall(\mathcal{F})$, and let C' be the clause derived by π'. If $v \in R_\mathcal{F}(u)$ then π is already ordered, so suppose $v \in L_\mathcal{F}(u)$. Let C_1 be the clause derived by π_1. We must have $u, v \in var(C_1)$, so $\pi_2 = red(\pi_1, u)$ is a $Q(D)$-resolution derivation of the clause $C_2 = C_1 \setminus \{u, \neg u\}$. The height of π_2 is at most k, so by induction hypothesis one can compute an ordered derivation π_3 of height at most k that derives C_2. Moreover, $v \in var(C_2)$ so $red(\pi_3, v)$ is an ordered $Q(D)$-resolution derivation of the clause $C_3 = C_2 \setminus \{v, \neg v\}$ with height at most h. Note that $C = (C_1 \setminus \{v, \neg v\}) \setminus \{u, \neg u\} = (C_1 \setminus \{u, \neg u\}) \setminus \{v, \neg v\} = C_3$. Suppose $\pi' \equiv res(\pi_1, \pi_2, e)$ where π_1 and π_2 derive C_1 and C_2, respectively, and $e \in var_\exists(\mathcal{F})$. For $i \in \{1, 2\}$, let $\pi'_i = red(\pi_i, u)$ if $u \in var(C_i)$ and $\pi'_i = \pi_i$ otherwise. Both π'_1 and π'_2 are $Q(D)$-resolution derivations of height at most k. Moreover, π_1 and π_2 are ordered because π' is. For $i \in \{1, 2\}$, let π''_i be a derivation given by applying the induction hypothesis to π'_i. It is straightforward to check that $\pi^* = res(\pi''_1, \pi''_2, e)$ is an ordered $Q(D)$-resolution derivation of C and that π^* has height at most h.

Proof of Lemma 8

Proof. The proof is by induction on the size s of π. The base case where π_1 and π_2 are of size 1 is immediate. Assume the statement of the lemma holds for derivations of size less than $k > 3$ and let $s = k$. Without loss of generality, we make the following assumptions.

1. If the sizes of π_1 and π_2 are both greater than 1, and y and y' are the variables reduced by the final reduction steps of π_1 and π_2, respectively, then $y \in L_\mathcal{F}(y') \cup \{y'\}$.
2. The size of π_1 is greater than 1.

Let y be the variable reduced by the final reduction step of π_1 (y is well defined by assumption 2). If $e \in L_\mathcal{F}(y)$ the derivation π is already ordered, so let $e \in R_\mathcal{F}(y)$. Suppose $y \in var_\forall(\mathcal{F})$. Then $\pi_1 \equiv red(\pi'_1, y)$ for a suitable π'_1 deriving a clause C'_1.

Note that $y, e \in var(C_1')$. Because π_1 is ordered and y is reduced in its final reduction step we must have $resolved(\pi_1') \subseteq R_{\mathcal{F}}(y)$. By Corollary 6 we get $(y, e) \in D^{\text{std}}_{\mathcal{F}}$, which contradicts π's being a $Q(D^{\text{std}})$-resolution derivation.

Thus $y \in var_{\exists}(\mathcal{F})$ and $\pi_1 \equiv res(\pi_1', \pi_1'', y)$ for suitable derivations π_1' and π_1''. Suppose π_1', π_1'', and π_2 derive the clauses C_1', C_1'', and C_2, respectively. We distinguish three cases.

1. Suppose $y \in var(C_2)$ and let $\ell_y \in C_2$ such that $var(\ell_y) = y$. Either C_1' or C_1'' must contain ℓ_y. Without loss of generality assume $\ell_y \in C_1'$. Since the final resolution step of π reduces e, there is an $\ell_e \in C_2$ such that $var(\ell_e) = e$, and $\overline{\ell_e} \in C_1' \cup C_1''$. If $\overline{\ell_e} \in C_1'$ we let $\pi^* = res(\pi_1', \pi_2, e)$. It is straightforward to verify that π^* is a $Q(D^{\text{std}})$-resolution derivation of size less than k that derives a clause $C' \subseteq C$. Moreover, both π_1' and π_2 are ordered. Applying the induction hypothesis, we obtain an ordered $Q(D^{\text{std}})$-resolution derivation of a clause $C'' \subseteq C' \subseteq C$. If $\overline{\ell_e} \notin C_1'$ then $C_1' \subseteq C$ and π_1' yields the desired derivation.
2. Suppose $y \notin var(C_2)$ and $e \in var(C_1') \cap var(C_1'')$. Then $\pi_a = res(\pi_1', \pi_2, e)$ and $\pi_b = res(\pi_1'', \pi_2, e)$ are well defined $Q(D^{\text{std}})$-resolution derivations of clauses C_a and C_b, respectively, and $res(\pi_a, \pi_b, y)$ is a $Q(D^{\text{std}})$-resolution derivation of C. Both π_a and π_b have size less than k, and π_1', π_1'', and π_2 are ordered. By induction hypothesis, we can compute ordered $Q(D^{\text{std}})$-resolution derivations π_a' and π_b' such that π_a' derives $C_a' \subseteq C_a$ and π_b' derives $C_b' \subseteq C_b$. If $y \notin var(C_a')$ or $y \notin var(C_b')$ then $C_a' \subseteq C$ or $C_b' \subseteq C$, and so π_a' or π_b' provides the desired derivation. Otherwise, we let $\pi^* = res(\pi_a', \pi_b', y)$. The derivation π^* is ordered by assumption 1 in combination with the fact that both π_a' and π_b' are ordered. Moreover, π^* derives a clause $C' \subseteq C$.
3. Suppose $y \notin var(C_2)$ and $e \notin var(C_1') \cap var(C_1'')$. Without loss of generality assume $e \in var(C_1') \setminus var(C_1'')$. Then $\pi_a = res(\pi_1', \pi_2, e)$ is a $Q(D^{\text{std}})$-resolution derivation of a clause C_a, and $res(\pi_a, \pi_1'', y)$ is a $Q(D^{\text{std}})$-resolution derivation of C. The derivation π_a has size less than k and both π_1' and π_2 are ordered. By induction hypothesis, we can compute an ordered $Q(D^{\text{std}})$-resolution derivation π_a' of a clause $C_a' \subseteq C_a$. If $y \notin var(C_a')$ then $C_a' \subseteq C$ and π_a' yields the desired derivation. Otherwise, let $\pi^* = res(\pi_a', \pi_1'', y)$. The derivation π^* is $Q(D^{\text{std}})$-resolution derivation of a clause $C' \subseteq C$, and it is ordered by assumption 1.

Proof of Theorem 9

Proof. Let π be a $Q(D^{\text{std}})$-resolution refutation of $\mathcal{F}$. We claim that we can compute an ordered $Q(D^{\text{std}})$-resolution refutation π' of $\mathcal{F}$. By Lemma 2 the derivation π' is a $Q(D^{\text{trv}})$-resolution refutation of $\mathcal{F}$. We prove the claim by induction on the number of unordered subderivations of π. If π contains no unordered subderivations, then in particular π itself is ordered. Otherwise, let π_u be an unordered subderivation of π such that π_u contains no unordered subderivations, and let C be the clause derived by π_u. We can apply either Proposition 7 or Lemma 8 to obtain an ordered $Q(D^{\text{std}})$-resolution derivation π_o that derives a clause $C' \subseteq C$. We construct a new $Q(D^{\text{std}})$-resolution refutation π' of $\mathcal{F}$ by replacing π_u with π_o in π and (possibly) removing redundant derivation steps. The number of unordered subderivations of π' is strictly smaller than the number of unordered subderivations of π, so we can apply the induction hypothesis to get an ordered $Q(D^{\text{std}})$-resolution refutation of $\mathcal{F}$.

Extending DPLL-Based QBF Solvers to Handle Free Variables

William Klieber[1], Mikoláš Janota[2], Joao Marques-Silva[2,3], and Edmund Clarke[1]

[1] Carnegie Mellon University, Pittsburgh, PA, USA
[2] IST/INESC-ID, Lisbon, Portugal
[3] University College Dublin, Ireland

1 Introduction

In a number of interesting applications (such as automatic synthesis of a reactive system from a formal specification [1]), one needs to consider *open* formulas, i.e., formulas with free (unquantified) variables. A solution to such a QBF is a formula equivalent to the given one but containing no quantifiers and using only those variables that appear free in the given formula. For example, a solution to the open QBF formula $(\exists x.\ (x \wedge y) \vee z)$ is the formula $y \vee z$.

In [10], it was shown how clause/cube learning for DPLL-based CDCL QBF solvers can be reformulated in terms of *sequents* and extended to non-CNF, non-prenex formulas. This technique uses *ghost variables* to handle non-CNF formulas in a manner that is symmetric between the existential and universal quantifiers. We show that this sequent-based technique can be naturally extended to handle QBFs with free variables.

A naïve way to recursively solve an open QBF Φ is shown in Figure 1. Roughly, we Shannon-expand on the free variables until we're left with only closed-QBF problems, which are then handed to a closed-QBF solver. If the free variables are always branched on in the same order, then the algorithm effectively builds an ordered BDD [6], assuming that the `ite` function is memoized and performs appropriate simplification. The naïve algorithm suffers from many inefficiencies. In terms of branching behavior, it is similar to the DPLL algorithm, but it lacks non-chronological bracktracking and an equivalent of clause learning. The main contribution of this extended abstract is to show how an existing closed-QBF algorithm can be modified to directly consider formulas with free variables. A more detailed treatment will be published at CP 2013.

```
function solve(Φ) {
    if (Φ has no free variables) {return closed_qbf_solve(Φ);}
    x := (a free variable in Φ);
    return ite(x, solve(Φ with x substituted with True),
                  solve(Φ with x substituted with False));
}
```

Fig. 1. Naive algorithm. The notation "`ite(x,` ϕ_1`,` ϕ_2`)`" denotes a formula with an *if-then-else* construct that is logically equivalent to $(x \wedge \phi_1) \vee (\neg x \wedge \phi_2)$.

In related work, various approaches [11,5,7] have been proposed for open QBFs of the form $\exists X.\,\phi$. For QBFs with arbitrary quantifier prefixes, the only other work of which we are aware is that of Becker, Ehlers, Lewis, and Marin [1], which uses computational learning to generate CNF, DNF, or CDNF formulas, and that of Benedetti and Mangassarian [3], which adapts sKizzo [2] for open QBF. The use of SAT solvers to build BDDs [13,9] has also been investigated.

1.1 Preliminaries

Input Formulas. We consider prenex formulas of the form $(Q_1X_1...Q_nX_n.\,\phi)$, where $Q_i \in \{\exists, \forall\}$ and ϕ is quantifier-free and represented as a DAG. The logical connectives allowed in ϕ are conjunction, disjunction, and negation.

Quantifier Order. In a formula such as $\forall x.\exists y.\,\phi$, where the quantifier of y occurs inside the scope of the quantifier of x, we say that y is **downstream** of x. All quantified variables in are considered downstream of all free variables.

Substitution. Given a partial assignment π, we define "$\Phi|\pi$" to be the result of the following: For every assigned variable x, we replace all occurrences of x in Φ with the assigned value of x (and delete the quantifier of x, if any).

2 Theory

We employ *ghost variables* to provide a modification of the Tseitin transformation that is symmetric between the two quantifer types. The idea of using a symmetric transformation was first explored in [14]. Similar ideas have been used to handle non-prenex formulas in [10] and "don't care" propagation in [8].

Where the Tseitin transform would introduce a Tseitin variable g to represent a subformula, we instead introduce two *ghost variables*: an existential variable $g^{\exists}$ and a universal variable $g^{\forall}$. Two new innermost quantification blocks are added to the formula for the ghost variables. For example, if the original quantifier prefix was $\exists e\,\forall u$, then the new prefix would be $\exists e\,\forall u\,\exists \boldsymbol{g}^{\exists}\,\forall \boldsymbol{g}^{\forall}$.

Definition 1 (Consistent assignment to ghost variable). Given a quantifier type $Q \in \{\exists, \forall\}$ and an assignment π, we say that a ghost literal g^Q is assigned **consistently** under π iff $g^Q|\pi =$ (the formula represented by $g^Q)|\pi$.

To define the semantics of QBF with ghost variables, we use a three-valued logic with a third value dontcare. We call it "don't care" because we need to deal with assignments with inconsistently assigned ghost variables, but we don't really care about such assignments. In our three-valued logic, a conjunction (or respectively disjunction) of boolean values evaluates to false (resp. true) if any conjunct is false (resp. if any disjunct is true), and otherwise it evaluates to dontcare if any conjunct (resp. disjunct) is dontcare. The negation of dontcare is dontcare.

Definition 2 (Semantics with Ghost Vars). Given an assignment π to all non-ghost variables and a subset of the ghost variables, we define $[\![\Phi]\!]_\pi$ as follows:

$$[\![\Phi]\!]_\pi := \begin{cases} \Phi|\pi & \text{if every ghost variable in } \pi \text{ is assigned consistently} \\ \mathsf{dontcare} & \text{otherwise} \end{cases}$$

For convenience in defining $[\![\Phi]\!]_\pi$ for a partial assignment π, we assume that the formula is prepended with a dummy "quantifier" block for free variables. For example, $(\exists e.\, e \wedge z)$ becomes $(\mathcal{F}z.\, \exists e.\, e \wedge z)$, where $\mathcal{F}$ denotes the dummy block for free variables. We define $[\![\Phi]\!]_\pi$ for a partial assignment π as follows:

$$
\begin{aligned}
[\![Qx.\, \Phi]\!]_\pi &= [\![\Phi]\!]_\pi \quad \text{if } x \in \text{vars}(\pi) \\
[\![\exists x.\, \Phi]\!]_\pi &= [\![\Phi]\!](\pi \cup \{x\}) \vee [\![\Phi]\!](\pi \cup \{\neg x\}) \quad \text{if } x \notin \text{vars}(\pi) \\
[\![\forall x.\, \Phi]\!]_\pi &= [\![\Phi]\!](\pi \cup \{x\}) \wedge [\![\Phi]\!](\pi \cup \{\neg x\}) \quad \text{if } x \notin \text{vars}(\pi) \\
[\![\mathcal{F}x.\, \Phi]\!]_\pi &= x \;?\; [\![\Phi]\!](\pi \cup \{x\}) \;:\; [\![\Phi]\!](\pi \cup \{\neg x\}) \quad \text{if } x \notin \text{vars}(\pi)
\end{aligned}
$$

The notation "$x \;?\; \phi_1 : \phi_2$" denotes a formula with an *if-then-else* construct that is logically equivalent to $(x \wedge \phi_1) \vee (\neg x \wedge \phi_2)$. Note that if Φ contains free variables unassigned by π, then $[\![\Phi]\!]_\pi$ is a formula in terms of these free variables.

Definition 3 (Sometimes-Dontcare). A formula ϕ is said to be **sometimes-dontcare** iff there is an assignment π under which ϕ evaluates to dontcare.

Definition 4 (Game-State Specifier, Match). A **game-state specifier** is a pair $\langle L^{\text{now}}, L^{\text{fut}} \rangle$ consisting of two sets of literals, L^{now} and L^{fut}. We say that $\langle L^{\text{now}}, L^{\text{fut}} \rangle$ **matches** an assignment π iff:

1. for every literal ℓ in L^{now}, $\ell|_\pi = \mathsf{true}$, and
2. for every literal ℓ in L^{fut}, either $\ell|_\pi = \mathsf{true}$ or $\ell \notin \text{vars}(\pi)$.

For example, $\langle \{e\}, \{u\} \rangle$ matches the assignments $\{e\}$ and $\{e, u\}$, but does not match $\{\}$ or $\{e, \neg u\}$. Note that, for any literal ℓ, if $\{\ell, \neg\ell\} \subseteq L^{\text{fut}}$, then $\langle L^{\text{now}}, L^{\text{fut}} \rangle$ matches an assignment π only if π doesn't assign ℓ.

Definition 5 (Sequent). The sequent "$\langle L^{\text{now}}, L^{\text{fut}} \rangle \models \Phi \Leftrightarrow \psi$" means "for all assignments π that match $\langle L^{\text{now}}, L^{\text{fut}} \rangle$, if $[\![\Phi]\!]_\pi$ is not sometimes-dontcare, then $[\![\Phi]\!]_\pi$ is logically equivalent to $\psi|_\pi$".

3 Algorithm

The top-level algorithm is based on the well-known DPLL algorithm, except that sequents are used instead of clauses. Similar to how SAT solvers maintain a *clause database*, our solver maintains a *sequent database*. A SAT solver's clause database is initialized to contain exactly the set of clauses produced by the Tseitin transformation of the input formula Φ_{in} into CNF. Each clause in the Tseitin transformation corresponds to two sequents in our initial sequent database (one with existential ghost variables, and one with universal). Literals are placed in L^{fut} if allowed by Proposition 2 of [10]; otherwise, in L^{now}.

When the current assignment matches a sequent in the database, the solver performs learning to add a new sequent of the form $\langle L^{\text{now}}, L^{\text{fut}} \rangle \models (\Phi_{\text{in}} \Leftrightarrow \psi)$ to the database. If L^{now} is empty, then the solver returns ψ as the final answer. Otherwise, it backtracks to the earliest decision level at which the newly learned sequent will trigger a forced literal in propagation.

Propagation. Propagation is similar to that of closed-QBF solvers. Consider a sequent $\langle L^{\text{now}}, L^{\text{fut}} \rangle \models (\Phi_{\text{in}} \Leftrightarrow \psi)$ in the sequent database. If, under π_{cur},

1. there is exactly one unassigned literal ℓ in L^{now}, and
2. no literals in $L^{\text{now}} \cup L^{\text{fut}}$ are assigned false, and
3. ℓ is not downstream of any unassigned literals in L^{fut},

then $\neg\ell$ is *forced* — it is added to the current assignment π_{cur}. Propagation ensures that the solver never re-explores areas of the search space for which it already knows the answer, ensuring progress and eventual termination.

Learning. The solver performs *learning* after the current assignment π_{cur} matches a sequent in the database. The learning procedure is based on the clause learning introduced for SAT in [12] and adapted for QBF in [15] and adapted for sequents in [10]. We use resolution-like inference rules to infer new sequents by 'resolving' two sequents: (1) a sequent containing a forced literal r and (2) the sequent that forced r (i.e., the *antecedent* of r). The rules for resolving on free variables and existential variables are shown below; the rule for universal variables is similar.

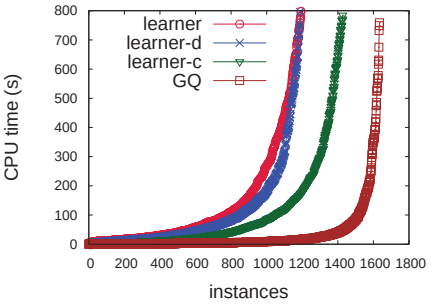
Resolving on a literal r that is existentially quantified:

$$\frac{\begin{array}{l}\langle L_1^{\text{now}} \cup \{r\},\, L_1^{\text{fut}}\rangle \models (\Phi_{\text{in}} \Leftrightarrow \mathsf{false}) \\ \langle L_2^{\text{now}} \cup \{\neg r\},\, L_2^{\text{fut}}\rangle \models (\Phi_{\text{in}} \Leftrightarrow \psi) \\ r \text{ is not downstream of any } \ell \text{ such that } \ell \in L_1^{\text{fut}} \text{ and } \neg\ell \in (L_1^{\text{fut}} \cup L_2^{\text{fut}})\end{array}}{\langle L_1^{\text{now}} \cup L_2^{\text{now}},\, L_1^{\text{fut}} \cup L_2^{\text{fut}} \cup \{\neg r\}\rangle \models (\Phi_{\text{in}} \Leftrightarrow \psi)}$$

Resolving on a literal r that is free in Φ_{in}:

$$\frac{\begin{array}{l}\langle L_1^{\text{now}} \cup \{r\},\, L_1^{\text{fut}}\rangle \models (\Phi_{\text{in}} \Leftrightarrow \psi_1) \\ \langle L_2^{\text{now}} \cup \{\neg r\},\, L_2^{\text{fut}}\rangle \models (\Phi_{\text{in}} \Leftrightarrow \psi_2)\end{array}}{\langle L_1^{\text{now}} \cup L_2^{\text{now}},\, L_1^{\text{fut}} \cup L_2^{\text{fut}} \cup \{r, \neg r\}\rangle \models (\Phi_{\text{in}} \Leftrightarrow (r\ ?\ \psi_1 : \psi_2))}$$

4 Experimental Results



We extended the existing closed-QBF solver GhostQ [10] to implement the techniques described in this paper. For comparison, we used the solvers and load-balancer benchmarks from [1]. With a time limit of 800 seconds, GhostQ solved 189 instances that none of the three solvers from [1] solved, whereas there were 18 instances solved by at least one solver from [1] but not by GhostQ. The graph to the right shows the number of non-trivial instances solved by GhostQ (about 1600) and the three "learner" solvers (about 1400 and 1200) from [1]. As future work, it may be useful to adapt closed-QBF preprocessing techniques such as blocked-clause elimination [4] to open QBF.

References

1. B. Becker, R. Ehlers, M. D. T. Lewis, and P. Marin. ALLQBF Solving by Computational Learning. In S. Chakraborty and M. Mukund, editors, *ATVA*, volume 7561 of *Lecture Notes in Computer Science*, pages 370–384. Springer, 2012.
2. M. Benedetti. sKizzo: A Suite to Evaluate and Certify QBFs. In *CADE*, 2005.
3. M. Benedetti and H. Mangassarian. QBF-Based Formal Verification: Experience and Perspectives. *JSAT*, 2008.
4. A. Biere, F. Lonsing, and M. Seidl. Blocked Clause Elimination for QBF. In *CADE*, 2011.
5. J. Brauer, A. King, and J. Kriener. Existential Quantification as Incremental SAT. In G. Gopalakrishnan and S. Qadeer, editors, *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 191–207. Springer, 2011.
6. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986.
7. E. Goldberg and P. Manolios. Quantifier elimination by Dependency Sequents. In G. Cabodi and S. Singh, editors, *FMCAD*, pages 34–43. IEEE, 2012.
8. A. Goultiaeva and F. Bacchus. Exploiting QBF Duality on a Circuit Representation. In *AAAI*, 2010.
9. J. Huang and A. Darwiche. Using DPLL for Efficient OBDD Construction. In *SAT*, 2004.
10. W. Klieber, S. Sapra, S. Gao, and E. M. Clarke. A Non-prenex, Non-clausal QBF Solver with Game-State Learning. In *SAT*, 2010.
11. K. L. McMillan. Applying SAT Methods in Unbounded Symbolic Model Checking. In E. Brinksma and K. G. Larsen, editors, *CAV*, volume 2404 of *Lecture Notes in Computer Science*, pages 250–264. Springer, 2002.
12. J. P. M. Silva and K. A. Sakallah. GRASP - a new search algorithm for satisfiability. In *ICCAD*, pages 220–227, 1996.
13. R. Wille, G. Fey, and R. Drechsler. Building free binary decision diagrams using SAT solvers. *Facta universitatis-series: Electronics and Energetics*, 20(3):381–394, 2007.
14. L. Zhang. Solving QBF by Combining Conjunctive and Disjunctive Normal Forms. In *AAAI 2006*.
15. L. Zhang and S. Malik. Towards a Symmetric Treatment of Satisfaction and Conflicts in Quantified Boolean Formula Evaluation. In *CP 2002*.

Certificate Extraction from Variable-Elimination QBF Preprocessors

Allen Van Gelder

University of California, Santa Cruz http://www.cse.ucsc.edu/~avg

Abstract. Recent solvers for quantified boolean formulas have the ability to produce a certificate for their conclusion, in the form of a Q-resolution proof that the input QBF is either true of false. However, in many cases the solver does not receive the original QBF, so the produced certificate does not verify the truth value of that original QBF. This paper addresses some aspects of the question of how to extend the certificate produced by the QBF solver for the preprocessed QBF so that it serves as a certificate for the original QBF.

1 Introduction

Solvers for Quantified Boolean Formulas (QBFs) are rapidly increasing in strength, and are now at the point that they are capable of solving significant problems and also producing certificates so that their conclusions can be checked during post-processing. [BJ12,NPL+12].

It is starting to be recognized that simply delivering a 0 or 1 answer is not good enough, for a solver. There has to be some formal proof system to back up claimed answers. More than just verifying a claimed answer, users want additional information relevant to their applications. While this is a very open-ended topic, this short paper concentrates on producing certificates that can verify the conclusions of the combined work of a preprocessor and a solver. More precisely it shows what records may be kept by a variable-elimination preprocessor that are sufficient to transform a satisfying assignment of the preprocessed formula into a satisfying assignment of the original formula.

2 Preliminaries

In general, *quantified boolean formulas* (QBFs) generalize propositional formulas by adding universal and existential quantification of boolean variables. See [KBL99] for a thorough introduction. This paper uses notation and definitions found in recent QBF papers by the same author. *PCNF* denotes a formula in prenex conjunctive normal form. Clauses are enclosed in [ ]. Disjoint union of sets is denoted by "+". Lowercase roman letters denote literals or variables, where the context calls for such. They are existential for letters near the beginning of the alphabet (e.g., d, e, f), universal for letters near the end of the alphabet

(e.g., u, v, w), and of indeterminate quantifier type near the middle of the alphabet (e.g., p, q). Lowercase greek letters, such as α, τ, etc., denote sequences of literals (or sets where order is immaterial). Exceptions are stated where they occur.

Definition 1 An ***assignment*** is a partial function from variables to truth values, and is usually represented as the set of literals that it maps to `true`. (This is sometimes called a *partial assignment*.) A ***total assignment*** assigns a truth value to every variable in the quantifier-free formula. Assignments are denoted by σ, τ, etc.

Applications of an assignment σ to a logical expression are denoted by $q\lceil_\sigma$, $C\lceil_\sigma$, $\mathcal{F}\lceil_\sigma$, etc. If σ assigns variables that are quantified in Ψ, those quantifiers are deleted in $\Psi\lceil_\sigma$, and their variables receive the assignment specified by σ. □

3 Variable-Elimination Preprocessors

We use `bloqqer` [BSL11] as an example of the type of preprocessor from which we wish to draw certificates. Although the *pure-literal rule*, *unit-clause rule*, and *variable-elimination resolution rule* are usually considered to be distinct actions on existential variables, they can all be characterized as the *variable-elimination resolution rule* (VER). The matrix is partitioned into three sets of clauses after an existential literal, say q, is chosen:

(A) clauses containing q,
(B) clauses containing $\overline{q}$,
(C) remaining clauses.

Recall that propositional VER forms all resolvents of a clause in set (A) and a clause in set (B), then discards tautologous resolvents and discards the sets (A) and (B). Finally, for QBF all resolvents are universally reduced.

If the set (B) is empty and (A) is nonempty, q is a pure literal, and the set of resolvents is empty. If the set (A) contains the unit clause $[q]$, this is a case of the unit-clause rule, and the set of resolvents consists of the clauses in set (B) with the literal $\overline{q}$ deleted. If the set (B) contains $[\overline{q}]$, the situation is symmetrical.

We formalize the actions of a variable-elimination (VE) preprocessor as a sequence of rounds, numbered by r for $1 \leq r \leq R$. In each round, one variable is eliminated and records are kept to permit a certificate to be constructed later. At this stage we are not interested in optimizations.

Variable-Elimination Resolution If the operation for round r is VER, record the existential variable q that is eliminated, the set of clauses containing q and the set of clauses containing $\overline{q}$; these two sets are deleted during round r. The non-tautologous resolvents are given fresh clause IDs and added to the formula; these do not need to be recorded in connection with round r but their derivations should be recorded (one resolution each).

The critical situation for generating a certificate is when the formula is `true` and a hitting set needs to be found to seed an "initial-reason cube". The hitting

set (technically consistent hitting set) τ_r must be a partial assignment such that every clause in the formula at the beginning of round r has a true literal. But the formula is a moving target, as it changes every round. So the precondition for extending the hitting set from the end of round r to the beginning of round r is that every clause in the formula at the end of round r is made true by τ_{r+1}. To extend τ_{r+1} to τ_r, all clauses deleted during round r must be made true.

The procedure to accomplish the extension from τ_{r+1} to τ_r in the propositional case is known from antiquity, but possibly not published archivally. However, Dechter and Rish give a closely related procedure [DR94]. Check each clause that was deleted during round r (due to VER) and if one is found that contains q (resp. $\overline{q}$) and all its other literals are false in τ_{r+1}, then add q (resp. $\overline{q}$) to τ_{r+1} giving τ_r. Call this clause C_0. Assume it contains q, as the case for $\overline{q}$ is similar. The precondition in the preceding paragraph ensures that clauses with both q and with $\overline{q}$ cannot be chosen.

If no clause qualifies for C_0 in the preceding paragraph, then choose any clause that is not made true by τ_{r+1} as C_0. Again, assume w.l.o.g. that $q \in C_0$. Now add q and the *negation* of every other literal in C_0 to τ_{r+1}, giving τ_r. (Some of these literals may already be in τ_{r+1}.) Heuristics may be used to try to improve the choice, but any choice works.

In both cases for C_0 it has no true literal in τ_{r+1}. So every clause containing $\overline{q}$ whose resolvent with C_0 is non-tautologous is hit (made true) by τ_{r+1}. Also every clause containing $\overline{q}$ whose resolvent with C_0 is *tautologous* is hit by τ_r. Finally all clauses containing q are hit by τ_r. So the postcondition at the beginning of round r is that every clause in the formula at the beginning of round r is hit by τ_r.

Universal-Variable Expansion If the operation for round r is *universal-variable expansion* (UVE), record the universal variable u that is eliminated, as well as the set of round-r clauses containing u, called $\Gamma_0(u)$, and the set of round-r clauses containing $\overline{u}$. called $\Gamma_1(u)$.

The details and notation of the UVE operation are now described. Note that there is no propositional analog of this operation. Define $\gamma_0(u)$ (resp. $\gamma_1(u)$), the *companion set for* u (resp. $\overline{u}$) to be the set of existential variables that occur (with either polarity) in $\Gamma_0(u)$ (resp. $\Gamma_1(u)$) and are later than u in the prenex.[1]

Next, define $\gamma_0^*(u)$ inductively as follows: $\gamma_0(u) \subseteq \gamma_0^*(u)$. If variable $e \in \gamma_0^*(u)$ and variable f is existential and later in the prenex than u and some clause contains both e and f (with either polarity), then $f \in \gamma_0^*(u)$. The inductive definition of $\gamma_1^*(u)$ is similar. Correctness of this construction can be shown with standard identities for quantifiers, such as

$$\forall u\,(F(u) \wedge G()) \equiv ((\forall u\,(F(u)) \wedge G()) \qquad \text{where } u \text{ does not occur in } G(),$$
$$\exists e\,(F(e) \wedge H()) \equiv ((\exists e\,(F(e)) \wedge H()) \qquad \text{where } e \text{ does not occur in } H(),$$

[1] Relaxing the condition to "e depends on u" based on a dependency scheme is future work.

as well as associativity and commutativity of "$\wedge$".

Supersets of $\gamma_0^*(u)$ and $\gamma_1^*(u)$ may be used soundly. In practice the set of all existential variables later in the prenex than u provides a simple over-approximation for $\gamma_0^*(u)$ and $\gamma_1^*(u)$.

Further, define $\Gamma_c(u)$ to be the set of round-r clauses not in $\Gamma_0(u)$ or $\Gamma_1(u)$, but which have a variable (with either polarity) in $(\gamma_0^*(u) \cup \gamma_1^*(u))$, and define $\Gamma_d(u)$ to be the remaining round-r clauses that are not in any of $\Gamma_0(u)$, $\Gamma_1(u)$, or $\Gamma_c(u)$. The clauses in $\Gamma_d(u)$ will persist into round $r+1$, while all other round-r clauses will be deleted during round r.[2]

The UVE step both deletes variables and clauses and inserts fresh variables and clauses, as follows. For each (existential) variable $e \in \gamma_0^*(u)$ create a fresh variable, say e_u^0, and record that e_u^0 evolved from e by instantiating $\overline{u} = 1$. For each (existential) variable $e \in \gamma_1^*(u)$ create a fresh variable, say e_u^1, and record that e_u^1 evolved from e by instantiating $u = 1$. For each clause $C \in (\Gamma_0(u) + \Gamma_c(u))$ create a fresh clause C_u^0 by deleting u if present and replacing each variable $f \in \gamma_0^*(u)$ by f_u^0. For each clause $C \in (\Gamma_1(u) + \Gamma_c(u))$ create a fresh clause C_u^1 by deleting $\overline{u}$ if present and replacing each variable $f \in \gamma_1^*(u)$ by f_u^1. Record how each C_u^0 and C_u^1 evolved, similarly to e_u^0 and e_u^1. Note that each clause in $\Gamma_c(u)$ evolves into two fresh clauses.

The critical situation for generating a certificate is when the formula is true and a set of hitting sets needs to be found to seed a set of "initial-reason cubes". Each hitting set (technically, consistent hitting set) must be a partial assignment such that every clause in the formula has a true literal. The key requirement on the set, following [VGWL12], is that if a hitting set in prenex order has a prefix α, u, then there must also be a hitting set with prefix α, $\overline{u}$, and *vice versa*.

So the precondition for updating each hitting set τ_{r+1} from the end of round r to a pair of hitting sets for the beginning of round r is that every clause in the formula at the end of round r is made true by τ_{r+1}. To update to τ_r^0 (possibly containing $\overline{u}$) and τ_r^1 (possibly containing u), all clauses deleted during round r must be made true using variables that exist at the beginning of round r.

The idea has some similarities to the VER extraction. Recall that u was expanded in round r. Partition the round $(r+1)$ clauses as follows:

1. Δ_u^0 consists of clauses with some existential variable of the form f_u^0 (which evolved from f during round r);
2. Δ_u^1 consists of clauses with some existential variable of the form f_u^1 (which evolved from f during round r);
3. Δ_u^d consists of the remaining clauses.

The hitting set for Δ_u^d in τ_{r+1} can be transferred directly into each of τ_r^0 and τ_r^1. Also, add $\overline{u}$ to τ_r^0 and add u to τ_r^1.

First we describe how to complete τ_r^0. For each clause $C_u^0 \in \Delta_u^0$, let C be the round-r clause that evolved into C_u^0. If some literal $q \in C$ already is true in τ_r^0, there is nothing to do. If this case does not apply, choose any existential literal

[2] The usual descriptions include several "optimizations" that are unrelated to correctness, and complicate certificate extraction.

of the form $e_u^0 \in C_u^0$ that is true in τ_{r+1} and add e to τ_r^0. This must be possible because $u \notin C_u^0$ and τ_{r+1} intersects C_u^0. For each clause $C_u^1 \in \Delta_u^1$, let C be the round-r clause that evolved into C_u^1. If $u \in C$, C_u^1 does not exist. If $\overline{u} \in C$, there is nothing to do. Otherwise, C already intersects τ_r^0 at some existential literal, and again there is nothing to do.

The completion of τ_r^1 is similar, except that Δ_u^1 is processed first. Thus each of τ_r^0 and τ_r^1 is a hitting set for the set of round-r clauses.

In the middle of the rounds a variable may have a sequence of tags, such as $(((e_w^0)_v^1)_u^0)_t^1$, where t, u, v and w are universal variables in the reverse order of their expansions. In the beginning of the round in which u was expanded this variable was $(e_w^0)_v^1$. Clause identifiers can similarly consist of an integer ID for the clause derived by resolution or in the original formula, modified by a sequence of tags to show where it was duplicated by some universal expansion.

4 Conclusion

A method to extend hitting sets backward through the rounds of a VE preprocessor (such as `bloqqer`) is described and the argument for correctness is sketched. The final result at the beginning of round 0 is required to contain only variables in the original formula (fed to the VE preprocessor) and to support a Q-refutation by cube resolution (AKA term resolution). The data that needs to be recorded to facilitate this extension is specified abstractly.

To realize the scope of the problem, it is perfectly possible that the initial set of hitting sets presented to the VE post-processor consists of *one* hitting set consisting of existential literals, *none of which occur in the original formula*.

Acknowledgment We thank the anonymous reviewer who pointed out errors in an earlier version.

References

[BJ12] V. Balabanov and J. R. Jiang. Unified QBF certification and its applications. *Formal Methods in System Design*, 41:45–65, 2012.

[BSL11] A. Biere, M. Seidl, and F. Lonsing. Blocked clause elimination for QBF. In *Proc. CADE*, 2011.

[DR94] R. Dechter and I. Rish. Directional resolution: the davis-putnam procedure, revisited. In *Proc. 4th Int'l Conf. on Principles of Knowledge Representation and Reasoning (KR'94)*, pages 134–145. Morgan Kaufmann, San Francisco, 1994.

[KBL99] H. Kleine Büning and T. Lettmann. *Propositional Logic: Deduction and Algorithms*. Cambridge University Press, 1999.

[NPL+12] A. Niemetz, M. Preiner, F. Lonsing, M. Seidl, and A. Biere. Resolution-based certificate extraction for QBF (tool presentation). In *Proc. SAT*, 2012.

[VGWL12] A. Van Gelder, S. B. Wood, and F. Lonsing. Extended failed literal detection for QBF. In *Proc. SAT*, 2012.

Benchmarks from Reduction Finding

Charles Jordan[1⋆] and Łukasz Kaiser[2]

[1] ERATO Minato Project, JST & Hokkaido University
[2] LIAFA, CNRS & Université Paris Diderot
skip@ist.hokudai.ac.jp, kaiser@liafa.univ-paris-diderot.fr

1 Introduction

Consider the problem of searching for a complexity-theoretic reduction between two decision problems, P and Q. Such a reduction is a function r satisfying $x \in P \iff r(x) \in Q$ for all structures x for which it makes sense to consider P. It is not difficult to show that determining the existence of reductions is undecidable in general. However, sufficiently restricting the class of reductions and structures considered results in a simpler, decidable variant that is still meaningful from a complexity-theory perspective.

Once we can encode these reductions and structures in a finite number of bits, the problem becomes a QBF instance with one quantifier alternation (2QBF):

$$\exists r \; \forall x \; (x \in P \iff r(x) \in Q)\,. \tag{1}$$

Therefore, reduction finding in this setting is essentially a Σ_2^p problem and it is possible to apply QBF solvers and ASP solvers supporting disjunctive logic programs. In this paper, we introduce various reduction-finding instances intended for benchmarking QBF solvers. For additional background, details and comparisons with other approaches, see [4]. Automatic reduction finding was first considered by [1]. More generally, reduction finding is a form of program synthesis, and similar approaches can be used more widely, see [3].

2 Background

Although polynomial-time reductions are perhaps the most traditional, we focus on a weaker class of quantifier-free logical reductions used in descriptive complexity. We do not introduce descriptive complexity and the full motivation for this class here, see [4] for the definitions we use and [2] for additional material.

The weaker class of logical reduction we consider has a number of advantages. First, the logical formulas defining the reduction make the explicit construction of (1) reasonable. Although the class of reductions we consider is extremely limited, it still suffices to characterize complexity classes and complete problems generally remain complete under these reductions. Proving that a computation

⋆ Supported in part by KAKENHI No. 25106501.

cannot be done in polynomial-time is notoriously difficult, while there are techniques for proving that something cannot be done in, e.g., first-order logic. So our reductions are strong enough that they suffice to determine the relationship between complexity classes, but weak enough to hope for progress.

2.1 Parameters

An instance of a reduction-finding problem is determined by several parameters, which thus determine the difficulty of the underlying problem. First, we must fix the two decision problems of interest: P and Q in (1). Our system supports defining these problems in first-order logic with equality extended by a reflexive transitive-closure (TC) operator, corresponding to the complexity class NL.

The choice of properties is important: choosing, e.g., the trivial properties *always true* and *always false* results in an easy instance that is not interesting. From the complexity-theoretic perspective, the hardness of properties defined in first-order logic (without TC) is fairly well-understood. Therefore, formulas using TC are most interesting and seem to be significantly harder.

Next, there are several parameters that determine the class of reductions being considered. Several parameters determine which atomic formulas are allowed to occur in the reduction. In particular, these parameters control whether successor is allowed between variables ($x = y + 1$), and whether certain numeric constants ($\min, \max$) are available. Then, there is a parameter k fixing the *dimension* of the reduction (informally, the dimension determines how much larger the output of a reduction can be compared to the original structure). Our reductions are defined by tuples of formulas in DNF – there is also a parameter c determining how many conjunctions may occur in the DNF.

Finally, there is a parameter n determining the size of structure considered in (1); we search only[3] for reductions that are correct on structures of size n.

Given these parameters, our generator constructs the corresponding QBF instance of (1) in several formats: qdimacs, negated qdimacs (to avoid an additional quantifier alternation from CNF conversion), qpro and lparse. Increasing c provides additional power to the class of reductions (instances become more positive), and increasing k is similar. Increasing n usually places more restrictions on the reduction (instances become more negative). A balance between parameters is important: large c with small n is unlikely to be meaningful because the reduction is too powerful for the small class of structures.

3 Instances

Let us describe how we instantiate the parameters described above in practice, and present some experimental results on a few parameter sets.

For the two decision problems of interest, P and Q in (1), we use one of 48 properties that were translated from the first paper on automatic reduction

[3] We support searching for reductions correct on a range of sizes $n_1 \leq n_2$ when using CEGAR, see [4].

finding [1] to allow to compare performance to their system (cf. [4]). The names we use for the problems correspond to the names used in the system from [1], and are always 6 characters long. Below, we give a list describing these 48 problems, to give an overview what is included.

- `trivial`, `ntrivil`, `trueque`, `falsequ`, `query15`, `query30` (trivial or almost trivial problems that we include for correctness testing)
- `reachqu` (reachability, NL-complete), `nreachq` (negated reachability, coNL-complete), `query10` (symmetric reachability, SL-complete),
- `query06`, `query31`, `query33`, `query34`, `query36` (L-complete), and `query71` (negated query06, coL-complete),
- `query42`, `query44` (non-reflexive reachqu/nreachq, NL/coNL-complete),
- `query48`, `query49` (non-reflexive SL/coSL-complete),
- `query54` (strongly connected, NL-complete), `query55` (symmetric query54),
- `query57`, `query58`, `query60`, `query64` (minor variations of the above),
- the others[4] are defined in plain FO and thus correspond to AC^0 problems.

Above, we mention the classes coNL, coL, SL and coSL, however, NL=coNL and L=coL=SL=coSL. Instances corresponding to discovering these famous collapses are interesting and appear hardest among the instances we consider.

We use the largest class of reductions supported: we allow min and max and successor in the atoms (`-min -max -succ`) and also the use of numbers when defining constants (`-nbrs`). As for the dimension k (`-dim`), number of conjunctions c (`-cls`), and structure size n (`-elems`), the most basic set of parameters we used was $k = 1, c = 1, n = 3$. Our generator is `ReductionTest.native`[4] and the following command can be used to generate the `qdimacs` file for the above parameters and $P =$ `nreachq`, $Q =$ `reachqu`.

```
./ReductionTest.native -min -max -succ -nbrs
  -from nreachq -to reachqu -dim 1 -cls 1 -elems 3 -qdimacs
```

The formula generated by the above command has 1731 variables and 18645 clauses, but most of the variables come from CNF conversion – only 66 are under the first existential quantifier (describing in r from eq. 1) and only 9 under the second universal one (describing $\mathfrak{A}$). Instances that don't use TC result in smaller formulas, but already setting `-elems 5` above returns 29878 variables and 530903 clauses. It is also possible to directly generate files for all problems we consider for a given parameter set using the `-gen [directory]` option.

```
./ReductionTest.native -min -max -succ -nbrs
  -dim 1 -cls 1 -elems 3 -gen .
```

In addition to generating files, the generator can test the existence of a reduction. This is done with CEGAR using a specified SAT-solver (at present `-minisat`, `-glueminisat` or the standard solver from Intel's Decision Procedure Toolkit). For example, one can check the existence of a reduction from $P =$ `nreachq` to $Q =$ `reachqu` with the specified parameters as follows.

[4] The generator with instructions, all instance names, and the collection of generated files we used for testing are available from `http://toss.sf.net/reductGen.html`.

```
./ReductionTest.native -min -max -succ -nbrs
  -from nreachq -to reachqu -dim 1 -cls 1 -elems 3 -glueminisat
```

From the 48 properties included, the method above can generate $48 \times 48 = 2304$ `qdimacs` files for each parameter set. To avoid an extra quantifier alternation due to CNF conversion in `qdimacs`, we also tested negating the QBF formula before CNF conversion – but results were worse than with the added alternation (cf. [4]). To give an intuition about the hardness of the generated QBF instances, we present below the results from [4] showing the number of timeouts on these 2304 instances for 5 QBF solvers. The tests were performed on an Opteron 1385 (using 1 core) and with a timeout of 120s.

	$c=1\ n=3$	$c=2\ n=3$	$c=3\ n=3$	$c=1\ n=4$	$c=2\ n=4$	$c=3\ n=4$
RAREQS	0	0	16	19	65	204
DEPQBF	0	142	547	16	297	711
QUBE	10	536	949	82	760	1082
CIRQIT	58	673	1138	511	1092	1357
SKIZZO	522	1058	1156	975	1327	1434

On our instances, RAREQS, a recently introduced CEGAR solver, performed best. For non-CEGAR solvers, DEPQBF and QUBE outperform SKIZZO and CIRQIT. Between DEPQBF and QUBE the situation is less clear, some instances work much better with one of these solvers, others with the other. The comparison between SKIZZO and CIRQIT is difficult as well. As to the dominance of DEPQBF and QUBE over SKIZZO and CIRQIT, it holds for almost all queries. Still, there are a few outliers such as the reduction from `query26` to `query01`, on which DEPQBF and QUBE time out, but SKIZZO answers almost immediately.

Of course, our generator is not restricted to these problems – any problem in NL can be defined in first-order logic with transitive closure, and these formulas can be directly used with the `-from` and `-to` options.

Although some interesting reductions can be expressed with dimension 1, usually this does not suffice. However, even parameters such as $k = 2, c = 1, n = 4$ or $k = 3, c = 1, n = 3$ seem to produce very hard instances. Instances with $k > 1$ where the properties use transitive-closure are often hard, but achieving better performance on such instances is an important step to finding *new* reductions and hopefully new complexity-theoretic knowledge.

References

1. Crouch, M., Immerman, N., Moss, J.E.B.: Finding reductions automatically. In: Fields of Logic and Computation – Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday. LNCS, vol. 6300, pp. 181–200. Springer (2010)
2. Immerman, N.: Descriptive Complexity. Springer-Verlag (1999)
3. Jordan, C., Kaiser, Ł.: Learning programs as logical queries. In: LTC 2013
4. Jordan, C., Kaiser, Ł.: Experiments with reduction finding. In: Proc. SAT 2013. LNCS, vol. 7962, pp. 192–207. Springer (2013)