Exploiting Quantifiers Structure in QBF Reasoning

Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella

DIST, Università di Genova, Viale Causa, 13 – 16145 Genova, Italy
{enrico,mox,tac}@dist.unige.it

Abstract. The best currently available solvers for quantified Boolean formulas (QBFs) process their input in prenex form, i.e., all the quantifiers are to appear in the prefix of the formula separated from the purely propositional part representing the matrix. However, in many QBFs deriving from applications, the propositional part is intertwined with the quantifiers structure. In this paper, we show that converting such instances into prenex form, and thus losing the structural information about the quantifiers, interferes with the solving process by thwarting branches that could otherwise have occurred. To tackle this problem we devise suitable extensions to the QBF search algorithm in order to exploit the quantifiers structure, and we implement the extended algorithm in the state-of-the-art solver QuBE. Our experimental analysis shows that substantial speedups can be obtained by working with a structured-prefix form rather than with the traditional prenex form.

1 Introduction

The use of quantified Boolean formulas (QBFs) as a logic language to encode problems arising from various application domains has attracted increasing interest in recent years (see, e.g., [1–5]). The application-driven quest for efficiency has in turn propelled the research on decision procedures in order to deal with the size and the complexity of the QBF encodings (see [6] for a recent account on the state of the art in QBF reasoning). Considering the best currently available QBF solvers, we see that all of them process their input in prenex form, i.e., all the quantifiers are to appear in the prefix of the formula separated from the purely propositional part representing the matrix. This representation has the advantage of simplicity without losing generality: indeed it is possible to convert efficiently any QBF into an equivalent one in prenex form. However, in many QBFs deriving from applications, the propositional part is intertwined with the quantifiers structure. Given the current state of the art in QBF solvers, the only way to deal with the QBF instances arising in such cases is to apply a conversion into prenex form, but this poses two issues: first, as shown in [7], there are several methods to obtain a prenex form starting from a non-prenex one, and the method delivering the best performances depends both on the kind of instances and on the internals of the QBF solver; second, as we experienced in some preliminary experiments, even the best performances can be unsatisfactory on the instances in prenex form.

In this paper, we introduce a viable and effective solution to both of the above issues. We start by showing that the conversion of QBF instances exhibiting quantifiers structure into prenex form is not always a fruitful approach. With the aid of an example, and assuming that the QBF engine is search-based, we demonstrate that converting non-prenex QBFs into prenex ones interferes with the search process by thwarting branches that can happen only when dealing directly with the non-prenex instance. From this result, we conjecture that carrying structural information about the prefix to the solving process can improve performances, similarly to what has been assessed for propositional satisfiability in [8]. In order to test our hypothesis, we devise a set of suitable extensions to the QBF search algorithm that take into account the quantifiers structure in all of its phases, namely backtracking, heuristics, and simplification. Then, we implement the extended algorithm in the solver QuBE [9] featuring state-of-the-art data structures, search heuristics and backtracking techniques, in order to provide a challenging baseline for our proof-of-concept implementation. Finally, we test the original version of QuBE, named QuBE(to), against the extended version QuBE(po) that can deal with QBFs in non-prenex form. Our experimental analysis shows that QuBE(po) can be substantially and consistently faster than QuBE(to), confirming our initial conjecture that important speedups can be obtained by working with a structured-prefix form rather than with the traditional prenex form.

The paper is structured as follows. After a brief review of the logic of QBFs, we demonstrate how losing structural information about the quantifiers during the conversion to prenex form impacts negatively on the solving phase. Then, we show the extensions that we devised in order to tackle this problem and how we implemented them in QuBE(po). Finally, we describe how we generated structured QBF instances and the results that we obtained experimenting with QuBE(to) and QuBE(po). We conclude the paper with some remarks, future and related work.

2 The logic of QBFs

Consider a set P of propositional letters. An *atom* is an element of P. A *literal* is an atom or the negation $\overline{z}$ of an atom z. In the following, for any literal l,

- $|l|$ is the atom occurring in l; and
- $\overline{l}$ is $\overline{l}$ if l is an atom, and is $|l|$ otherwise.

A *clause* is a finite disjunction of literals. Finally,

- if $c_1, \ldots, c_n$ are clauses, $(c_1 \wedge \ldots \wedge c_n)$ is a QBF,
- if Φ is a QBF and z is an atom, $Qz\Phi$ is a QBF, where Q is either an existential quantifier ($\exists$) or a universal quantifier ($\forall$). In $Qz\Phi$, Φ is called the *scope* of Qz, and z is the atom *bound* by Q.
- if $\Phi_1, \ldots, \Phi_n$ are QBFs $(\Phi_1 \wedge \ldots \wedge \Phi_n)$ is a QBF.

For simplicity, we restrict our attention to *closed QBFs*, i.e., to QBFs in which each atom is bound by a quantifier. For example,

$$\begin{array}{r}\forall y_1(\exists x_1 \forall y_2 \exists x_2((\overline{y}_1 \vee \overline{y}_2 \vee x_2) \wedge (\overline{y}_1 \vee \overline{y}_2 \vee \overline{x}_2) \wedge \\ (\overline{y}_1 \vee y_2 \vee \overline{x}_2) \wedge (y_1 \vee x_1 \vee x_2) \wedge \\ (y_1 \vee \overline{x}_1 \vee x_2) \wedge (y_1 \vee \overline{x}_1)) \wedge \\ \exists x_3 \forall y_3 \exists x_4((\overline{y}_1 \vee y_3 \vee \overline{x}_4) \wedge (\overline{y}_1 \vee x_3 \vee y_3 \vee x_4)))\end{array} \tag{1}$$

is a closed QBF. Further, we assume that inside a QBFs there are no two distinct quantifiers that bind the same atom. With this assumption, we can represent any QBF as a pair

- the *prefix*, consisting of a partial order on the set of pairs $\langle quantifier, boundatom \rangle$: $Q_1 z_1$ and $Q_2 z_2$ are in partial order (and we write $z_1 \preceq z_2$) if and only if $Q_2 z_2$ occurs in the scope of $Q_1 z_1$, and
- the *matrix*, consisting of the set of clauses in φ.

Given that we will use x_i (resp. y_i) to denote an existentially (resp. universally) quantified atom, we can omit the quantifier from the specification of the partial order and write, e.g., $x \preceq y$ instead of $\exists x \preceq \forall y$. Thus, the prefix of (1) corresponds to the partial order

$$y_1 \preceq x_1, x_1 \preceq y_2, y_2 \preceq x_2 \qquad y_1 \preceq x_3, x_3 \preceq y_3, y_3 \preceq x_4. \tag{2}$$

About the matrix, in the following we write $\{l_1, \ldots, l_n\}$ for the clause $(l_1 \vee \ldots \vee l_n)$. Thus, the matrix of (1) is

$$\begin{array}{c}\{\{\overline{y}_1, \overline{y}_2, x_2\}, \{\overline{y}_1, \overline{y}_2, \overline{x}_2\}, \{\overline{y}_1, y_2, \overline{x}_2\}, \{y_1, x_1, x_2\}, \\ \{y_1, \overline{x}_1, x_2\}, \{y_1, \overline{x}_1\}\}, \{\overline{y}_1, y_3, \overline{x}_4\}, \{\overline{y}_1, x_3, y_3, x_4\}\}\end{array} \tag{3}$$

Consider a QBF φ with prefix $\preceq_\varphi$ and matrix Φ. The semantics of φ can be defined recursively as follows. Define an atom z to be *minimal in* φ if for no other atom z' in φ, $z' \preceq_\varphi z$. If the prefix is empty, then φ's value is defined according to the truth tables of propositional logic. If z is minimal in φ and z is existential (respectively universal), φ is true if and only if the QBF φ_x or (respectively and) $\varphi_{\overline{x}}$ are true. If l is a literal, φ_l is the QBF

- whose matrix is obtained from Φ by substituting l with TRUE and $\overline{l}$ with FALSE, and
- whose prefix is obtained from $\preceq_\varphi$ by removing the pairs $|l|, z$ such that $|l| \preceq_\varphi z$.

The above representation of QBFs differs from the traditional one in which the prefix corresponds to a total order. Indeed, any QBF with prefix $\preceq_\varphi$ has the same value of a QBF with the same matrix and prefix extending $\preceq_\varphi$.

```
bool SOLVE(P, M)
 1 ⟨P, M⟩ ← SIMPLIFY(P, M)
 2 if M = ∅ then return TRUE
 3 if {} ∈ M then return FALSE
 4 l ← CHOOSE-LITERAL(P, M)
 5 if l is existential then
 6   return SOLVE(P, M ∪ {l}) or
             SOLVE(P, M ∪ {l̄})
 7 else
 8   return SOLVE(P, M ∪ {l}) and
             SOLVE(P, M ∪ {l̄})

set SIMPLIFY(P, M)
12 while {l} ∈ M do
13   P ← REMOVE(P, |l|)
14   for each c ∈ M s.t. l ∈ c do
15     M ← M \ {c}
16   for each c ∈ M s.t. l̄ ∈ c do
17     M ← (M \ {c}) ∪ {c \ {l̄}}
18 return ⟨P, M⟩
```

Fig. 1. Basic QBF search algorithm.

3 Exploiting the Quantifiers Structure

In order to show that the conversion of QBF instances exhibiting quantifiers structure into prenex form is not guaranteed to be a fruitful approach, we present a basic QBF search algorithm (Figure 1), and then we show two traces of its operation (Figures 2- 3) on the following QBF:

$$\begin{aligned}\forall y_0(\exists x_1 \forall y_1 \exists x_3((y_0 \vee \overline{y}_1 \vee x_3) \wedge (y_0 \vee \overline{y}_1 \vee \overline{x}_3) \wedge \\ (y_0 \vee y_1 \vee \overline{x}_3) \wedge (\overline{y}_0 \vee x_1 \vee x_3) \wedge \\ (\overline{y}_0 \vee \overline{x}_1 \vee x_3) \wedge (\overline{y}_0 \vee \overline{x}_1 \vee \overline{x}_3)) \wedge \\ \exists x_2 \forall y_2 \exists x_4((y_0 \vee \overline{x}_2 \vee \overline{x}_4 \vee y_2) \wedge (y_0 \vee x_2 \vee \overline{x}_4 \vee \overline{y}_2) \wedge \\ (\overline{y}_0 \vee y_2 \vee x_4) \wedge (\overline{y}_0 \vee \overline{y}_2 \vee \overline{x}_4)))\end{aligned} \tag{4}$$

when the search strategy is constrained as it would be using a corresponding prenex QBF. In the particular case of (4), we conclude that constrained search strategies are bound to perform worse than dynamic strategies which can exploit the quantifiers structure. Hence, in general, we can conclude that conversion to prenex form can degrade the performances of search-based solvers.

By looking at Figure 1 we can see that the basic QBF search algorithm is comprised of two functions. The main one, SOLVE, takes as input the parameters P and M corresponding to the prefix $\preceq_\varphi$ and the matrix Φ of some QBF φ; SOLVE returns TRUE if φ is true and FALSE otherwise. The auxiliary function SIMPLIFY takes as input a prefix P and a matrix M (we abuse notation using the terms prefix and matrix to denote the *variables* encoding them) and returns a set comprised of a simplified prefix and matrix. We also define:

- a *unit literal* l (denoted by $\{l\}$) as any literal l such that there exist a clause $c \in M$ where l is the only existential literal in c and there is no other universal literal $l' \in c$ such that $|l'| \preceq |l|$ in P;
- an *empty clause* (denoted by $\{\}$) as any clause that does not contain existential literals;
- an *empty matrix* (denoted by $\emptyset$) as a matrix that does not contain any clause.

A short description of the behavior of SOLVE and SIMPLIFY follows: all the line numbers are cited with reference to Figure 1. SOLVE starts by simplifying P and M (line 1) and then selecting one of the following cases:

- If the matrix is empty (line 2) then SOLVE returns TRUE, i.e., the algorithm backtracks to the last recursive invocation over a universal literal (line 8);
- If the matrix contains an empty clause (line 3) then SOLVE returns FALSE, i.e., the algorithms backtracks to the last recursive invocation over an existential literal (line 9);
- If none of the above is true, then a literal l is selected (line 4) from M such that $|l|$ is minimal in P and M: if l is existential, then SOLVE is invoked recursively adding l as a unit literal; if the first call returns FALSE, then a second call is performed with $\overline{l}$ added as a unit literal (line 6); if l is universal, then the recursive invocation follows a similar pattern, but the second call occurs only if the first call returns TRUE(line 8).

In the following, we call *branch* the step performed by SOLVE in lines 6 and 8, and we call *branching literal* the one added as a unit literal to M in the recursive call; a branch is of type *OR* if the branching literal is existential, and it is of type *AND* otherwise. SIMPLIFY loops while M contains some unit literal (line 12). For each such literal l, SIMPLIFY invokes the function REMOVE to expunge all the pairs $|l|, |l'|$ such that $|l| \preceq |l'|$ from P (line 13) and then simplifies the matrix M by removing all the clauses containing l (lines 14-15), and by removing $\overline{l}$ from the other clauses. Finally, SIMPLIFY returns the set $\langle P, M\rangle$ with the modified prefix P and matrix M (line 18).

The algorithm presented in Figure 1 can be used to simulate the solution of the QBF (4) in prenex form by fixing a priori two different strategies implemented by CHOOSE-LITERAL:

- "Universal First" (UF for short), always preferring universal literals, and
- "Existential First" (EF for short), always preferring existential literals.

The above strategies are used to select a branching literal when there is more than one literal l such that $|l|$ is minimal in the current QBF. For the time being, we assume that both EF and UF always branch on positive literals first, where a literal is *positive* if $|l| = l$ and *negative* otherwise. To understand why using UF and EF is the same as solving (4) in prenex form, consider that, although several extensions to the prefix of (4) are possible, we are compelled to choose y_0 as the first atom in any total order, and then we get to choose how to interleave the partial orders $x_1 \preceq y_1, y_1 \preceq x_3$ and $x_2 \preceq y_2, y_2 \preceq x_4$. In this process, the crucial step is deciding whether universal atoms should have priority over existential atoms, corresponding to UF, or the other way around, corresponding to EF.

Figure 2 shows the tree implicitly constructed by SOLVE when deciding (4) using the UF strategy. In the Figure, each node of the tree contains a matrix and is numbered according to the order in which SOLVE explores the search space; the root node contains the input matrix (4), and the other nodes contain the matrices resulting from the simplifications performed along the path from the

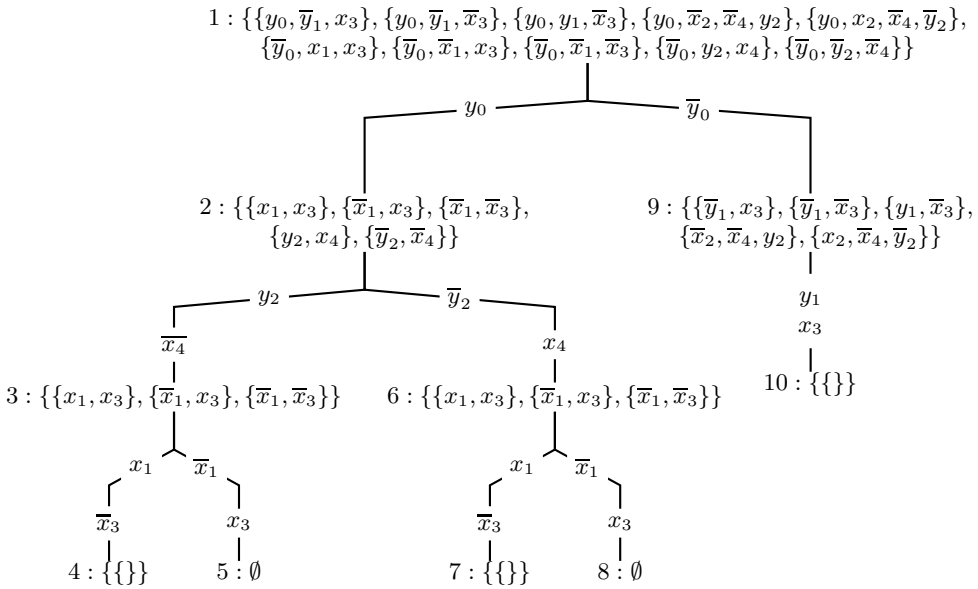

Fig. 2. Search tree using the UF strategy on the QBF with the matrix at the root and the prefix corresponding to $y_0 \preceq x_1, y_0 \preceq x_2, x_1 \preceq y_1, y_1 \preceq x_3, x_2 \preceq y_2, y_2 \preceq x_4$.

root to each of them; notice that all the leaves are either *solutions*, i.e., $\emptyset$, or *conflicts*, i.e., $\{\{\}\}$, the latter denoting any matrix that contains at least an empty clause; branches in the tree correspond to the choice of a literal performed by Choose-Literal whenever both values of the branching literal must be tried; straight lines stand for unit literals or branching literals that are not subject to backtracking. With reference to Figure 2, we can see that Solve attacks the QBF (4) by branching on y_0 first, as no other choice for a branching literal is possible at this point. From node 2, Choose-Literal could pick either x_1 or y_2, but y_2 is chosen first in accordance with the UF strategy. The subsequent simplification causes the unit literal $\overline{x}_4$ to arise and the matrix to be simplified as shown in node 3. From such node, either x_1 or x_3 could be picked as branching literal, but in both cases we get to the empty matrix (node 5). Backtracking from node 5 reaches node 2, where $\overline{y}_2$ must be tried in order to close the AND branch. As it can be seen, the subtree rooted at node 6 is identical to the one rooted at node 3, and thus the algorithms backtracks to node 1, where $\overline{y}_0$ is tried and node 9 is reached. From node 9 we can conclude with one additional branch (y_1) and a unit literal (x_3) that the QBF (4) is indeed false. From the example in Figure 2 we can conclude that, had Choose-Literal picked x_1 instead of y_2, we could have saved the additional, and useless, branch on y_2. Therefore, constraining the search with the UF strategy resulted in a less efficient exploration of the first part of the search space.

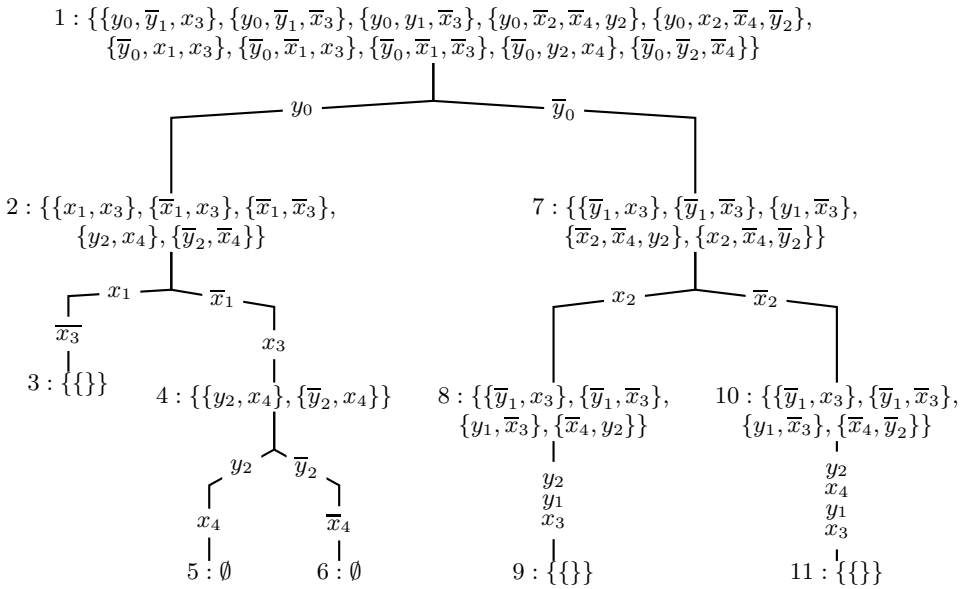

Fig. 3. Search tree using the EF strategy on the QBF with the matrix at the root and the prefix corresponding to $y_0 \preceq x_1, y_0 \preceq x_2, x_1 \preceq y_1, y_1 \preceq x_3, x_2 \preceq y_2, y_2 \preceq x_4$.

Figure 3 shows the search tree explored by Solve when deciding (4) using the EF strategy. The presentation follows the same conventions used in Figure 2. By looking at Figure 3 we can see that EF is better than UF on the subtree rooted at node 2 since EF ends up exploring a smaller search space than UF. Nevertheless, if we look at the subtree rooted at node 7 in Figure 3, and we contrast it with the subtree rooted at node 9 in Figure 2, we can also see that EF is worse than UF in this part of the search space. In particular, had Choose-Literal picked y_1 instead of x_2 as a branching literal, the search could have terminated with one additional branch on y_1 and the subsequent unit literal x_3. From Figure 3 we can conclude that EF has the same limitations of UF, and therefore it is not always possible to impose some total order on the prefix without degrading the performances of Solve.

Two further observations are in order about Figure 2 and Figure 3. First, it could be argued that, had Choose-Literal selected $\overline{y}_0$ as the first branch, the search space would have included only the rightmost subtree of node 1 both in Figure 2 and in Figure 3. Incidentally, this would also mean that UF performs better than EF on the QBF (4). This argument is countered by considering that it is sufficient to swap y_0 and $\overline{y}_0$ in (4) to obtain a QBF such that Solve explores the same amount of search space as in Figures 2 and 3 precisely when Choose-Literal selects negative literals first. Second, it could be argued that, had Solve used a more advanced strategy for backtracking, e.g., backjumping [10],

the search space could be pruned a posteriori and the effects of using a ineffective heuristic could be mitigated. Indeed, in the case of (4), using backjumping does not help when using the UF strategy (cf. Figure 2), but it does help when using the EF strategy (cf. Figure 3). To understand why, we recall that backjumping works by computing reasons, i.e., subsets of the literals assigned along the paths from the root to the leaves of the search tree: intuitively, a literal is in the current reason iff it is responsible for the current conflict (resp. solution); if a branching literal is not in the reason, then branching over it will cause a conflict (resp. solution) to arise again. Hence, backjumping can prune irrelevant parts of the search space by skipping branching literals that are not contained in the current reason. By looking at node 7 in Figure 3, we can observe that x_2 is not in the reason of the conflict detected at node 9 (the reason being $\{y_0, \overline{y}_1\}$, i.e., an empty clause) and therefore, the rightmost subtree of node 7 could be pruned by backjumping. However, it is sufficient to consider the following QBF:

$$\begin{aligned}\forall y_0(\exists x_1 \forall y_1 \exists x_3((y_0 \vee \overline{y}_1 \vee x_3) \wedge (y_0 \vee \overline{y}_1 \vee \overline{x}_3) \wedge \\ (y_0 \vee y_1 \vee \overline{x}_3) \wedge (\overline{y}_0 \vee x_1 \vee x_3) \wedge \\ (\overline{y}_0 \vee \overline{x}_1 \vee x_3) \wedge (\overline{y}_0 \vee \overline{x}_1 \vee \overline{x}_3)) \wedge \\ \exists x_2 \forall y_2 \exists x_4((y_0 \vee \overline{x}_2 \vee \overline{x}_4 \vee y_2) \wedge (y_0 \vee x_2 \vee \overline{x}_4 \vee \overline{y}_2) \wedge \\ (\overline{y}_0 \vee y_2 \vee x_4) \wedge (\overline{y}_0 \vee \overline{y}_2 \vee \overline{x}_4) \wedge \\ (y_0 \vee x_2 \vee x_4) \wedge (y_0 \vee \overline{x}_2 \vee \overline{x}_4) \wedge (y_0 \vee \overline{x}_2 \vee x_4)))\end{aligned} \tag{5}$$

obtained by (4) adding the clauses in the last line of (5). Deciding (5) with Solve using the EF strategy and backjumping is as expensive as it is without backjumping. In particular, the search tree is similar to the one shown in Figure 3 where the reason for the conflict detected after branching on $\overline{y}_0$ and x_2 now contains x_2, so the branch on $\overline{x}_2$ cannot be skipped. Therefore, even in the case of backjumping and the EF strategy, we can come up with examples where the conversion to prenex form is bound to impact negatively on the search process. In Section 5, we show the experimental data obtained with the implementation of Solve on top of our solver QuBE described in the next section. Our analysis confirms that the degradation in the performances of search-based solvers when converting the input instances to prenex form is an important phenomenon, at least on the QBFs that exhibiting quantifiers structure.

4 Implementation in QuBE

We implemented the algorithm described in the previous section on top of our solver QuBE [9]. QuBE reads instances in prenex form and features state-of-the-art backtracking techniques, heuristics and data structures that provide a challenging baseline for our proof-of-concept implementation. To describe QuBE's features prior to this work, we use the following terminology:

- QuBE(to) to denote the version of QuBE solving QBF instances in prenex form, and QuBE(po) to denote the version of QuBE modified to exploit the quantifiers structure.

– A *term* is a finite conjunction of literals (terms arise when learning from solutions [11]); clauses and terms are also collectively referred to as constraints.
– A *reason*, as outlined in Section 3, is a subset of the literals assigned along a path in the search tree which is responsible for the conflict or the solution in the leaf (see [10] for more details).
– The *prefix level* of an atom in a totally ordered prefix is 1 + the number of expressions $Q_j z_j Q_{j+1} z_{j+1}$ in the prefix with $j \geq i$ and $Q_j \neq Q_{j+1}$

We now briefly describe the three main components of QuBE(to) and how we have modified them to obtain QuBE(po) (for a more detailed survey of various aspects of the latest implementation of QuBE see [9, 12]).

Backtracking QuBE(to) features learning as introduced for QBFs by [11], i.e., while backtracking the solver computes and stores additional constraints to avoid repeating the same search. The fundamental problem with learning is that adding constraints may lead to an exponential memory blow up, and thus QuBE(to) uses criteria (i) for limiting the constraints that have to be learned, and/or (ii) for unlearning some of them. The basic implementation of (i) in QuBE(to) works as follows. Assume that we are backtracking on a literal l assigned at decision level n, where the *decision level* of a literal is the number of nodes before l. The clause (resp. term) corresponding to the reason is learned only if:

1. l is existential (resp. universal),
2. all the assigned literals in the reason except l, are at a decision level strictly smaller than n, and
3. there are no open universal (resp. existential) literals in the reason that are before l in the (totally ordered) prefix.

Once QuBE(to) has learned the constraint, it backjumps to the node at the maximum decision level among the literals in the reason, excluding l. In QuBE(po), condition 3 above is changed as follows:

3. there are no open universal (resp. existential) literals l' in the reason such that $l' \preceq l$ in the prefix.

The new condition 3 ensures that also in QuBE(po) the learned constraint corresponds to a valid unit literal at the decision level where the solver jumps to.

Backtracking-based heuristic Choose-Literal is implemented in QuBE(to) by associating two counters to each literal: the number of clauses c such that $l \in c$, and the number of terms t such that $l \in t$. Each time a constraint is added, the counters are incremented; when a learned constraint is removed, the counters are decremented. In order to choose a branching literal, QuBE(to) stores the literals in a priority queue according to (i) the prefix level of the corresponding atom, (ii) the score and (iii) the numeric ID. Periodically, QuBE(to) rearranges the priority queue by updating the score of each literal l: this is done by halving the old score and summing to it the variation in the number of constraints k

such that $l \in k$, if l is existential, or the variation in the number of constraints k such that $\overline{l} \in k$, if l is universal. In QuBE(PO), the ordering of the priority queue cannot be maintained using the same set of conditions $(i - iii)$ above. In particular, since there is no total order on the prefix levels, the condition that top-priority literals in the queue must correspond to minimal atoms in the current QBF must be enforced by modifying the score as follows. First, we consider the set S of *maximal* atoms, i.e., all the atoms $|l|$ such that there is no $|l'|$ where $|l| \preceq |l'|$ in the prefix, and we assign them the basic score. Then, we consider the set S' of all the atoms $|l|$ such that $|l| \preceq |l'|$ precisely when $|l'| \in S$, i.e., $|l|$ is maximal: for each such literal l, we add to the basic score the maximum score among the literals l' such that $|l| \preceq |l'|$. We repeat the process, each time by letting $S = S'$ and computing the new S' as above. In this way, we guarantee that any two literals l, l' that are *incomparable*, i.e., such that neither $|l| \preceq |l'|$ nor $|l'| \preceq |l|$ are selected according to the heuristic scores, at the same time respecting the condition that each branching literal l corresponds to a minimal atom $|l|$.

Simplification QuBE(TO) simplification algorithm is based on the three literal watching (3LW) and the clause watching (CW) lazy data structures presented in [13] and extended in [12] to deal with terms. No modification was required to CW in QuBE(PO). In the case of 3LW, the identification of a unit literal l in a clause c (resp. a term t) requires the implementation of an additional data structure that can efficiently detect whether there is some other universal (resp. existential) literal l' in c (resp. t) such that $|l'| \preceq |l|$ in the prefix.

5 Experimental Analysis

The QBFs used for our experiments encode problems of checking derivability of a nested counterfactual from a given propositional knowledge base, and have been first proposed and used in [7]. Intuitively, a counterfactual $p > q$ is a conditional query stating that if p would be true in the knowledge base, then q would also be true. Accordingly, a nested counterfactual $p_1 > (p_2 > ....(p_n > q)....)$ corresponds to conditional queries involving a sequence of revisions. The QBFs resulting from this problem are appealing since they are not in prenex-cnf form and admit a large number of different prenex strategies. In figure 4 we shown the experimental setting used for comparing QuBE(PO) and QuBE(TO):

- MKNCF is a generator for random nested counter factual (NCF) problems. Four arguments are required in input to the generator: (i) VAR: the maximum number of distinct variables used in the formula; (ii) CLS: the number of clauses in the theory; (iii) LPC: the number of literals per theory clause; (iv) DEP: the nesting depth.
- TRAQLA is a translator from the ncf problem into a non-cnf, non-prenex QBF,

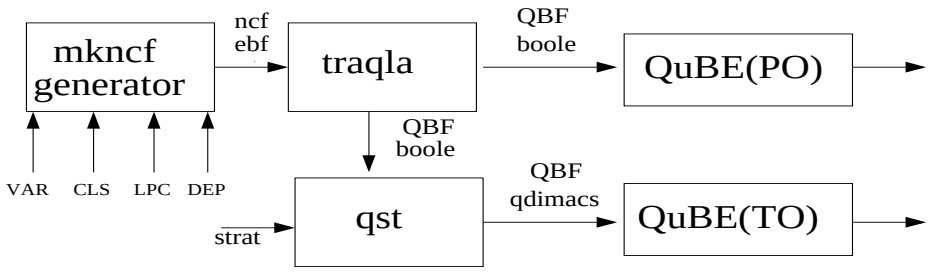

Fig. 4. Experimental setting;

- QST is the CNF translator, it takes in input a non-cnf, non-prenex QBF and generates a cnf, prenex QBF. It takes also in input a parameter *strat* that fixes which strategy is used for construct the prenex ordering. Intuitively, the strategies differ in the way they deal with existential and universal atoms that are incomparable, and thus they generate different total orders starting from the same partial one.

Looking at the figure 4 we can see that MKNCF randomly generates a NCF problem according with the fixed parameters (VAR, CLS, DEP, LPC) and then the NCF problem is translated into a non-cnf, non-prenex QBF. Here we have two possibilities:

1. give the non-cnf, non-prenex QBF problem as input of QuBE(PO) and solve it, or
2. translate the non-cnf, non-prenex QBF problem into a cnf, prenex QBF, and solve it with the QuBE(TO).

Finally we compare the time results of the two solvers. The MKNCF parameters have been set according to what has been done in [7]: In particular,

- the nested depth DEP is fixed to 6;
- VAR is varied in $\{4, 8, 16\}$;
- CLS is varied in such a way to have the ratio CLS/VAR in $\{1, 2, 3, 4, 5\}$;
- LPC is varied in $\{3, 4, 5\}$.

For each setting of DEP, VAR, CLS, LPC, 100 problems are generated. About the strategy *strat* used to generate a total order from the input partial order, we consider the four different strategies described in [7] and named $\exists\downarrow\forall\uparrow$, $\exists\uparrow\forall\downarrow$, $\exists\uparrow\forall\uparrow$, $\exists\downarrow\forall\downarrow$ by the authors. All these strategies are ensured to produce a QBF in total order with the minimum possible number of alternations in the prefix. In our setting, the *number of alternations* of a QBF in partial order corresponds to the partial order length of the prefix. For more details about the benchmarks and the different prenexing strategies, see [7].

As for the configuration of QuBE, in our experiments we use the common setup for QuBE(TO) and QuBE(PO) as described in Section 4. All the experimental data are obtained by running QuBE(TO) and QuBE(PO) on a farm of

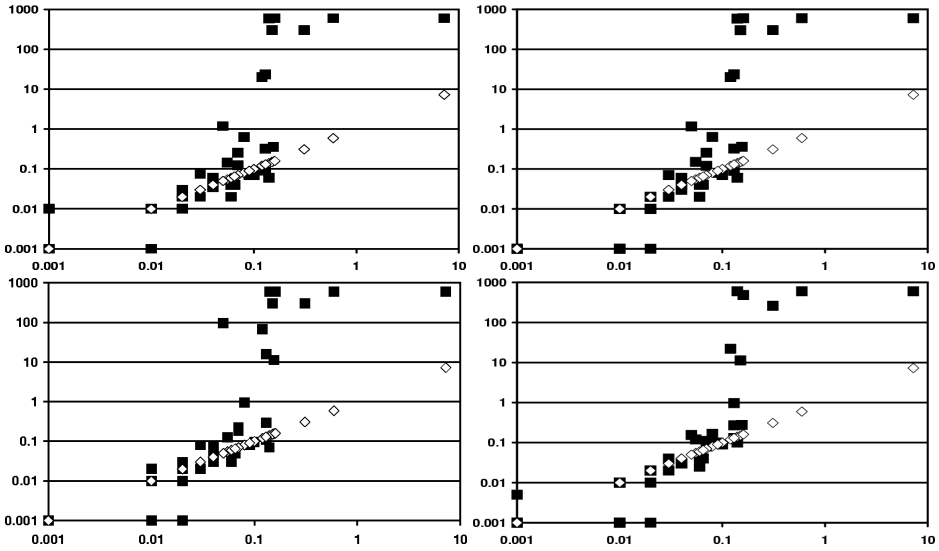

Fig. 5. QuBE(PO) vs. QuBE(TO); the plots correspond to the four different prenexing strategies $\exists\downarrow\forall\downarrow$, $\exists\uparrow\forall\uparrow$, $\exists\downarrow\forall\uparrow$, $\exists\uparrow\forall\downarrow$ (clockwise, from top-left)

10 identical rack-mount PCs, each one equipped with a 3.2Ghz PIV processor, 1GB of main memory and running Debian/GNU Linux. The timeout after which a solver is stopped has been set to 600s. In each plot in Figure 5, each solid-fill square dot represents an instance, QuBE(PO) solving time is on the x-axis (log scale), while QuBE(TO) solving time is on the y-axis (log scale). The diagonal (outlined diamond boxes) represents the solving time of QuBE(PO) against itself and serves as reference: the dots above the diagonal are instances where QuBE(PO) performs better than QuBE(TO), while the dots below are the instances where QuBE(PO) performances are worse than QuBE(TO). Each point on the plots is the mediam time over the 100 samples generated for a particular setting of the MKNCF parameters. As we can see from figure 5, QuBE(PO) compares very well with respect to QuBE(TO). In particular the performance gap in favor of QuBE(PO) is more important for the non trivial instances (i.e., with a running time ≥ 0.1s). In particular, QuBE(PO) median time never exceeds the timeout, while this happens several times for QuBE(TO), no matter which of the 4 different prenexing strategies is used. In figure 6 we also show the comparison between QuBE(PO) vs QuBE(TO) using the best prenexing strategies. For each benchmark, the best prenexing strategies is defined as the one that leads to the best performances of QuBE(TO). QuBE(PO) is competitive even in this setting, and, more importantly, QuBE(TO) median time exceeds the timeout for some values of the parameters. This last fact confirms our conjecture that reducing the QBF in total order does not pay off.

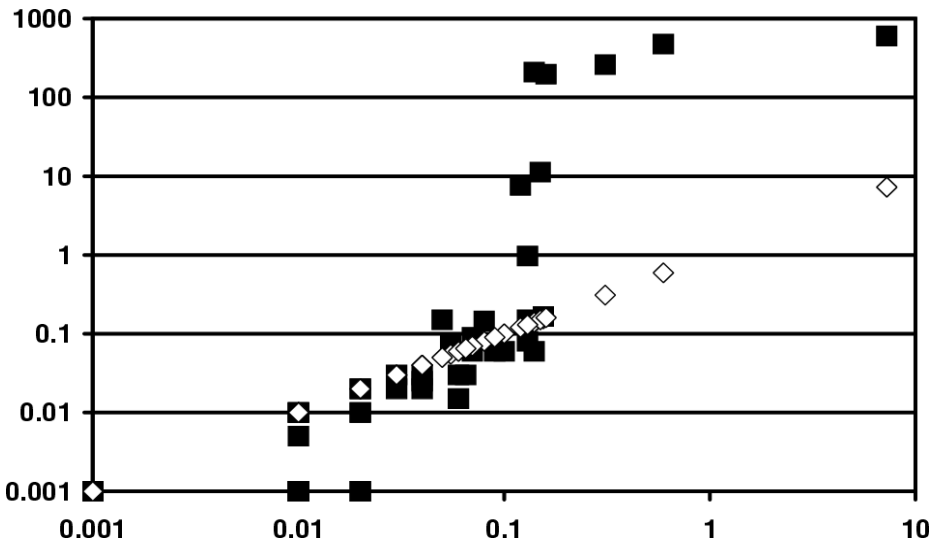

Fig. 6. Sota Solver vs QuBE(po)

6 Conclusions and Future Work

The main points touched in the paper can be summarized as follows:

- The conversion of QBF instances exhibiting quantifiers structure into prenex form is not always guaranteed to be a fruitful approach in the case of search-based solvers, even when advanced backtracking techniques are used.
- The basic search algorithm of QBF solvers can be extended to take into account the quantifiers structure; while such extensions required some effort to be implemented on top of our state-of-the-art solver QuBE, they can pay off in terms of performances.
- Controlled experiments on the specifically devised counterfactual instances reveal that by taking into account the quantifiers structure we can get dramatic improvements in the performance of the qbf solver, thereby confirming our initial conjecture.

Given the effectiveness of QuBE(po) we are currently experimenting with it to solve the challenging test cases described in [14], where QBF reasoning is applied to model checking of early software requirements. The challenge in such test cases is due to the fact that the resulting QBF encodings are hardly in the reach of QBF solvers using prenex instances only.

7 Related work

Some works are related to the one presented in this paper. For example the already cited [7] where they have shown that the prenexing strategy used in the

translation from non prenex to a prenex form could make the difference during the search. This works different from our approach since what we have shown that impose a total order during a cnf conversion does not pay off.

The work mostly related to ours is [15]. In this work, the author tries to re-construct the original non prenex structure of the formula starting from the instance in total order. A similar thing is also done in [16]. The essential difference between [15, 16] and our work is that our solver is based on search, while the solvers in [15] and [16] are mainly based on quantifier elimination. For solvers based on quantifier elimination, recovering or keeping the original quantifier structure is fundamental in order to reduce the size of each quantifier elimination operation. However, the solver SKIZZO described in [15] is not entirely based on quantifier elimination. In fact, SKIZZO uses different strategies –including search– for trying to solve each problem. However, search is the last attempted and thus the least used strategy. Further, the type of search performed by SKIZZO is different from ours, and it is not clear how SKIZZO uses the quantifier structure during the search.

Acknowledgments

We would like to thank Uwe Egly, Stefan Woltran and Micheal Zolda for providing us the MKNCF, TRAQLA and QST tools. In particular we would like to thank Micheal Zolda for his advices about how to use the three tools. This work is partially supported by MIUR.

References

1. U. Egly, T. Eiter, H. Tompits, and S. Woltran. Solving Advanced Reasoning Tasks Using Quantified Boolean Formulas. In *Seventeenth National Conference on Artificial Intelligence (AAAI 2000)*, pages 417–422. The MIT Press, 2000.
2. C. Scholl and B. Becker. Checking equivalence for partial implementations. In *38th Design Automation Conference (DAC'01)*, 2001.
3. Abdelwaheb Ayari and David Basin. Bounded model construction for monadic second-order logics. In *12th International Conference on Computer-Aided Verification (CAV'00)*, number 1855 in LNCS, pages 99–113. Springer-Verlag, 2000.
4. C. Castellini, E. Giunchiglia, and A. Tacchella. SAT-based planning in complex domains: Concurrency, constraints and nondeterminism. *Artificial Intelligence*, 147:85–117, 2003.
5. M. Mneimneh and K. Sakallah. Computing Vertex Eccentricity in Exponentially Large Graphs: QBF Formulation and Solution. In *Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2919 of *Lecture Notes in Computer Science*. Springer Verlag, 2003.
6. D. Le Berre, M. Narizzano, L. Simon, and A. Tacchella, editors. *Second QBF solvers evaluation*. Pacific Institute of Mathematics, 2004. Available on-line at `www.qbflib.org`.
7. U. Egly, M. Seidl, H. Tompits, and M. Zolda. Comparing Different Prenexing Strategies for Quantified Boolean Formulas. In *Sixth International Conference*

on Theory and Applications of Satisfiability Testing (SAT 2003), volume 2919 of *Lecture Notes in Computer Science*. Springer Verlag, 2003.
8. C. Thiffault, F. Bacchus, and T. Walsh. Solving Non-clausal Formulas with DPLL Search. In *10th Conference on Principles and Practice of Constraint Programming (CP 2004)*, volume 3258 of *Lecture Notes in Computer Science*, pages 663–678. Springer Verlag, 2004.
9. E. Giunchiglia, M. Narizzano, and A. Tacchella. QuBE++: an Efficient QBF Solver. In *5th Formal Methods in Computer Aided Design conference (FMCAD 2004)*, volume 3312 of *Lecture Notes in Computer Science*. Springer Verlag, 2004.
10. E. Giunchiglia, M. Narizzano, and A. Tacchella. Backjumping for Quantified Boolean Logic satisfiability. *Artificial Intelligence*, 145:99–120, 2003.
11. E. Giunchiglia, M. Narizzano, and A. Tacchella. Learning for Quantified Boolean Logic Satisfiability. In *18th National Conference on Artificial Intelligence (AAAI 2002)*. AAAI Press/MIT Press, 2002.
12. E. Giunchiglia, M. Narizzano, and A. Tacchella. QBF reasoning on real-world instances. In *7th International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, Lecture Notes in Computer Science. Springer Verlag. To appear.
13. I.P. Gent, E. Giunchiglia, M. Narizzano, A. Rowley, and A. Tacchella. Watched Data Structures for QBF Solvers. In *Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2919 of *Lecture Notes in Computer Science*. Springer Verlag, 2003.
14. A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, and P. Traverso. Specifying and analyzing early requirements in Tropos. *Requirements Engineering*, 9(2):132–150, 2004.
15. Marco Benedetti. Quantifier Trees for QBFs. In *8th International Conference on Theory and Applications of Satisfiability Testing (SAT 2005)*, volume 3569 of *Lecture Notes in Computer Science*. Springer Verlag, 2005.
16. A. Biere. Resolve and Expand. In *Seventh Intl. Conference on Theory and Applications of Satisfiability Testing*, 2004. Extended Abstract.