I N F S Y S

R E S E A R C H

R E P O R T

INSTITUT FÜR INFORMATIONSSYSTEME

ARBEITSBEREICH WISSENSBASIERTE SYSTEME

A SOLVER FOR QBFS IN NEGATION NORMAL FORM

Uwe Egly Martina Seidl Stefan Woltran

INFSYS RESEARCH REPORT 1843-08-03

MARCH 2008

Institut für Informationssysteme
AB Wissensbasierte Systeme
Technische Universität Wien
Favoritenstrassße 9-11
A-1040 Wien, Austria
Tel: +43-1-58801-18405
Fax: +43-1-58801-18493
sek@kr.tuwien.ac.at
www.kr.tuwien.ac.at

TU WIEN

INFSYS RESEARCH REPORT
INFSYS RESEARCH REPORT 1843-08-03, MARCH 2008

A SOLVER FOR QBFS IN NEGATION NORMAL FORM

Uwe Egly[1] and Martina Seidl[2] and Stefan Woltran[3]

Abstract. Various problems in artificial intelligence 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 describe 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 several problems from the area of artificial intelligence. We prove properties of the algorithms generalized to non-clausal form by using a novel approach based on a sequent-style formulation of the calculus.

[1]Institute of Information Systems, Knowledge-Based Systems Group, TU Vienna, Favoritenstraße 9-11, A-1040 Vienna, Austria. Email: uwe@kr.tuwien.ac.at

[2]Institute of Software Technology and Interactive Systems, Business Informatics Group, TU Vienna, Favoritenstraße 9-11, A-1040 Vienna, Austria. Email: seidl@big.tuwien.ac.at

[3]Institute of Information Systems, Database and AI Group, TU Vienna, Favoritenstraße 9-11, A-1040 Vienna, Austria. Email: woltran@dbai.tuwien.ac.at

Acknowledgements: This work was supported by the Austrian Science Fund (FWF) under grant P18019, the Austrian Academic Exchange Service (ÖAD) under grant Amadée 2/2006, and by the Austrian Federal Ministry of Transport, Innovation and Technology BMVIT and the Austrian Research Promotion Agency FFG under grant FIT-IT-810806.

Contents

1 Introduction

Formal frameworks are often suitable for the representation of application problems (like planning, scheduling, formal verification, etc.) which can then be solved by automated reasoning tools. Many such problems 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. In practice, QBFs have been proven to be a useful framework for the rapid implementation of reasoning tasks from these areas (see, e.g., [2, 7, 13, 25, 26, 30, 33, 37]), mainly because there has been made a significant progress in the development of QBF solvers within the last years (cf., e.g., [28, 31] for an overview and evaluation of state-of-the-art systems).

Almost all of these solvers, however, expect the input formula to be in *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). Of course, this restriction facilitates the handling of the formula because certain assumptions about the structure can be made. But the encodings of real world problems barely result in formulas obeying such a normal form, hence an extra transformation is required. This transformation usually is performed in two steps, namely *prenexing* and afterwards transforming the resulting purely propositional matrix into CNF. The main drawbacks of this transformation procedure are:

- The prenexing operation is *not* deterministic.
- The translation into CNF results in an increase of the formula size as well as in an increase of the number of variables in the formula.
- The structure of the formula may be disrupted and scopes of quantifiers are artificially extended.

To avoid this preliminary transformation step (together with its occurring problems) in reasoning systems of the aforementioned kind, a QBF solver which allows for the processing of arbitrary QBFs (or with only weak restrictions on the syntax) is needed. In this paper, we present such a prover, qpro, which works on formulas in *negation normal form* (NNF). Such formulas are characterized by the property that negation occurs in front of atoms only, but the usage of quantifiers, conjunction, and respectively, disjunction remains unrestricted. The basic procedure of qpro is a *generalized* variant of the DPLL algorithm with enhanced dependency-directed backtracking (DDB) techniques. While the extension of the basic DPLL procedure to QBFs in NNF is rather straightforward, the interplay of simplification, and respectively, DDB techniques turns out to be technically involving. We thus investigate these aspects also in an abstract, proof-theoretic manner. Compared to other QBF systems for arbitrary formulas (for instance, those based on binary decision diagrams) the space requirements for qpro are modest, since qpro runs in polynomial space (wrt the length of the input formula).

1.1 Translations to Normal Form—Advantages and Pitfalls

As pointed out, the motivation to circumvent the restriction to formulas in PCNF and to work with QBFs in NNF instead, is the problem to generate "good" normal forms. In contrast to, for instance, first-order logic, the main problem here is the handling of quantifiers at the places where they occur. In the following we explain some aspects of normalization for different logics and discuss why QBFs are problematic in this aspect.

It is well known how a propositional (or first-order formula) can be translated into a (un)satisfiability-equivalent CNF, such that the structural information is retained by new atoms [36, 10, 9]. 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 [4, 11]. Moreover, as experiments showed [14, 32], 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 in [1].

But the problem to construct a prenex form of a QBF is still present. In particular, the prenexing transformation cannot be carried out deterministically and the chosen normalization strategy crucially influences the runtimes (also depending on the concrete solver used), see e.g., [15, 44]. 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) [5, 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 [24].

However, there is a certain difference between first-order logic and the language of QBFs in the treatment of quantifiers while prenexing a formula. 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.[1]

1.2 Towards a Practical Efficient Non-Normal Form QBF Solver

As already pointed out, in order to provide a solver for QBFs which saves us from the step of prenexing we propose as input format *negation normal form*. In fact, negation normal form is a good candidate because of several reasons.

- The negation normal form of an arbitrary QBF is unique.
- The translation of a QBF into its negation normal form is easy and runs in linear time.
- The structure of the formula is essentially retained.
- The quantifiers stay "in place".
- Contrary to the use of arbitrary QBFs, polarity considerations are restricted to atoms, which avoids numerous technical difficulties. This holds for formal details, but even more for implementation issues.

However, the price that we have to pay for a decision procedure not based on PCNFs is the necessity to generalize its optimization mechanisms to the broader class of QBFs in NNF in order to obtain an efficient implementation of the DPLL algorithm. Indeed, our goal is to provide a solver which is comparably efficient on PCNFs but, in general, better on non-prenex formulas compared to the combination of a transformation to normal form and the application of a PCNF solver.

[1] There is one solver, sKizzo [6], which introduces Skolem functions in an intermediate step.

The basic techniques we shall consider in a generalized manner for our solver are the following. First, we need a mechanism to recognize special kinds of literals in order to significantly simplify the currently processed formulas. Two such methods are well known. The concept of a *unit literal* amounts to identify clauses with a single literal. Similarly, it is desirable to find an atom that occurs only positively or only negatively in the formula (a *pure literal*). In both cases, we can immediately force the "correct" assignment to the corresponding atom. Both detections are very easy for CNFs, and moreover, the respective manipulation of the formula can be handled efficiently using suitable data structures. For QBFs in NNF, a general detection of unit-like literals is more involved. In fact, we shall distinguish two forms of unit literals, called *global* and *local*. The detection of pure literals remains rather simple by the fact that we deal with QBFs in NNF. However, the simplification of formulas is a bit more complicated than for QBFs in PCNF. As a benefit, one can make use of further techniques which are not possible in QBF solvers which require the input formula to be in PCNF. We mention here the concept of *miniscoping* which tries to reduce the scope of quantifiers (i.e., by shifting quantifiers further into the formula) which in turn may allow further simplifications.

Another technique we want to exploit is the realization of (different forms of) dependency-directed backtracking (DDB). The basic idea is to avoid unnecessary evaluations by identifying whether an atom contributes to a current evaluation of the formula. Again, the generalization from PCNF to NNF provides some technical difficulties (especially in connection with the simplification techniques mentioned before), but there are also advantages compared to PCNF solvers. In particular, we are able to apply DDB techniques to false *and* true subproblems.

1.3 Related Work

Most state-of-the-art QBF solvers process only formulas in PCNF, thus they require an additional transformation step when the formula is not available in the required format. As discussed above, the transformation to the PCNF version of a formula comes usually with certain undesirable side effects like the increasing of the number of variables.

An analysis of the prenexing operation and its disadvantages has been performed in [15, 44]. In order to avoid the discovered disadvantages, certain restrictions of the PCNF have be leveraged to find a trade-off to reuse the knowledge and experience gained in PCNF solving and to overcome the restrictions of the normal form transformation. [24], for instance, abandon the separation between quantifier prefix and and propositional matrix to minimize the scope of the quantifiers and report that this technique allows for more efficient solving of certain QBFs. In other words, the input basically remains in PCNF but now it comes together with information on quantifier dependencies. Also in [6], a speed-up gained by the reconstruction of the quantifier tree is reported. The concept of miniscoping is implicitly also performed in expansion-based systems like `quantor` [8] to avoid unnecessary duplications of certain formula parts. However, all these techniques still have the disadvantage that an arbitrary input has to be transformed into PCNF or a PCNF-like format (see [24]) first.

To abstain from the CNF structure has also been proposed in BDD-based approaches (see, e.g., [20]), and for the solver QuBos [3] which eliminates one type of variables by expansion and numerous optimizations (including miniscoping), and then passes the resulting formula to a SAT solver. However, both approaches require exponential space in the worst case.

The impact of allowing a CNF structure combined with a DNF (disjunctive normal form structure) is explored in [42, 38] where the formulas consist of a CNF part and a DNF part. This is a very natural way to include learned clauses and their dual part but nevertheless the formula structure is very restricted and the problematic normal form transformations cannot be circumvented.

1.4 Organization

The outline of the paper is as follows: Section 2 introduces necessary definitions and notations. In Section 3, we present generalizations of elimination rules for unit and pure literals. We consider the formal underpinnings of our solver in Section 4, where we follow a proof-theoretical approach by using the means of a sequent calculus. Afterwards, we present our system qpro, which implements the decision procedure. In Section 6, we first discuss the impact of enabling/disabling the previously discussed optimization techniques. Then we compare our solver with state-of-the-art provers on benchmarks including problems from the area of AI, viz. counterfactual reasoning [21, 17] and correspondence checking [18, 34] in answer-set programming [19], as well as known benchmarks from the area of modal logic [35]. Finally, we discuss the obtained results and give further pointers to related and future work in Section 7.

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 parentheses, 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 $\mathcal{P}$ of variables 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 $\phi \in \mathcal{L}_\mathcal{P}$ and $x \in \mathcal{P}$, 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. Note that we allow negation only in front of a variable or a truth constant. Formulas which obey this restriction are usually referred to be in *negation normal form* (NNF). However, the NNF of any arbitrary QBF, i.e., formulas where negations may occur anywhere in the formula, can be obtained by iteratively 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 the restricted language $\mathcal{L}_\mathcal{P}$. Unless stated otherwise, we assume that the occurrence of a quantifier is unique for any QBF. Hence, for each $x \in \mathcal{P}$, there is at most one occurrence of $\mathsf{Q}x$ allowed in a QBF. Again, this is not a serious restriction, since any QBF can be brought into this form by a suitable variable renaming.

The *scope* of a quantifier $\mathsf{Q}x$ in a QBF ϕ is defined to be 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 *free* in a QBF ϕ if it is not located within the scope of the quantifier $\mathsf{Q}x$ in ϕ. By $\mathsf{free}(\phi)$ we denote the set of variables occurring free in ϕ. A QBF ϕ is *closed* if there are no free variables in ϕ. A variable x is called existential (resp. universal) in ϕ if it is located in ϕ within the scope of a quantifier $\exists x$ (resp. $\forall x$). Variables which are either existential or universal in ϕ are also said to be *bound* in ϕ. We use, for a variable x, $\overline{x} = \neg x$ and $\overline{\neg x} = x$. For a literal l of the form x or $\neg x$, the function $\mathsf{var}(l)$ returns the variable x. Furthermore, the function $\mathsf{pol}(\phi, x)$ returns the polarity of a propositional variable x in a QBF ϕ. Possible values of $\mathsf{pol}(\phi, x)$ are *pos*, *neg*, *both*, and *none*. Finally, given a QBF ϕ, a variable $x \in \mathcal{P}$, and $y \in \mathcal{P} \cup \{\top, \bot\}$, we denote by $\phi[x/y]$ the result of substituting each occurrence of x ϕ by y.

The *propositional skeleton* $\mathsf{psk}(\phi)$ of a QBF ϕ provides a corresponding quantifier-free formula of ϕ and is recursively defined as follows:

$$\mathsf{psk}(\phi) = \begin{cases} \phi & \phi \in \{x, \neg x \mid x \in \mathcal{P}\} \cup \{\top, \bot\} \\ \mathsf{psk}(\psi_1) \circ \mathsf{psk}(\psi_2) & \text{if } \phi = \psi_1 \circ \psi_2, \circ \in \{\vee, \wedge\} \\ \mathsf{psk}(\psi) & \text{if } \phi = \mathsf{Q}x\ \psi, \mathsf{Q} \in \{\exists, \forall\} \end{cases}$$

To define the semantics of QBFs, we introduce an evaluation function $\upsilon_{\mathcal{S}}$ with respect to a set $\mathcal{S}$ of literals assigning the truth value for free variables. We sometimes omit the subscript $\mathcal{S}$, if it is clear from the context. In particular, this can be done for any closed QBF, on which we shall focus later. However, since we also deal with open (mostly quantifier-free, i.e., propositional) formulas, we define the semantics in such a general way. In what follows we use $\mathcal{S} \uplus \{l\}$ as a shorthand for $(\mathcal{S} \setminus \{\bar{l}\}) \cup \{l\}$, where l is a literal.

Definition 1 (Evaluation Function) *Let ϕ be a QBF and let $\mathcal{S}$ be a set of literals such that for each $x \in \mathsf{free}(\phi)$, either $x \in \mathcal{S}$ or $\neg x \in \mathcal{S}$. The* evaluation function $\upsilon_{\mathcal{S}} : \mathcal{L}_{\mathcal{P}} \to \{1, 0\}$ *is given as follows:*

1. $\upsilon_{\mathcal{S}}(\top) = \upsilon_{\mathcal{S}}(\neg\bot) = 1$ *and* $\upsilon_{\mathcal{S}}(\bot) = \upsilon_{\mathcal{S}}(\neg\top) = 0$
2. $\upsilon_{\mathcal{S}}(l) = \begin{cases} 1 & \text{if } l \in \mathcal{S} \\ 0 & \text{if } \bar{l} \in \mathcal{S} \end{cases}$
3. $\upsilon_{\mathcal{S}}(\phi_1 \vee \phi_2) = \begin{cases} 1 & \text{if } \upsilon_{\mathcal{S}}(\phi_1) = 1 \text{ or } \upsilon_{\mathcal{S}}(\phi_2) = 1 \\ 0 & \text{otherwise} \end{cases}$
4. $\upsilon_{\mathcal{S}}(\phi_1 \wedge \phi_2) = \begin{cases} 1 & \text{if } \upsilon_{\mathcal{S}}(\phi_1) = 1 \text{ and } \upsilon_{\mathcal{S}}(\phi_2) = 1 \\ 0 & \text{otherwise} \end{cases}$
5. $\upsilon_{\mathcal{S}}(\forall x\phi') = \begin{cases} 1 & \text{if } \upsilon_{\mathcal{S}'}(\phi') = 1, \text{ for all } \mathcal{S}' \in \{\mathcal{S} \uplus \{x\},\ \mathcal{S} \uplus \{\neg x\}\} \\ 0 & \text{otherwise} \end{cases}$
6. $\upsilon_{\mathcal{S}}(\exists x\phi') = \begin{cases} 1 & \text{if } \upsilon_{\mathcal{S}'}(\phi') = 1, \text{ for some } \mathcal{S}' \in \{\mathcal{S} \uplus \{x\},\ \mathcal{S} \uplus \{\neg x\}\} \\ 0 & \text{otherwise} \end{cases}$

Basically, this definition amounts to both closed QBFs and QBFs with free variables. For closed QBFs, we can give an alternative syntactic-driven top-down characterization, which uses (1), (3), and (4) from above (recall that for closed QBFs, there is no need to refer to an assignment $\mathcal{S}$), together with

- $\upsilon(\forall x\ \phi) = \begin{cases} 1 & \text{if } \upsilon(\phi[x/\top]) = 1 \text{ and } \upsilon(\phi[x/\bot]) = 1 \\ 0 & \text{otherwise} \end{cases}$
- $\upsilon(\exists x\ \phi) = \begin{cases} 1 & \text{if } \upsilon(\phi[x/\top]) = 1 \text{ or } \upsilon(\phi[x/\bot]) = 1 \\ 0 & \text{otherwise} \end{cases}$

This characterization calls for more direct handles to treat truth constants. To this end, one can consider some basic simplifications obtained from the following equivalence-preserving transformations:

(S1) $\neg\top \Rightarrow \bot; \quad \neg\bot \Rightarrow \top;$

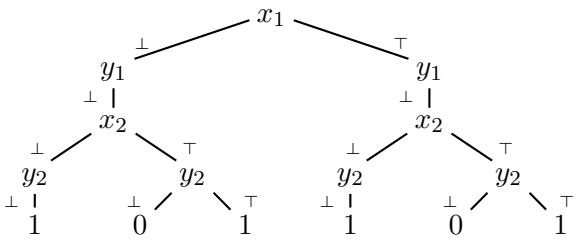

Figure 1: The branching tree of the example QBF ϕ.

(S2) $\top \wedge \phi \Rightarrow \phi$; $\quad \bot \wedge \phi \Rightarrow \bot$; $\quad \phi \wedge \top \Rightarrow \phi$; $\quad \phi \wedge \bot \Rightarrow \bot$;

(S3) $\top \vee \phi \Rightarrow \top$; $\quad \bot \vee \phi \Rightarrow \phi$; $\quad \phi \vee \top \Rightarrow \top$; $\quad \phi \vee \bot \Rightarrow \phi$;

(S4) $(\mathsf{Q}x\,\phi) \Rightarrow \phi$ where $\mathsf{Q} \in \{\forall, \exists\}$, x does not occur in ϕ;

Definition 2 *A formula is called* cleansed *if none of the simplifications (S1) – (S4) is applicable.*

The sequence of the variable assignments when evaluating a closed QBF ϕ can be illustrated by a *branching tree*. The nodes contain the *branching 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. Moreover, we usually assume that after each such step the resulting formula is additionally brought into its cleansed form. The leaves contain the resulting truth values. Hence, each branch amounts to a truth assignment for the (quantified) variables and the leaf indicates the truth value of $\mathsf{psk}(\phi)$ under this assignment.

We sometimes omit branches: If x is existentially (resp., universally) quantified and the first subproblem evaluates to true (resp., false), then the second subproblem can faithfully be omitted.

Example 1 *The branching tree of the true formula*

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

is shown in Fig. 1, and is read as follows:

1. *First, the variable x_1 is replaced by $\bot$ and cleansed via the simplifications (S2) and (S3). This yields the formula*
$$\phi' \;=\; \exists y_1 \big(\forall x_2 \exists y_2 ((x_2 \vee \neg y_2) \wedge (\neg x_2 \vee y_2))\big).$$
2. *The variable y_1 is set to an arbitrary truth value; the value $\bot$ has been chosen in Fig. 1. Alternatively, the quantification $\exists y_1$ can be removed due to (S4) because all occurrences of y_1 have been eliminated in Step (1).*
3. *Then we set the variable x_2 to $\bot$ and we obtain as a cleansed formula $\exists y_2(\neg y_2)$.*
4. *If we replace y_2 by $\bot$, the formula evaluates to 1. As y_2 is an existential variable, it is not necessary to consider the dual assignment for y_2.*
5. *Since x_2 is universally quantified, we also have to consider the second subproblem where x_2 is set to $\top$. Now, $\exists y_2 y_2$ is obtained after cleansing.*
6. *When y_2 is set to $\bot$, the formula evaluates to 0, otherwise the formula evaluates to 1.*

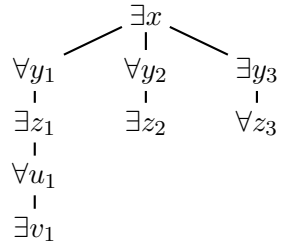

Figure 2: A quantifier dependency tree.

7. *So when x_1 is replaced by $\bot$, the resulting subproblem evaluates to true. Therefore the second branch, where x_1 is set to $\top$, has to be considered as well, since x_1 is universally quantified. This part of the evaluation works accordingly.*

The reader may already have observed that at some point the traversal of the right-hand side branch is performed exactly in the same manner as for the left-hand side. We shall make use of such situations later for advanced backtracking techniques.

Finally, we introduce prenex normal forms for QBFs. To this end, we define the following concept: Consider a QBF ϕ of the form $\exists x_1 \ldots \exists x_n \psi$ where ψ itself is *not* of the form $\exists y \psi'$. Then, we write ϕ also as $\exists X \psi$, where $X = \{x_1, \ldots, x_n\}$. For $X = \emptyset$, $\exists X \psi$ amounts to ψ. Accordingly, this concept is defined for $\forall$. In other words, we will abbreviate maximal sequences of quantifiers $\mathsf{Q}x_1 \ldots \mathsf{Q}x_n$ of the same type in a QBF by $\mathsf{Q}X$. For matters of presentation, we occasionally write $\mathsf{Q}x_1 \ldots \mathsf{Q}x_n$ also as $\mathsf{Q}x_1 \ldots x_n$.

Definition 3 *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 given in conjunctive normal form, ϕ is said to be in prenex conjunctive normal form (PCNF).

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, and the evaluation problem of QBFs $\forall X_1 \exists X_2 \ldots \mathsf{Q}_i X_i \phi$ is Π_i^P-complete.

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, 44]. 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. We say that a QBF has *depth* m, if the sequences of depending quantifiers provide at most $m-1$ alternations. The quantifier dependencies are illustrated in the *quantifier dependency tree*. To avoid a formal definition, we illustrate the basic ideas by an example.

Example 2 *Consider the QBF*

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

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 $\forall y_1$. Observe that ψ has depth 5 as witnessed by the sequence $\exists x \forall y_1 \exists z_1 \forall u_1 \exists v_1$. The quantifier dependency tree of the formula is shown in Fig. 2. The aim

of prenexing is to "linearize" *quantifier dependencies (which in fact form a partial order) without increasing the depth of the QBF, i.e., without increasing the number of quantifier alternations.*[2]

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:*

$$\begin{aligned}
\uparrow &: \exists xy_3\ \forall y_1y_2z_3\ \exists z_1z_2\ \forall u_1\ \exists v_1\ (\psi_1 \wedge \psi_2 \wedge \psi_3);\\
\exists\uparrow\forall\downarrow &: \exists xy_3\ \forall y_1y_2\ \exists z_1z_2\ \forall u_1z_3\ \exists v_1\ (\psi_1 \wedge \psi_2 \wedge \psi_3);\\
\exists\downarrow\forall\uparrow &: \exists x\ \forall y_1y_2\ \exists z_1y_3\ \forall u_1z_3\ \exists v_1z_2\ (\psi_1 \wedge \psi_2 \wedge \psi_3);\\
\downarrow &: \exists x\ \forall y_1\ \exists z_1y_3\ \forall u_1y_2z_3\ \exists v_1z_2\ (\psi_1 \wedge \psi_2 \wedge \psi_3).
\end{aligned}$$

Most of the currently available systems require their input to be in prenex conjunctive normal form, i.e., the input formulas are prenex forms and the quantifier matrix is a conjunction of disjunctions of literals. These disjunction of literals are often called *clauses*. If the input format is such a PCNF, then two steps are necessary to evaluate a formula: (i) the transformation to the normal form and (ii) the actual solving. Hence, if an arbitrary QBF has to be solved, one particular transformation into PCNF has to be chosen by the user. This choice may crucially influence the running time. For a thorough analysis of this problem and further prenexing strategies, see [44]. Note that if we only require a transformation into QBFs in NNF, this problem does not arise.

Remark. In the following, we consider closed QBFs only (unless stated otherwise).

3 Generalizing Unit and Pure Literal Detection

We already have introduced some very basic equivalence-preserving transformations (S1)–(S4) for QBFs which can be used to simplify the processed QBF and thus to decrease the search space. We recall that such transformations can indeed be applied also within a QBF. Next, we introduce three further simplifications which, in combination with (S1)–(S4), are even more crucial to increase the efficiency of solvers.

In fact, most PCNF DPLL-based QBF solvers implement two special equivalence-preserving optimization techniques: the elimination of *pure* and *unit* literals. If one of those rules is applicable on a variable, the variable can immediately be assigned, even if this variable does not appear in the outermost quantifier block.

The concept of a pure literal naturally generalizes to QBFs in NNF: A literal l is called *pure* in a QBF ϕ if its complement $\overline{l}$ does not occur in ϕ.

Theorem 1 (Pure Literal Elimination) *Let the literal $l = x$ (resp. $l = \neg x$) be pure in a closed QBF ϕ. Then*

$$\phi \text{ is equivalent to } \begin{cases} \phi[x/\top] \text{ (resp. } \phi[x/\bot]) & \text{if } x \text{ is existential;} \\ \phi[x/\bot] \text{ (resp. } \phi[x/\top]) & \text{if } x \text{ is universal.} \end{cases}$$

[2] However, this is not always possible. In fact, if we have a Boolean combination of QBFs, one additional alternation has to be taken into account. As the simplest example consider $\forall V\phi \wedge \exists V'\phi'$ with ϕ and ϕ' purely propositional. Then "minimal" linearizations are of the form $\forall V\exists V'(\phi \wedge \phi')$ or $\exists V'\forall V(\phi \wedge \phi')$. Both options, however, possess an alternation of quantifiers, whereas the original QBF does not.

Proof. We show that ϕ is equivalent to $\phi[x/\top]$ under the assumption that all occurrences of x are positive in ϕ and that x is an existential variable. The other cases are similar. Due to the Equivalence Replacement Theorem of classical logic, it suffices to prove that the subformula $\exists x\, \phi'$ of ϕ is equivalent to $\phi'[x/\top]$. (Recall that all occurrences of x in ϕ are located in the subformula ϕ'). We show that, under the given restrictions, $\exists x\, \phi' \rightarrow \phi'[x/\top]$ and $\phi'[x/\top] \rightarrow \exists x\, \phi'$ hold.
(1) $\exists x\, \phi'[x/x] \rightarrow \exists x\, \phi'[x/\top]$ is valid due to the Monotonic Replacement Theorem (i.e., $\psi \rightarrow \top$ is a tautology for all ψ). The quantifier on the right-hand side of the implication can be omitted by (S4) because all occurrences of x are removed.
(2) By semantics, $(\phi'[x/\top] \rightarrow \exists x\, \phi')$ is equivalent to $(\phi'[x/\top] \rightarrow (\phi'[x/\top] \vee \phi'[x/\bot]))$ which is clearly valid. □

A clause of a QBF in PCNF is called *unit* if it contains exactly one existential literal l. An existential literal is called unit if it occurs in a unit clause. Then such a literal can be immediately set to $\top$, and consequently, the then satisfied clause is removed.

For QBFs in NNF, we consider two cases: (i) a unit literal occurs somewhere inside the formula or (ii) a unit literal occurs in the subformula directly after the quantifier block which binds the literal's variable. Furthermore we apply the rule for disjunction and conjunction in a dual manner and we obtain the following theorems:

Theorem 2 (Local Unit Literal Elimination) *Let $(l \circ \phi')$, with $\circ \in \{\vee, \wedge\}$ be a subformula of a closed QBF ϕ and let $l = x$ (resp. $l = \neg x$). Then we can faithfully replace ϕ' in ϕ by*

$$\begin{array}{ll} \phi'[x/\top] \textit{ (resp. by } \phi'[x/\bot]\textit{)} & \textit{for } \circ = \wedge; \\ \phi'[x/\bot] \textit{ (resp. by } \phi'[x/\top]\textit{)} & \textit{for } \circ = \vee. \end{array}$$

Proof. We show the case for a conjunctive subformula $x \wedge \phi'$. The other cases are similar. It is sufficient to establish that $x \wedge \phi'$ is equivalent to $x \wedge \phi'[x/\top]$. First, in case x is set to true, we clearly have $(x \wedge \phi')[x/\top] = (x \wedge \phi'[x/\top])[x/\top]$. In case, x is set to false, $(x \wedge \phi')[x/\bot]$ is equivalent to $(x \wedge \phi'[x/\top])[x/\bot]$, since $(x \wedge \phi')[x/\bot] = \bot \wedge \phi'[x/\bot]$ reduces to $\bot$ and so does $(x \wedge \phi'[x/\top])[x/\bot] = \bot \wedge (\phi'[x/\top])[x/\bot]$. □

The combination of the pure and local literal elimination can be summarized as a so-called global unit literal elimination rule, which is defined as follows:

Theorem 3 (Global Unit Literal Elimination) *Let $\mathsf{Q}x(l \circ \phi')$ be a subformula of a closed QBF ϕ where $l = x$ (resp. $l = \neg x$). Then*

$$\phi \textit{ is equivalent to } \begin{cases} \phi[x/\top] \textit{ (resp. } \phi[x/\bot]\textit{)} & \textit{if } x \textit{ is existential in } \phi; \\ \phi[x/\bot] \textit{ (resp. } \phi[x/\top]\textit{)} & \textit{if } x \textit{ is universal in } \phi. \end{cases}$$

Proof. Suppose the case of a subformula $\exists x(x \circ \phi')$ in ϕ. The other cases proceed accordingly. Recall that all occurrences of x in ϕ have to be located in the subformula $\exists x(x \circ \phi')$.

If $\circ = \vee$, we obtain, by applying local unit literal elimination, $\exists x(x \vee \phi'[x/\bot])$ which turns x to be pure in ϕ. By Theorem 1, we can turn the subformula into $\exists x(\top \vee \phi'[x/\bot])$ which is however equivalent to $\exists x(\top \vee \phi'[x/\top])$, see (S2).

If $\circ = \wedge$, local unit literal elimination yields $\exists x(x \wedge \phi'[x/\top])$. Again this turns x to a pure literal in ϕ. Now, Theorem 1 immediately yields that the desired subformula $\exists x(\top \wedge \phi'[x/\top])$ is equivalent to $\exists x(x \wedge \phi')$. □

Example 3 *Consider the QBF*

$$\forall x_2 \exists x_1 \exists y_1 (((x_2 \vee \neg y_1) \wedge (\neg x_2 \vee y_1)) \vee (x_1 \wedge \neg y_1)).$$

This QBF can be evaluated as follows:

- *The pure variable x_1 can be replaced by $\top$. After simplifications, we obtain the formula $\forall x_2 \exists y_1 (((x_2 \vee \neg y_1) \wedge (\neg x_2 \vee y_1)) \vee \neg y_1)$.*
- *Now, $\neg y_1$ is global unit and all occurrences of y_1 can therefore be replaced by $\bot$.*
- *We can delete the quantifier $\exists y_1$ and obtain $\forall x_2 (((x_2 \vee \neg \bot) \wedge (\neg x_2 \vee \bot)) \vee \neg \bot)$. A few further basic simplifications show that the QBF evaluates to* 1.

4 A Sequent Calculus for QBFs

One of the most prominent and successful decision procedures for SAT is the method developed by Davis, Putnam, Logemann, and Loveland (DPLL for short). The DPLL procedure has been adapted for QBFs (in PCNF) and is implemented in most state-of-the art solvers. We generalize DPLL in such a manner that it can be used for formulas in NNF and allows to omit the transformation to prenex conjunctive normal form.

In the literature, DPLL is usually presented in a procedural manner. Because of its close relation to the semantics of QBFs, elaborations on formal properties are neglected. In this work, we present a declarative validity characterization for QBFs by the means of a sequent calculus. We call this calculus GQBF. The implementation of this calculus however results in a generalized variant of DPLL, which is presented in the following section. We consider this approach because it allows us to abstract from many control aspects of the concrete algorithm and to focus on the actual decision procedure. In particular, this will allow us to show the correctness of properties, which are central to dependency-directed backtracking, in a very straightforward way.

4.1 The Basic Decision Procedure

We characterize the calculus GQBF for closed QBFs in NNF in terms of the logical rules shown in Fig. 3. Subsequently, we use *indexed QBFs*, i.e., $\top$, $\bot$, and each variable $x \in \mathcal{P}$ carry indices which are (possibly empty) sets $\sigma \subseteq \mathcal{P}$ of variables. For the matter of presentation, we shall refer to and manipulate such indices in a string-like fashion. For an indexed element s_σ, $\overline{s}_\sigma$ is as expected, i.e., $\overline{\top}_\sigma = \bot_\sigma$, $\overline{\bot}_\sigma = \top_\sigma$, $\overline{x}_\sigma = \neg x_\sigma$, $\neg \overline{x}_\sigma = x_\sigma$. Also note that variable symbols in quantifiers will remain without an index, and that the same variable may contain different indices for different occurrences.

Our basic calculus uses such indices only to mark truth constants with the variable which it has been replaced, but later when simplifications are added to the calculus, we will make use of indices in a more involved manner to recognize which of the variables are relevant for the current proof.

Sequents of the form $\vdash \phi$ (where ϕ is an indexed QBF) are manipulated until no rule application is possible anymore. In inference rules, the sequents above a line are called the *premises* and the sequent below is called the *conclusion*. The single formula in the conclusion is called the *principal formula* of the inference.

A sequent $\vdash \phi$ is called *valid* if ϕ is valid. Our sequents consist of exactly one formula in NNF. Therefore, we cannot prove formulas like $x \vee \neg x$ directly in this calculus but only $\exists x\,(x \vee \neg x)$. This restriction is possible since we deal with closed formulas only.

$$\frac{\vdash \phi}{\vdash \phi \vee \psi}\ (\vee') \qquad \frac{\vdash \psi}{\vdash \phi \vee \psi}\ (\vee'') \qquad\qquad \frac{\vdash \phi \quad \vdash \psi}{\vdash \phi \wedge \psi}\ (\wedge)$$

$$\frac{\vdash \phi[x_\sigma/\bot_{\sigma x}]}{\vdash \exists x\phi}\ (\exists') \qquad \frac{\vdash \phi[x_\sigma/\top_{\sigma x}]}{\vdash \exists x\phi}\ (\exists'') \qquad\qquad \frac{\vdash \phi[x_\sigma/\bot_{\sigma x}] \quad \vdash \phi[x_\sigma/\top_{\sigma x}]}{\vdash \forall x\phi}\ (\forall)$$

Figure 3: The logical rules of the calculus GQBF.

A *derivation* in this calculus is a tree generated by the bottom-up application of the rules. The root of this tree is called the *end-sequent*. The leaves are sequents which contain only a (possibly negated) truth constant. A *proof* is a derivation whose leaves are labeled with axioms. *Axioms* are sequents of the form $\vdash \top_\sigma$ or $\vdash \neg\bot_\sigma$. If a formula or a formula part is unsatisfiable, the corresponding branches result in *non-axioms*, i.e. , in $\vdash \bot_\sigma$ or $\vdash \neg\top_\sigma$.

An example of a proof in the GQBF calculus is given in Fig. 4; note that this proof is not the smallest possible. All rules operate on the main connective of the principal formula but nevertheless the evaluation of a formula is not completely done deterministically. When we have a disjunction or an existential quantifier as main connective and the first branch does not result in a proof, we have to try the second branch.[3]

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash \neg\bot_y \quad \vdash \top_z}{\vdash \neg\bot_y \wedge \top_z}\ (\wedge)}{\vdash (\neg\bot_y \wedge \top_z) \vee \bot_x}\ (\vee')}{\vdash \exists z((\neg\bot_y \wedge z) \vee \bot_x)}\ (\exists'')}{\vdash \exists y\exists z((\neg y \wedge z) \vee \bot_x)}\ (\exists') \qquad \dfrac{\dfrac{\dfrac{\dfrac{\vdash \neg\bot_y \quad \vdash \top_z}{\vdash \neg\bot_y \wedge \top_z}\ (\wedge)}{\vdash (\neg\bot_y \wedge \top_z) \vee \top_x}\ (\vee')}{\vdash \exists z((\neg\bot_y \wedge z) \vee \top_x)}\ (\exists'')}{\vdash \exists y\exists z((\neg y \wedge z) \vee \top_x)}\ (\exists')}{\vdash \forall x\exists y\exists z((\neg y \wedge z) \vee x)}\ (\forall)$$

Figure 4: A proof for the formula $\forall x\exists y\exists z((\neg y \wedge z) \vee x)$.

The soundness of GQBF can be easily proven by showing that the application of a rule to its valid premise(s) leads to a valid conclusion. Completeness is established accordingly the other way round (i.e. bottom-up) by structural induction showing that a proof of the end-sequent $\vdash \phi$ can be constructed, whenever the QBF ϕ is valid. Proving the termination is established by arguing that the formula in the conclusion of each rule is more complex then the formula(s) in the premise(s). Moreover, in each rule, we have at most two possible choices for the premise. We do not give the proofs for correctness and completeness here as they are standard for sequent calculi.

Theorem 4 *The calculus* GQBF *is sound, complete, and terminating.*

[3]When disproving a formula, it would be necessary to consider the second branch for the according rules for conjunction and universal quantifier if the first branch is a proof. However, in what follows we omit a discussion on disproving, which can be considered in a fully dual manner.

This simple calculus shall provide the basis for a powerful decision procedure. In the rest of the section, we discuss necessary techniques to prune the search space (for finding proofs) and to make the proofs as small as possible which is crucial for the practical application in a solver.

4.2 Dependency-Directed Backtracking

Consider again the proof in Fig. 4 for the formula $\forall x\phi = \forall x \exists y \exists z((\neg y \wedge z) \vee x)$. The rule which is applied backwards to the end-sequent is ($\forall$) and results in two branches. The left branch contains the proof for $\phi[x/\bot_x]$, whereas the proof of $\phi[x/\top_x]$ is shown on the right side. The two proofs are of the same form except for the replacement of x. Furthermore, the variable x does not play a role for the completion of the proof because none of the truth constants in the axioms have been introduced due to the replacement of x. So x turns out to be *irrelevant* for the proof of $\phi[x/\bot_x]$. As a consequence, the replacement of x by $\top_x$ would result in a proof anyway, because $\bot_x$ does not appear in any axiom. Consequently, the second branch of the application of the ($\forall$)-rule can be faithfully omitted. This technique of the identification of irrelevant variables is known as *dependency-directed backtracking* (DDB).

Dependency-directed backtracking (or synonymously back-jumping) is an important pruning technique for DPLL and known to be crucial for the efficient practical application of DPLL in the PCNF case (see for example [29, 23]). DDB can also be generalized for non-PCNF formulas. Here it becomes even more powerful because it works for true and false problems dually. In contrast, when dealing with PCNF-formulas, a distinction has to be made. This is due to the fact that a PCNF-formula evaluates to false when *one clause* is unsatisfiable whereas it evaluates to true when *one literal* in *each clause* makes the whole formula true.

We now introduce the concept of relevant variables for a proof S, which we also call a *reason for* S.

Definition 4 *Let S be a proof and $\mathsf{ax}(S)$ the set of all axioms of S. Then the* reason $\mathsf{R}(S)$ *of a proof S is defined as*

$$\mathsf{R}(S) = \{x \mid x \in \sigma \text{ and } \vdash \top_\sigma \in \mathsf{ax}(S) \text{ or } \vdash \neg\bot_\sigma \in \mathsf{ax}(S)\}.$$

The central observation is now as follows.

Theorem 5 *Let S be a proof of $\phi[x/s_x]$ with $s \in \{\top, \bot\}$, and $x \notin \mathsf{R}(S)$. Then there exists a proof S' for $\phi[x/\overline{s}_x]$.*

Proof. Assume there exists a proof for $\phi[x/s_x]$, $x \notin \mathsf{R}(S)$, but there exists no proof for $\phi[x/\overline{s}_x]$. Since $\phi[x/s_x]$ and $\phi[x/\overline{s}_x]$ are exactly of the same structure besides the replacements of the occurrences of the variable x, we can obtain a derivation with $\phi[x/\overline{s}_x]$ as end-sequent where exactly the same rules as in the proof of $\phi[x/s_x]$ are used. This derivation must contain some non-axioms of the form $\vdash \bot_y$ or $\vdash \neg\top_y$, since we assume that there is no proof for $\phi[x/\overline{s}_x]$. However, since the only change we perform compared to the proof of $\phi[x/s_x]$ is the exchange of $\top_x$ and $\bot_x$, the only candidate for y is in fact x. But then $x \notin \mathsf{R}(S)$ cannot be the case and a contradiction occurs. Therefore $\phi[x/\overline{s}_x]$ also has a proof. □

This observation allows in particular to "duplicate" the proof of $\phi[x/s_x]$. In other words, in constructing the proof for a formula $\forall x\phi$, it is possible to "omit" the other branch in case we found a proof S for the first branch with $x \notin \mathsf{R}(S)$.

In the case that the formula is unsatisfiable, this method works dually. Then the pruning is done on existential variables, i.e., if the first problem does not have a proof and the corresponding variable does not

occur in any of the non-axioms (which must be explicitly stated now), then the examination of the second problem can be omitted.

4.3 Simplifications

We now show that our main result on dependency-directed backtracking also holds if numerous simplifications are considered in this calculus. These simplifications are crucial to prune the search space in order to speed-up our proposed procedure.

Our strategy here is as follows. We add to the logical rules of the calculus (which operate on the main connective of the principal formula) simplification rules (which operate inside such a formula). We then show certain permutation properties between logical rules and simplification rules. Under a permutation we understand that "shifting up" simplification rules does not severely modify the proof unless simplifications are "absorbed" by standard rules. We are then able to compile away simplifications when we iteratively treat the simplifications in such a way that we start with the upper most (i.e., such that there is no further simplification rule between the axioms and its application). The resulting proof without simplifications can be longer than the original proof, but its reason is not increased.

For the simplification rules, we use the following notation

$$\frac{\vdash \phi(\psi')}{\vdash \phi(\psi)}\ (s)\,\text{Cond}$$

which states that a subformula ψ is replaced by the logically equivalent formula ψ' in the premise, and Cond states an additional condition when this rule is allowed to be applied.

Moreover, we write ϕ_σ to denote the formula which results from ϕ by appending σ to each index π of both variables and truth constants in ϕ. For instance, if $\top_\pi$ occurs in ϕ then this occurrence is rewritten to $\top_{\pi\sigma}$ in ϕ_σ.

We next formulate the simplifications (S1)–(S4) as well as local unit, global unit, and pure as such rules in Fig. 5. However, we omit some symmetric cases. Note that the so extended calculus remains correct, complete, and terminating.

As already discussed above, we permute simplifications towards the axioms until they disappear or we can replace them by a small proof without simplifications. Eventually, we obtain a (possibly longer) proof without simplifications. Moreover, the reason for this resulting proof is a subset of the original one.

The basic tool to "bubble-up" simplifications is the permutation of adjacent inferences[4] We consider permutation schemata (R_s, R_l) where R_s is one of the simplification rules and R_l is one of the logical rules.

Definition 5 *A simplification rule R_s is* permutable over *a logical rule R_l (towards the axioms), if, for all applications r_s of R_s of the form*

$$\frac{\vdash \phi(\psi')}{\vdash \phi(\psi)}\ (s)\ \textit{Cond}$$

and all applications r_l of R_l such that the principal formula of r_s is closed, r_l occurs immediately above r_s, and such that ψ' is not the consequent formula of r_l, it holds that there is a proof of the consequent of r_s from the premises of r_l in which zero or more applications of R_s occur above application of R_l and no other rules are applied.

[4]This is a common technique; see [27, 41].

$$\frac{\vdash \phi(\bot_\sigma)}{\vdash \phi(\neg\top_\sigma)}\ \text{(S1a)} \quad \frac{\vdash \phi(\top_\sigma)}{\vdash \phi(\neg\bot_\sigma)}\ \text{(S1b)}$$

$$\frac{\vdash \phi(\psi_\sigma)}{\vdash \phi(\top_\sigma \wedge \psi)}\ \text{(S2a)} \quad \frac{\vdash \phi(\bot_\sigma)}{\vdash \phi(\bot_\sigma \wedge \psi)}\ \text{(S2b)}$$

$$\frac{\vdash \phi(\top_\sigma)}{\vdash \phi(\top_\sigma \vee \psi)}\ \text{(S3a)} \quad \frac{\vdash \phi(\psi)}{\vdash \phi(\bot_\sigma \vee \psi)}\ \text{(S3b)}$$

$$\frac{\vdash \phi(\psi)}{\vdash \phi(\mathsf{Q}x\psi)}\ \text{(S4) no occurrences of } x_\sigma \text{ in } \psi$$

$$\frac{\vdash \phi(x_\sigma \wedge \psi[x_{\sigma'}/\top_{\sigma\sigma'x}])}{\vdash \phi(x_\sigma \wedge \psi)}\ \text{(LU1a)} \quad \frac{\vdash \phi(\neg x_\sigma \wedge \psi[x_{\sigma'}/\bot_{\sigma\sigma'x}])}{\vdash \phi(\neg x_\sigma \wedge \psi)}\ \text{(LU1b)}$$

$$\frac{\vdash \phi(x_\sigma \vee \psi[x_{\sigma'}/\bot_{\sigma\sigma'x}])}{\vdash \phi(x_\sigma \vee \psi)}\ \text{(LU2a)} \quad \frac{\vdash \phi(\neg x_\sigma \vee \psi[x_{\sigma'}/\top_{\sigma\sigma'x}])}{\vdash \phi(\neg x_\sigma \vee \psi)}\ \text{(LU2b)}$$

$$\frac{\vdash \phi((x_\sigma \circ \psi)[x_{\sigma'}/\top_{\sigma'x}])}{\vdash \phi(\exists x(x_\sigma \circ \psi))}\ \text{(GU1a)} \quad \frac{\vdash \phi((\neg x_\sigma \circ \psi)[x_{\sigma'}/\bot_{\sigma'x}])}{\vdash \phi(\exists x(\neg x_\sigma \circ \psi))}\ \text{(GU1b)}$$

$$\frac{\vdash \phi((x_\sigma \circ \psi)[x_{\sigma'}/\bot_{\sigma'x}])}{\vdash \phi(\forall x(x_\sigma \circ \psi))}\ \text{(GU2a)} \quad \frac{\vdash \phi((\neg x_\sigma \circ \psi)[x_{\sigma'}/\top_{\sigma'x}])}{\vdash \phi(\forall x(\neg x_\sigma \circ \psi))}\ \text{(GU2b)}$$

$$\frac{\vdash \phi(\psi[x_\sigma/\top_{\sigma x}])}{\vdash \phi(\exists x\psi)}\ \text{(P1a) no occurrence of } \neg x_{\sigma'} \text{ in } \psi$$

$$\frac{\vdash \phi(\psi[x_\sigma/\bot_{\sigma x}])}{\vdash \phi(\exists x\psi)}\ \text{(P1b) no occurrence of } x_{\sigma'} \text{ in } \psi$$

$$\frac{\vdash \phi(\psi[x_\sigma/\bot_{\sigma x}])}{\vdash \phi(\forall x\psi)}\ \text{(P2a) no occurrence of } \neg x_{\sigma'} \text{ in } \psi$$

$$\frac{\vdash \phi(\psi[x_\sigma/\top_{\sigma x}])}{\vdash \phi(\forall x_\sigma\psi)}\ \text{(P2b) no occurrence of } x_{\sigma'} \text{ in } \psi$$

Figure 5: The simplification rules of the calculus GQBF.

Lemma 1 *Any simplification rule is permutable over a logical rule (towards the axioms), except in the cases, where the simplification rule is of type local unit (LU) on variable x and the logical rule is a quantifier rule on the same variable x.*

Proof. In the following, we first consider the cases of valid permutations, where we denote the simplification rule by (s). Recall that, in all these rules, the original subformula ψ to be replaced and the replacement ψ' are logically equivalent. The permutation schemata are as follows (we omit $\vee''$ and $\exists''$).

Case $r_l = (\vee')$. Let ψ occur in both disjuncts. Then we have:

$$\dfrac{\dfrac{\vdash \phi_1(\psi')}{\vdash \phi_1(\psi') \vee \phi_2(\psi')}\,(\vee')}{\vdash \phi_1(\psi) \vee \phi_2(\psi)}\,\text{(s)} \qquad \Longrightarrow \qquad \dfrac{\dfrac{\vdash \phi_1(\psi')}{\vdash \phi_1(\psi)}\,\text{(s)}}{\vdash \phi_1(\psi) \vee \phi_2(\psi)}\,(\vee')$$

If ψ occurs in ϕ_1 only, then the above figures can be adapted easily by replacing $\phi_2(\psi)$ by ϕ_2. If ψ occurs only in ϕ_2, then (s) is superfluous in the original proof; it can be safely omitted.

Case $r_l = (\wedge)$. We show the case where ψ occurs in both conjuncts. If it occurs only in one, then apply (s) only in the corresponding branch.

$$\dfrac{\dfrac{\vdash \phi_1(\psi') \quad \vdash \phi_2(\psi')}{\vdash \phi_1(\psi') \wedge \phi_2(\psi')}\,(\wedge)}{\vdash \phi_1(\psi) \wedge \phi_2(\psi)}\,\text{(s)} \qquad \Longrightarrow \qquad \dfrac{\dfrac{\vdash \phi_1(\psi')}{\vdash \phi_1(\psi)}\,\text{(s)} \quad \dfrac{\vdash \phi_2(\psi')}{\vdash \phi_2(\psi)}\,\text{(s)}}{\vdash \phi_1(\psi) \wedge \phi_2(\psi)}\,(\wedge)$$

Case $r_l = (\exists')$. The simplification (s) is not (L*) with variable x.

$$\dfrac{\dfrac{\vdash \phi(\psi')[x_\sigma/\bot_{\sigma x}]}{\vdash \exists x \phi(\psi')}\,(\exists')}{\vdash \exists x \phi(\psi)}\,\text{(s)} \qquad \Longrightarrow \qquad \dfrac{\dfrac{\vdash \phi(\psi')[x_\sigma/\bot_{\sigma x}]}{\vdash \phi(\psi)[x_\sigma/\bot_{\sigma x}]}\,\text{(s)}}{\vdash \exists x \phi(\psi)}\,(\exists')$$

Case $r_l = (\forall)$. The simplification (s) is not (L*) with variable x. Let $\phi^t(\psi)$ ($t \in \{\top, \bot\}$) denote $\phi(\psi)[x_\sigma/t_{\sigma x}]$. We show the case where ψ occurs in both substitution instances. If it occurs only in one, then apply (s) only in the corresponding branch.

$$\dfrac{\dfrac{\vdash \phi^{\bot}(\psi') \quad \vdash \phi^{\top}(\psi')}{\vdash \forall x \phi(\psi')}\,(\forall)}{\vdash \forall x \phi(\psi)}\,\text{(s)} \qquad \Longrightarrow \qquad \dfrac{\dfrac{\vdash \phi^{\bot}(\psi')}{\vdash \phi^{\bot}(\psi)}\,\text{(s)} \quad \dfrac{\vdash \phi^{\top}(\psi')}{\vdash \phi^{\top}(\psi)}\,\text{(s)}}{\vdash \forall x \phi(\psi)}\,(\forall)$$

For a non-permutable case, consider the following situation

$$\dfrac{R\colon\ \dfrac{\vdash \phi(x_\sigma \wedge \psi[x_{\sigma'}/\top_{\sigma\sigma' x}])[x_\sigma/\bot_{\sigma x}]}{\vdash \exists x \phi(x_\sigma \wedge \psi[x_{\sigma'}/\top_{\sigma\sigma' x}])}\,(\exists')}{\vdash \exists x \phi(x_\sigma \wedge \psi)}\,\text{(LU1a)} \qquad \Longrightarrow \qquad \dfrac{\dfrac{?}{\vdash \phi(x_\sigma \wedge \psi)[x_\sigma/\bot_{\sigma x}]}}{\vdash \exists x \phi(x_\sigma \wedge \psi)}\,(\exists')$$

where the ? indicates that it is impossible to apply (LU1a) in such a way that R is obtained. □

Lemma 2 *Let S be a proof for ϕ with simplifications. Then there exists a proof S' for ϕ without simplifications and $\mathsf{R}(S') \subseteq \mathsf{R}(S)$.*

Proof. Consider an arbitrary branch in S with a top-most simplification rule (s). If no such branch exists, S does not contain any simplification and $S' = S$. Otherwise, permute this (s) into all branches towards the axioms until no permutation is possible. Then we have the following cases.

Case 1: (s) is applied to an axiom. This is the case for (S1b) with $\psi = \neg\bot_\sigma$ and $\psi' = \top_\sigma$. Then (s) can simply be omitted without influencing $\mathsf{R}(\cdot)$.

Case 2: (s) has been permuted below an axiom. Then the indicated inner formula μ of (s) is the sequent formula and the simplifications can be replaced by a proof of μ without simplifications. If (s) is (S2a), (S3a), (S3b), (S4), then we replace (s) by $(\wedge)$, $(\vee')$, $(\vee'')$, or a corresponding quantifier rule. In the first, third and fourth case, ψ has to be instantiated either to $\top$ or to $\neg\bot$. If we compare the reasons for corresponding inference figures, we see that they are the same.

The inference rules (P1a) or (P1b) can also occur directly below an axiom if $\phi(\exists x\psi) = \exists x x$ or $\phi(\exists x\psi) = \exists x\neg x$. In this case, (P1a) and (P1b) are replaced by $(\exists'')$ and $(\exists')$ and corresponding inference figures have the same reason. Observe that (S1a), (S2b), (P2a), (P2b) do not occur here, because they cannot occur directly below axioms. The same is true for (L*) and (G*).

Case 3: The logical rule above (s) has a subformula occurrence of the inner formula of (s) as its principal formula. This implies that the inner formula of (s) occurs as the sequent formula, but, contrary to Case 2, the upper sequent is not an axiom. Observe that the inner formula cannot be a sequent formula for all rules (L*), because all sequent formulas are closed. Moreover, (S1a) cannot occur here, because S is a proof. Additionally, (S1b) and (S3a) are not relevant for this case, because their premise would be an axiom. For the other simplifications (S*) and (P1*), the handling is similar to Case 2. Let us have a look at (P2a), which is replaced by $(\forall)$.

$$\dfrac{\overset{\alpha}{\vdash \psi[x_\sigma/\bot_{\sigma x}]}}{\vdash \forall x_\sigma\psi}\ \text{(P2a)} \quad\Longrightarrow\quad \dfrac{\overset{\alpha}{\vdash \psi[x_\sigma/\bot_{\sigma x}]} \quad \overset{\beta}{\vdash \psi[x_\sigma/\top_{\sigma x}]}}{\vdash \forall x_\sigma\psi}\ (\forall)$$

Recall that α is simplification-free, because the current (s) is the topmost in the branch. The question is whether we can construct β in such a way that $\mathsf{R}(\beta) \subseteq \mathsf{R}(\alpha)$ holds. This is possible, because x_σ is replaced by $\bot_{\sigma x}$ and $\neg x$ does not occur in ψ by the condition in (P2a). Consequently, the introduced $\bot_{\sigma x}$ cannot occur in any axiom in α. Hence, when we take α and change the substitution for x_σ to $\top_{\sigma x}$, the resulting tree is also a proof and the newly introduced $\top_{\sigma x}$ do not occur in axioms. The argumentation works dually for (P2b). For the case of (GU1a), (GU1b), we can replace them by $(\exists'')$, $(\exists')$. Let us have a look at (GU2a), which is replaced by $(\forall)$. (The case for (GU2b) is symmetric.) We assume that $\neg x$ occurs in ψ, since otherwise (GU2a) can be replaced by (P2a) and the solution for the latter can be applied.

$$\dfrac{\overset{\alpha}{\vdash (x_\sigma \circ \psi)[x_{\sigma'}/\bot_{\sigma' x}]}}{\vdash \forall x(x_\sigma \circ \psi)}\ \text{(GU2a)} \quad\Longrightarrow\quad \dfrac{\overset{\alpha}{\vdash (x_\sigma \circ \psi)[x_{\sigma'}/\bot_{\sigma' x}]} \quad \overset{\beta}{\vdash (x_\sigma \circ \psi)[x_{\sigma'}/\top_{\sigma' x}]}}{\vdash \forall x(x_\sigma \circ \psi)}\ (\forall)$$

The connective $\circ$ has to be $\vee$, because otherwise the premise of (GU2a) is not provable. If x is not relevant for the proof α, we can modify α and get β with $\mathsf{R}(\alpha) = \mathsf{R}(\beta)$. If x is relevant in α (i.e. , $\neg\bot_{\sigma x}$ is an axiom), then $\mathsf{R}(\alpha)$ contains at least σx and the proof β consists of $(\vee')$ resulting in the axiom $\vdash \top_{\sigma x}$.

Case 4: The permutation is blocked because of one of the non-permutable cases involving (L*). Let us consider the case (LU1a, $\exists'$). (The case (LU1b, $\exists''$) is symmetric.)

$$\dfrac{\dfrac{\dfrac{\overset{\alpha}{\vdash \phi(x_\sigma \wedge \psi[x_{\sigma'}/\top_{\sigma\sigma' x}])[x_\sigma/\bot_{\sigma x}]}}{\vdash \exists x \phi(x_\sigma \wedge \psi[x_{\sigma'}/\top_{\sigma\sigma' x}])}\,(\exists')}{\vdash \exists x \phi(x_\sigma \wedge \psi)}\,\text{(LU1a)} \qquad \Longrightarrow \qquad \dfrac{\dfrac{\overset{\beta}{\vdash \phi(x_\sigma \wedge \psi)[x_\sigma/\bot_{\sigma x}]}}{\vdash \exists x \phi(x_\sigma \wedge \psi)}\,(\exists'')$$

Observe that α is simplification-free and that a substitution instance of $\bot_{\sigma x} \wedge \psi[x_{\sigma'}/\top_{\sigma\sigma' x}]$ does not occur as a sequent formula in α, because it is unprovable and α is a proof. Consequently, we can replace (LU1a) and $(\exists')$ by $(\exists'')$ and obtain $\mathsf{R}(\beta) \subseteq \mathsf{R}(\alpha)$.

We continue with the case (LU2a, $\exists''$). (The case (LU2b, $\exists'$) is symmetric.)

$$\dfrac{\dfrac{\dfrac{\overset{\alpha}{\vdash \phi(x_\sigma \vee \psi[x_{\sigma'}/\bot_{\sigma\sigma' x}])[x_\sigma/\top_{\sigma x}]}}{\vdash \exists x \phi(x_\sigma \vee \psi[x_{\sigma'}/\bot_{\sigma\sigma' x}])}\,(\exists'')}{\vdash \exists x \phi(x_\sigma \vee \psi)}\,\text{(LU2a)} \qquad \Longrightarrow \qquad \dfrac{\dfrac{\overset{\beta}{\vdash \phi(x_\sigma \vee \psi)[x_\sigma/\top_{\sigma x}]}}{\vdash \exists x \phi(x_\sigma \vee \psi)}\,(\exists')$$

Again, α is simplification-free. If a substitution instance of $\top_{\sigma x} \vee \psi[x_{\sigma'}/\bot_{\sigma\sigma' x}]$ occurs as a sequent formula in α, then we have two possibilities. If we take the axiom $\vdash \top_{\sigma x}$ in α, then we take it also in β. If we take the instance of ψ in α, then $\bot_{\sigma\sigma' x}$ can occur in an axiom or not (the index set may even be larger). If it occurs in an axiom, we can safely take $\top_{\sigma x}$ in β because this axiom has a smaller index set. If $\bot_{\sigma\sigma' x}$ does not occur in any axiom, we can safely replace it by $\top_{\sigma x}$. In all the cases, $\mathsf{R}(\beta) \subseteq \mathsf{R}(\alpha)$ holds.

We continue with the case (LU1a, $\forall$). (The case (LU1b, $\forall$) is symmetric.)

$$\dfrac{\dfrac{\dfrac{\overset{\alpha_1}{\vdash \phi(x_\sigma \wedge \psi[x_{\sigma'}/\top_{\sigma\sigma' x}])[x_\sigma/\bot_{\sigma x}]} \quad \overset{\alpha_2}{\vdash \phi(x_\sigma \wedge \psi[x_{\sigma'}/\top_{\sigma\sigma' x}])[x_\sigma/\top_{\sigma x}]}}{\vdash \forall x \phi(x_\sigma \wedge \psi[x_{\sigma'}/\top_{\sigma\sigma' x}])}\,(\forall)}{\vdash \forall x \phi(x_\sigma \wedge \psi)}\,\text{(LU1a)} \qquad \Longrightarrow$$

$$\dfrac{\overset{\beta_1}{\vdash \phi(x_\sigma \wedge \psi)[x_\sigma/\bot_{\sigma x}]} \quad \overset{\beta_2}{\vdash \phi(x_\sigma \wedge \psi)[x_\sigma/\top_{\sigma x}]}}{\vdash \forall x \phi(x_\sigma \wedge \psi)}\,(\forall)$$

Observe that α_1 is simplification-free and that a substitution instance of $\bot_{\sigma x} \wedge \psi[x_{\sigma'}/\top_{\sigma\sigma' x}]$ does not occur as a sequent formula in α_1, because it is unprovable and α_1 is a proof. Then we can safely modify α_1 to get β_1. Moreover, we can safely modify α_2 to get β_2 because the index set of the indicated verum in β_2 is included in the index sets indicated in α_2. We obtain $\mathsf{R}(\beta_1) \subseteq \mathsf{R}(\alpha_1)$ and $\mathsf{R}(\beta_2) \subseteq \mathsf{R}(\alpha_2)$.

Finally, we consider the case (LU2a, $\forall$). (The case (LU2b, $\forall$) is symmetric.)

$$\dfrac{\dfrac{\dfrac{\overset{\alpha_1}{\vdash \phi(x_\sigma \vee \psi[x_{\sigma'}/\bot_{\sigma\sigma' x}])[x_\sigma/\bot_{\sigma x}]} \quad \overset{\alpha_2}{\vdash \phi(x_\sigma \vee \psi[x_{\sigma'}/\bot_{\sigma\sigma' x}])[x_\sigma/\top_{\sigma x}]}}{\vdash \forall x \phi(x_\sigma \vee \psi[x_{\sigma'}/\bot_{\sigma\sigma' x}])}\,(\forall)}{\vdash \forall x \phi(x_\sigma \vee \psi)}\,\text{(LU2a)} \qquad \Longrightarrow$$

$$\dfrac{\overset{\beta_1}{\vdash \phi(x_\sigma \vee \psi)[x_\sigma/\bot_{\sigma x}]} \quad \overset{\beta_2}{\vdash \phi(x_\sigma \vee \psi)[x_\sigma/\top_{\sigma x}]}}{\vdash \forall x \phi(x_\sigma \vee \psi)}\,(\forall)$$

Again, α_1 is simplification-free. The proof α_1 can safely be transformed to β_1 because the indicated truth constants are identical and the index set becomes smaller. For α_2, the argumentation is similar to the case (LU2a,$\exists''$). Therefore, we obtain $\mathsf{R}(\beta_1) \subseteq \mathsf{R}(\alpha_1)$ and $\mathsf{R}(\beta_2) \subseteq \mathsf{R}(\alpha_2)$.

For all the other cases involving (L*) and existential quantifier rules, the simplification can safely deleted without increasing the reason. □

With Lemma 2, we immediately get the following result making use of Theorem 5.

Theorem 6 *Let S be a proof with simplifications of $\phi[x/s_x]$ with $s \in \{\top, \bot\}$ and $x \notin \mathsf{R}(S)$. Then there exists a proof S' for $\phi[x/\overline{s}_x]$.*

Note that this result provides the theoretical basis of dependency-directed backtracking in the sense that the search of a proof for $\phi[x/\overline{s}_x]$ can be faithfully omitted in case we found a proof S for $\phi[x/s_x]$ with x not being a reason for S. In particular, this effect can be exploited in searching a proof for a QBF $\forall x \phi$, as the following example illustrates.

Example 4 *Assume we want to prove the QBF*

$$\forall x \exists y \exists z \exists p \forall q \big(((\neg p \vee y) \wedge (y \vee \neg z) \wedge (p \vee (x \wedge \neg y))) \vee (\neg x \wedge q) \vee (x \wedge \neg q)\big).$$

The proof is shown in Fig. 6. We obtain only one single axiom, namely $\vdash \top_{yp}$. So the reason are the variables y and p, but not the variable x. Therefore, it is not necessary to consider the second branch of the low-most application of the $(\forall)$-rule.

Furthermore observe that it is in general not unique which simplification rule should be applied. Instead of the pure rule (P1a), we could also have applied the unit rule. Since we have already a complete calculus without the simplification rules, it is even possible to completely omit those rules. For example, we could have applied the $(\wedge)$-rule instead of the last (S2a)-rule. Then we would have obtained the two axioms $\vdash \top_y$ and $\vdash \top_p$. Nevertheless the reason remains the same.

$$
\begin{array}{cl}
\vdash \top_{yp} & \\
\hline
\vdash \top_y \wedge \top_p & \text{(S2a)} \\
\hline
\vdash \exists p(\top_y \wedge p) & \text{(GU1a)} \\
\hline
\vdash \exists p(\top_y \wedge \top_y \wedge p) & \text{(S2a)} \\
\hline
\vdash \exists z \exists p(\top_y \wedge \top_y \wedge p) & \text{(S4)} \\
\hline
\vdash \exists z \exists p\big((\neg p \vee \top_y) \wedge (\top_y \vee \neg z) \wedge p\big) & 2 \times \text{(S3a)} \\
\hline
\vdash \exists y \exists z \exists p\big((\neg p \vee y) \wedge (y \vee \neg z) \wedge p\big) & \text{(P1a)} \\
\hline
\vdash \exists y \exists z \exists p\big((\neg p \vee y) \wedge (y \vee \neg z) \wedge (p \vee \bot_x)\big) & \text{(S3b)} \\
\hline
\vdash \exists y \exists z \exists p\big((\neg p \vee y) \wedge (y \vee \neg z) \wedge (p \vee (\bot_x \wedge \neg y))\big) & \text{(S2b)} \\
\hline
\vdash \exists y \exists z \exists p\big(((\neg p \vee y) \wedge (y \vee \neg z) \wedge (p \vee (\bot_x \wedge \neg y))) \vee (\neg \bot_x \wedge \bot_q)\big) & \text{(S2b), (S3b)} \\
\hline
\vdash \exists y \exists z \exists p \forall q\big(((\neg p \vee y) \wedge (y \vee \neg z) \wedge (p \vee (\bot_x \wedge \neg y))) \vee (\neg \bot_x \wedge q)\big) & \text{(P2a)} \\
\hline
\vdash \exists y \exists z \exists p \forall q\big(((\neg p \vee y) \wedge (y \vee \neg z) \wedge (p \vee (\bot_x \wedge \neg y))) \vee (\neg \bot_x \wedge q) \vee \bot_x\big) & \text{(S3b)} \\
\hline
\vdash \exists y \exists z \exists p \forall q\big(((\neg p \vee y) \wedge (y \vee \neg z) \wedge (p \vee (\bot_x \wedge \neg y))) \vee (\neg \bot_x \wedge q) \vee (\bot_x \wedge \neg q)\big) & \text{(S2b)} \\
\hline
\vdash \forall x \exists y \exists z \exists p \forall q\big(((\neg p \vee y) \wedge (y \vee \neg z) \wedge (p \vee (x \wedge \neg y))) \vee (\neg x \wedge q) \vee (x \wedge \neg q)\big) & (\forall)
\end{array}
$$

Figure 6: Proof of ϕ with simplifications.

5 The Implementation of the Solver qpro

In the last section, we have introduced all building blocks necessary to create an efficient solver for formulas in NNF in a formal manner. The basic decision procedure can be realized by a simple search-based backtracking algorithm which works in polynomial space with respect to the size of the input formula. In the implementation, we simply apply the rules of the previously introduced calculus for a systematic proof search. In the following, we use a procedural C-like language for the pseudo-code. First of all, we start with the basic decision procedure including the optimizations achieved by the simplification rules. Then we improve this decision procedure by including dependency-directed backtracking. In combination with the backtracking algorithm, this technique is harder to realize than explained in the description of the sequent calculus because we cannot keep the whole proof in the memory and simply read off the reasons from the axioms. Furthermore we even distinguish between two different implementation variants of DDB in the solver qpro.

5.1 A Generalization of the DPLL Procedure

In Fig. 7, we present a simplified version of the basic algorithm used in our solver qpro. In each call of that procedure, which we refer to as `split`, we first apply a further function, `simplify`, in order to reduce the formula size by simplifications along the lines of the rules in Fig. 5. Then a logical rule of GQBF (see Fig. 3) is applied according to the main connective of the resulting formula. This is different to the previously introduced sequent calculus where simplifications can be performed at any stage of the proof or it can even be completely omitted. But the main difference to the calculus is the way the formula is processed. Now the search strategy is explicitly stated and if a branch does not result in a proof, then a backtracking mechanism leads to the investigation of alternative paths. Indeed, it is not the goal to construct a proof but only to obtain the evaluation result. This is, of course, crucial when implementing a solver which works with polynomial space requirements (with respect to the length of the input).

In contrast to the calculus where the way how a proof is found is of minor interest, we are now concerned with figuring out a search strategy which yields the truth value of QBFs. The procedure `split` is straightforward but we have to face three sources of indeterminism within the `switch` statement: when `simplify` returns a QBF of the form $\phi_1 \circ \phi_2$, we have to select (i) which subformula to evaluate first; this part differs from PCNF solvers where such a decision is not necessary. If a formula $\mathsf{Q}X\phi$ ($\mathsf{Q} \in \{\forall, \exists\}$) with a quantifier as a main connective is returned, then there is a certain freedom of choice concerning the variable selection. It is (ii) possible to choose any variable from the first quantifier block without changing the evaluation result. The last indeterminism (iii) concerns the choice of the variable assignment which is considered first during splitting. To keep the pseudo-code as simple as possible, we avoid such considerations and process the formula "from left to right". In practical implementations, selection heuristics can be incorporated in order to improve the algorithm accordingly.

5.2 The Implementation of `simplify`

The function `simplify`, shown in Fig. 8, is applied to the QBF ϕ recursively, until we reach a fix point with respect to the simplifications in order to reduce the formula size. Since `simplify` also triggers the application of the unit and pure rules, a concrete order of the simplifications has to be chosen. The order is as follows:

```
BOOLEAN split(φ) {
/* In:   closed QBF φ in NNF */
/* Out:  {1,0} */

  φ' = simplify(φ);

  switch(φ')
    case ⊤          :   return 1;
    case ⊥          :   return 0;

    case (φ1 ∨ φ2)  :  return (split(φ1) ∥ split(φ2));
    case (φ1 ∧ φ2)  :  return (split(φ1) && split(φ2));

    case ∃xψ        :  return (split(ψ[x/⊥]) ∥ split(ψ[x/⊤]));
    case ∀xψ        :  return (split(ψ[x/⊥]) && split(ψ[x/⊤]));
}
```

Figure 7: The basic algorithm.

- For a formula $\psi_1 \circ \psi_2$, it is first checked whether ψ_1 or ψ_2 is a literal to apply the local unit rule, otherwise we apply simplifications to each of the two subformulas.
- Similarly, in case of a formula $\mathsf{Q}x\psi$, it is first checked whether ψ is either of the form $l \circ \psi$ (the global unit rule is then applied) or x occurs in a single polarity in ψ (the pure literal rule is then applied); otherwise the algorithm proceeds by simplifying the formula ψ.

The subprocedures of `simplify`, namely `lunit`, `gunit`, and `pure` are depicted in Fig. 9. In addition, these procedures make use of auxiliary functions $\mathsf{pol}(\phi, x)$ and $\mathsf{var}(l)$ as defined in Section 2. In particular, we use $\mathsf{pol}(l, \mathsf{var}(l))$ to determine whether a literal is positive or negative.

5.3 Dependency-Directed Backtracking

Dependency-directed backtracking in the sequence calculus was straightforward. When a quantifier rule is applied and when the first subproblem has been solved, then the second subproblem can be omitted, if the corresponding variable is not included in a reason. Depending on whether the subproblem evaluates to true or false, we mean by a reason either the set of variables occurring in the index sets of the axioms (see Definition 4), or the set of variables occurring in the index sets of *all* non-axioms.

The practical realization of DDB is not that easy because, when we use the algorithm above, we do *not* get a complete proof for a subproblem due to depth first search. Such a search regime is necessary to get a procedure which runs in polynomial space. One possibility to circumvent the problem is to collect the variables which are responsible for the evaluation result for one concrete assignment $\mathcal{S}$ in *relevance sets*. Roughly speaking, this amounts to the indexing of truth constants like in the quantifier rules in Fig. 3. Instead of the composition of indices (like in the simplification rules in Fig. 5), we evaluate the propositional skeleton of the input QBF with respect to $\mathcal{S}$. During the search we can thus assemble these relevance sets to the reasons. If a branching variable is not included in the reason, then it is not necessary to consider the second subproblem.

```
QBF simplify(φ) {
/* In:   closed QBF φ in NNF */
/* Out:  simplified QBF equivalent to φ */

  φ' = φ;

  switch(φ) {

    /* local unit */
    case (l ∘ ψ)          :  if (var(l) ∈ ψ) then φ' = simplify(lunit(ψ,l,∘));


    case (ψ1 ∧ ψ2)        :  φ' = (simplify(ψ1) ∧ simplify(ψ2));
    case (ψ1 ∨ ψ2)        :  φ' = (simplify(ψ1) ∨ simplify(ψ2));


    /* global unit */
    case (Qx(l ∘ ψ))      :  if (x==var(l)) then
                               φ' = simplify(gunit(l ∘ ψ,l,Q));

    /* pure literal */
    case (Qxψ)            :  if (pol(ψ,x) != both) then
                               ψ = pure(ψ,Q,x);

                             ψ' = simplify(ψ);
                             if x ∈ free(ψ') then φ' = Qxψ';
                             else φ' = ψ';

  }

  if (φ' != φ) then φ' = simplify(φ');


  switch(φ') {

    case (¬⊤)    :  return(⊥);     case (¬⊥)    :  return(⊤);
    case (⊥ ∧ ψ)  :  return(⊥);     case (⊤ ∨ ψ)  :  return(⊤);
    case (⊤ ∧ ψ)  :  return(ψ);     case (⊥ ∨ ψ)  :  return(ψ);
    otherwise    :  return(φ');

  }
}
```

Figure 8: The function `simplify`.

```
(QBF) lunit(ψ, l, ∘) {
/* In:   QBF ψ, literal l, connective ∘ */
/* Out:  QBF of the form l ∘ ψ' */

  if (∘ == ∧) then {
    if (pol(l, var(l)) == pos) then  return (l ∧ ψ[l/⊤]);
    else                             return (l ∧ ψ[l̄/⊥]);
  }
  if (pol(l, var(l)) == pos) then  return (l ∨ ψ[l/⊥]);
  else                             return (l ∨ ψ[l̄/⊤]});
}

(QBF) gunit(ψ, l, Q) {
/* In:   QBF ψ, literal l, quantifier Q */
/* Out:  QBF ψ with var(l) substituted by a truth constant */

  if (Q == ∀) then {
    if (pol(l, var(l)) == pos) then  return (ψ[l/⊥]);
    else                             return (ψ[l̄/⊤]);
  }
  if (pol(l, var(l)) == pos) then  return (ψ[l/⊤]);
  else                             return (ψ[l̄/⊥]);
}

(QBF) pure(ψ, Q, x) {
/* In:   QBF ψ, quantifier Q, variable x */
/* Out:  QBF ψ with var(l) substituted by a truth constant */

  if (Q == ∀) then {
    if (pol(ψ, x) == pos) then  return (ψ[x/⊥]);
    else                        return (ψ[x/⊤]);
  }
  if (pol(ψ, x) == pos) then  return (ψ[x/⊤]);
  else                        return (ψ[x/⊥]);
}
```

Figure 9: Auxiliary functions `lunit`, `gunit`, and `pure`.

Definition 6 (Set of Relevant Variables) *Let ϕ be a QBF and $\mathcal{S}$ a set of literals. We define the set of relevant variables $\mathsf{RV}_\phi(\mathcal{S})$ of $\mathcal{S}$ with respect to ϕ as $\mathsf{R}^\phi_{\mathcal{S}}(\mathsf{psk}(\phi))$, where $\mathsf{R}^\phi_{\mathcal{S}}(\psi)$ is defined as follows:*

1. *If* $v_{\mathcal{S}}(\mathsf{psk}(\phi)) = 1$ *then*

$$\mathsf{R}^{\phi}_{\mathcal{S}}(\psi) = \begin{cases} \{\mathsf{var}(l)\} & \text{if } \psi = l \text{ and } \mathsf{var}(l) \text{ is universal in } \phi; \\ \mathsf{R}^{\phi}_{\mathcal{S}}(\psi_i) & \text{if } \psi = \psi_1 \vee \psi_2,\ v_{\mathcal{S}}(\psi_i) = 1 \text{ for some } i \in \{1,2\}; \\ \mathsf{R}^{\phi}_{\mathcal{S}}(\psi_1) \cup \mathsf{R}^{\phi}_{\mathcal{S}}(\psi_2) & \text{if } \psi = \psi_1 \wedge \psi_2; \\ \{\} & \text{otherwise}. \end{cases}$$

2. *If* $v_{\mathcal{S}}(\mathsf{psk}(\phi)) = 0$ *then*

$$\mathsf{R}^{\phi}_{\mathcal{S}}(\psi) = \begin{cases} \{\mathsf{var}(l)\} & \text{if } \psi = l \text{ and } \mathsf{var}(l) \text{ is existential in } \phi; \\ \mathsf{R}^{\phi}_{\mathcal{S}}(\psi_i) & \text{if } \psi = \psi_1 \wedge \psi_2,\ v_{\mathcal{S}}(\psi_i) = 0 \text{ for some } i \in \{1,2\}; \\ \mathsf{R}^{\phi}_{\mathcal{S}}(\psi_1) \cup \mathsf{R}^{\phi}_{\mathcal{S}}(\psi_2) & \text{if } \psi = \psi_1 \vee \psi_2; \\ \{\} & \text{otherwise}. \end{cases}$$

In the definition above, we do not consider simplifications involving unit and pure literal elimination. To incorporate them, it is necessary to check whether a variable x which is included in the set of relevant variables has been assigned a truth value due to one of those rules. If this is the case, all variables which are responsible that x becomes unit or pure have to be included too. Note that the set of relevant variables is not necessarily unique. In the computation of $\mathsf{R}^{\phi}_{\mathcal{S}}(\psi)$ with $v_{\mathcal{S}}(\mathsf{psk}(\phi)) = 1$, the case for $\psi = \mathsf{psk}(\phi)$ and $\psi = \psi_1 \vee \psi_2$ where both ψ_1 and ψ_2 evaluate to true under $\mathcal{S}$ results in a choice that has to be made. In fact it would not be incorrect to include the relevant variables of both formulas, but later we will need the set of relevant variables to construct reasons to decide whether we must consider the second subproblem of a certain variable. The smaller the reason is, the better, because a variable missing in this set indicates that the second problem can be omitted. Observe that we also distinguish between variables of different quantification: if we calculate the set of relevant variables for a true subproblem, we only collect universally quantified variables, otherwise we only collect existentially quantified variables. This is because we skip the second branch in the semantic tree of a universally quantified variable only if the subproblem has been evaluated to true—otherwise we would omit the second problem anyway.

In contrast to the sequent calculus where we collect the variables included in the reason during the search, we calculate here the necessary set only if we have proven a subproblem for a certain variable assignment. This has the advantage that we obtain only the variables which are included in the reason at the end and that we do not collect variables which are later thrown away due to some optimizations or because they occur in a branch which turns out to be useless for the proof. In what follows, we present two versions of dependency-directed backtracking quite similar to the algorithms implemented in the solver semprop [29]. The first one, DDB by *labeling*, is a weaker version but with less implementational overhead, whereas the second one, DDB by *relevance sets*, has higher potential in decreasing the search space. We show how to integrate them to our core procedure `split`.

5.3.1 Dependency-Directed Backtracking by Labeling

In Fig. 10, we present our algorithm extended by *dependency-directed backtracking by labeling*. The procedure now contains two additional parameters, a set $\mathcal{S}$ of literals and a QBF Φ. The former keeps track of the current assignment and will be used to compute the set of relevant variables. The latter stands for the original input QBF and remains unchanged, whereas the first parameter ϕ is local to the procedure representing

```
BOOLEAN split(φ,S,Φ) {
/* In:   closed QBF φ in NNF, set S of literals, the input QBF Φ */
/* Out:  {1,0} */

  φ' = simplify(φ);

  switch(φ')
    case ⊤          :   for all x ∈ RV_Φ(S) setRelevant(x);
                        return 1;
    case ⊥          :   for all x ∈ RV_Φ(S) setRelevant(x);
                        return 0;

    case (φ1 ∨ φ2)  :  return (split(φ1,S,Φ) || split(φ2,S,Φ));
    case (φ1 ∧ φ2)  :  return (split(φ1,S,Φ) && split(φ2,S,Φ));

    case ∃xψ        :  setIrrelevant(x)
                       if ((split(ψ[x/⊥],S ∪ {¬x},Φ) == 0) {
                         if isIrrelevant(x) return 0;
                         else return split(ψ[x/⊤],S ∪ {x},Φ);
                       }
                       return 1;

    case ∀xψ        :  setIrrelevant(x)
                       if ((split(ψ[x/⊥],S ∪ {¬x},Φ) == 1) {
                         if isIrrelevant(x) return 1;
                         else return split(ψ[x/⊤],S ∪ {x},Φ);
                       }
                       return 0;
}
```

Figure 10: Procedure `split` enhanced by DDB by labeling.

the currently processed formula, i.e., the result of simplifications, variable substitutions, etc. To evaluate a QBF Φ, `split` is initially called via `split`$(\Phi, \emptyset, \Phi)$. The idea is as follows.

1. When branching on a variable x, we mark x as irrelevant using *setIrrelevant*(x).
2. If a leaf of the branching tree is reached, the set $\mathsf{RV}_\Phi(\mathcal{S})$ of relevant variables (see Definition 6) is determined with respect to the currently used assignment $\mathcal{S}$ of variables. Moreover, each variable $x \in \mathsf{RV}_\Phi(\mathcal{S})$ is now marked as relevant by *setRelevant*(x); other variables remain irrelevant.
3. If backtracking returns to a variable x, it is checked whether x actually has been set to relevant or not. If x is still irrelevant, the second subproblem can be omitted and one can immediately continue to backtrack.

We illustrate the algorithm on a concrete formula.

Example 5 *Let ϕ be the formula from Example 1, i.e.,*

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

The branching tree of ϕ has been shown in Fig. 1. Observe that the same subtree occurs on the left-hand side and on the right-hand side below x_1. The algorithm performs as follows (to keep the example simple, we omit unit and pure literal detection). To start we call `split`*$(\phi, \emptyset, \phi)$.*

1. *Since we omit the unit and pure rules, initially no simplification has to be performed. We branch on the variable x_1 and start with $\phi'[x_1/\bot]$ (ϕ' denotes ϕ without $\forall x_1$). Since x_1 is set to $\bot$, we put $\neg x_1$ into $\mathcal{S}$. We call* `split`*$(\phi'[x_1/\bot], \{\neg x_1\}, \phi)$. By replacing x_1 by $\bot$ in ϕ', we get the formula $\exists y_1 (\forall x_2 \exists y_2 ((x_2 \vee \neg y_2) \wedge (\neg x_2 \vee y_2)))$ after simplification.*

2. *The quantification $\exists y_1$ can be removed because y_1 does not occur in the scope of $\exists y_1$ in ϕ' anymore. This removal is performed by the function* `simplify`*. We thus do not include y_1 in the set $\mathcal{S}$ of literals.*

3. *Next we branch on the variable x_2, set it to $\bot$, and include $\neg x_2$ in the set $\mathcal{S}$. The simplification function after calling* `split` *with the according arguments returns $\exists y_2 (\neg y_2)$.*

4. *If we replace y_2 by $\bot$, a final call of* `split` *simplifies the formula to $\top$ and thus we compute the set of relevant variables, $\mathsf{RV}_\phi(\mathcal{S})$ with $\mathcal{S} = \{\neg x_1, \neg x_2, \neg y_2\}$. Since $\mathsf{psk}(\phi) = ((x_2 \vee \neg y_2) \wedge (\neg x_2 \vee y_2)) \vee (x_1 \wedge y_1)$, it can be checked that the only universal variable relevant for making $\mathsf{psk}(\phi)$ true under $\mathcal{S}$ is x_2.*

5. *Since x_2 is universally quantified and labeled relevant, we also have to consider the subproblem where the variable x_2 is replaced by $\top$. We thus have to call now* `split`*$(\exists y_2 ((\top \vee \neg y_2) \wedge (\neg \top \vee y_2)), \{\neg x_1, x_2\}, \phi)$ and after simplifications and the final replacement of y_2 by $\bot$, the formula evaluates to 0.*

6. *But setting y_2 to $\top$ yields 1. Again, the set of relevant variables with respect to the current assignment $\{\neg x_1, x_2, y_2\}$ in this branch consists only of x_2.*

7. *As x_1 is universal and the solution of the first subproblem is true, we usually should also consider the second subproblem when we backtrack. But now DDB comes into play and since x_1 was never included in the reasons for $\phi'[x_1/\bot]$, it is not necessary to consider the subproblem, where x_1 is set to $\top$.*

5.3.2 Dependency-Directed Backtracking by Relevance Sets

As already pointed out, the smaller the set of variables labeled as relevant the better it is for the solving process. If a variable is irrelevant then the second subproblem can be skipped under any circumstances. But consider the following case: we branch on an existentially quantified variable x and the first subproblem evaluates to false. As x is labeled as relevant, we have to consider the second subproblem too. Assume that also the second subproblem evaluates to false but x is not relevant for the solution of the second subproblem. However, x remains labeled and therefore x remains relevant for the problem where $\mathsf{Q}x$ is handled. If we had considered the second subproblem first, the situation would have been different: (1) the other problem could have been omitted, (2) the set of the variables labeled as relevant would have been smaller.

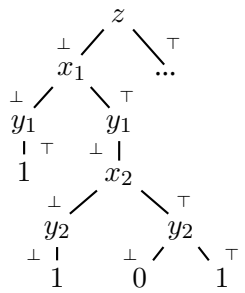

Figure 11: The branching tree of $\forall z \forall x_1 \exists y_1 (\forall x_2 \exists y_2 ((x_2 \vee \neg y_2) \wedge (\neg x_2 \vee y_2)) \vee (\neg x_1 \wedge \neg y_1 \wedge \neg z))$.

To obtain (1), it is too late, we made the "wrong" choice when deciding which subproblem to consider first. But we can get at least the smaller variable set like in (2). We collect the relevant variables of the subproblems in different sets, make some case distinctions according to the branching variable, and construct the current reason accordingly. In the sequent calculus, we are not faced with this problem because this optimization involves only the processing order of the branches. Since we have the whole proof in the sequent calculus, we can also distinguish between the two different sets because of the different available branches.

The following examples illustrates the algorithm.

Example 6 *Let ϕ be the formula*

$$\forall z \forall x_1 \exists y_1 (\forall x_2 \exists y_2 ((x_2 \vee \neg y_2) \wedge (\neg x_2 \vee y_2)) \vee (\neg x_1 \wedge \neg y_1 \wedge \neg z)).$$

The branching tree of ϕ is shown in Fig. 11. Again we omit unit and pure literal elimination to keep the example simple.

1. *We start with* `split`*$(\phi, \emptyset, \phi)$. Applying simplifications has no effect.*
2. *We branch on the variable z and start with* `split`*$(\phi'[z/\bot], \{\neg z\}, \phi)$, where ϕ' denotes ϕ without $\forall z$. We get $\forall x_1 \exists y_1 (\forall x_2 \exists y_2 ((x_2 \vee \neg y_2) \wedge (\neg x_2 \vee y_2)) \vee (\neg x_1 \wedge \neg y_1))$ after the application of* `simplify`*.*
3. *We continue by branching on the variable x_1 and set x_1 to $\bot$. We obtain the formula $\exists y_1 (\forall x_2 \exists y_2 ((x_2 \vee \neg y_2) \wedge (\neg x_2 \vee y_2)) \vee \neg y_1)$ after the according call of* `split` *and the simplifications.*
4. *Then we branch on the variable y_1 and set y_1 to $\bot$. As usual, we call the function* `split`*$(\forall x_2 \exists y_2 ((x_2 \vee \neg y_2) \wedge (\neg x_2 \vee y_2)) \vee \neg \bot, \{\neg z, \neg x_1, \neg y_1\}, \phi)$. After the simplifications, the formula evaluates to 1. The set of relevant variables includes only the universal ones, z and x_1, because they occur in the subformula of the uppermost disjunction which makes* `psk`*(ϕ) true under the current assignment $\{\neg z, \neg x_1, \neg y_1\}$.*
5. *As x_1 is universal in ϕ and marked relevant, we have also to consider the subproblem where x_1 is set to $\top$. The variable y_1 is handled as in step (2) of Example 1. So, we get $\forall x_2 \exists y_2 ((x_2 \vee \neg y_2) \wedge (\neg x_2 \vee y_2))$ after some simplifications.*
6. *We continue with branching on x_2 and set it to $\bot$ which results in $\exists y_2 \neg y_2$ after the call* `split`*$(\exists y_2 ((\bot \vee \neg y_2) \wedge (\neg \bot \vee y_2)), \{\neg z, x_1, \neg x_2\}, \phi)$ and* `simplify`*.*

7. *If we replace y_2 by $\bot$, the formula evaluates to 1 in the next step. The only variable included in the set of relevant variables is x_2 because together with the existential variable y_2, it is responsible that ϕ evaluates to 1 under this specific variable assignment.*

8. *Since x_2 is universally quantified and included in the current reason, we have also to consider the second subproblem of $\forall x_2 \exists y_2((x_2 \vee \neg y_2) \wedge (\neg x_2 \vee y_2))$ where the variable x_2 is replaced by $\top$. Simplification yields $\exists y_2 y_2$.*

9. *Setting y_2 to $\bot$ results in 0, but setting y_2 to $\top$ yields 1 after calling* `split` *and* `simplify`*. Again, the only variable included in the set of relevant variables is x_2.*

10. *When we return to x_1 during backtracking, we notice that x_1 is not relevant in the second subproblem where the variable has been replaced by $\top$. That was bad luck—if we had chosen the second subproblem as the first one to consider instead, we could have omitted setting x_1 to $\bot$ when using DDB by labeling. In DDB by relevance sets, such cases are recognized. Here the current reason is only the reason of the subproblem where x_1 is set to $\top$. DDB by labeling could not have made this distinction and the current reason would be the union of the reason of the first subproblem and the reason of the second subproblem. For DDB by relevance sets, the reason is now a smaller one: for example, the variable z is now not included in the current reason and it is not necessary to call* `split`$(\phi'[z/\top], \{z\}, \phi)$.

The pseudo-code of the implementation of `split` incorporating DDB by relevance sets is shown in Fig. 12. Again we collect the current assignment to calculate the relevance set from the original input QBF. The main difference to DDB by labeling is that we now directly calculate the current reason and return it in order to be able to minimize its size.

5.3.3 Pure and Unit Literal Elimination in Dependency-Directed Backtracking

To illustrate some of the difficulties when combining different pruning techniques, let us consider the following example.

Example 7 *Given the QBF*

$$\phi = \exists x \exists z \exists y((x \vee y) \wedge ((\neg y \wedge z) \vee (\neg y \wedge \neg z))),$$

ϕ should be evaluated by a variant of `split` *which implements the removal of truth constants and unit literals, as well as DDB by labeling. We do not include all pruning techniques in order to keep this example simple. The algorithm performs as follows.*

1. *The variable x is replaced by $\bot$. We obtain $\exists z \exists y(y \wedge ((\neg y \wedge z) \vee (\neg y \wedge \neg z)))$.*

2. *Now the unit rule can be applied on y and y is replaced by $\top$. The formula evaluates to false.*

3. *In the next step, the relevant variables have to be detected and marked. The right subformula of the outermost conjunction evaluates to false, so y and z are labeled relevant.*

4. *As x is still irrelevant, it is not necessary to consider the second subproblem where x is set to $\top$—the formula evaluates to false.*

```
(BOOLEAN, relset) split(φ,S,Φ) {
/* In:   closed QBF φ in NNF, set S of literals, the input QBF Φ */
/* Out:  ({1,0}, reason) */

  φ' = simplify(φ);

  switch(φ')
    case ⊤          :   return (1,RV_Φ(S));
    case ⊥          :   return (0,RV_Φ(S));

    case (φ1 ∨ φ2)  :  (r1,R1) = split(φ1,S,Φ);
                       if (r1 == 1) then return (1,R1);
                       (r2,R2) = split(φ2,S,Φ);
                       if (r2 == 1) then return (1,R2);
                       return (0,R1 ∪ R2);

    case (φ1 ∧ φ2)  :  (r1,R1) = split(φ1,S,Φ);
                       if (r1 == 0) then return (0,R1);
                       (r2,R2) = split(φ2,S,Φ);
                       if (r2 == 0) then return (0,R2);
                       return (1,R1 ∪ R2);

    case ∃xψ        :  (r1,R1) = (split(ψ[x/⊥],S ∪ {¬x},Φ);
                       if (r1 == 1) then return (1,R1);
                       if (x ∉ R1) then return (0,R1);

                       (r2,R2) = (split(ψ[x/⊤],S ∪ {x},Φ);
                       if (r2 == 1) then return (1,R2);
                       if (x ∉ R2) then return (0,R2);
                       return (0,R1 ∪ R2);

    case ∀xψ        :  (r1,R1) = (split(ψ[x/⊥],S ∪ {¬x},Φ);
                       if (r1 == 0) then return (0,R1);
                       if (x ∉ R1) then return (1,R1);

                       (r2,R2) = (split(ψ[x/⊤],S ∪ {x},Φ);
                       if (r2 == 0) then return (0,R2);
                       if (x ∉ R2) then return (1,R2);
                       return (1,R1 ∪ R2);
}
```

Figure 12: Procedure `split` enhanced by DDB by relevance sets.

Obviously, this result is wrong—if we had set x to $\top$ and y to $\bot$ first, the result would have been different. This is because the variable y has been assigned a truth value by the application of a special rule,

namely unit in this case. This rule is *not* applicable at any time, but only if certain preliminaries are satisfied. To understand why a formula has taken a certain value, i.e., to calculate the set of relevant variables, it is necessary to include the reasons which allowed the application of such special rules like unit or pure. In the example above, the variable x is responsible that y becomes unit, and so x has to be labeled relevant too. Then we also obtain the correct result.

The solver qpro implements a more involved variant of `simplify` than the one shown in Fig. 8. In particular, to make `simplify` work for DDB in combination with pure and unit literal elimination, the final version of `simplify` has as result not only the simplified QBF ϕ' but also an accordingly changed set $\mathcal{S}'$ of literals because pure and unit elimination contribute to the set of variable assignments.

6 Experimental Evaluation

In the following, we present the experimental evaluation of our solver. First, we investigate the impact of enabling/disabling different options of qpro; afterwards, we compare qpro to four state-of-the-art systems.

All tests were performed on an Intel Xeon 3 GHz with 4 GB of RAM with a timeout set to 100 seconds. Figures 14–19 depict the outcome of the test runs: they show the percentage of solved formulas on the ordinate related to the solving time on the abscissa. Observe that the scale of the x-coordinate is logarithmic whereas the scale of the other axis is linear. Before presenting details, we shortly describe the benchmark formulas we used.

6.1 Description of the Benchmarks

To test our implementation we have chosen four sets of benchmarks: (i) encodings of the modal logic K; (ii) encodings of nested counterfactuals; (iii) encodings of answer-set correspondence tests which are on the fourth level of the polynomial hierarchy (ASC-4); and (iv) encodings of easier answer-set correspondence tests which are on the second level of the polynomial hierarchy (ASC-2).[5] We use here the non-PCNF versions of these benchmarks, which are the direct outcome of the encodings. The four types of benchmarks differ in the complexity of the formula structure as well as in the quantifier depth. The structural differences of those sets are best illustrated by the quantifier dependencies of the formulas (see Fig. 13).

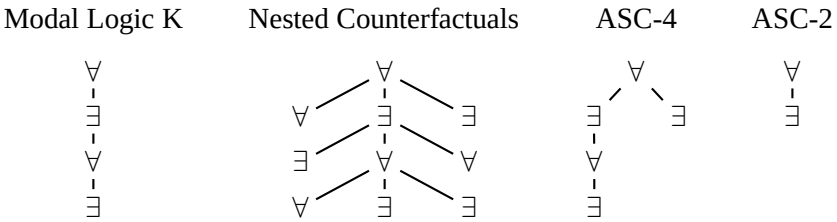

Figure 13: Quantifier dependencies of the different benchmarks.

6.1.1 Modal Logic K

This set of benchmarks contains instances which were also used in the TANCS'98 comparison of provers for modal logics. Applying the encoding from [35] yields QBFs with a linear dependency among the quantifiers.

[5] Prenexed forms of such benchmarks except the last one, i.e., ASC-2, are frequently used benchmarks and have been included in QBF solver competitions, see e.g., [31].

Hence, the translation to prenex normal form is fully determined (there is just one way to shift the quantifiers in front of the formula).

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.

6.1.2 Nested Counterfactuals

The formulas of this benchmark set encode the problem of reasoning over nested counterfactuals. The depth of the resulting QBFs ranges from four to eight. The quantifier dependency tree for depth four is shown in Fig. 13. obviously there are different ways to linearize such a QBF (see [15] for the details). For each depth, we created 50 instances, where the QBFs contain 183, 245, 309, 375, and 443 variables. The transformation to PCNF increases the number of variables to 464, 600, 786, 934, and have about 60% true and 40% false instances.

6.1.3 Answer-Set Correspondence

The formulas in this set encode correspondence tests between propositional logic programs under the answer-set semantics. It is checked whether two programs provide equally projected answer-sets under any program extension over a specified alphabet, cf. [40].

The first subset comprises 1000 instances (465 are true and 535 are false). Furthermore, for each problem, we have two different encodings: S and T. The problem of answer-set correspondence is Π_4^P-complete, and thus all QBFs in this set have depth four. As Fig. 13 indicates there are two different possibilities to obtain a linear quantifier prefix. 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 benchmarks in the second subset rely on an easier subclass of program comparisons [34] which are complete for Π_2^P. Thus the resulting QBF encodings are here of depth two. The idea of the benchmarks is to compare a program with itself but having a randomly selected rule dropped. The interesting feature of this set is that the encoded problem contains a lot of structural information, having duplicated program rules at several occasions within the encoding. It is thus interesting to see, whether this structural information is easier exploited by qpro compared to PCNF-solvers, for which this information is not directly accessible due to the preceeding transformation to normal form. We have eight different sets of such program comparisons, each containing 100 elements. The QBFs possess 127, 151, 175, 199, 223, 247, 271, and 295 variables, the translation into PCNF yields between 1000 and 1800 variables. In total, one half of the instances evaluates to true.

6.2 Internal Comparisons

First, we investigate how the enabling/disabling of different simplification rules influences the runtime behavior of qpro. We consider only the encodings of modal logic K and the encodings of the nested counterfactuals because they reflect the typical behavior of the different variants of qpro. Recall that the first test set contains only formulas with a linear quantifier tree, whereas the formulas of the other set have a complicated structure of quantifier dependencies (and thus are very "different" from being in PCNF).

We ran six different versions of qpro, namely

- qproNone: all possible simplification options disabled;
- qproUP: unit and pure literal detection enabled;
- qproL: DDB by labeling enabled;
- qproS: DDB by relevant sets enabled;
- qproUPL: unit and pure literal detection as well as DDB by labeling enabled; and
- qproUPS: unit and pure literal detection as well as DDB by relevant sets enabled.

In what follows, we briefly summarize our observations.

6.2.1 Modal Logic K

Fig. 14 shows that the more options are enabled, the better qpro performs: qproUPL and qproUPS show definitely the best runtime behavior whereas qproUPS is slightly better in average. When disabling unit and pure, the difference between DDB by labeling and DDB by relevance sets is more obvious. Furthermore, DDB turns out to be less important here than unit and pure literal detection.

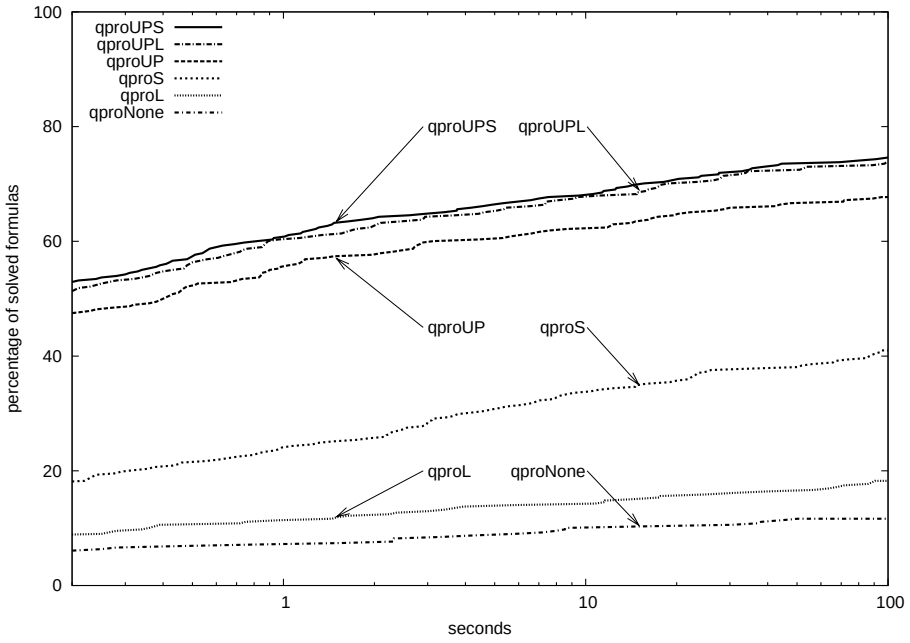

Figure 14: Different version of qpro applied to encodings of modal logic K.

6.2.2 Nested Counterfactuals

In Fig. 15, there are two outliers: qproNone and qproS. Without any options enabled, qpro is able to solve just a few formulas. Much more surprisingly, qproS clearly outperforms all other variants. This phenomenon

may have two different origins: On the one hand, the application of the simplifications involves additional search. For this set, these simplifications seem to result in an overhead rather than in an optimization. On the other hand, the elimination of unit and pure literals also influences the actual relevant sets in the computation. Consider the following example: a variable y has become unit because—let us say the existential—variable x was assigned a truth value during the splitting process. So when the unit rule is applied and y is included in a reason, then x must be included too. Therefore, the second subproblem may not be omitted if the first one resulted in false when returning to x during the backtracking. Otherwise, if the unit rule had not been applied on y and it is assigned a truth value by the ordinary splitting process, then it is possible that x is not included in the reason. In this case, the subproblem where x is set to the dual value can be omitted.

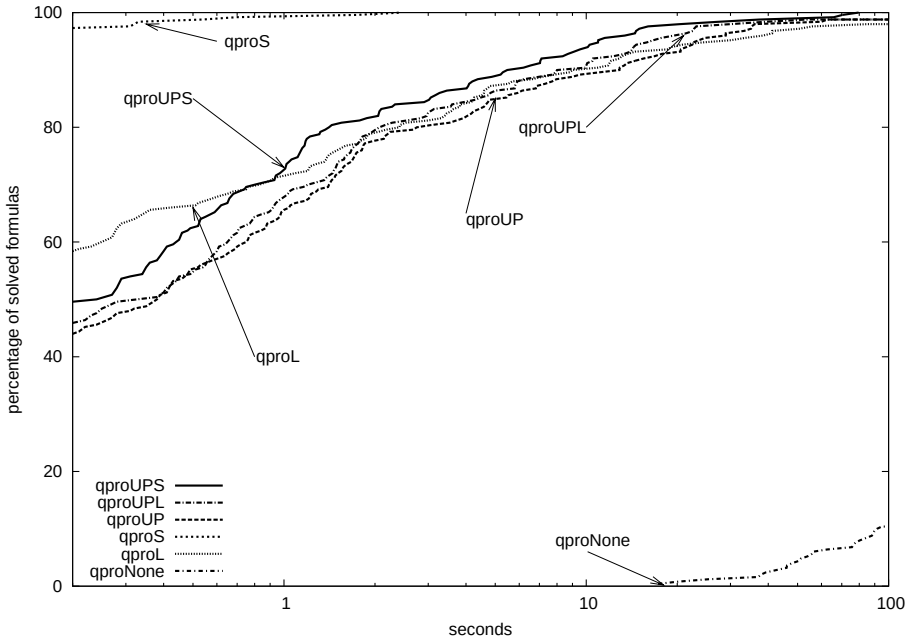

Figure 15: Different version of qpro applied to encodings of nested counterfactuals.

6.3 Comparison with State-of-the-Art Systems

Next, we compare the performance of our solver qpro against the established systems QuBE−BJ (version v1.2) [23], sKizzo (version v0.8.2) [6], semprop (release 24/02/02) [29], and quantor (version v3.0) [8]. These solvers have been selected because they have shown to be competitive in previous QBF evaluations and moreover, they have been the most robust ones in previous test runs, i.e., they did not deliver wrong results on our benchmarks. Moreover, QuBE−BJ and semprop implement backtracking techniques, similar to the ones used in qpro. Finally, sKizzo and quantor try to extract original quantifier dependencies from a PCNF. Hence the latter solvers may detect similar structural information on the input formula as qpro has got *a priori* from the input of the corresponding non-prenex formula.

All solvers except qpro require the input to be in PCNF. We thus apply the following test strategy: Given a benchmark QBF ϕ, (i) translate ϕ into NNF and use that formula as input to qpro; (ii) translate ϕ into PCNF and provide the outcome as input to the other solvers. The latter translation is performed in two

steps, namely the prenexing and the conversion of the resulting purely propositional matrix into CNF by the application of the structure-preserving transformation to normal form as described in [44].

All solvers are used with their predefined standard options, for qpro we use the variant qproUPS.

6.3.1 Modal Logic K

Many of the formulas of this benchmark set are solved immediately by all solvers. However, for some of the formulas, none of the solvers delivered a result within the timeout of 100 seconds. For this test set, the results of qpro are located in the middle of the field. As the quantifier dependency tree is linear, qpro cannot gain advantage of the quantifier dependency structure and so it has to deal with formulas almost in prenexed form. Still, the results show that the more complex data structures (compared to the PCNF solvers) which qpro has to handle, do not lead to a significant overhead.

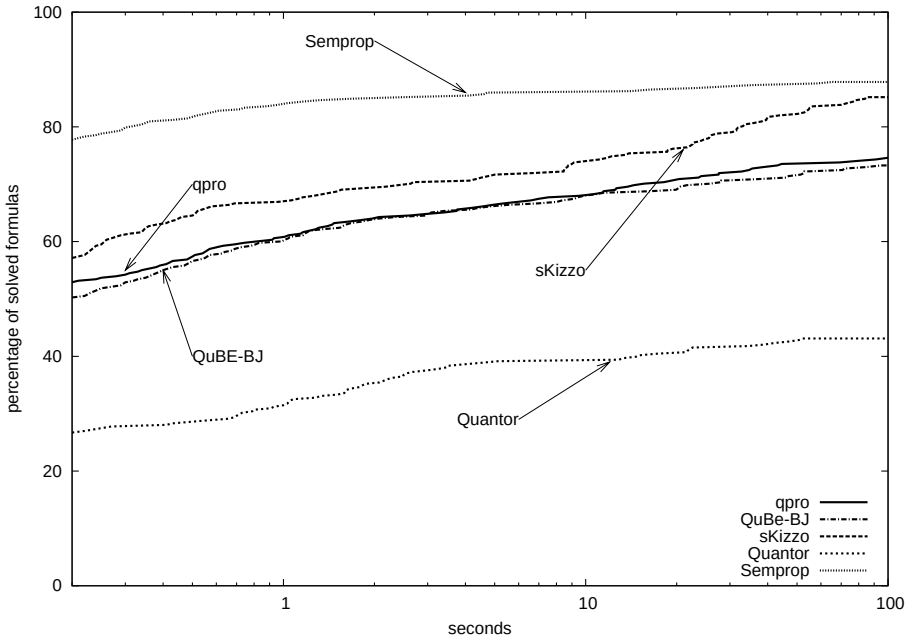

Figure 16: Encodings of modal logic K.

6.3.2 Nested Counterfactuals

As indicated by Fig. 13, the quantifier dependencies allow for several different translations into PCNF. We have applied each strategy presented in [15] to the PCNF solvers in previous test runs. To be fair, we show only the results for the best strategy of each solver.

Our solver clearly outperforms the other solvers, actually it is the only solver able to solve all formulas of the set. This indicates that the more information on quantifier dependencies is lost due to the prenexing (even if the "right" prenexing strategy is used), the more competitive qpro turns out to be. With increasing quantifier depth, the formulas get harder to solve. Only qpro can handle the augmenting hardness, whereas

the other solvers fail. Consequently, their curves remain quite flat, except for semprop for which the curve ascends starting from the 5th second quite steeply.

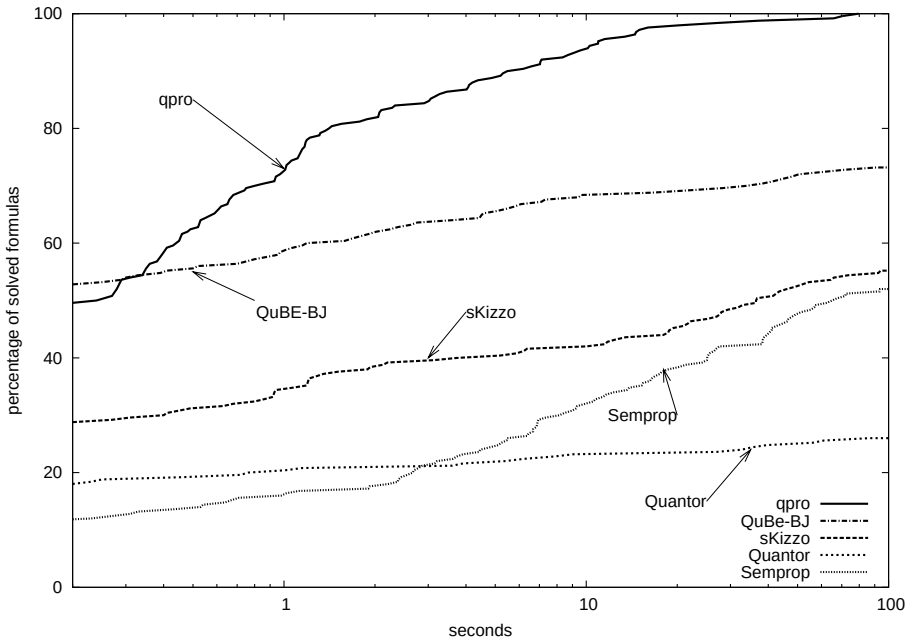

Figure 17: Encodings of nested counterfactuals.

6.3.3 Answer-Set Correspondence

For each formula of the first answer-set correspondence benchmark set (ASC-4), we obtain four different runtimes when using PCNF solvers. We distinguish between the different encodings S and T and the two possible shifting strategies when building a linear quantifier prefix, namely "↑" and "↓" (see Section 2). As the transformation step to prenex normal form is not necessary for qpro, we have only two different runtimes here.

The results of all possible combinations are presented in [16]; here, we show only the worst (upper picture in Fig. 18) and the best case (lower picture in Fig. 18) for all solvers. In fact, encoding T is always better than S and it is preferable to use "↓". In fact, the upper diagram illustrates the runtimes for encoding S together with strategy "↑" (for the PCNF solvers), whereas the lower diagram depicts T together with "↓". The curves of qpro differ only slightly, which is an indicator that qpro is less dependent on the particular encodings in contrast to the PCNF solvers. Fig. 18 also demonstrates clearly the importance of choosing a suitable prenexing strategy; indeed, all the PCNF solvers show problematic performance with the "↑" strategy but are much faster with the "↓" strategy.

In the second subset (ASC-2), we have QBFs of depth 2, thus there is only one prenex form. The results for different solvers are summarized in Fig. 19. Among all solvers, qpro performs most competitive although, in this case, it cannot gain advantage of the quantifier dependencies. Nevertheless the non-CNF structure seemingly supports qpro's solving method and explains the good results. The form of qpro's curve illustrates very clearly the structured nature of this problems and increasing difficulty of the benchmarks

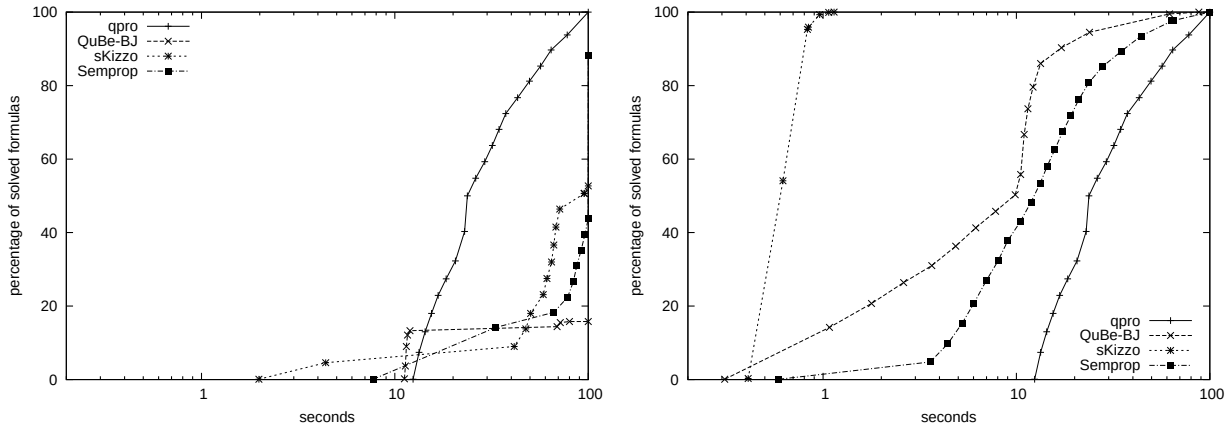

Figure 18: Encodings of answer-set correspondence checking (ASC-4).

with the increasing number of variables. Most formulas of one variable number are almost equally hard to solve, so we can observe the steep ascent in qpro's curve in Fig. 19.

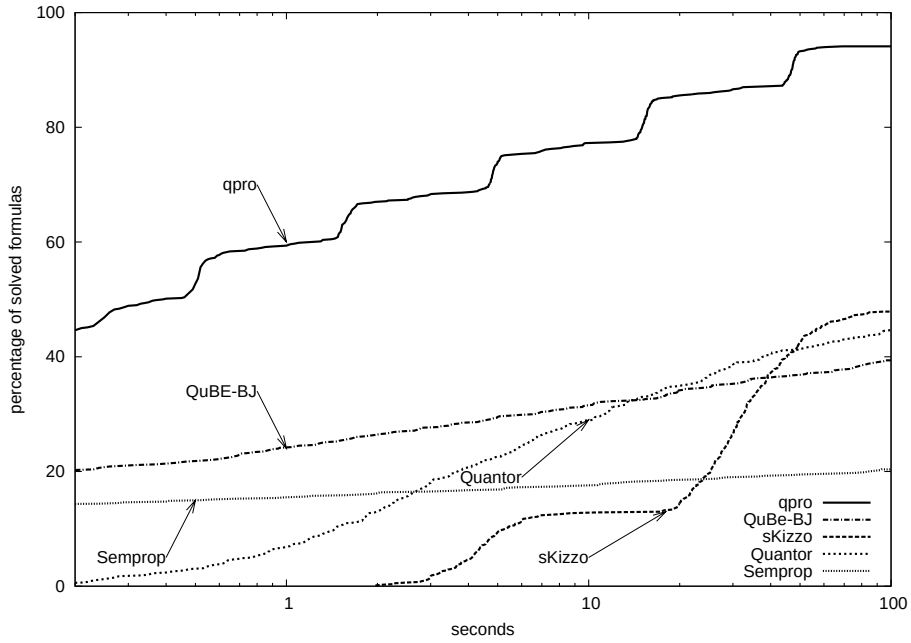

Figure 19: Encodings of answer-set correspondence checking (ASC-2).

6.4 Discussion

The chosen benchmarks provide an increasing complexity in their structure, and therefore, an increasing disruption of the structure during the normal form transformation can be expected. In particular, for the

encodings of modal logic K and ASC-2, we just can investigate the effect of applying the transformation of one shifting strategy, since the prefix is already determined by the encoding. The ASC-4 problems allow to analyze the effect of prenexing if there is only a small deviation from a linear quantifier dependency. In what follows, we briefly discuss three main observations from our experiments.

- The more information of the formula structure is lost due to the transformation to normal form, the more competitive qpro turns out to be. Fig. 17 illustrates this effect for benchmark formulas with complex quantifier dependencies. Although we compare qpro here against the PCNF solvers together with their *best* suited strategy, qpro significantly outperforms all other solvers.

- Fig. 19 indicates that not only the prenexing causes a loss of structural information which improves the solving process but also the transformation to CNF has a great impact on the runtime. So qpro can also outperform the other solvers if structural information is not only obtained from quantifier dependencies but symmetries in the general structure of the formula are present (which gets blurred in the transformation of the propositional matrix to CNF).

- Fig. 18 presents results for two different encodings (namely S and T) of the same problem where one is an explicit but rather simple optimization of the other. These results indicate that qpro is less depending on the chosen encoding, whereas the performance of PCNF solvers differs much more. A similar observation in a different context is observed in [39]. In fact, qpro performs better on the unoptimized encoding, in the case the "wrong" prenexing strategy "↑" is used for the PCNF solvers.

Overall, these results are a further justification that a transformation to normal form may have crucial impacts on the performance.

7 Conclusion

We presented a new QBF solver, qpro, which significantly differs from previous approaches by its ability to process QBFs in negation normal form (NNF) instead of QBFs in prenex conjunctive prenex normal (PCNF). We generalized the DPLL procedure to handle such QBFs in NNF and we discussed implemented performance-improving techniques like different forms of dependency-directed backtracking. The system together with the benchmarks used in this paper can be found at

`http://www.big.tuwien.ac.at/staff/seidl/qpro`.

The motivation for the development of qpro was as follows: In practical applications, QBF solvers can be used as a black box in reasoning systems to solve problems stemming from a diversity of formalisms. Such problems are encoded as QBFs but natural encodings do not directly result in any normal form. However, since most solvers only accept the input to be in such a restricted format like PCNF, an additional translation to normal form is often required. As we have shown in our experiments, using the solver qpro, and thus avoiding the additional translation into PCNF, often results in much better performance.

Other standard optimization techniques found in PCNF solvers like learning (see, e.g., [22, 43, 42]) have not been included yet in qpro but this extension of the solver is subject of ongoing work. Furthermore, the currently implemented selection heuristics for variables and subformulas, are very basic and more sophisticated approaches have to be considered in future work.

There are a few further solvers, namely QuBoS [3], boole[6], and zqsat [20] in the literature, 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 we encountered some problems on certain formulas. Finally, zqsat implements DPLL using zero-compressed BDDs. The comparison to zqsat is subject to future work; as well, a detailed comparison of our approach with the one suggested in [24] to extend QuBE is on our agenda.

References

[1] C. Ansótegui, C. Gomes, and B. Selman. The Achilles' Heel of QBF. In *Proceedings of the 20th National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI 2005)*, pages 275–281. AAAI Press / MIT Press, 2005.

[2] O. Arieli and M. Denecker. Reducing Preferential Paraconsistent Reasoning to Classical Entailment. *Journal of Logic and Computation*, 13(4):557–580, 2003.

[3] A. Ayari and D. Basin. QUBOS: Deciding Quantified Boolean Logic Using Propositional Satisfiability Solvers. In *Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2002)*, volume 2517 of *LNCS*, pages 187–201. Springer, 2002.

[4] M. Baaz, C. Fermüller, and A. Leitsch. A Non-Elementary Speed Up in Proof Length by Structural Clause Form Transformation. In *Proceedings of the 9th Annual IEEE Symposium on Logic in Computer Science (LICS 1994)*, pages 213–219. IEEE Computer Society Press, 1994.

[5] M. Baaz and A. Leitsch. On Skolemization and Proof Complexity. *Fundamenta Informaticae*, 20:353–379, 1994.

[6] M. Benedetti. sKizzo: A Suite to Evaluate and Certify QBFs. In *Proceedings of the 21th International Conference on Automated Deduction (CADE 2005)*, volume 3632 of *LNCS*, pages 369–376. Springer, 2005.

[7] P. Besnard, T. Schaub, H. Tompits, and S. Woltran. Representing Paraconsistent Reasoning via Quantified Propositional Logic. In *Inconsistency Tolerance*, volume 3300 of *LNCS*, pages 84–118. Springer, 2005.

[8] A. Biere. Resolve and Expand. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, volume 3542 of *LNCS*. Springer, 2005.

[9] T. Boy de la Tour. An Optimality Result for Clause Form Translation. *Journal of Symbolic Computation*, 14(4):283–302, 1992.

[10] E. Eder. *Relative Complexities of First-Order Calculi*. Artificial Intelligence. Vieweg Verlag, 1992.

[11] U. Egly. On Different Structure-preserving Translations to Normal Form. *Journal of Symbolic Computation*, 22:121–142, 1996.

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

[12] U. Egly. Quantifiers and the System KE: Some Surprising Results. In *Proceedings of the 12th International Workshop on Computer Science Logic (CSL 1998)*, volume 1584 of *LNCS*, pages 90–104. Springer, 1999.

[13] U. Egly, T. Eiter, H. Tompits, and S. Woltran. Solving Advanced Reasoning Tasks Using Quantified Boolean Formulas. In *Proceedings of the 17th National Conference on Artificial Intelligence and the 12th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI 2000)*, pages 417–422. AAAI Press / MIT Press, 2000.

[14] U. Egly and T. Rath. On the Practical Value of Different Definitional Translations to Normal Form. In *Proceedings of the 13th International Conference on Automated Deduction (CADE 1996)*, volume 1104 of *LNCS*, pages 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 *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2919 of *LNCS*, pages 214–228. Springer, 2004.

[16] U. Egly, M. Seidl, and S. Woltran. A Solver for QBFs in Nonprenex Form. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006)*, pages 477–481. IOS Press, 2006.

[17] T. Eiter and G. Gottlob. The Complexity of Nested Counterfactuals and Iterated Knowledge Base Revisions. *Journal of Computer and System Sciences*, 53(3):497–512, 1996.

[18] T. Eiter, H. Tompits, and S. Woltran. On Solution Correspondences in Answer Set Programming. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 97–102. Professional Book Center, 2005.

[19] M. Gelfond and N. Leone. Logic Programming and Knowledge Representation - The A-Prolog Perspective. *Artificial Intelligence*, 138(1-2):3–38, 2002.

[20] M. GhasemZadeh, V. Klotz, and C. Meinel. Embedding Memoization to the Semantic Tree Search for Deciding QBFs. In *Proceedings of the 17th Australian Joint Conference on Artificial Intelligence (AI 2004)*, volume 3339 of *LNCS*, pages 681–693. Springer, 2004.

[21] M. Ginsberg. Counterfactuals. *Artificial Intelligence*, 30:35–79, 1986.

[22] E. Giunchiglia, M. Narizzano, and A. Tacchella. Learning for Quantified Boolean Logic Satisfiability. In *Proceedings of the 17th National Conference on Artificial Intelligence and the 14th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI 2002)*, pages 649–654. AAAI Press, 2002.

[23] E. Giunchiglia, M. Narizzano, and A. Tacchella. Backjumping for Quantified Boolean Logic satisfiability. *Artificial Intelligence*, 145:99–120, 2003.

[24] E. Giunchiglia, M. Narizzano, and A. Tacchella. Quantifier Structure in Search Based Procedures for QBFs. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2006)*, pages 812–817. European Design and Automation Association, 2006.

[25] T. Jussila and A. Biere. Compressing BMC Encodings with QBF. *Electronic Notes in Theoretical Computer Science*, 174(3):45–56, 2007.

[26] J. Katz, Z. Hanna, and N. Dershowitz. Space-Efficient Bounded Model Checking. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2005)*, pages 686–687. IEEE Computer Society, 2005.

[27] S. C. Kleene. Permutability of Inferences in Gentzen's Calculi LK and LJ. *Memoirs of the AMS*, 10:1–26, 1952.

[28] D. Le Berre, M. Narizzano, L. Simon, and A. Tacchella. The Second QBF Solvers Comparative Evaluation. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT 2004), Revised Selected Papers*, volume 3542 of *LNCS*, pages 376–392. Springer, 2005.

[29] R. Letz. Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2002)*, volume 2381 of *LNCS*, pages 160–175. Springer, 2002.

[30] A. Ling, D. Singh, and S. D. Brown. FPGA Logic Synthesis Using Quantified Boolean Satisfiability. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT 2005)*, volume 3569 of *LNCS*, pages 444–450. Springer, 2005.

[31] M. Narizzano, L. Pulina, and A. Tacchella. Report of the Third QBF Solvers Evaluation. *Journal of Satisfiability, Boolean Modeling and Computation*, 2:145–164, 2006.

[32] A. Nonnengart, G. Rock, and C. Weidenbach. On Generating Small Clause Normal Forms. In *Proceedings of the 15th International Conference on Automated Deduction (CADE 1998)*, volume 1421 of *LNCS*, pages 397–411. Springer, 1998.

[33] J. Oetsch, M. Seidl, H. Tompits, and S. Woltran. ccT: A Tool for Checking Advanced Correspondence Problems in Answer-Set Programming. In *Proceedings of the 15th International Conference on Computing (CIC 2006)*, pages 3–10. IEEE Computer Society Press, 2006.

[34] E. Oikarinen and T. Janhunen. Verifying the Equivalence of Logic Programs in the Disjunctive Case. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2004)*, volume 2923 of *LNCS*, pages 180–193. Springer, 2004.

[35] G. Pan and M. Vardi. Optimizing a BDD-Based Modal Solver. In *Proceedings of the 20th International Conference on Automated Deduction (CADE 2003)*, volume 2741 of *LNCS*, pages 75–89. Springer, 2003.

[36] D. Plaisted and S. Greenbaum. A Structure Preserving Clause Form Translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986.

[37] J. Rintanen. Constructing Conditional Plans by a Theorem Prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.

[38] A. Sabharwal, C. Anstegui, C. P. Gomes, J. W. Hart, and B. Selman. QBF Modeling: Exploiting Player Symmetry for Simplicity and Efficiency. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT 2006)*, volume 4121 of *LNCS*, pages 382–395. Springer, 2006.

[39] H. Samulowitz, J. Davies, and F. Bacchus. Preprocessing QBF. In *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP 2006)*, volume 4204 of *LNCS*, pages 514–529. Springer, 2006.

[40] H. Tompits and S. Woltran. Towards Implementations for Advanced Equivalence Checking in Answer-Set Programming. In *Proceedings of the 21st International Conference on Logic Programming (ICLP 2005)*, volume 3668 of *LNCS*, pages 189–203. Springer, 2005.

[41] A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*, volume 43 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1996.

[42] L. Zhang. Solving QBF by Combining Conjunctive and Disjunctive Normal Forms. In *Proceedings of the 21st National Conference on Artificial Intelligence and the 18th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI 2006)*. AAAI Press, 2006.

[43] L. Zhang and S. Malik. Towards a Symmetric Treatment of Satisfaction and Conflicts in Quantified Boolean Formula Evaluation. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP 2002)*, volume 2470 of *LNCS*, pages 200–215. Springer, 2002.

[44] M. Zolda. Comparing Different Prenexing Strategies for Quantified Boolean Formulas. Master's thesis, Technische Universität Wien, Institut für Informationssysteme, 2004.