A Solver for QBFs in Nonprenex Form[1]

Uwe Egly[2] and **Martina Seidl**[3] and **Stefan Woltran**[2]

Abstract. Various problems in AI can be solved by translating them into a quantified boolean formula (QBF) and evaluating the resulting encoding. In this approach, a QBF solver is used as a black box in a rapid implementation of a more general reasoning system. Most of the current solvers for QBFs require formulas in prenex conjunctive normal form as input, which makes a further translation necessary, since the encodings are usually not in a specific normal form. This additional step increases the number of variables in the formula or disrupts the formula's structure. Moreover, the most important part of this transformation, prenexing, is not deterministic. In this paper, we focus on an alternative way to process QBFs without these drawbacks and implement a solver, qpro, which is able to handle arbitrary formulas. To this end, we extend algorithms for QBFs to the non-normal form case and compare qpro with the leading normal-form provers on problems from the area of AI.

1 Introduction

Formal frameworks are often suitable for the representation of application problems (like planning, scheduling, etc.) which can then be solved by automated reasoning tools. Many important problems in artificial intelligence (AI) (like problems in knowledge representation) can be encoded efficiently using quantified boolean formulas (QBFs), which are an extension of classical propositional formulas, permitting existential and universal quantifications over propositional atoms. QBFs have been proven to be a powerful framework for the rapid implementation of reasoning tasks from these areas (see, e.g., [6, 13, 24]), mainly because there has been made a significant progress in the development of QBF solvers in the last few years [19]. Almost all of these solvers expect the input formula to be in a certain *prenex conjunctive normal form* (PCNF), requiring all quantifiers to be in front of a purely propositional formula, which has to be in conjunctive normal form (CNF). However, natural encodings of problems from AI do not yield QBFs in such a normal form, and thus the particular instances have to be transformed. The transformation is performed in two steps, namely *prenexing* and transformation of the resulting purely propositional matrix into CNF. The drawbacks of this transformation are an increase in both formula size and variable number, or, even worse, the formula's structure is disrupted.

In this paper, we present a prover, qpro, which works on arbitrary QBFs. Its basic procedure is a generalized DPLL algorithm with enhanced dependency-directed backtracking techniques. The space requirements for qpro are modest; it runs in polynomial space (wrt the length of the input formula). The motivation to circumvent formulas in PCNF and to work with arbitrary QBFs is the problem to generate "good" normal forms in this logic. The main problem here is the handling of quantifiers at the places where they occur. This is in contrast to first-order logic. We explain in the following aspects of normalization for different logics and discuss why QBFs are problematic.

It is well known how a propositional (or first-order formula) can be translated into a satisfiability-equivalent CNF, such that the structural information is retained by new atoms [23, 10, 8]. Together with their definition, such new atoms can mimic the effect of the analytic cut rule in full calculi like Gentzen systems resulting in drastically shorter proofs [3, 11]. Moreover, as experiments showed [14, 21], such structure-preserving translations are not only beneficial from a theoretical point of view, but can also speed-up automated theorem provers for practical problems. In the last few years, similar results have been obtained for the case of prenex QBFs and an optimized handling of the newly introduced atoms has been proposed [1].

We consider the problem to construct a prenex form of a QBF. The prenexing transformation cannot be carried out deterministically; the chosen normalization strategy crucially influences the runtimes (also depending on the concrete solver used), see e.g., [15, 26]. In fact, this phenomenon mirrors a similar observation from classical theorem proving in *first-order logic*, where classes of formulas exist for which different quantifier shifting strategies (resulting in different prenex forms) yield a non-elementary difference of proof size (and search-space size) [4, 12]. Clearly, the impact of the prenex forms is less drastic for QBFs because of the simpler underlying logic, but there are indications that prenexing impacts the runtime of highly optimized state-of-the art solvers [18].

In first-order logic, skolemization can be used to encode the properties of (usually) existential quantifiers by Skolem functions. In a nutshell, skolemization gets rid of existential quantifiers "in place". The introduced Skolem functions encode two properties of quantifier rules in full first-order calculi: (i) the eigenvariable condition and (ii) non-permutabilities between quantifier rules. Condition (i) is satisfied by the requirement to introduce a globally new function symbol, and condition (ii) is handled by the occur check in the unification algorithm. Due to the weaker syntax of QBFs, the introduction of Skolem functions is not possible and therefore this conceptually simple tool is not directly applicable in the context of QBFs.

The outline of the paper is as follows: Section 2 introduces necessary definitions and notations. Sections 3 and 4 are devoted to the formal underpinnings of the prover. Section 5 provides experiments and comparisons. Finally, we discuss the results as well as related and future work in Section 6.

2 Background

We introduce the language $\mathcal{L}_{\mathcal{P}}$ of QBFs as an extension of the language of propositional logic. The alphabet of $\mathcal{L}_{\mathcal{P}}$ consists of paren-

[1] This work was supported by FWF under grant P18019, ÖAD under grant Amadée 2/2006, and FFG under grant FIT-IT-810806.

[2] Institut für Informationssysteme 184/3, Technische Universität Wien, Favoritenstraße 9–11, A-1040 Vienna, Austria.

[3] Institut für Softwaretechnik und Interaktive Systeme 188/3, Technische Universität Wien, Favoritenstraße 9–11, A-1040 Vienna, Austria.

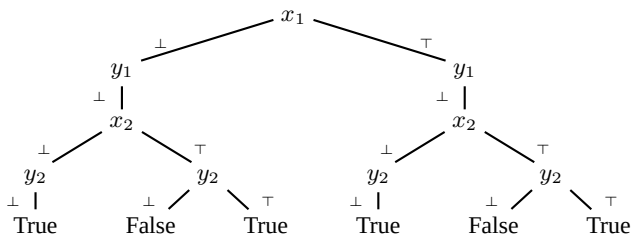

Figure 1. The branching tree of the example QBF ϕ.

theses, the truth constants $\top$ and $\bot$, a countable set of variables $\mathcal{P}$, the unary connective $\neg$ (negation), the binary connectives $\vee$ (disjunction) and $\wedge$ (conjunction), and the quantifier symbols $\forall$ (universal) and $\exists$ (existential). A literal is a variable or a negated variable.

We define the language of *quantified propositional logic* over a set of variables $\mathcal{P}$ as the smallest set, $\mathcal{L}_\mathcal{P}$, satisfying the conditions:

1. If $x \in \mathcal{P} \cup \{\top, \bot\}$, then $x \in \mathcal{L}_\mathcal{P}$ and $(\neg x) \in \mathcal{L}_\mathcal{P}$;
2. if $\phi, \psi \in \mathcal{L}_\mathcal{P}$, then $(\phi \circ \psi) \in \mathcal{L}_\mathcal{P}$, where $\circ \in \{\vee, \wedge\}$;
3. if $\phi \in \mathcal{L}_\mathcal{P}$, $x \in \mathcal{P}$, and $\mathsf{Q}x$ does not occur in ϕ, then $(\mathsf{Q}x\,\phi) \in \mathcal{L}_\mathcal{P}$, where $\mathsf{Q} \in \{\forall, \exists\}$.

Any element of $\mathcal{L}_\mathcal{P}$ is called a *quantified boolean formula* (QBF). If no ambiguities arise, we omit parentheses when convenient. Moreover, for a set $X = \{x_1, \ldots, x_n\}$ of variables, and $\mathsf{Q} \in \{\forall, \exists\}$, we write $\mathsf{Q}X\phi$ or $\mathsf{Q}x_1 \ldots x_n\phi$ as a short-hand for $\mathsf{Q}x_1 \cdots \mathsf{Q}x_n\phi$.

Note that we allow negation only in front of a variable or a truth constant. Formulas which obey this restriction are said to be in *negation normal form* (NNF). The NNF of any QBF can be obtained by applying DeMorgan's laws and the removal of double negation. Since this transformation can be done deterministically and since the increase of the formula size is negligible, we only consider QBFs in NNF. From a technical point of view, QBFs in NNF have the advantage that polarities of complex subformulas do not come into play.

The scope of a quantifier (occurrence) $\mathsf{Q}x$ in a QBF ϕ is defined as ψ, where $\mathsf{Q}x\,\psi$ is the subformula corresponding to the occurrence of $\mathsf{Q}x$ in ϕ. An occurrence of a variable x is called existential (resp. universal) if it is located within the scope of a quantifier $\exists x$ (resp. $\forall x$). Unless stated otherwise, we consider *closed* QBFs, i.e., each occurrence of a variable x is located in the scope of a quantifier $\mathsf{Q}x$.

We denote by $\phi[x/\psi]$ the result of substituting each occurrence of a variable x in a QBF ϕ by a QBF ψ. The semantics of a QBF is then given as follows: A QBF $\exists x\,\psi$ is true iff $\psi[x/\bot]$ or $\psi[x/\top]$ is true. Dually, a QBF $\forall x\,\psi$ is true iff $\psi[x/\bot]$ and $\psi[x/\top]$ are true. The other connectives are treated according to the standard evaluation rules of propositional logic, providing the usual recursive definition of the truth value of a QBF. Two closed QBFs are called equivalent if they possess the same truth value.

The sequence of the variable assignments when evaluating a QBF can be illustrated by a branching tree. The nodes contain the variables and the two subtrees of a node x correspond to the subproblems, where x is replaced by $\bot$ or $\top$, as indicated by the labels of the arcs. The leaves contain the resulting truth values. If x is existentially (resp. universally) quantified and one subproblem evaluates to true (resp. false), then the second subproblem can be omitted. As an example, we show in Figure 1 the branching tree of the true formula

$$\phi \;=\; \forall x_1 \exists y_1 \big(\forall x_2 \exists y_2 ((x_2 \vee \neg y_2) \wedge (\neg x_2 \vee y_2)) \vee (x_1 \wedge y_1)\big).$$

Definition 1 *A QBF ϕ is given in* prenex normal form (PNF) *if ϕ is of the form $\mathsf{Q}_1 X_1 \ldots \mathsf{Q}_m X_m \psi$, where $\mathsf{Q}_i \in \{\forall, \exists\}$ and ψ is purely propositional. Moreover, if ψ is in conjunctive normal form, ϕ is said to be in* prenex conjunctive normal form (PCNF).

```
BOOLEAN split(QBF φ in NNF) {

switch (simplify (φ)):
  case ⊤: return True;
  case ⊥: return False;
  case (φ1 ∨ φ2): return (split(φ1) || split(φ2));
  case (φ1 ∧ φ2): return (split(φ1) && split(φ2));
  case (QXψ): select x ∈ X;
    if Q = ∃ return (split(ψ[x/⊤]) || split(ψ[x/⊥]));
    if Q = ∀ return (split(ψ[x/⊤]) && split(ψ[x/⊥]));
}
```

Figure 2. The basic algorithm.

The PCNF format is required by most of the available QBF solvers.

Any QBF can be translated into an equivalent QBF in PNF, but there are several ways to do this. The concept of different *prenexing strategies* is discussed in [15, 26]. We give here only some intuition.

First, the *dependencies* between the quantifiers in a QBF are given by common occurrences on paths in the formula tree. To avoid a formal definition, consider the following example

$$\psi = \exists x\big((\forall y_1\, \exists z_1\, \forall u_1\, \exists v_1\, \psi_1) \wedge (\forall y_2\, \exists z_2\, \psi_2) \wedge (\exists y_3\, \forall z_3\, \psi_3)\big),$$

where the ψ_i's are propositional formulas. Then, $\forall y_1$ depends on $\exists x$, $\exists z_1$ depends on $\forall y_1$ as well as on $\exists x$, $\forall y_2$ depends on $\exists x$, etc. but, e.g., $\exists z_2$ does not depend on $\forall y_1$. We say that a QBF has *depth* m, if the sequences of depending quantifiers provide at most $m-1$ alternations. Observe that ψ has depth 5 as witnessed by the "path" $\exists x \forall y_1 \exists z_1 \forall u_1 \exists v_1$. The aim of prenexing is to "*linearize*" quantifier dependencies (which in fact form a partial order) without increasing the depth of the QBF. We consider here four different *prenexing strategies*, namely "$\uparrow$", "$\downarrow$", "$\exists\downarrow\forall\uparrow$", and "$\exists\uparrow\forall\downarrow$". Hereby, "$\uparrow$" (resp. "$\downarrow$") denotes that any quantifier is placed as outermost (resp. innermost) as possible in the prefix. "$\exists\downarrow\forall\uparrow$" and "$\exists\uparrow\forall\downarrow$" follow the same concept but now the handling is depending on the particular quantifier, i.e., whether it concerns an existential or a universal one. Thus, for our example formula ψ, we derive different PNFs of ψ having the same depth:

$$\begin{aligned}
\uparrow \;&:\; \exists x y_3\; \forall y_1 y_2 z_3\; \exists z_1 z_2\; \forall u_1\; \exists v_1\; (\psi_1 \wedge \psi_2 \wedge \psi_3);\\
\exists\uparrow\forall\downarrow \;&:\; \exists x y_3\; \forall y_1 y_2\; \exists z_1 z_2\; \forall u_1 z_3\; \exists v_1\; (\psi_1 \wedge \psi_2 \wedge \psi_3);\\
\exists\downarrow\forall\uparrow \;&:\; \exists x\; \forall y_1 y_2\; \exists z_1 y_3\; \forall u_1 z_3\; \exists v_1 z_2\; (\psi_1 \wedge \psi_2 \wedge \psi_3);\\
\downarrow \;&:\; \exists x\; \forall y_1\; \exists z_1 y_3\; \forall u_1 y_2 z_3\; \exists v_1 z_2\; (\psi_1 \wedge \psi_2 \wedge \psi_3).
\end{aligned}$$

QBFs in PNF are prototypical problems for complexity classes in the *polynomial hierarchy*. In fact, the evaluation problem of QBFs $\exists X_1 \forall X_2 \ldots \mathsf{Q}_i X_i \phi$ is Σ_i^P-complete with $\mathsf{Q}_i = \exists$ if i is odd and $\mathsf{Q}_i = \forall$ if i is even. Dually, evaluating $\forall X_1 \exists X_2 \ldots \mathsf{Q}_i X_n \phi$ is Π_i^P-complete, with $\mathsf{Q}_i = \forall$ if i is odd, and $\mathsf{Q}_i = \exists$ if i is even.

3 A Generalization of the DPLL Procedure

In this section we present the most important formal underpinnings of the presented solver qpro which relies on a generalized variant of the decision procedure due to Davis, Putnam, Loveland, and Logemann (DPLL for short). The DPLL procedure [9] represents one of the most successful algorithms to decide the truth value of a propositional formula. This method has been adapted for QBFs (in PCNF format) and it is used in many state-of-the-art solvers.

We generalize DPLL in such a way that it is applicable to arbitrary formulas in $\mathcal{L}_\mathcal{P}$, i.e., such that DPLL can be directly applied to QBFs

in NNF without additional transformations to PCNF. Figure 2 shows a simplified version of the program code of our decision procedure.

Note that DPLL is a direct implementation of the semantics for QBFs. The formula is split into subproblems, whose return values are treated according to the connective, the variables are replaced by truth constants, and simplifications are applied until a truth value is obtained. The basic decision procedure is thus a simple search-based backtracking algorithm which works in polynomial space with respect to the size of the input formula.

The function `simplify`(ϕ) in Figure 2 returns a formula which results from ϕ by applying numerous equivalence-preserving transformations, including, e.g., the following ones:

(a) $\neg\top \Rightarrow \bot; \quad \neg\bot \Rightarrow \top;$

(b) $\top \wedge \phi \Rightarrow \phi; \quad \bot \wedge \phi \Rightarrow \bot; \quad \top \vee \phi \Rightarrow \top; \quad \bot \vee \phi \Rightarrow \phi;$

(c) $(\mathsf{Q}x\,\phi) \Rightarrow \phi$, $\mathsf{Q} \in \{\forall, \exists\}$, x does not occur in ϕ;

(d) $\forall x(\phi \wedge \psi) \Rightarrow (\forall x\phi) \wedge (\forall x\psi)$;

(e) $\forall x(\phi \vee \psi) \Rightarrow (\forall x\phi) \vee \psi$, whenever x does not occur in ψ;

(f) $\exists x(\phi \vee \psi) \Rightarrow (\exists x\phi) \vee (\exists x\psi)$;

(g) $\exists x(\phi \wedge \psi) \Rightarrow (\exists x\phi) \wedge \psi$, whenever x does not occur in ψ.

Rewritings (d)–(g) are known as *miniscoping*. Note that the application of miniscoping is dynamic within the algorithm, due to the repetitive substitution of variables and simplifications of subformulas. Further simplifications are derived from generalizations of other well known concepts.

Definition 2 *Let ϕ be a QBF, ψ a subformula of ϕ, $\mathsf{Q} \in \{\forall, \exists\}$, and $\circ \in \{\vee, \wedge\}$. A literal $l \in \{x, \neg x\}$ is called*

- local unit *(wrt ψ) in ϕ, if ψ is of the form $(l \circ \psi')$;*
- global unit *(wrt ψ) in ϕ, if ψ is of the form $\mathsf{Q}x(l \circ \psi')$;*
- pure *(wrt ψ) in ϕ, if ψ is of the form $\mathsf{Q}x\psi'$, and $\overline{l}$ does not occur in ψ', where $\overline{x} = \neg x$ and $\overline{\neg x} = x$.*

Proposition 1 *Let ϕ be a closed QBF and let $l = x$ (resp. $l = \neg x$) be a (i) local-unit (ii) global-unit (iii) pure literal wrt a subformula ψ in ϕ, where ψ is given according to Definition 2. Then, ϕ is equivalent to the QBF resulting from ϕ by replacing ψ in case of*

$$(i) \quad \text{by} \begin{cases} l \circ \psi'[x/\top] \text{ (resp. } l \circ \psi'[x/\bot]) & \text{if } \circ = \wedge; \\ l \circ \psi'[x/\bot] \text{ (resp. } l \circ \psi'[x/\top]) & \text{if } \circ = \vee; \end{cases}$$

$$(ii) \quad \text{by} \begin{cases} (l \circ \psi')[x/\top] \text{ (resp. } (l \circ \psi')[x/\bot]) & \text{if } x \text{ is existential;} \\ (l \circ \psi')[x/\bot] \text{ (resp. } (l \circ \psi')[x/\top]) & \text{if } x \text{ is universal;} \end{cases}$$

$$(iii) \text{ by} \begin{cases} \psi'[x/\top] \text{ (resp. } \psi'[x/\bot]) & \text{if } x \text{ is existential;} \\ \psi'[x/\bot] \text{ (resp. } \psi'[x/\top]) & \text{if } x \text{ is universal.} \end{cases}$$

Note that (ii) and (iii) delete $\mathsf{Q}x$ in ψ by an implicit application of (c) (since all occurrences of x in the scope of $\mathsf{Q}x$ have been replaced by $\bot$ or $\top$). Also observe the following difference between (i) and (ii). In (i), the literal occurrence l in $(l \circ \psi)$ is not affected by the substitution of x, while in (ii) the substitution of x concerns also l in $(l \circ \psi)$. Note that (ii) yields—together with (a) and (b)—a replacement of the entire subformula $\psi = \mathsf{Q}x(l \circ \psi')$ in ϕ by $\top$ (resp. $\bot$), in the case $\mathsf{Q} = \exists$, $\circ = \vee$ (resp. $\mathsf{Q} = \forall$, $\circ = \wedge$). Finally, observe that (ii) also applies to formulas of the form $\psi = \mathsf{Q}x\mathsf{Q}_1X_1 \cdots \mathsf{Q}_nX_n(l \circ \psi'')$ since, by definition of QBFs, $x \notin X_i$, and with (e) and (g), we can transform ψ to be of form $\mathsf{Q}x(l \circ \psi')$.

The remaining part of the extended DPLL procedure in Figure 2 is straightforward but we have to face three sources of indeterminism within the `switch` statement: if `simplify`(ϕ) returns a QBF $\phi_1 \vee \phi_2$ or $\phi_1 \wedge \phi_2$, we have to select (i) which subformula to evaluate first; this part differs from PCNF solvers, where such a decision is not necessary; if `simplify`(ϕ) returns $\mathsf{Q}X\psi$, with $\mathsf{Q} \in \{\forall, \exists\}$, we have to select (ii) on which variable $x \in X$ to branch, and (iii) which subproblem (i.e., $\psi[x/\top]$ or $\psi[x/\bot]$) to consider first. Choice (ii) is obviously restricted by the dependencies between the quantifiers, since universal and existential quantifiers must not be permuted.

4 Dependency-Directed Backtracking

In this section we briefly describe an important technique called *dependency-directed backtracking* (DDB), which is known to be crucial for the efficiency of the DPLL procedure. In QBF solvers for PCNF, this technique only works for false subproblems. Starting from [20], we generalize this technique to arbitrary formulas, for which it can be applied to *true and false subproblems*. DDB for false subproblems applies to existential variables, whereas DDB for true subproblems applies to universal variables.

When we inspect a branching tree, we may notice that some branches (together with the corresponding subtrees) can be omitted without influencing the (partial) evaluation result. In this case, the result is independent from the assignments of some variables. If we have set such a variable x to one truth value, we can safely omit the assignment of x to the other truth value at this point in the branching tree during backtracking. Consider Figure 1 as an example and observe that the tree is *symmetric*, i.e., the left and the right subtree of x_1 (with root y_1) are identical. Suppose the prover has finished the left half of the branching tree. Under the assignment $\bot$ for x_1, the resulting simplified formula $\phi' : \forall x_2 \exists y_2((x_2 \vee \neg y_2) \wedge (\neg x_2 \vee y_2))$ evaluates to true. Since x_1 is a universal variable, the next assignment for it is $\top$. A clever solver notices that x_1 has no influence on the truth value of ϕ' with the consequence that ϕ' is true under the new assignment. So, no further assignments are performed because we can utilize the result of the left subtree with root y_1. What we have described here is DDB on true subproblems (neglecting substitutions of unit and pure literals, however, for the matter of presentation). The more common DDB on false subproblems works as expected.

We have implemented two different sound and complete backtracking techniques, namely *labeling* and *relevance sets*. Since the latter is superior to the former and since all experiments have been performed using the latter, we only describe *DDB by relevance sets* here. We present this technique only for true subproblems, but it works dually for false ones.

Let R_y denote the *relevance set* for a node y. If the solver reaches a leaf l in the branching tree, only those variables whose assignments determine the truth value of the formula (i.e., the label of l) form R_l. Assume we return from a subtree P_1 with root x_1 to the variable x directly above x_1 during backtracking and P_1 is true. If x is existential, then $R_x = R_{x_1}$. If x is universal, then we check whether x is contained in R_{x_1}. If $x \notin R_{x_1}$ holds, then the other subproblem P_1' is true and $R_x = R_{x_1}$. Otherwise, the relevance set R'_{x_1} of P_1' has to be considered. We construct R_x as follows. If $x \notin R'_{x_1}$ holds, then $R_x = R'_{x_1}$. Otherwise, R is set to $R_{x_1} \cup R'_{x_1}$.

5 Experimental Evaluation

In this section we compare the performance of our new solver qpro against the established systems QuBE−BJ [17] (v1.2), sKizzo [5] (v0.4), semprop [20] (rel. 24/02/02), and quantor [7] (rel. 25/01/04). These solvers have been selected because they have shown to be

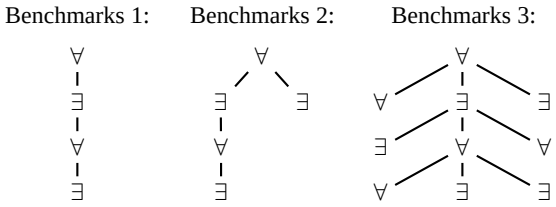

Figure 3. Quantifier dependencies of the different benchmarks.

competitive in previous QBF evaluations and do not deliver wrong results on our benchmarks. Moreover, QuBE−BJ and semprop implement backtracking techniques, similar to the one used in qpro. Finally, sKizzo and quantor try to extract original quantifier dependencies from a PCNF, and thus may detect similar structural information on the input formula as qpro has got *a priori* from the input of the corresponding nonprenex formula.

All solvers except qpro require the input to be in PCNF. We thus apply the following test strategy: Given a benchmark QBF ϕ not in PCNF, we (i) provide ϕ as input to qpro; (ii) translate ϕ into PCNF and provide the outcome as input to the other solvers. The exact way we obtained the PCNFs is discussed below. We have chosen the following benchmark formulas (not in PCNF).[4]

1. Encodings for satisfiability in modal logic K.
2. Encodings for correspondence tests in answer-set programming.
3. Encodings of reasoning with nested counterfactuals.

In what follows, we briefly describe the benchmarks. For detailed information, the reader is referred to the according references.

The first set of benchmarks contains instances which were also used in the TANCS'98 comparison of provers for modal logics. Applying the encoding from [22] yields QBFs with a linear dependency among the quantifiers. Hence, the translation to PNF is fully determined (there is just one way to shift the quantifiers in front of the formula).

The benchmarks in the second set encode correspondence tests between propositional logic programs under the answer-set semantics, where it is checked whether two programs provide equal projected answer-sets under any program extension over a specified alphabet, cf. [25]. For the benchmarks, we use two strategies, namely "↑" and "↓" to obtain formulas in PCNF.

The benchmarks in the third set encode the problem of reasoning over nested counterfactuals [15]. For this set of formulas, the quantifier dependencies allow for several different translations into PCNF. We have applied each strategy from [15] to the PCNF solvers. To be fair, we show only the results for the best strategy of each solver.

The structural differences between these sets are best illustrated by the quantifier dependencies of the formulas. We depict them for the case of QBFs of depth 4 in Figure 3. Hence, the chosen benchmarks provide an increasing complexity in their structure, and therefore, an increasing disruption of the structure during the transformation to PNF can be expected. In particular, for Benchmarks 1, we just can investigate the effect of applying the transformation of one shifting strategy, since the prefix is already determined by the encoding. Benchmarks 2 allow to analyze the effect of prenexing if there is only a small deviation from a linear quantifier dependency. In fact, using either strategy "↑" or "↓", one can place the group of existential quantifiers from the right path either together with the upper or the lower set of existential quantifiers from the left path. Finally, Benchmarks 3 provide formulas, for which different PNFs can be obtained.

[4] PCNF versions are part of the QBFLIB, `http://www.qbflib.org/`.

	number of timeouts					average runtimes (in sec.)				
	qpro	QuBE−BJ	semprop	sKizzo	quantor	qpro	QuBE−BJ	semprop	sKizzo	quantor
01-*	10	7	7	0	18	52.34	37.43	38.29	**12.65**	67.90
02-*	2	0	0	0	18	23.90	6.92	**0.08**	0.89	71.67
03-*	14	10	14	17	17	70.30	**52.92**	68.27	80.72	80.69
04-*	9	8	0	0	17	45.06	45.22	**0.01**	0.18	79.79
05-*	0	0	0	7	13	**0.01**	**0.00**	0.11	37.25	63.09
06-*	0	0	0	0	12	**0.00**	**0.00**	0.32	0.29	54.90
07-*	0	0	0	0	8	**0.00**	**0.00**	**0.00**	0.17	43.10
08-*	0	0	0	4	8	**0.00**	**0.00**	**0.01**	0.12	40.24
09-*	0	0	0	1	2	**0.00**	**0.00**	**0.01**	2.33	15.09
10-*	0	0	0	0	0	**0.00**	**0.00**	**0.01**	**0.00**	**0.07**
11-*	9	8	0	0	16	45.09	43.63	**0.14**	0.22	76.02
12-*	6	6	0	0	15	34.28	32.67	**0.07**	0.21	72.10
13-*	0	4	0	0	1	0.22	26.74	**0.02**	1.00	6.79
14-*	13	13	12	10	11	72.26	64.57	58.89	**50.13**	56.22
15-*	0	0	0	0	0	3.90	**0.01**	0.39	**0.04**	**0.05**
16-*	0	0	0	0	19	0.27	0.28	**0.01**	0.57	73.96
17-*	16	0	12	4	20	78.74	**0.72**	61.29	47.83	82.57
18-*	13	0	0	6	18	65.63	0.26	**0.04**	41.46	79.95

Table 1. Benchmarks 1: Modal Logic K.

	number of timeouts					average runtimes (in sec.)				
	qpro	QuBE−BJ	semprop	sKizzo	quantor	qpro	QuBE−BJ	semprop	sKizzo	quantor
S↑	–	842	117	527	1000	–	87.34	81.82	74.67	100
S↓	–	90	6	0	1000	–	43.21	27.60	**2.67**	100
T↑	–	43	38	0	1000	–	20.85	54.86	2.97	100
T↓	–	0	0	0	1000	–	9.26	16.90	**1.13**	100
S	29	–	–	–	–	33.37	–	–	–	–
T	0	–	–	–	–	17.87	–	–	–	–

Table 2. Benchmarks 2: Answer Set Correspondence.

We ran our tests on an Intel Xeon 3 GHz with 4GB of RAM. All solvers are used with their predefined standard options. For each instance, we set a timeout of 100 seconds. We report both the number of timeouts per set of benchmarks, as well as the average runtimes, where formulas with timeout are considered to be solved in 100 seconds. In what follows, we present our results on the benchmarks.[5]

Benchmarks 1: Modal Logic K. This set contains 378 formulas arranged in 18 subsets, with 21 formulas each. Half of the formulas evaluates to true. Depending on the modal depth of the original formula, the depth of the encodings ranges from 5 to 133; the number of variables ranges from less than 40 to more than 4300. Due to the transformation into PCNF, the number of variables increases up to more than 12800 in the worst case. The results are given in Table 1.

Benchmarks 2: Answer-Set Correspondence. Here, the test series comprise 1000 instances (465 are true and 535 are false). We ran each problem on two different encodings, S and T, where T is an explicit optimization of S (see [25] for details). The problem of answer-set correspondence is Π_4^P-complete, and thus all QBFs in this set have depth 4. The quantifier dependencies, as depicted in Figure 3, are the same for S and T and suggest to distinct between two strategies for obtaining PCNFs, viz. ↑ and ↓. The QBFs possess, in case of S, 200 variables and, in case of T, 152 variables. The additional translation into PCNF yields, in case of S, QBFs over 2851 variables and, in case of T, QBFs over 2555 variables. The results for each solver on each combination of the chosen translation and (in the case of PCNF solvers) prenexing strategy are given in Table 2.

Benchmarks 3: Nested Counterfactuals. The final set of benchmarks are encodings of nested counterfactual reasoning, separated

[5] We highlight the best runtime for each test set (discrepancies due to measurement inaccuracy are ignored).

by the depth of the resulting QBFs which ranges from 4 to 8. For each depth, we created 50 instances, where the QBFs contain 183, 245, 309, 375, 443 variables. The transformation to PCNF increases the number of variables to 464, 600, 786, 934, and 1132. In total, we have about 60% true and 40% false instances. As mentioned above, these encodings allow for several different prenexing strategies. For space reasons, we do not present the results for each strategy here, but Table 3 shows the results for the *best* strategy which has been derived for each solver on the entire set of QBFs. Recall that for qpro we do not need to apply any such strategy.

	number of timeouts					average runtimes (in sec.)				
	qpro	QuBE−BJ ∃↑∀↓	semprop ↑	sKizzo ∃↓∀↑	quantor ∃↓∀↑	qpro	QuBE−BJ ∃↑∀↓	semprop ↑	sKizzo ∃↓∀↑	quantor ∃↓∀↑
4	0	1	8	9	31	**0.41**	5.10	39.39	22.30	86.27
5	0	3	10	13	30	**1.06**	9.35	28.69	32.72	88.62
6	0	4	34	26	42	**2.06**	11.66	69.01	57.20	82.87
7	0	8	32	28	41	**2.34**	20.45	63.34	60.06	82.87
8	0	12	45	38	41	**6.81**	32.08	79.72	78.55	90.47

Table 3. Benchmarks 3: Nested Counterfactuals.

6 Discussion and Conclusion

We presented a new QBF solver, qpro, which significantly differs from previous approaches by its ability to process arbitrary QBFs instead of QBFs in PCNF. We sketched generalizations of the DPLL procedure necessary to handle arbitrary QBFs and briefly discussed implemented performance-improving techniques like different forms of dependency-directed backtracking. Future work calls for an analysis how the different pruning techniques influence performance.

In practical applications, QBF solvers can be used as a black box in reasoning systems to solve encodings of the problems considered. Usually such encodings are not in PCNF and, as we have shown in our experiments, avoiding the additional translation into PCNF may result in much better performance. In what follows, we briefly discuss two main observations from our experiments.

- The more information on quantifier dependencies is lost due to prenexing, the more competitive qpro turns out to be. Table 3 contains those benchmarks where this effect is most apparent. Although we compare qpro here against the PCNF solvers together with their *best* suited strategy, qpro significantly outperforms all other solvers.
- Table 2 presents results for two different encodings of the same problem where T is an explicit optimization of S. These results show that qpro is less depending on the chosen encoding, whereas the performance of PCNF solvers differs much more. In fact, qpro performs better on the unoptimized encoding S, in the case the "wrong" prenexing strategy "↑" is used for the PCNF solvers.

Finally, we briefly discuss related systems.

- There are a few further solvers, namely QUBOS [2], boole[6], and zqsat [16], which also allow arbitrary QBFs as input, but rely on different techniques. QUBOS simplifies the QBF and then constructs an equivalent propositional formula which is evaluated by SAT solvers, whereas boole is based on binary decision diagrams (BDDs). Thus both need exponential space in the worst case. We have included boole in our pre-tests, but it was not competitive at the benchmarks. We also neglected QUBOS, because it yielded wrong results on some problems. Finally, zqsat implements DPLL using zero-compressed BDDs. The comparison to zqsat is subject to future work.
- In [18], an extension to QuBE was suggested, where the input is a QBF in PCNF *together with information on quantifier dependencies*. Contrary to this approach, ours avoids the bounded renaming of variables during the prefix construction together with the necessity to transform the matrix into CNF. This enables us to dynamically (i) apply miniscoping and (ii) recognize independent subproblems which can be solved in parallel. Nonetheless, the results in [18] lead to observations in the same direction as ours. Future work will include a detailed comparison of the two approaches, which will be done as soon as the modified QuBE system is available.

[6] http://www.cs.cmu.edu/~modelcheck/bdd.html.

References

[1] C. Ansótegui, C. Gomes, and B. Selman, 'The Achilles' Heel of QBF', in *Proc. AAAI'05*, pp. 275–281. AAAI Press, (2005).
[2] A. Ayari and D. Basin, 'QUBOS: Deciding Quantified Boolean Logic Using Propositional Satisfiability Solvers', in *Proc. FMCAD'02*, pp. 187–201. Springer, (2002).
[3] M. Baaz, C. Fermüller, and A. Leitsch, 'A Non-Elementary Speed Up in Proof Length by Structural Clause Form Transformation', in *Proc. LICS'94*, pp. 213–219. IEEE Computer Society Press, (1994).
[4] M. Baaz and A. Leitsch, 'On Skolemization and Proof Complexity', *Fundamenta Informaticae*, **20**, 353–379, (1994).
[5] M. Benedetti, 'sKizzo: A Suite to Evaluate and Certify QBFs', in *Proc. CADE-21*, pp. 369–376. Springer, (2005).
[6] P. Besnard, T. Schaub, H. Tompits, and S. Woltran, 'Representing Paraconsistent Reasoning via Quantified Propositional Logic', in *Inconsistency Tolerance*, 84–118, Springer, (2005).
[7] A. Biere, 'Resolve and Expand', in *Proc. SAT'04*, pp. 59–70. Springer, (2005).
[8] T. Boy de la Tour, 'An Optimality Result for Clause Form Translation', *Journal of Symbolic Computation*, **14**(4), 283–302, (1992).
[9] M. Davis, G. Logemann, and D. Loveland, 'A Machine Program for Theorem Proving', *Comm. of the ACM*, **5**(7), 394–397, (1962).
[10] E. Eder, *Relative Complexities of First-Order Calculi*, Vieweg, 1992.
[11] U. Egly, 'On Different Structure-preserving Translations to Normal Form', *Journal of Symbolic Computation*, **22**, 121–142, (1996).
[12] U. Egly, 'Quantifiers and the System KE: Some Surprising Results', in *Proc. CSL'98*, pp. 90–104. Springer, (1999).
[13] U. Egly, T. Eiter, H. Tompits, and S. Woltran, 'Solving Advanced Reasoning Tasks Using Quantified Boolean Formulas', in *Proc. AAAI'00*, pp. 417–422, (2000).
[14] U. Egly and T. Rath, 'On the Practical Value of Different Definitional Translations to Normal Form', in *Proc. CADE-13*, pp. 403–417. Springer, (1996).
[15] U. Egly, M. Seidl, H. Tompits, S. Woltran, and M. Zolda, 'Comparing Different Prenexing Strategies for Quantified Boolean Formulas', in *Proc. SAT'03*, pp. 214–228, (2004).
[16] M. Ghasemzadeh, V. Klotz, and C. Meinel, 'Embedding Memoization to the Semantic Tree Search for Deciding QBFs.', in *Proc. AI'05*, pp. 681–693. Springer, (2004).
[17] E. Giunchiglia, M. Narizzano, and A. Tacchella, 'Backjumping for Quantified Boolean Logic Satisfiability', *AIJ*, **145**, 99–120, (2003).
[18] E. Giunchiglia, M. Narizzano, and A. Tacchella, 'Quantifier Structure in Search Based Procedures for QBFs', in *Proc. of DATE*, (2006).
[19] D. Le Berre, M. Narizzano, L. Simon, and A. Tacchella, 'The Second QBF Solvers Comparative Evaluation', in *Proc. SAT'04*, pp. 376–392. Springer, (2005).
[20] R. Letz, 'Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas', in *Proc. TABLEAUX'02*, pp. 160–175. Springer, (2002).
[21] A. Nonnengart, G. Rock, and C. Weidenbach, 'On Generating Small Clause Normal Forms', in *Proc. CADE-15*, pp. 397–411. Springer, (1998).
[22] G. Pan and M. Vardi, 'Optimizing a BDD-Based Modal Solver', in *Proc. CADE-19*, pp. 75–89. Springer, (2003).
[23] D. A. Plaisted and S. Greenbaum, 'A Structure Preserving Clause Form Translation', *Journal of Symbolic Computation*, **2**(3), 293–304, (1986).
[24] J. Rintanen, 'Constructing Conditional Plans by a Theorem Prover', *Journal of Artificial Intelligence Research*, **10**, 323–352, (1999).
[25] H. Tompits and S. Woltran, 'Towards Implementations for Advanced Equivalence Checking in Answer-Set Programming', in *Proc. ICLP'05*, pp. 189–203. Springer, (2005).
[26] Michael Zolda, *Comparing Different Prenexing Strategies for Quantified Boolean Formulas*, Master's thesis, TU Wien, 2004.