qbf2epr: A Tool for Generating EPR Formulas from QBF*

Martina Seidl[1,2], Florian Lonsing[1,3] and Armin Biere[1]

[1] Institute for Formal Models and Verification,
Johannes Kepler University, Austria
firstname.lastname@jku.at

[2] Business Informatics Group
Vienna University of Technology, Austria

[3] Knowledge-Based Systems Group
Vienna University of Technology, Austria

Abstract

We present the tool qbf2epr which translates quantified Boolean formulas (QBF) to formulas in effectively propositional logic (EPR). The decision problem of QBF is the prototypical problem for PSPACE, whereas EPR is NEXPTIME-complete. Thus QBF is embedded in a formalism, which is potentially more succinct. The motivation for this work is twofold. On the one hand, our tool generates challenging benchmarks for EPR solvers. On the other hand, we are interested in how EPR solvers perform compared to QBF solvers and if there are techniques implemented in EPR solvers which would also be valuable in QBF solvers and vice versa. Furthermore, we provide an improved encoding of QBF in EPR based on dependency schemes which are a powerful concept in QBF solving.

1 Motivation

Propositional logic has proven itself to be an extremely valuable formalism for solving a wide range of reasoning problems of industrial scale. With its decision problem (SAT) being the prototypical problem for the complexity class NP, propositional logic serves not only as the host language for a wide range of application problems like planning and verification, but also is an enabling technology for large reasoning frameworks [20]. Building around this success story, research has been focused on related formalisms which have the same expressive power as SAT, but whose additional language features allow exponentially smaller problem encodings.

Two of such formalisms are *quantified Boolean formulas* (QBF) [17] and *effectively propositional logic* (EPR) [14]. While QBF extends the language of propositional logic with quantifiers over propositional variables, EPR is a syntactically restricted subset of first-order logic. As consequence, the majority of recent QBF solvers extend techniques found in modern SAT solvers, whereas EPR solvers are strongly inspired by first-order theorem provers. From a practical point of view, it would be interesting to know if there are benefits in transferring the reasoning techniques of one formalism to the other formalism. In order to directly compare QBF and EPR solvers, benchmarks are required which can be run by both kinds of solvers. One way of obtaining such benchmarks is embedding the "weaker" QBF in the "stronger" EPR. To this end, we present the tool qbf2epr which performs such an encoding. Therefore, we shortly revisit the basics of EPR and QBF in the next section, which are required for the embedding of QBF in EPR presented in Section 3. This embedding is implemented in the tool qbf2epr. Experiments

*This work was partially funded by the Vienna Science and Technology Fund (WWTF) under grant ICT10-018 and by the Austrian Science Fund (FWF) under NFN Grants S11408-N23 and S11409-N23 (RiSE).

performed with recent QBF and EPR solvers are presented in Section 4. For improving the embedding of QBF in EPR, we then consider *dependency schemes* in Section 5. Dependency schemes are a powerful concept in QBF solving, which provide a relaxed notion of quantifier dependencies. We conclude this paper with a discussion of future research directions.

2 Background

In the following, we introduce the concepts and terminology used in this paper necessary to describe the transformation of a QBF to an equisatisfiable EPR formula. In particular, we recap the formalisms EPR and QBF. We assume familiarity with propositional logic as well as with first-order logic.

Effectively Propositional Logic (EPR). The formulas of EPR (also known as Bernays-Schoenfinkel class) form a subset of first-order predicate logic (FOL) consisting of formulas with the structure

$$\exists X \forall Y.\, \rho \;\equiv\; \exists X \forall Y. \bigwedge_{i=0}^{n} \bigvee_{j=0}^{m_i} l_{ij}$$

where X and Y are disjoint sets of variables and ρ is a function free first-order formula in conjunctive normal form over X and Y. When transforming such a formula to Skolem normal form, the existentially quantified variables are simply replaced by new constants. If the universally quantified variables are grounded by all combinations of these constants, then the resulting formula is an exponentially larger representation of the EPR formula in propositional logic. In this paper, we use standard first-order semantics, which is specified over a domain $\mathcal{D}$ and an interpretation function $\iota : FOL \to \{\mathbb{T}, \mathbb{F}\}$. For our purposes, the domain of interest is binary. Several important use cases for EPR have been identified, including LTL bounded model checking [18] or reasoning with quantified bit vectors [9, 27].

Quantified Boolean Formulas (QBF). QBF extend propositional logic with quantifiers over propositional variables, i.e., a QBF may contain a subformula $\forall^q x.\psi$ or $\exists^q x.\psi$. The formula ψ is again a QBF and is called *scope* of variable x. In this paper, we mainly consider QBF $\Pi.\psi$ in prenex conjunctive normal form (PCNF) with $\Pi = Q_1 X_1 \dots Q_n X_n$, where the X_i are disjoint sets of variables, i denotes the quantification level of the variables in X_i, $Q_j \in \{\forall^q, \exists^q\}$, and ψ is a propositional formula in conjunctive normal form. A formula in conjunctive normal form consists of a conjunction of clauses. A clause is a disjunction of literals. A literal is a variable or the negation of a variable. Note that we annotate the connectives, quantifiers, and truth constants with the superscript ^{q} in order to distinguish them from the corresponding symbols in EPR. A QBF $\forall^q x \Pi.\psi$ is satisfiable iff $\Pi.\psi[x \backslash \bot^q]$ and $\Pi.\psi[x \backslash \top^q]$ is true. Respectively, a QBF $\exists^q x \Pi.\psi$ is true iff $\Pi.\psi[x \backslash \bot^q]$ or $\Pi.\psi[s \backslash \top^q]$ is true. QBF find their application for instance in various verification scenarios [3].

3 From QBF to EPR

In this section, we discuss the translation of QBFs in prenex conjunctive normal form to EPR formulas. This translation is based on the results presented in [2, 21]. Benedetti [2] introduced the Skolemization-based QBF solver sKizzo which rewrites QBF to a first-order formula, before reducing it to a propositional formula. Piskac et al. illustrate the encoding of quantified formulas

with equality over a finite domain to EPR [21] which we specialize to standard QBF. The translation is straightforward, but optimized with respect to specific QBF properties.

3.1 Transformation of a QBF to a First-Order Formula

In this first step, we reformulate the QBF problem as an instance of first-order predicate logic as follows.

Definition 1. *The embedding* $[\![]\!]_p : QBF \to FOL$ *with respect to the unary predicate symbol* p *is given by*

$$\begin{array}{rcl@{\qquad}rcl} [\![\exists^q x.\phi]\!]_p & = & \exists x.[\![\phi]\!]_p & [\![\forall^q x.\phi]\!]_p & = & \forall x.[\![\phi]\!]_p \\ [\![\phi \vee^q \psi]\!]_p & = & [\![\phi]\!]_p \vee [\![\psi]\!]_p & [\![\phi \wedge^q \psi]\!]_p & = & [\![\phi]\!]_p \wedge [\![\psi]\!]_p \\ [\![x]\!]_p & = & p(x) & [\![\neg^q x]\!]_p & = & \neg p(x) \\ [\![\top^q]\!]_p & = & p(\mathit{true}) & [\![\bot^q]\!]_p & = & p(\mathit{false}) \end{array}$$

The embedding wraps the predicate p around the variables and QBF truth constants $\top^q$ and $\bot^q$ are mapped to the dedicated function symbols *true* and *false*. Whereas in predicate logic variables and in consequence the quantifiers operate on the term level, the situation is different in QBF. Here the variables incarnate predicates. To lift the variables to term level, we introduce a predicate p of arity one, for which it holds that $\iota(p(\mathit{true})) = \mathbb{T}$ and $\iota(p(\mathit{false})) = \mathbb{F}$.

Lemma 1. *Let* ϕ *be a QBF and let* p *be a unary predicate symbol. Then* ϕ *is satisfiable iff the first-order formula* $([\![\phi]\!]_p \wedge p(\mathit{true}) \wedge \neg p(\mathit{false}))$ *is satisfiable.*

Lemma 1 can be easily shown by induction over the formula size. It describes the embedding of arbitrary structured QBF to FOL. In the following, we consider QBFs in prenex conjunctive normal form only. This is no severe restriction, because the transformation to PCNF is polynomial and most QBF benchmarks are only available in PCNF anyhow. We impose no restriction on the number of quantifier alternations. If a QBF ϕ has a prefix with n quantifier alternations, then the FOL formula $[\![\phi]\!]_p$ obviously has n quantifier alternations as well.

3.2 Elimination of Existential Quantifiers

In order to obtain a FOL formula with the prefix structure $\exists X \forall Y$, we substitute existentially quantified variables which are in the scope of universally quantified variables $a_1, \ldots, a_m$ by a Boolean function with $a_1, \ldots, a_m$ as arguments. This technique of symbolic representation of quantifier dependencies is know as Skolemization [24]. In automated theorem proving, Skolemization is used to eliminate one type of quantifiers in a satisfiability preserving manner.

Definition 2. *Let* $\chi = \exists x_1 \ldots \exists x_n \forall a_1 \ldots \forall a_m \exists y \Pi.\rho$ *be a FOL formula in PCNF. Then the quantifier* $\exists y$ *can be eliminated as follows*

$$\exists x_1 \ldots \exists x_n \forall a_1 \ldots \forall a_m \exists y \Pi.\rho \rightsquigarrow \exists x_1 \ldots \exists x_n \forall a_1 \ldots \forall a_m \Pi.\rho[y \backslash f_y(a_1, \ldots, a_m)]$$

where f_y *is a function symbol not occurring in* χ*. The function* $\mathsf{sk}(\chi)$ *applies this substitution on FOL formula* χ *until fixpoint.*

Obviously, in Definition 2 an existential variable is replaced by a function with all universal variables as arguments which have a lower quantification level. In Section 5, we will provide an optimization of this substitution in order to obtain functions with less parameters.

Lemma 2. *Let ρ be a FOL formula in prenex conjunctive normal form and let $\rho' = \mathsf{sk}(\rho)$. Then it holds that (i) the prefix of ρ' contains at most one quantifier alternation, (ii) if ρ' has one quantifier alternation, then it has the structure $\exists X \forall Y$, (iii) ρ' is satisfiable iff ρ is satisfiable.*

Points (i) and (ii) of Lemma 2 follow from the construction of $\mathsf{sk}(.)$, and point (iii) is an application of Skolemization [24]. With the introduction of function symbols, we move further away not only from the language of QBF but also from EPR, although we have now the required structure of the quantifier prefix. To obtain an EPR formula, the function symbols must be eliminated.

3.3 Removal of Function Symbols

In the last step, we have introduced function symbols which allowed us to symbolically represent quantifier dependencies and to reduce the prefix to the required structure having only one alternation. Since the language of EPR does not allow function symbols, these have to be removed from the formula again.

Definition 3. *The function $\mathsf{func}(\rho)$ applies the following rewriting rule*

$$p(f(a_1, \ldots, a_m)) \rightsquigarrow p_f(a_1, \ldots, a_m)$$

on FOL formula ρ until fixpoint where p_f is a predicate and f is a function symbol.

We are interested in interpretations of the functions over a two valued domain, hence function symbols can also be seen as predicates. Therefore we can remove the enclosing predicate symbol for such functions.

Lemma 3. *Over a two valued domain, the rewriting function $\mathsf{func}(\rho)$ preserves satisfiability.*

Finally, we have all the building blocks required to transform a QBF to an EPR formula which manifests in the following proposition.

Proposition 1. *Let ϕ be a formula in prenex conjunctive normal form and let p be a symbol not occurring in ϕ. Then $(\mathsf{func}(\mathsf{sk}([\![\phi]\!]_p)) \wedge p_{true} \wedge \neg p_{false})$ is an EPR formula which is equisatisfiable to ϕ.*

Proposition 1 follows from Lemma 1, Lemma 2, and Lemma 3. The following example illustrates the translation of a QBF to an EPR formula.

Example 1. *Given the QBF $\phi = \exists^q a \exists^q b \forall^q x \forall^q y \exists^q c \exists^q d.(a \vee^q x \vee^q c) \wedge^q (a \vee^q b) \wedge^q (b \vee^q y \vee^q d)$ the following three steps have to be performed.*

1. Transformation to FOL.
   We embbed ϕ in first-order logic w.r.t. predicate p and obtain

   $$[\![\phi]\!]_p = \exists a \exists b \forall x \forall y \exists c \exists d.(p(a) \vee p(x) \vee p(c)) \wedge (p(a) \vee p(b)) \wedge (p(b) \vee p(y) \vee p(d)).$$

2. Elimination of Existential Quantifiers.
   In order to obtain the required prefix structure, we apply Skolemination and eliminate all existential quantifiers not occurring in the first quantifier block, i.e., $\exists c$ and $\exists d$. Then

   $$\mathsf{sk}([\![\phi]\!]_p) = \exists a \exists b \forall x \forall y.(p(a) \vee p(x) \vee p(f_c(x,y))) \wedge (p(a) \vee p(b) \wedge (p(b)) \vee p(y) \vee p(f_d(x,y))).$$

3. Elimination of Function Symbols.
   Finally, we lift the function symbols to predicats resulting in the formula

   $$\mathsf{func}(\mathsf{sk}([\![\phi]\!]_p)) = \exists a \exists b \forall x \forall y.(p(a) \vee p(x) \vee f_c(x,y)) \wedge (p(a) \vee p(b) \wedge (p(b)) \vee p(y) \vee f_d(x,y)).$$

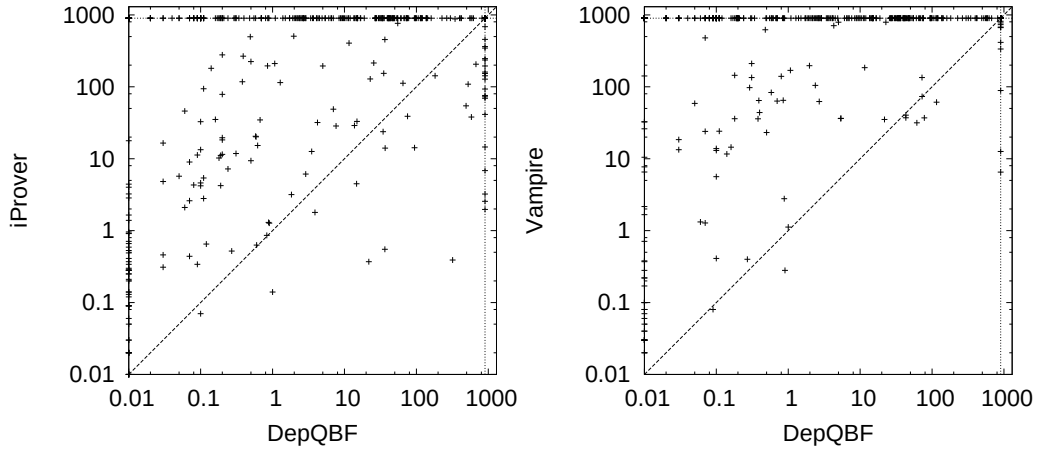

Figure 1: Comparison DepQBFwith iProver and Vampire.

4 Evaluation

The embedding of QBF in EPR as described in the previous section is implemented in the tool qbf2epr [1], with input format QDIMACS, the standard format for QBFs in prenex conjunctive normal form [10]. The produced formulas are formulated in the TPTP language [25]. We investigated how a state-of-the-art EPR solver performs on the QBF benchmark set of the evaluation 2010 [19]. As EPR solver we used iProver [12], which won the EPR devision of the CASC competition [26] in the last years. As a reference QBF solver, we ran DepQBF [16]. DepQBF is a search-based solver relying on the QDPLL algorithm for QBF [7]. QDPLL is a QBF-specific variant of the DPLL algorithm for propositional logic. Given a QBF in QDIMACS format, DepQBF processes the formula without any modifications whereas iProver operates on the EPR embedding generated by qbf2epr. All considered QBFs are in PCNF already, hence there was no need for explicit clausification in iProver in our experiments. Table 1 summarizes our main results.[1] The benchmark set contains 568 formulas.

As probably expected, the QBF solver DepQBF outperforms iProver in terms of the number of solved instances as well as in terms of time and memory requirements. DepQBF solved more than twice as many instances in half of average time spent by iProver on the instances translated by qbf2epr (cf. left subfigure of Figure 1). The translation to EPR failed on four out of 568 formulas due to limited time or memory. This does not influence the overall picture as none of these four formulas were solved by DepQBF. On average, translation time spent by qbf2epr was 13.33 seconds (in parentheses next to 673.50, the average run time of iProver). Average memory usage of DepQBF is lower by a factor of 40. Note that iProver ran out of memory on 237 instances, which also contributes to the median time of 900 seconds, as we treat these situations like running out of time.

If we focus on instances which were solved by both DepQBF and iProver, then again instances were solved faster and with less memory by DepQBF in their native QBF encoding. The performance of iProver is closest to DepQBF on unsatisfiable instances, where iProver spends 50% more time on average. This observation is also related to 21 instances which were solved

[1]Setup: 64-bit Ubuntu Linux 9.04, Intel®Q9550 2.83 GHz with 900 seconds / 7 GB total time and memory limit. Exceeding the memory limit is counted as a time out. We used iProver version 0.8.1 and an internal version of DepQBF. Binaries of all tools and log-files are available online [1].

	Solved	*SAT*	*UNSAT*	*Avg. Time*	*Med. Time*	*Avg. Mem.*	*Med. Mem.*
DepQBF	372	166	206	334.60	29.92	93.3674	22.2
iProver	155	51	104	673.50 (13.33)	900	3924.48	3673.0
Only formulas solved by both iProver and DepQBF							
DepQBF	134	44	90	28.08	0.09	19.056	2.55
iProver				54.87	3.62	432.5	85.60
Only satisfiable formulas solved by both iProver and DepQBF							
DepQBF	44	44	–	2.60	< 0.01	4.02	< 0.01
iProver				36.94	0.48	120.548	29.10
Only unsatisfiable formulas solved by both iProver and DepQBF							
DepQBF	90	–	90	40.54	0.18	26.4067	7.35
iProver				63.63	5.54	585.01	167.15
21 formulas solved by iProver but not by DepQBF							
iProver	21	7	14	166.43	127.89	1505.07	616.9

Table 1: Performance comparison of DepQBF and iProver.

by iProver but *not* by DepQBF. From those 21 instances, 14 are unsatisfiable. Successful QBF preprocessing techniques like blocked clause elimination [5] showed only little improvements on the performance of the EPR solver.

Apart from the QDPLL-based solver DepQBF, we considered the QBF solver Quantor [4], which is based on quantifier elimination. Quantor solved 203 instances (99 satisfiable, 104 unsatisfiable) in 590.15 seconds average time. Although Quantor solves fewer instances than DepQBF, the results indicate that QDPLL and quantifier elimination, the two major approaches for QBF solving, perform better on QBF than iProver does on translated instances.

Note that iProver solved 37 unsatisfiable instances which were *not* solved by Quantor. A closer look at the set of formulas exclusively solved by iProver (and by none of the two QBF solvers) seems to suggest that techniques applied in iProver are particularly beneficial for solving unsatisfiable QBFs. Many of these exclusively solved formulas encode problems of black box bounded model checking (BMC) [11] (family `biu*` in the benchmark set). Solving BMC encodings in QBF and EPR is in general considered to be challenging [3, 9].

Additionally, we were interested how a first-order theorem prover handles the translated QBFs. We therefore ran the same experiment with the state-of-the-art theorem prover Vampire [22]. Vampire solved 65 formulas (24 satisfiable, 41 unsatisfiable). A comparison to DepQBF is shown in the right part of Figure 1. Again several instances of black box bounded model checking are included in the set of solved formulas.

5 From QBF to EPR with Dependencies

The embedding of QBF in EPR as introduced in Section 3 always included *all* universal variables of lower quantification level in the Skolem function of an existential variable, even if they were independent of each other. In order to reduce number of arguments the Skolem functions, we clarify the notion of (in)dependence based on a QBF specific concept called *dependency scheme* and evaluate the impact of this optimization on the runtime of the EPR solver.

5.1 QBF Dependency Schemes

Most state-of-the-art QBF solvers use prenex conjunctive normal form as introduced in Section 2 as input format. On the one hand, PCNF supports sophisticated reasoning techniques and allows for highly optimized data structures, but on the other hand structural information which might be valuable for the solving process is blurred by the transformation to PCNF. For example, in general it is not given that a formula consists of a quantifier prefix and the propositional matrix. To obtain the required structure, *prenexing* has to be performed which refers to the satisfiability preserving transformation of shifting all quantifiers outside the formula.

Whereas in the original formula the quantifiers are arranged in a tree established by the nesting of the scopes, prenexing flattens this tree into a linear list. So a much stronger order is imposed on the variables, which is restrictive for the solving process. Since prenexing is not deterministic, multiple prenexing strategies are often applicable. It has been shown that the selection of the wrong strategy adversely influences the solving time [8]. Therefore, solvers are strongly dependent on the normalform transformation tool, if the prefix is simply processed from left to right.

To overcome this restriction, dependency schemes [23] have been introduced which allow for relaxing the prefix ordering without changing the truth value of a formula. A dependency scheme D is a binary relation over the variables of a QBF expressing (in)dependence of two variables. A variable y depends on variable x iff $(x, y) \in D$. Consider the QBF $\phi = \forall^q x \exists^q y.\psi$ with $(x, y) \notin D$. Then ϕ is equivalent to $\exists^q y \forall^q x.\psi$, i.e., if two variables are independent, then their quantifiers may be swapped. The prenex directly implies a trivial dependency scheme P where $(x, y) \in P$ iff variable x occurs before variable y in the prefix. As in the case of P, dependency schemes may contain spurious dependencies. These spurious dependencies may be eliminated by a stronger dependency scheme, but dependencies crucial for the truth value of a formula may never be omitted. Less spurious dependencies contained in a dependency scheme imply more freedom for variable selection. It can be shown that there exists one unique optimal dependency scheme, but its computation is in PSPACE. Since the computation of the optimal dependency scheme is as hard as solving a QBF, it is not practically feasible. Therefore, non-optimal, but nevertheless powerful dependency schemes are used for QBF solving. For a detailed survey on theory and practice of dependency schemes in QBF solver, we kindly refer to [15].

5.2 Translation to EPR with Dependency Schemes

In the following, we extend Definition 2 with a dependency scheme in order to reduce the number of arguments of the Skolem functions.

Definition 4. *Let ϕ be a QBF in PCNF with a dependency scheme D. Further, let $\chi = [\![\phi]\!]_p = \exists X \forall A \exists y \Pi.\rho$ be an embedding of ϕ in FOL, where X and A are sets of variables (and Π the rest of the quantifier prefix). Then the quantifier $\exists y$ can be eliminated as follows*

$$\chi \rightsquigarrow \exists X \forall A \Pi.\rho[y \backslash f_y(a_1, \ldots, a_k)]$$

where $(a_i, y) \in D$, $|\{a_i | (a_i, y) \in D\}| = k$, and f_y is a function symbol not occurring in χ. The function $\mathsf{sk}^{\mathsf{D}}(\chi)$ applies this substitution on FOL formula χ until fixpoint.

Note that Definition 4 does not refer to any specific dependency scheme. If we would use the dependency scheme induced by the quantifier prefix, then we would obtain the same translations as with Definition 2.

Lemma 4. *Let ρ be a FOL formula in prenex conjunctive normal form obtained from a QBF ϕ with dependency scheme D. Further, let $\rho' = \mathsf{sk}^{\mathsf{D}}(\rho)$. Then it holds that (i) the prefix of ρ' contains at most one quantifier alternation, (ii) if ρ' has one quantifier alternation, then it has the structure $\exists X \forall Y$, (iii) ρ' is satisfiable iff ρ is satisfiable.*

Whereas (i) and (ii) directly follow from the construction of $\mathsf{sk}^{\mathsf{D}}()$, (iii) holds because it can be shown that two QBFs in PCNF are equivalent if they have the same matrix and the same quantifiers, and the quantifier ordering of both obey the same dependency scheme.

Example 2. *The QBF $\exists^q a \exists^q b \forall^q x \forall^q y \exists^q c \exists^q d.(a \vee^q x \vee^q c) \wedge^q (a \vee^q b) \wedge^q (b \vee^q y \vee^q d)$ with dependency scheme $D = \{(a,x),(x,c),(b,y),(y,d)\}$ is equisatisfiable to the EPR formula*

$$\exists a \exists b \forall x \forall y.(p(a) \vee p(x) \vee f_c(x)) \wedge (p(a) \vee p(b)) \wedge (p(b) \vee p(y) \vee f_d(y)).$$

5.3 Implementation and Evaluation

The QBF solver DepQBF [16] which we already applied for the evaluation in Section 4 uses the *standard dependency schema* [23] in a DPLL-based decision procedure and provides a very efficient implementation for its calculation. The standard dependency scheme is calculated by analyzing the structure of a formula in terms of connections between variables in sequences of clauses. The dependency scheme of Example 2 is a standard dependency scheme. There are dependencies between a and x as well as between x and c, because these variables occur in the same clause. The same holds for b and y and y and d. However, there is no dependency between d and x as well as c and y. In consequence, we may replace c by $f_c(x)$ and d by $f_d(y)$. A formal description of the standard dependency scheme can be found in [23].

We extended DepQBF in such a way that it provides a dependency service, i.e., it can be called with a formula as argument, and then compute pairs of dependent variables. qbf2epr uses this service to obtain the universal variables on which a given existential variable depends. Then only these variables are included in the Skolem function.

With the improved translation, we performed the same evaluation as before using iProver as EPR solver. With the dependency scheme enabled, 104 unsatisfiable formulas as well as 51 satisfiable formulas were solved, e.g. exactly the same number as before without dependencies. However, the usage of dependency schemes reduced the over all runtime by more than 1000 seconds. Details are available at [1].

6 Conclusion and Future Work

We showed how to translate QBF to EPR resulting in challenging benchmarks for testing and evaluating EPR solver. The tool qbf2epr is available at [1]. EPR as well as QBF are promising formalisms for the encoding of a multitude of verification problems [9, 18, 20, 27] for which probably no succinct encoding in SAT can be found. Although both are closely related to SAT, QBF and EPR solving approaches differ profoundly. In first-order theorem proving for example, instantiation-based approaches result quite naturally in a decision procedure for EPR [13] which might also be interesting for QBF. In our experiments we observed that there exist formulas in the QBF standard benchmark set which can be solved by the EPR solver iProver but not by specialized QBF solvers.

This points out that there might be techniques used in EPR solving which might also be valuable for QBF solvers and confirms observations of [27] where a similar behavior for quantified

bit vector formulas was experienced. Therefore, more experiments have to be conducted and the implemented inference techniques have to be compared rigorously. QBF solvers quite substantially outperform EPR solvers on the considered QBF benchmarks, and thus it would be interesting to investigate if and how EPR solving can benefit from QBF solving techniques also on other instances. Besides the additional benchmarks, qbf2epr offers an additional benefit to the developers of EPR solvers. Now QBF specific development tools like fuzzers and delta-debuggers [6] can be directly used for the development of EPR solvers.

We further showed, how the presented embedding is extended to take independencies between variables into account. First, we considered only the order of the variables imposed by the prefix. In an improved variant of the embedding, we use dependency schemes for reducing the number of arguments of the Skolem functions.

However, the comparison between the QBF and EPR solvers was not fair in the following sense. EPR allows for a potentially more succinct encoding of application problems than QBF due to the richer language. Nevertheless, we provided a direct translation of the QBF encoding to the EPR solver, which does not benefit from EPR language features. In future work, it would be therefore interesting to investigate if more sophisicated translations of the QBFs are possible. For a fair evaluation, application problems shall be directly encoded in QBF and EPR allowing a direct comparison of the solvers.

Acknowledgements. The authors would like to thank Robert Aistleitner and Gregor Dorfbauer for implementing the original version of the presented tool as part of a student project.

References

[1] qbf2epr Project Page. Website. `http://fmv.jku.at/qbf2epr/`.

[2] M. Benedetti. sKizzo: a QBF Decision Procedure based on Propositional Skolemization and Symbolic Reasoning. Tech.Rep. 04-11-03, ITC-irst, 2004.

[3] M. Benedetti and H. Mangassarian. Qbf-based formal verification: Experience and perspectives. *JSAT*, 5(1-4):133–191, 2008.

[4] A. Biere. Resolve and Expand. In *SAT'04*, pages 59–70, 2004.

[5] A. Biere, F. Lonsing, and M. Seidl. Blocked Clause Elimination for QBF. In *CADE'11*, volume 6803 of *LNCS*, pages 101–115. Springer, 2011.

[6] R. Brummayer, F. Lonsing, and A. Biere. Automated Testing and Debugging of SAT and QBF Solvers. In *Proc. of SAT 2010*, volume 6175 of *LNCS*, pages 44–57. Springer, 2010.

[7] M. Cadoli, A. Giovanardi, and M. Schaerf. An Algorithm to Evaluate Quantified Boolean Formulae. In *AAAI/IAAI'98*, pages 262–267, 1998.

[8] U. Egly, M. Seidl, H. Tompits, S. Woltran, and M. Zolda. Comparing Different Prenexing Strategies for Quantified Boolean Formulas. In *Proc. of SAT 2003*, volume 2919 of *LNCS*, pages 214–228. Springer, 2004.

[9] M. Emmer, Z. Khasidashvili, K. Korovin, and A. Voronkov. Encoding industrial hardware verification problems into effectively propositional logic. In *FMCAD'10*, pages 137–144. IEEE, 2010.

[10] E. Giunchiglia, M. Narizzano, and A. Tacchella. Quantified Boolean Formulas satisfiability library (QBFLIB), 2001. `www.qbflib.org`.

[11] M. Herbstritt and B. Becker. On Combining 01X-Logic and QBF. In *EUROCAST'07*, volume 4739 of *LNCS*, pages 531–538. Springer, 2007.

[12] K. Korovin. iProver - An Instantiation-Based Theorem Prover for First-Order Logic (System Description). In *IJCAR'08*, pages 292–298, 2008.

[13] K. Korovin. Instantiation-based automated reasoning: From theory to practice. In *CADE'09*, volume 5663 of *LNCS*, pages 163–166. Springer, 2009.

[14] H.R. Lewis. Complexity results for classes of quantificational formulas. *Journal of Computer and System Sciences*, 21(3):317–353, 1980.

[15] F. Lonsing. *Dependency Schemes and Search-Based QBF Solving: Theory and Practice*. PhD thesis, JKU Linz, 2012.

[16] F. Lonsing and A. Biere. DepQBF: A Dependency-Aware QBF Solver. *JSAT*, 7(2-3):71–76, 2010.

[17] A.R. Meyer and L.J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *SWAT'72*, pages 125–129. IEEE, 1972.

[18] J.A. Navarro Pérez and A. Voronkov. Encodings of Bounded LTL Model Checking in Effectively Propositional Logic. In *CADE'07*, volume 4603 of *LNCS*, pages 346–361. Springer, 2007.

[19] C. Peschiera, L. Pulina, A. Tacchella, U. Bubeck, O. Kullmann, and I. Lynce. The Seventh QBF Solvers Evaluation (QBFEVAL'10). In *SAT*, volume 6175 of *LNCS*, pages 237–250. Springer, 2010.

[20] M. R. Prasad, A. Biere, and A. Gupta. A survey of recent advances in SAT-based formal verification. *STTT*, 7(2):156–173, 2005.

[21] R. Piskac and L. de Moura and N. Bjørner. Deciding Effectively Propositional Logic with Equality. Technical Report: MSR-TR-2008-181.

[22] A. Riazanov and A. Voronkov. The design and implementation of vampire. *AI Commun.*, 15(2-3):91–110, 2002.

[23] Marko Samer and Stefan Szeider. Backdoor sets of quantified boolean formulas. *J. Autom. Reasoning*, 42(1):77–97, 2009.

[24] Th. Skolem. *Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit und Beweisbarkeit mathematischen Sätze nebst einem Theoreme über dichte Mengen.* Translation in: From Frege to Gödel, van Heijenoort, Harvard Univ. Press, 1971.

[25] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[26] G. Sutcliffe and C. Suttner. The State of CASC. *AI Comm.*, 19(1):35–48, 2006.

[27] C.M. Wintersteiger, Y. Hamadi, and L. de Moura. Efficiently solving quantified bit-vector formulas. In *FMCAD'10*, pages 239–246. IEEE, 2010.