Quantifier structure in search based procedures for QBFs

Enrico Giunchiglia, Massimo Narizzano, Armando Tacchella

Abstract

The best currently available solvers for Quantified Boolean Formulas (QBFs) process their input in prenex form, i.e., all the quantifiers have 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 quantifier structure. To tackle this problem, the standard approach is to convert such QBFs in prenex form, thereby loosing structural information about the prefix. In the case of search based solvers, the prenex form conversion introduces additional constraints on the branching heuristic, and reduces the benefits of the learning mechanisms.

In this paper we show that conversion to prenex form is not necessary: current search based solvers can be naturally extended in order to handle non prenex QBFs and to exploit the original quantifier structure. We highlight the two mentioned drawbacks of the conversion in prenex form with a simple example, and we show that our ideas can be useful also for solving QBFs in prenex form. To validate our claims, we implemented our ideas in the state-of-the-art search based solver QUBE*, and conducted an extensive experimental analysis. The results show that very substantial speedups can be obtained.*

***Index Terms*— Formal Verification, Satisfiability, Quantified Boolean Formulas**

Manuscript received April 3, 2006; revised August 8, 2006. This work was partially supported by MIUR.

E. Giunchiglia, M. Narizzano and A. Tacchella are with the Dipartimento di Informatica Sistemistica e Telematica of the Universitá di Genova, Viale Causa 13, 16145 Genova, Italy

I. Introduction

The use of Quantified Boolean Formulas (QBFs) to encode problems arising from Formal Verification (see, e.g., [1], [2], [3], [4]) and Artificial Intelligence (see, e.g., [5], [6]), has attracted increasing interest in recent years. The application-driven quest for efficiency has in turn propelled the research on solvers, and each year a comparative evaluation among the different available QBF solvers takes place, see [7] for the report of the last event in the serie. All the best currently available solvers assume that

1) the input QBF is in prenex form, i.e., all the quantifiers have to appear in the prefix of the formula separated from the purely propositional part, and
2) the input QBF is in conjunctive normal form (CNF), i.e., the propositional part of the formula (called matrix) consists of a set of clauses.

However, in many QBFs deriving from applications in computer aided verification and artificial intelligence, the propositional part is intertwined with the quantifier structure and the matrix is not in CNF. Examples of such applications are diameter calculation of sequential circuits (see, e.g., [3], [8]), model checking of early requirements (see, e.g., [9]) and formal equivalence checking of partial implementations (see, e.g., [1]). The situation is simpler in the propositional satisfiability (SAT) case, corresponding to QBFs in which all the quantifiers are existential: in SAT, the first problem does not show up, and several papers have been dedicated to efficient and effective conversions to CNF and/or to the implementation of SAT solvers able to handle non CNF formulas (see, e.g., [10], [11] for two recent papers on these issues). The solutions devised in SAT to handle non CNF formulas can be easily lifted to the more complex QBF case. Still, in the QBF case we are left with the first issue. Indeed, the standard solution is to convert any non prenex QBF into a prenex one using standard quantifier rewriting rules like

$$(\exists x \varphi(x) \wedge \forall y \psi(y)) \mapsto \exists x \forall y (\varphi(x) \wedge \psi(y))$$

or

$$(\exists x \varphi(x) \wedge \forall y \psi(y)) \mapsto \forall y \exists x (\varphi(x) \wedge \psi(y)).$$

However, in the resulting QBF, we loose the information that x and y are not one in the scope of the other. Further, as the above simple example shows, there can be more than one rule applicable at each step, and the prefix of the resulting formula depends on the order with which rules are applied. In general, given a non prenex QBF φ, there can be exponentially many QBFs

1) in prenex form,
2) equivalent to φ, and
3) each of them obtainable from φ using the above mentioned rewriting rules.

Thus, it is not clear which of these exponentially many QBFs is best, i.e., which one leads to the best performances of the QBF solver. Egly, Seidl, Tompits, Woltran and Zolda [12] define four strategies which are optimal in the sense that the resulting QBF is guaranteed to belong to the lowest possible complexity class in the polynomial hierarchy. However, the optimal reasoning strategies that they define are not the only possible ones: there can be exponentially many optimal strategies, and thus it is again not clear which of them is best. Further, the experimental analysis conducted in [12] on a series of instances encoding knowledge representation problems and involving state-of-the-art QBF solvers based on search, showed that the strategy delivering the best performances depends both on the kind of instances and on the internals of the QBF solver. Even assuming that the best (on average) strategy has been identified, the conversion of a QBF φ into a QBF φ' in prenex form causes that, for each pair of distinct variables z and z' in φ, in φ' either z will be in the scope of z' or vice versa: in the case of the QBF $\exists x \varphi(x) \wedge \forall y \psi(y)$ after the conversion to prenex form, either x will be in the scope of y or vice versa. The a priori decision about which variable should be in the scope of the other is not optimal for search based solvers. Indeed, the branching heuristic has to take into account the prefix: in the case of $\exists x \forall y (\varphi(x) \wedge \psi(y))$ (resp. $\forall y \exists x (\varphi(x) \wedge \psi(y))$) the branching heuristic has to select x (resp. y) before y (resp. x). Further, the prefix is also taken into account by the learning mechanism, and an a priori fixed ordering on the variables may reduce its effectiveness.

In this paper we show that conversion to prenex form is not necessary: current search based solvers can be naturally extended in order to handle non prenex QBFs and to exploit the original quantifier structure. We highlight the drawbacks caused by the conversion on the branching heuristic and on the learning mechanisms of search based solvers with a simple example. As the example shows, these drawbacks can lead to the exploration of search spaces bigger than the space explored by solvers handling non prenex QBFs. To validate our claims, we implemented our ideas in our solver QuBE [13], and conducted an extensive experimental analysis on QBFs not in prenex form. We also show that our ideas can be useful also for solving QBFs in prenex form, once suitably preprocessed in order to minimize the scope of each quantified variable. The results show that very substantial speedups can be obtained by using a solver reasoning on non prenex QBFs.

The paper is structured as follows. After a brief review of the logic of QBFs (Section II) and of the standard approach to solve QBFs in prenex form (Section III), we show that the procedure for prenex QBFs can be extended to handle non prenex QBFs (Section IV). The advantages of reasoning with non prenex QBFs are discussed in Section V. The implementation of these ideas in QuBE is described in Section VI, and the experimental analysis in Section VII. We conclude the paper with some remarks and related work (Section VIII).

This paper is based on and extends [14].

II. Quantified Boolean Logic

Since we focus on the issue of exploiting the quantifier structure, we consider QBFs in which the quantifiers may not be in prenex form, but in which the matrix is in CNF.

Consider a set P of *variables*. A *literal* is a variable or the negation $\overline{z}$ of a variable z. In the following, for any literal l,

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

A *clause* C is a finite disjunction $(l_1 \vee \ldots \vee l_n)$ $(n \geq 0)$ of literals such that for each pair of literals l_i, l_j in C, $|l_i| \neq |l_j|$. Finally, the set of *QBFs* is defined as the smallest set such that

- if $C_1, \ldots, C_n$ are clauses $(n \geq 0)$, $(C_1 \wedge \ldots \wedge C_n)$ is a QBF,
- if φ is a QBF and z is a variable, $Qz\varphi$ is a QBF, where Q is either the existential quantifier "$\exists$" (in which case we say that z and $\overline{z}$ are *existential*) or the universal quantifier "$\forall$" (in which case we say that z and $\overline{z}$ are *universal*). In $Qz\varphi$, φ is called the *scope* of z, and z is the variable *bound* by Q.
- if $\varphi_1, \ldots, \varphi_n$ are QBFs, $(\varphi_1 \wedge \ldots \wedge \varphi_n)$ is a QBF.

As an additional requirement on the syntax, without loss of generality and to make the formal treatment simpler, we assume that in every QBF each variable is bound by exactly one quantifier. Indeed, given an arbitrary QBF φ,

1) If φ contains a variable which is bound by more than one quantifier, it is always possible to rewrite (in linear time) φ into a new QBF φ' in which each variable is bound at most once and such that φ and φ' have the same semantics.

2) If φ contains a variable x which is not bound by any quantifier, φ can be treated as if it were $\exists x\varphi$.

For example, the expression

$$\begin{array}{r}\exists x_0(\forall y_1\exists x_1\exists x_2((x_0 \vee \overline{x}_1 \vee \overline{x}_2) \wedge (y_1 \vee \overline{x}_1 \vee x_2)\wedge \\ (x_1 \vee \overline{x}_2) \wedge (x_0 \vee x_1 \vee x_2))\wedge \\ \forall y_2\exists x_3\exists x_4((\overline{x}_0 \vee \overline{x}_3 \vee \overline{x}_4) \wedge (y_2 \vee \overline{x}_3 \vee x_4)\wedge \\ (x_3 \vee \overline{x}_4) \wedge (\overline{x}_0 \vee x_3 \vee x_4)))\end{array} \quad (1)$$

is a QBF meeting the above requirements.

On the assumption that each variable is quantified exactly once, we can represent any QBF φ as a pair $\langle$prefix,matrix$\rangle$ in which

- the *prefix* is a (partially) ordered set $\langle S, \prec\rangle$ such that
  1) each element of the set S has the form $\langle Q, z\rangle$ where Q is a quantifier, z is a variable, and Qz occurs in φ; and
  2) two elements $\langle Q_1, z_1\rangle$ and $\langle Q_2, z_2\rangle$ in the set are in (partial) order (and we write $z_1 \prec z_2$) if and only if
     a) either $Q_2 z_2$ occurs in the scope of z_1 and $Q_1 \neq Q_2$,
     b) or there exists Qz with $Q \neq Q_1$, $Q \neq Q_2$, $Q_2 z_2$ occurring in the scope of z and Qz occurring in the scope of z_1.
- the *matrix* is a set of clauses, and a clause (as in SAT) is represented as the set of literals in it.

Since we will use x_i (resp. y_i) to denote an existentially (resp. universally) quantified variable, we can simply represent a prefix $\langle S, \prec\rangle$ with the order $\prec$. From a formal point of view, this corresponds to assuming that the set P of variables is partitioned in two disjoint sets $\{x, x_1, x_2, \ldots\}$ and $\{y, y_1, y_2, \ldots\}$, being respectively the set of existentially and universally quantified variables. Then, the prefix of a QBF φ will be represented via a set of expressions of the form

$$z_1^0, \ldots, z_{m_0}^0 \prec z_1^1, \ldots, z_{m_1}^1 \prec \ldots \prec z_1^n, \ldots, z_{m_n}^n \quad (2)$$

($n, m_0, \ldots, m_n \geq 1$), standing for the set of elements $z_j^i \prec z_l^k$ with z_j^i and z_l^k in (2) and $i < k$. Thus, for example, the prefix of (1), is

$$x_0 \prec y_1 \prec x_1, x_2 \qquad x_0 \prec y_2 \prec x_3, x_4 \quad (3)$$

while the matrix of (1) is written as

$$\begin{array}{l}\{\{x_0, \overline{x}_1, \overline{x}_2\}, \{y_1, \overline{x}_1, x_2\}, \{x_1, \overline{x}_2\}, \\ \{x_0, x_1, x_2\}, \{\overline{x}_0, \overline{x}_3, \overline{x}_4\}, \\ \{y_2, \overline{x}_3, x_4\}, \{x_3, \overline{x}_4\}, \{\overline{x}_0, x_3, x_4\}\}.\end{array} \quad (4)$$

The above representation of QBFs generalizes the standard definition and representation of QBFs used, e.g., in [15], [16], [17], [18], [19], in which formulas are restricted to be in prenex form. In our setting, a QBF φ is in *prenex form* if for each existential variable x and universal variable y, either $x \prec y$ or $y \prec x$. The prefix of a QBF in prenex form can be represented via a single expression of the form (2), assuming that it contains at least one existential and one universal variable (if φ is a QBF in which all the quantifiers are of the same type, our representation of the prefix will be the empty set).[1]

Consider a QBF φ with prefix $\prec$ and matrix Φ.

We define the *prefix level of a variable* z as the length of the longest chain $z_1 \prec z_2 \prec \ldots \prec z_n \prec z_{n+1}$ ($n \geq 0$) in the prefix such that $z_{n+1} = z$. The *prefix level of a QBF* φ is the maximum of the prefix levels of the variables in φ.[2] For instance, in (1) the prefix level of x_0 is 1; both x_1 and x_2 have prefix level 3; and the prefix level of (1) is also 3. A variable z is *top in* φ if it has prefix level 1, i.e., if it is existential (resp. universal) and it is not in the scope of an universal (resp. existential) variable.

The *value* or *semantics of* φ can be defined recursively as follows:

- If the matrix of φ is empty, then φ is true.
- If the matrix of φ contains an empty clause, then φ is false.
- If z is a top existential (resp. universal) variable in φ, φ is true if and only if the QBF φ_z or (resp. and) $\varphi_{\overline{z}}$ are true.

If l is a literal and ψ is a QBF, then

1) the matrix of ψ_l is obtained from the matrix Ψ of ψ by (i) eliminating the clauses C such that $l \in C$, and eliminating $\overline{l}$ from the other clauses in Ψ;
2) the prefix of ψ_l is obtained from the prefix $\prec$ of ψ by removing the pairs $|l|, z$ such that $|l| \prec z$ or $z \prec |l|$.

Two QBFs are *equivalent* if they have the same value.

III. *Q-DLL* for QBFs in prenex form

Most of the available QBF solvers assume that the input formula is in prenex form. Consider a QBF φ in prenex form, with prefix $\prec$ and matrix Φ.

A simple recursive procedure for determining the value of φ, starts with φ and simplifies the current φ to $(\varphi_z \vee \varphi_{\overline{z}})$ (resp. $(\varphi_z \wedge \varphi_{\overline{z}})$), where z is a heuristically chosen top existential (resp. universal) variable in φ. On the basis of the values of φ_z and $\varphi_{\overline{z}}$, the value of φ can be determined according to the semantics of QBFs. The base case occurs when either the empty clause or the empty set of clauses are produced.

[1] Notice that two different QBFs can have the same representation. For instance, $\forall y\exists x_1\exists x_2((x_1 \vee y) \wedge (x_2 \vee y))$ and $\forall y\exists x_2(\exists x_1(x_1 \vee y) \wedge (y \vee x_2))$ are both represented with the pair $\langle\{y \prec x_1, x_2\}, \{\{x_1, y\}, \{x_2, y\}\}$. However, two QBFs with the same representation have the same semantics.

[2] For a QBF φ in prenex form, the prefix level of φ is equal to its *number of alternations*.

```
0 function Q-DLL(φ)
1   if (⟨a contradictory clause is in φ⟩) return FALSE;
2   if (⟨the matrix of φ is empty⟩) return TRUE;
3   if (⟨l is unit in φ⟩) return Q-DLL(φ_l);
4   l := ⟨a top literal in φ⟩;
5   if (⟨l is existential⟩)
      return Q-DLL(φ_l) or Q-DLL(φ_l̄);
6   else
      return Q-DLL(φ_l) and Q-DLL(φ_l̄).
```

Fig. 1. The algorithm of *Q-DLL*.

Cadoli, Giovanardi and Schaerf [15] introduced various improvements to this basic procedure on the basis of two lemmas reported here for the sake of reference.[3] The first improvement is that we can immediately conclude that φ is false if its matrix contains a contradictory clause. A clause C is *contradictory* if it does not contain existential literals. An example of a contradictory clause is the empty clause.

Lemma 1: Let φ be a QBF in prenex form. If the matrix of φ contains a contradictory clause, then φ is false.

Proof: Lemma 2.1 in [16]. □

The second improvement allows us to immediately simplify φ to φ_l if l is unit in φ. A literal l is *unit* in φ if l is existential and for some $m \geq 0$,

- a clause $\{l, l_1, \ldots, l_m\}$ belongs to the matrix of φ; and
- each literal l_i ($1 \leq i \leq m$) is universal and such that $|l| \prec |l_i|$, i.e., l_i is in the scope of l.

Lemma 2: Let φ be a QBF in prenex form. Let l be a literal which is unit in φ. φ and φ_l are equivalent.

Proof: Lemma 2.6 in [16]. □

A simple recursive presentation of a procedure including the two improvements above is presented in Figure 1. Given a QBF φ

1) FALSE is returned if a contradictory clause is in the matrix of φ (line 1); otherwise
2) TRUE is returned if the matrix of φ is empty (line 2); otherwise
3) at line 3, φ is recursively simplified to φ_l if l is unit; otherwise
4) at line 4 a top literal l is chosen (and we say that l has been *assigned as a branch*) and
   - if l is existential (line 5), the "**or**" of the results of the evaluation of φ_l and $\varphi_{\overline{l}}$ is returned;
   - otherwise (line 6), l is universal, and the "**and**" of the results of the evaluation of φ_l and $\varphi_{\overline{l}}$ is returned.

It is easy to see that *Q-DLL*, if given a QBF without universal quantifiers, is the same as the famous Davis-Logemann-Loveland procedure *DLL* [20] for (SAT). *Q-DLL* is correct: it returns TRUE if the input QBF is true and FALSE otherwise, as stated by the following theorem.

Theorem 1: Let φ be a QBF in prenex form. *Q-DLL*(φ) returns TRUE if φ is true and FALSE otherwise.

Proof: The proof is by induction on the size of the prefix of φ. The base case if trivial, while the step case is an easy consequence of Lemma 1, Lemma 2 and of the semantics of QBFs. □

Current state-of-the-art solvers based on search extend the procedure described in Figure 1 by implementing some form of good and/or nogood, static and/or dynamic learning [21], [22], [19], [23] and pure literal fixing [15].

In nogood learning, new clauses (called *nogoods*) are (temporarily) added to the matrix: these new clauses are obtained by resolving clauses in the input matrix or previously learned clauses. In good learning, *goods* (i.e., conjunctions of literals) are added as if in disjunction with the matrix:

1) the initial goods correspond to sets S of literals propositionally entailing the matrix (i.e., such that for each clause C in the matrix $C \cap S \neq \emptyset$) and are usually computed when the matrix of φ becomes empty (line 2 in Figure 1);
2) then, additional goods may be obtained by resolving already computed goods (see [23] for more details).

In static learning, nogoods and goods are computed before the search starts and they are usually added to the matrix permanently. In dynamic learning, nogoods and goods are computed during the search and they are usually added to the matrix temporarily. Independently from the type of learning implemented, goods and nogoods are used to prune the search space and can produce very significant improvements in the performances [21], [22], [19], [23].

Pure literal fixing allows to assign an existential (resp. universal) literal l if $\overline{l}$ (resp. l) does not belong to any clause in the matrix. In the presence of learning, care has to be taken in the definition and implementation of such rule [24]. Given its effectiveness in pruning the search space, pure literal fixing is used by most state-of-the-art solvers.

IV. *Q-DLL* for arbitrary QBFs

Consider a QBF φ, with prefix $\prec$ and matrix Φ, and assume that $\prec$ is arbitrary, i.e., not necessarily in prenex form.

As we anticipated in the introduction, in order to decide the value of φ, the standard approach is to first convert

[3] In [15], [16], the procedure presented by Cadoli, Giovanardi, Giovanardi and Schaerf included other simplification rules, like pure literal fixing and trivial truth, that we do not discuss here since they are not relevant for the goals of this paper.

φ into prenex form, and then use one of the available solvers. This is the approach followed, e.g., in [4], [3]. The conversion can be carried out by extending the prefix till we get a total order. However, this can have some serious drawbacks (detailed in the next section) and it is not the only possible approach.

The first important observation is that it is possible to generalize Lemmas 1 and 2 to arbitrary QBFs, not necessarily in prenex form. Indeed, the corresponding generalizations of the two lemmas are a consequence of the following fact, first proved in [25], for QBFs in prenex form.

Lemma 3: Let φ be a QBF with prefix $\prec$ and matrix Φ. Let C be a clause in Φ, and let C' be the clause obtained from C by removing the universal literals $l \in C$ such that there is no existential literal $l' \in C$ with $|l| \prec |l'|$. Let φ' be the QBF obtained from φ by replacing C with C'. φ and φ' are equivalent.

Proof: In the hypotheses of the lemma, let $C = \{l_1, \ldots, l_n, l_{n+1}, \ldots, l_m\}$. Assume $C \setminus C' = \{l_{n+1}, \ldots, l_m\}$ and that $m \geq n+1$ (if $m = n$ the proof is trivial). Further, without loss of generality, we assume that $l_i \prec l_{i+1}$, $1 \leq i < m$. Then, φ has the form ($p \geq m$)

$$\ldots Q_1|l_1| \ldots \forall |l_m| \ldots Q_p z_p \{\ldots \{l_1, \ldots, l_m\}, \ldots\}, \quad (5)$$

where $Q_p z_p$ is such that C is in its scope, and there is no other $Q_{p+1} z_{p+1}$ in the scope of $Q_p z_p$ such that C is in the scope of $Q_{p+1} z_{p+1}$. Equation (5) stands for

$$\ldots Q_1|l_1| \ldots \forall |l_m| \ldots Q_p z_p (\ldots (l_1 \vee \ldots \vee l_m) \ldots).$$

Then, since each variable quantified in the scope of $\forall |l_m|$ does not occur in C, given the associativity and commutativity of "$\wedge$", φ can be rewritten as

$$\ldots Q_1|l_1| \ldots \forall |l_m| \ldots Q_p z_p ((l_1 \vee \ldots \vee l_m) \ldots),$$

i.e., φ has the form

$$\ldots Q_1|l_1| \ldots \forall |l_m| \ldots Q_p z_p ((l_1 \vee \ldots \vee l_m) \wedge \psi).$$

Again, since each variable which is quantified in the scope of $\forall |l_m|$ does not occur in C, by applying standard rewriting rules for quantifiers, the above equation can be rewritten as

$$\ldots Q_1|l_1| \ldots (\forall |l_m| (l_1 \vee \ldots \vee l_m) \wedge \forall |l_m| \ldots Q_p z_p \psi),$$

equivalent to

$$\ldots Q_1|l_1| \ldots ((l_1 \vee \ldots \vee \forall |l_m| |l_m) \wedge \forall |l_m| \ldots Q_p z_p \psi),$$

equivalent to

$$\ldots Q_1|l_1| \ldots ((l_1 \vee \ldots \vee l_{m-1}) \wedge \forall |l_m| \ldots Q_p z_p \psi).$$

In the above equation, the literal l_m no longer belongs to the clause C. By iterating the above reasoning process, all the literals in $C \setminus C'$ can be eliminated from C. Since all the operations we performed can be reversed, we can get back to the original QBF in which all the literals in $C \setminus C'$ have been eliminated from C, and hence the thesis. □

The generalization of Lemma 1 to the case of arbitrary QBFs can be stated as follows.

Lemma 4: Let φ be a QBF. If the matrix of φ contains a contradictory clause, then φ is false.

Proof: By Lemma 3, if $C = (l_1 \vee \ldots \vee l_n)$ is a contradictory clause in the matrix of φ, we can substitute C with the empty clause, and hence the thesis. □

Before we generalize Lemma 2 to arbitrary QBFs, we need to generalize the definition of unit to the case in which φ is not in prenex form. A literal l is *unit* in φ if l is existential and for some $m \geq 0$,

- a clause $\{l, l_1, \ldots, l_m\}$ belongs to the matrix of φ; and
- each literal l_i ($1 \leq i \leq m$) is universal and such that $|l_i| \not\prec |l|$, i.e., l is not in the scope of l_i.

Notice that the above definition generalizes the one given in the previous section: If φ is in prenex form, the two definitions are equivalent.

Lemma 5: Let φ be a QBF. Let l be a literal which is unit in φ. φ and φ_l are equivalent.

Proof: Let $C = \{l, l_1, \ldots, l_m\}$ be a clause that causes l to be unit in φ. By Lemma 3, we can substitute C with $\{l\}$ in φ and obtain an equivalent QBF φ'. Consider φ'. Since each variable quantified in the scope of $\exists |l|$ does not occur in $\{l\}$, and given the associativity and commutativity of "$\wedge$", with manipulations analogous to those described in the proof of Lemma 3, φ' can be rewritten as

$$\ldots \exists |l| ((l) \ldots),$$

i.e., φ has the form

$$\ldots \exists |l| ((l) \wedge \psi).$$

The thesis follows from the fact that

$$\exists |l| ((l) \wedge \psi) \quad (6)$$

is logically equivalent to ψ_l in second order propositional logic. Therefore we can replace ψ_l for (6) in φ' and obtain an equivalent QBF. Hence the thesis. □

Theorem 2: Let φ be a QBF. *Q-DLL*(φ) returns TRUE if φ is true and FALSE otherwise.

Proof: Analogously to Theorem 1, the proof is by induction on the size of the prefix of φ. The base case if trivial, while the step case is an easy consequence of Lemma 4, Lemma 5 and of the semantics of QBFs. □

The above theorem states that *Q-DLL* in Figure 1 maintains its correctness even when the input QBF φ is not in prenex form. The introduction of learning and pure literal fixing does not produce complications and the techniques used in the prenex case can be easily generalized to the non prenex case, so we take them for granted.

A possible execution of *Q-DLL* on (1) is represented by the tree in Figure 2. In the Figure, each node of the tree is labeled with a set of clauses and it is numbered according to the order in which *Q-DLL* explores the search space; the root node has the input matrix (4) as label, and the other nodes contain the matrices resulting from assigning the literals marked along the path from the root to each of them; each leaf is marked with $\{\{\}\}$ to denote that the resulting set of clauses contains at least an empty clause.

V. Prenex vs non prenex solvers

Consider a non prenex QBF φ with prefix $\prec$ and matrix Φ. The standard solution to decide the value of φ is to first convert φ in prenex form and then use a standard QBF solver as the ones described in [16], [17], [19]. Conversion in prenex form can be done in linear time using standard rewriting rules for quantifiers, like those mentioned in the introduction. The resulting QBF

1) has a prefix $\prec'$ extending $\prec$,
2) has the same matrix Φ of φ, and
3) is guaranteed to be equivalent to φ.

However, the prefix of the resulting QBF may vary depending on the order with which rules are applied. In particular, if φ_1 and φ_2 are two versions of φ in prenex form, it may be the case that the prefixes of φ_1 and φ_2 are different, and even that φ_1 and φ_2 have different prefix levels.

Given this fact, it is not clear which of the total orders extending $\prec$ is best, i.e., which one leads to the best performances of the QBF solver at hand. In [12], the authors define four prenexing strategies which, given a QBF φ, are guaranteed to lead to a *prenex-optimal QBF (w.r.t. φ)*, i.e., to a QBF

- in prenex form;
- with the same matrix of φ;
- whose prefix extends the prefix of φ; and
- whose prefix level is equal to the prefix level of φ, assuming that the variables in φ with maximum prefix level are existential.[4]

[4]The assumption that in a QBF the variables with maximum prefix level are existential is not a restriction. Indeed, if y is a universal variable in a QBF φ having maximum prefix level, all the occurrences of y and $\overline{y}$ in the matrix of φ can be safely removed (Lemma 3), and hence also $\forall y$ can be removed from the prefix of φ.

In the case of the QBF (1) the prefix of a prenex-optimal QBF is

$$x_0 \prec y_1, y_2 \prec x_1, x_2, x_3, x_4, \tag{7}$$

while a QBF with prefix

$$x_0 \prec y_1 \prec x_1, x_2 \prec y_2 \prec x_3, x_4 \tag{8}$$

is non prenex-optimal. Prefix (7) is better than (8) –at least from a theoretical point of view– since the computational effort to evaluate QBFs is related to the prefix level of the QBF: the lower the prefix level, the lower is the complexity class which the formula belongs to. Thus, in general, it is reasonable to choose strategies which yield prenex QBFs having the same prefix level of the initial QBF, i.e., prenex-optimal QBFs.

The four strategies defined by [12] –denoted by $\exists^{\uparrow}\forall^{\uparrow}$, $\exists^{\uparrow}\forall^{\downarrow}$, $\exists^{\downarrow}\forall^{\uparrow}$, and $\exists^{\downarrow}\forall^{\downarrow}$– rely on a different treatment of existential and universal quantifiers. Intuitively, $\exists^{\uparrow}$ (resp. $\exists^{\downarrow}$) means that existential quantifiers are shifted in such a way that they are placed in the resultant quantifier prefix as high (resp. low) as possible yielding a quantifier ordering which is compatible with the original one. A similar meaning applies to $\forall^{\uparrow}$ and $\forall^{\downarrow}$.

For example, given a QBF of the form

$$\exists x(\forall y_1 \exists x_1 \forall y_2 \exists x_2 \varphi_0 \wedge \forall y_1' \exists x_1' \varphi_1 \wedge \exists x_1'' \varphi_2), \tag{9}$$

the prefixes of the QBFs in prenex form resulting from the application of the 4 strategies correspond to:

$$\begin{array}{l} \exists^{\uparrow}\forall^{\uparrow} = \exists x \exists x_1'' \;\; \forall y_1 \forall y_1' \;\; \exists x_1 \exists x_1' \;\; \forall y_2 \;\; \exists x_2, \\ \exists^{\uparrow}\forall^{\downarrow} = \exists x \exists x_1'' \;\; \forall y_1 \forall y_1' \;\; \exists x_1 \exists x_1' \;\; \forall y_2 \;\; \exists x_2, \\ \exists^{\downarrow}\forall^{\uparrow} = \exists x \;\; \forall y_1 \forall y_1' \;\; \exists x_1 \;\; \forall y_2 \;\; \exists x_2 \exists x_1' \exists x_1'', \\ \exists^{\downarrow}\forall^{\downarrow} = \exists x \;\; \forall y_1 \;\; \exists x_1 \;\; \forall y_2 \forall y_1' \;\; \exists x_2 \exists x_1' \exists x_1''. \end{array} \tag{10}$$

(See [12] for more details). However these 4 strategies are not the only possible ones: indeed, there can be exponentially many strategies leading to prenex-optimal QBFs. For example, considering (9), the QBF

$$\exists x \;\; \forall y_1 \;\; \exists x_1 x_1'' \;\; \forall y_2 y_1' \;\; \exists x_2 x_1' \;\; (\varphi_0 \wedge \varphi_1 \wedge \varphi_2)$$

is also prenex-optimal and its prefix is different from those in (10). Thus, it is again not clear which of the prenex-optimal QBFs/strategies should be considered. Further, the experimental analysis conducted in [12] shows that even considering the 4 optimal strategies therewith defined, the strategy delivering the best performances depends both on the kind of instances and on the internals of the QBF solvers.

In addition to the considerations above, converting a non prenex QBF to a prenex form, has two drawbacks.

First, when deciding which literal to assign as a branch, the selection is restricted among the top literals: imposing a total order on the prefix can severely limit the choice up to the point that the branching heuristic becomes useless.

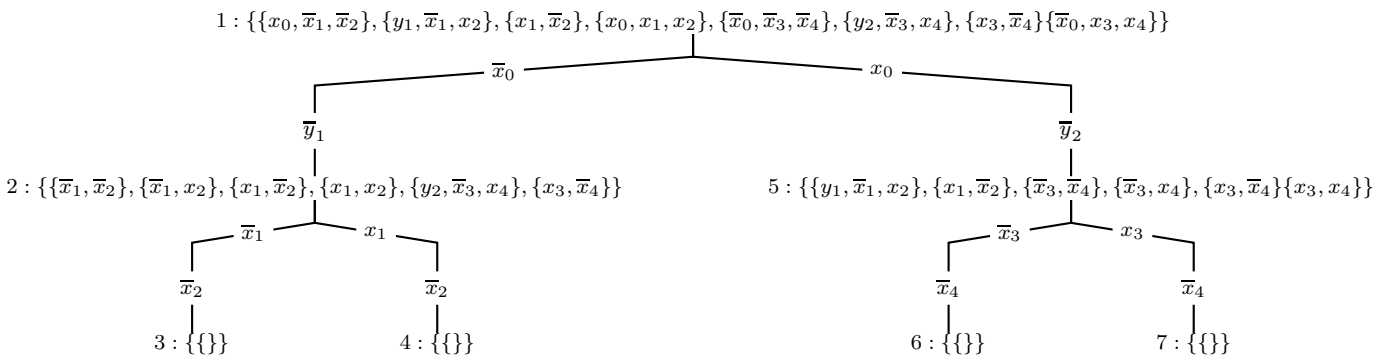

Fig. 2. Search tree of *Q-DLL* on the QBF with the matrix at the root and the prefix corresponding to $x_0 \prec y_1 \prec x_1, x_2$ and $x_0 \prec y_2 \prec x_3, x_4$.

Considering, e.g., an instance φ with three variables x_1, x_2 and y and prefix either $x_1 \prec x_2$ or $y \prec x_2$: the prefix according to which $x_1 \prec y \prec x_2$ imposes a fixed ordering on the variables to branch on. In other words, extending the order $\prec$ amounts to impose additional constraints on the variables to be used for branching, and thus to limit the power of the dynamic branching heuristic. This may cause the exploration of necessarily bigger search trees than those that might have been explored if given the original non prenex QBF. Consider for example the search tree in Figure 2 representing a possible trace of *Q-DLL* if given the non prenex QBF 1. As it can be seen from the figure, *Q-DLL* may assign x_1 as a branch without having assigned y_2 before (and this in a total order setting would imply $x_1 \prec y_2$) and assign y_2 later[5] as a branch without having assigned x_1 before (and this in a total order setting would imply $y_2 \prec x_1$): since it is not possible to have both $x_1 \prec y_2$ and $y_2 \prec x_1$, the search tree showed in Figure 2 cannot be explored by *Q-DLL* if run on a QBF with the same matrix and a total order prefix extending (3). Given that the search tree in the Figure is optimal (any other search tree possibly explored by *Q-DLL* on (1) has a bigger number of literals assigned as branches) it follows that extending (1) to a total order will necessarily cause the exploration of a search tree bigger than the one in Figure 2.

Second, for each existential variable x and universal variable y in a clause C in the matrix of φ, either $x \prec y$ or $y \prec x$. Because of this, the pruning induced by each clause in the matrix (corresponding to Lemmas 4 and 5) is unaffected if φ is converted in prenex form. However, if the solver implements some form of learning, it may be the case that some nogood and/or good Z is added to the matrix and Z contains an existential variable x and universal variable y with neither $x \prec y$ nor $y \prec x$. This happens, for example, if the nogood

$$\{y_1, x_2, x_3, x_4\} \tag{11}$$

(obtained by resolving the second, fourth and last of the clauses in (1)) is added to the matrix of φ, e.g., because the solver implements static nogood learning. In these cases converting the formula in prenex form may limit the pruning effects of learning. In the case of the nogood (11), if, e.g., x_2 and x_3 are assigned to false, x_4 can be propagated as unit: This would not happen if the prefix is extended with $y_1 \prec x_4$.[6] Analogously, assume that the good

$$\{y_1, x_1, \overline{x}_2, y_2, x_3, \overline{x}_4\} \tag{12}$$

is added to the matrix of φ when the matrix becomes empty after assigning $\overline{x}_0$ and the literals in (12) to true. Because of (12), y_1 (resp. y_2) could be directly assigned to false (as unit) in any branch in which y_2 (resp. y_1) is assigned to true, and this would not necessarily happen if the prefix is extended with $x_3 \prec y_1$ or $x_4 \prec y_1$ (resp. $x_1 \prec y_2$ or $x_2 \prec y_2$). See [23] for more details on good and nogood learning.

In Section VII-C we further discuss the disadvantages of the prenex form conversion in the case of diameter calculation problems.

VI. Exploiting Quantifier Structure in QuBE

As it is the case in SAT, real implementations of *Q-DLL* extend the basic algorithm with more powerful simplifica-

[5]Considering the QBF (1), it can be objected that both y_1 and y_2 could be eliminated during the preprocessing since they are pure literals: a slightly more complicated example in which this critique does not apply and all the considerations we make still hold, can be obtained by simply adding the two clauses $\{\overline{y}_1, \overline{x}_1, x_2\}$ and $\{\overline{y}_2, \overline{x}_3, x_4\}$ to the matrix.

[6]It can be objected that once x_2 and x_3 are assigned, so it is also y_1. This is due to the extreme simplicity of our example. It is relatively easy to build a more complex one, with more variables and clauses, in which x_2 and x_3 will be assigned as unit.

tion rules (e.g., pure literal fixing), static and/or dynamic nogood and/or good learning, and sophisticated dynamic branching heuristics for deciding on which literal to branch on. Examples of solver featuring the above characteristics are QUBE [24], YQUAFFLE [22], and SEMPROP [19]: see the respective papers for more details.

We implemented the algorithm described in the previous section on top of our solver QUBE [24]. The most recent version of QUBE participated "hors concours" to the 2006 QBF evaluation, and, according to the publicly available results, it was very competitive with all the other solvers in the evaluation.

QUBE reads instances in prenex form and features state-of-the-art backtracking techniques, heuristics and data structures. In the following, we use QUBE(TO) to denote the old version of QUBE solving QBF instances in prenex form, and QUBE(PO) to denote the version of QUBE modified in order to exploit the quantifier structure.

The main difference between QUBE(PO) and QUBE(TO) lies in the branching heuristic. The branching heuristic in QUBE(TO) is implemented by associating a counter to each literal l storing the number of constraints in which l appears. Each time a constraint is added, the counter is incremented; when a learned constraint is removed, the counter is decremented. In order to choose a branching literal, QUBE(TO) stores the literals in a priority queue according to

1) the prefix level of the corresponding variable;
2) the value of the counter; and
3) the numeric id.

Initially the score of each literal is set to the value of the associated counter. 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. Thus, QUBE(TO) branching heuristic resembles the Variable State Independent Decaying Sum (VSIDS) decision heuristic from ZCHAFF [26]: in the case of a SAT instance, QUBE(TO) branching heuristic behaves as VSIDS.

In QUBE(PO), the ordering of the priority queue cannot be maintained using the mechanisms of QUBE(TO). Indeed, variables can no longer be sorted according to their prefix level. However, for efficiency reason, it is important to sort literals to be used for branching in a priority queue taking into account both their position in the prefix and their score. In QUBE(PO) this problem has been solved by associating a counter and a score to each literal l:

- the counter is initialized and maintained as in QUBE(TO); and
- the value of the score of l is (assuming the prefix level of l is k)
  - the value of the counter associated to l if there is no literal with prefix level $k+1$ and in the scope of l; and
  - the value of the counter associated to l plus the maximum of the scores of the literals having prefix level $k+1$ and in the scope of l, otherwise.

Then, literals are sorted in the priority queue according to their score. In this way, we guarantee that

1) if $|l| \prec |l'|$ then the score of l is greater than the score of l', and this ensures that $|l|$ is assigned before $|l'|$; and
2) in the case of a SAT instance, then the score of each literal is equal to the value of its counter, and this ensures that literals are selected according to the VSIDS heuristic.

The other essential modification has been the implementation of a data structure allowing to efficiently check whether for any two variables z and z', $z \prec z'$ holds. In QUBE(TO) it is sufficient to store the prefix level $pl(z)$ of each variable z: for any two variables z and z', $z \prec z'$ if and only if $pl(z) < pl(z')$. This is no longer the case if the prefix is not total. However, it is sufficient to associate to each variable z two counters $d(z)$ and $f(z)$:

1) $d(z)$ represents the time stamp of when a depth first search (DFS) of the tree representing the formula first discovers Qz (where Q is a quantifier as usual),
2) $f(z)$ represents the time stamp of when the same DFS used for setting $d(z)$ finishes to visit the subtree of the formula with root Qz.

If we assume

1) that the time stamp is initially set to 1 and that 1 is also the value $d(z)$ associated to the first quantified variable z met by the DFS;
2) that z' represents the variable with the greatest $d(z')$ so far, and whose $f(z')$ is not yet set; and
3) that the DFS increments the time stamp when it finds a quantifier different from the one binding z',

the values associated to each variable in (1) are:

$$\begin{array}{llll} d(x_0) = 1, & d(y_1) = 2, & d(x_1) = 3, & d(x_2) = 3, \\ & f(y_1) = 3, & f(x_1) = 3, & f(x_2) = 3, \\ & d(y_2) = 4, & d(x_3) = 5, & d(x_2) = 5, \\ f(x_0) = 5, & f(y_2) = 5, & f(x_3) = 5, & f(x_2) = 5. \end{array}$$

As an easy consequence of the parenthesis theorem [27] we get that for any two variables z and z', $z \prec z'$ if and only if

$$d(z) < d(z') \leq f(z). \tag{13}$$

Notice that the above condition generalizes the test $pl(z) < pl(z')$ (sufficient if the QBF is in total order): indeed, in the case of a QBF in total order, for each variable z, $f(z)$ is equal to the prefix level of the formula, and $d(z)$ is equal to $pl(z)$. It is also obvious that given a QBF in prenex

form, the overhead produced by testing (13) instead of $pl(z) < pl(z')$ is absolutely negligible.

VII. Experimental Analysis

All the results presented in this section are obtained on a farm of 10 identical rack-mount PCs, each one equipped with a 3.2GHz PIV processor, 1GB of main memory and running Debian/GNU Linux. The amount of CPU time granted to the solvers has been set to 600 seconds (10 minutes), unless specified otherwise. When the run time of a solver exceeds the time limit, we terminate it and report a "timeout" condition. We used two versions of QuBE, QuBE(TO) for prenex QBF evaluation and QuBE(PO) for non prenex QBF evaluation. Both systems are written in ANSI C++ and compiled on the platforms above using `g++` with level 3 optimization. The plots presented have been obtained using the `calc` program from the `openoffice` suite.

To compare the performances of QuBE(PO) and QuBE(TO), we have used three suites of non prenex QBF instances, and also the set of prenex QBFs used in QBFEVAL 2006 [28].

A. Nested CounterFactual Problems (NCF)

These instances are the same used in [12] resulting from the QBF encoding of nonmonotonic reasoning problems. These instances suit well the needs of our experimental evaluation since they are automatically generated in non prenex form, and each one is then converted in prenex form according to the 4 different optimal strategies $\exists^{\uparrow}\forall^{\uparrow}$, $\exists^{\downarrow}\forall^{\downarrow}$, $\exists^{\downarrow}\forall^{\uparrow}$, $\exists^{\uparrow}\forall^{\downarrow}$ defined in [12].[7] The generator takes four parameters $\langle$DEP, VAR, CLS, LPC$\rangle$ which have been set as follows: 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 $\{3, 4, 5, 6\}$. For each setting of $\langle$DEP, VAR, CLS, LPC$\rangle$ we have generated 100 instances, and for each instance we obtained 4 different prenex QBFs and one non prenex QBF. Finally we have run QuBE(TO) and QuBE(PO) on the prenex and non prenex instances respectively.

On all such instances, QuBE(PO) compares very well with respect to QuBE(TO). Table I gives an overview of the results. In the Table,

- "Strategy" is the optimal strategy used to convert the formula in prenex form;
- "$>$" (resp. "$<$") is the number of instances for which QuBE(TO) is slower (resp. faster) than QuBE(PO) of more than 1s;
- "$=\pm 1s$" is the number of instances for which QuBE(TO) is within 1s from QuBE(PO) plus the number of instances on which both solvers time out;
- "$\gg$" (resp. "$\ll$") is the number of instances for which QuBE(TO) (resp. QuBE(PO)) times out while QuBE(PO) (resp. QuBE(TO)) does not;
- "$\bowtie$" is the number of instances for which both QuBE(TO) and QuBE(PO)) exceed the timeout;
- "$>10\times$" (resp. "$10\times<$") is the number of instances which are solved by both systems, but for which QuBE(TO) is at least 1 order of magnitude slower (resp. faster) than QuBE(PO).

[7] Both the generator and the converter to prenex form have been gently provided by the authors of [12].

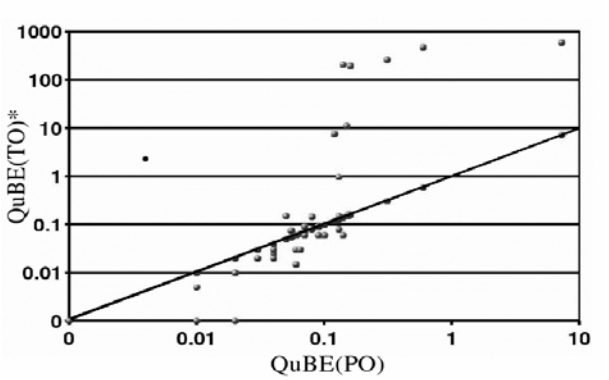

Fig. 3. QuBE(TO)*** vs.** QuBE(PO) **on NCF instances**

As it can be seen from the first four rows of Table I, on the NCF suite QuBE(PO) outperforms QuBE(TO) no matter which prenexing strategy is used. Further, among the four prenexing strategies used, $\exists^{\uparrow}\forall^{\uparrow}$ is the one leading to the best performances of QuBE(TO).

The plot in Figure 3 further highlights QuBE(PO) good performances. Here we compare QuBE(PO) and the ideal solver QuBE(TO)* that always fares the best time among those obtained by QuBE(TO) using the various prenexing strategies. In the plot in Figure 3, each bullet represents a setting of the parameters $\langle$DEP, VAR, CLS, LPC$\rangle$, QuBE(PO) median solving time is on the x-axis (log scale), while QuBE(TO)* median solving time, is on the y-axis (log scale). The diagonal serves as reference: the bullets above the diagonal are settings where QuBE(PO) performs better than QuBE(TO)*, while the dots below are the settings where QuBE(PO) is worse than QuBE(TO)*. The bullets on the diagonal represent the solving time of QuBE(PO). Even in such disadvantageous scenario, QuBE(PO) is competitive with QuBE(TO)*: QuBE(TO)* median time exceeds the timeout for some setting of the parameters while this is never the case for QuBE(PO).

Suite	Strategy	$>$	$<$	$=\pm 1s$	$\gg$	$\ll$	$\bowtie$	$> 10\times$	$10\times <$
NCF	$\exists^{\uparrow}\forall^{\uparrow}$	746	7	5247	370	1	1323	587	1
	$\exists^{\downarrow}\forall^{\downarrow}$	1061	0	4939	441	0	1324	847	0
	$\exists^{\downarrow}\forall^{\uparrow}$	1001	0	4999	425	0	1324	758	0
	$\exists^{\uparrow}\forall^{\downarrow}$	999	0	5001	425	0	1324	757	0
FPV	$\exists^{\uparrow}\forall^{\uparrow}$	627	208	70	68	43	44	190	0
DIA	$\exists^{\uparrow}\forall^{\uparrow}$	52	0	39	29	0	9	46	0
PROB	$\exists^{\uparrow}\forall^{\uparrow}$	42	0	7	0	0	0	1	0
FIXED	$\exists^{\uparrow}\forall^{\uparrow}$	12	6	7	4	0	119	1	1

TABLE I. QuBE(TO) **vs** QuBE(PO)**. Rows 1-4 are nonmonotonic reasoning problems. Row 5 are formal verification of early requirements problems. Row 6 are diameter calculation of models from** NuSMV**. Rows 7 and 8 are instances in the probabilistic and fixed classes of the 2006 QBF evaluation.**

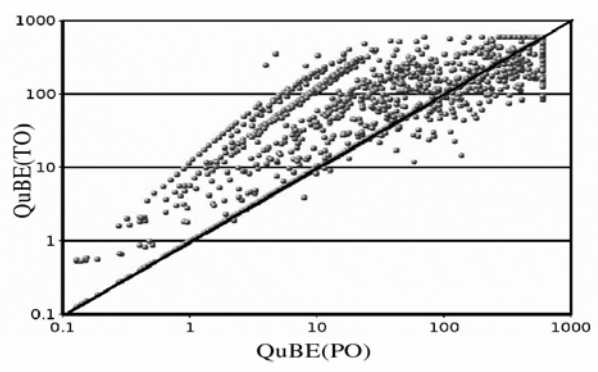

Fig. 4. QuBE(TO) **vs.** QuBE(PO) **on FPV instances**

B. Formal Property Verification Problems (FPV)

This suite is comprised of 905 QBFs obtained from the application described in [9], [29], where QBF reasoning is applied to model checking early requirements on the behavior of Web services' compositions. Each model checking problem corresponds to a set of non prenex QBFs, that can be solved by QuBE(PO). As in the case of the NCF suite, each non prenex QBF can be converted to a prenex one using some prenexing strategy, and then fed as input to QuBE(TO). Here and on the following benchmarks, we use the $\exists^{\uparrow}\forall^{\uparrow}$ prenexing strategy, i.e., the one leading to the best performances of QuBE(TO) on the NCF suite.

Looking at the results on the FPV suite in Figure 4 (same layout as Figure 3), we can see that the odds are on the side of QuBE(PO) also in this case, although the performance gap is less impressive than on the NCF suite. Notice that QuBE(TO) is sometimes faster than QuBE(PO), and this can be explained by the fact that the two solvers branch on different literals, and sometimes the choices done by QuBE(TO) may lead to smaller trees (this despite the drawbacks on the learning mechanism discussed in Section V). However, looking also at the details on the fifth row of Table I, we can see that QuBE(PO) performs at least one order of magnitude better than QuBE(TO) on 258 problems, compared to the 43 problems where the opposite happens (this count includes also the instances solved by only one system).

C. Diameter Calculation Problems (DIA)

This suite is comprised of 91 QBFs that compute the state space diameter of models bundled in the NuSMV [30] distribution. In particular, we considered the model of a counter (`counter`), the model of a chain of inverters (`ring`), the model of a distributed mutual exclusion protocol (`dme`), and the model of a semaphore-based mutual exclusion protocol (`semaphore`). First, we derived parametric versions of all the models considered by suitably modifying the original examples, e.g., from the `counter` example we obtained the `counter<N>` models with `N` (number of state bits) varied in $\{4, 5, 6, 7, 8\}$. Then, for each such model M, we computed the QBFs required to evaluate the diameter of M as follows. Let $\underline{s}$ be the vector of Boolean state variables of M, $I(\underline{s})$ be a propositional formula which is satisfiable exactly when $\underline{s}$ is an initial state of M, and $T(\underline{s}, \underline{s}')$ be a propositional formula which is satisfiable exactly when $\underline{s}'$ is a successor state of $\underline{s}$ in the transition relation of M.[8] Now consider the following QBF ϕ_n (assuming that we allow for arbitrary combinations of conjunctions "$\wedge$", disjunctions "$\vee$", negations "$\neg$", equivalences "$\equiv$", and quantifiers):

$$\begin{array}{c}\exists \underline{x}^{n+1}(\exists \underline{x}^0 \ldots \exists \underline{x}^n (I(\underline{x}^0) \wedge \bigwedge_{i=0}^{n} T(\underline{x}^i, \underline{x}^{i+1})) \wedge \\ \forall \underline{y}^0 \ldots \forall \underline{y}^n \neg (I(\underline{y}^0) \wedge \bigwedge_{i=0}^{n-1} T'(\underline{y}^i, \underline{y}^{i+1}) \wedge \\ \underline{x}^{n+1} \equiv \underline{y}^n))\end{array} \tag{14}$$

[8] We extract the propositional representation of the initial states I and of the transition relation T, using the BMC tool of NuSMV [31].

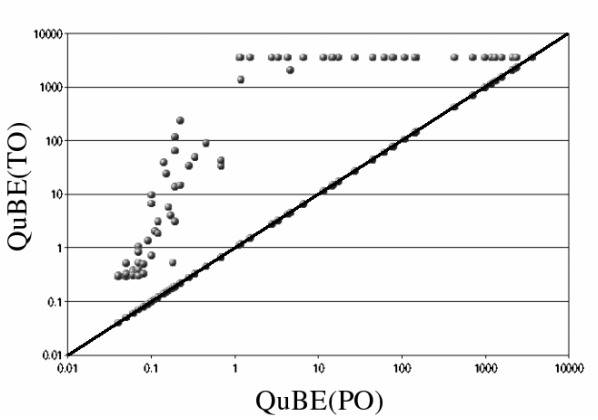

Fig. 5. QuBE(TO) **vs.** QuBE(PO) **on DIA instances**

where

$$T'(\underline{s}, \underline{s}') = (I(\underline{s}) \wedge I(\underline{s}')) \vee T(\underline{s}, \underline{s}'). \quad (15)$$

Assuming that d is the state space diameter of M, the distinctive property of ϕ_n is that ϕ_n is true exactly when $n < d$ and false exactly when $n \geq d$. Intuitively, the formula in (14) evaluates whether there exists a path of length $n+1$ from some initial state to a state $\underline{x}^{n+1}$, such that there is no path of length *at most* n that reaches $\underline{x}^{n+1}$ from some initial state. The rationale of using the transition relation T' obtained from the original T according to (15), is that a self loop on the initial states must be introduced to ensure that we are indeed checking that no paths of length less than n exists to reach $\underline{x}^{n+1}$ from some initial state. The prenex form of (14) obtained by applying the strategy $\exists^{\uparrow}\forall^{\uparrow}$ is

$$\begin{array}{l} \exists \underline{x}^{n+1} \exists \underline{x}^0 \ldots \exists \underline{x}^n \forall \underline{y}^0 \ldots \forall \underline{y}^n (\\ \quad I(\underline{x}^0) \wedge \bigwedge_{i=0}^{n} T'(\underline{x}^i, \underline{x}^{i+1}) \wedge \\ \quad \neg (I(\underline{y}^0) \wedge \bigwedge_{i=0}^{n-1} T'(\underline{y}^i, \underline{y}^{i+1}) \wedge \underline{x}^{n+1} \equiv \underline{y}^n)) \end{array} \quad (16)$$

which is a formulation similar to the ones considered in [8]. Finally, we would like to emphasize that our selection of models from those available in the NuSMV distribution is based on two requirements:

- the need of comparing QuBE(TO) and QuBE(PO) on QBF instances that are proportioned to their capacity in terms of problem size, and
- the need of understanding how the two solvers scale on instances of growing size and/or complexity.

Although we have tried all the examples bundled with the NuSMV distribution, the ones that we selected are the only ones that fulfill both the requirements.

Finally, we consider the results on the NuSMV suite in Figure 5 (same layout as Figures 3 and 4). The first observation is that we raised the time limit to 3600 seconds (one hour) on these experiments in order to get more data points, particularly in the case of QuBE(TO). The second observation is that on these instances QuBE(PO) is substantially and consistently faster than QuBE(TO). Looking at the detailed results in the sixth row of Table I we see that it is never the case that QuBE(TO) is faster than QuBE(PO), while QuBE(PO) is at least one order of magnitude faster than QuBE(TO) on 50% of the instances in the suite. Still from Table I, we can see that there are 9 instances where both QuBE(TO) and QuBE(PO) exceed the time limit. These instances include 2 QBFs obtained from the `counter8` model and 7 instances from the `dme` models. In particular, out of a total of 10 `dme` instances, QuBE(PO) manages to solve only 3 of them, while QuBE(TO) exceeds the time limit on all such instances.

In Figure 6 we present the results about an analysis involving the `counter<N>` and `semaphore<N>` models used to generate instances in the DIA suite. `counter<N>` models, where N is the number of state bits, have a known diameter size of $d = 2^N$. Therefore, they can be used to check how QuBE(PO) and QuBE(TO) scale while increasing the size of the QBF instances because of an increasing length of the diameter to be tested. On the other hand, `semaphore<N>` models, where N is the number of processes competing for the critical section, have a known diameter size of $d = 3$ for $N \geq 3$. Therefore, they can be used to check how QuBE(PO) and QuBE(TO) scale while increasing the size of the QBF instances because of an increasing size of the model to be tested. Both plots in Figure 6 report the length tested on the x-axis, and the CPU time consumed for the test on the y-axis. Triangles correspond to QuBE(PO) and squares to QuBE(TO). The lines between the bullets have the purpose of joining the data points related to the tests involving a specific `counter<N>` or `semaphore<N>` model. The number of bits (resp. processes) in the model is also indicated on the rightmost data point of each sequence of bullets, which marks also the largest instance solved before exceeding the time limit. As we can see from Figure 6 (left), QuBE(PO) is able to compute the diameter up to `counter<7>`, while QuBE(TO) fails to compute the diameter on `counter<5>` already. Even considering `counter<4>`, we see that QuBE(TO) run times grow much faster than the corresponding QuBE(PO) run times. The case of the `semaphore<N>` model shown in Figure 6 (right) confirms the good scaling properties of QuBE(PO) vs. QuBE(TO).

The bad behavior of QuBE(TO) with respect to QuBE(PO) is mainly due to the drawbacks described in Section V.

Indeed, in both (14) and (15), the variables in $\underline{x}^{n+1}$ have to be assigned before any universal variable is selected for

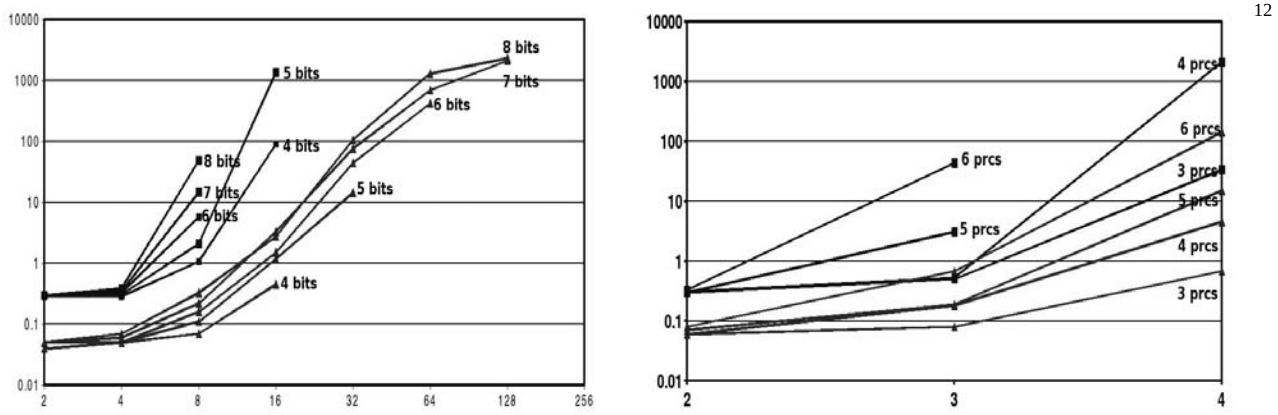

Fig. 6. QUBE(TO) **(squares) vs.** QUBE(PO) **(triangles) on diameter calculation for** `counter` **(left) and** `semaphore` **(right) models.**

branching. However, in (15) also $\underline{x}^0, \ldots, \underline{x}^n$ precede the universal variables in the prefix, and thus the computation of QUBE(TO) proceeds in two steps:

1) first it proves that there is a sequence of $n+1$ states leading to $\underline{x}^{n+1}$, and
2) then it shows that the state $\underline{x}^{n+1}$ cannot be reached in less than $n+1$ steps.

Such a fixed strategy is not necessarily followed by QUBE(PO) which, assuming that the branching heuristic first selects the variables in $\underline{x}^{n+1}$, can either

1) first check that $\underline{x}^{n+1}$ is indeed reachable in $n+1$ steps, or
2) first check that $\underline{x}^{n+1}$ is not reachable in n (or less) steps.

In other words, QUBE(PO) may first reject an assignment to the variables in $\underline{x}^{n+1}$ by showing that there is a sequence of n states leading to $\underline{x}^{n+1}$ starting from an initial state: this does not require finding a path of $n+1$ states leading to $\underline{x}^{n+1}$ and can be easier to show. As an extreme example, consider the case in which the set of initial states is the whole set of states: in this case, the diameter is 0 and for $n = 0$ (14) reduces to

$$\exists \underline{x}^1 (\exists \underline{x}^0 T(\underline{x}^0, \underline{x}^1) \wedge \forall \underline{y}^0 \neg (\underline{x}^1 \equiv \underline{y}^0)). \tag{17}$$

It is clear that for any assignments to the variables in $\underline{x}^1$, (17) can be easily shown to be false by setting each variable in $\underline{y}^0$ equal to the value of the corresponding variable in $\underline{x}^1$.

Further, the good learning mechanism of the solver behaves differently when using (16) instead of (14). Consider for example the simple problem with initial condition

$$I(s_1, s_2) = \neg s_1 \wedge \neg s_2$$

and transition relation

$$T(s_1, s_2, s_1', s_2') = \neg(\neg s_1 \wedge \neg s_2 \wedge s_1' \wedge s_2').$$

The diameter of this circuit is 2, and

1) the matrix of the QBFs (14) and (16) for $n = 1$, converted in CNF using the conversion described in [10], is (x is a variable introduced by the CNF conversion):

$$\begin{gathered}\{\overline{x}_1^0\}, \{\overline{x}_2^0\}, \\ \{x_1^0, x_2^0, \overline{x}_1^1, \overline{x}_2^1\}, \{x_1^1, x_2^1, \overline{x}_1^2, \overline{x}_2^2\}, \\ \{y_1^0, y_2^0, y_1^1, x\}, \{y_1^0, y_2^0, y_2^1, x\}, \\ \{x_1^2, x_2^2, y_1^1, y_2^1, \overline{x}\}, \{x_1^2, \overline{x}_2^2, y_1^1, \overline{y}_2^1, \overline{x}\}, \\ \{\overline{x}_1^2, x_2^2, \overline{y}_1^1, y_2^1, \overline{x}\}, \{\overline{x}_1^2, \overline{x}_2^2, \overline{y}_1^1, \overline{y}_2^1, \overline{x}\};\end{gathered}$$

2) the prefix of (14) is

$$x_1^2, x_2^2 \prec y_1^0, y_2^0, y_1^1, y_2^1 \prec x; \tag{18}$$

3) while the prefix of (16) is

$$x_1^0, x_2^0, x_1^1, x_2^1, x_1^2, x_2^2 \prec y_1^0, y_2^0, y_1^1, y_2^1 \prec x. \tag{19}$$

If we assume that the sequence of literals assigned by QUBE(PO) and QUBE(TO) is $\overline{x}_1^0, \overline{x}_2^0, \overline{x}_1^1, x_2^1, y_1^0, x$ assigned as unit, unit, branch, branch, branch and pure respectively, the empty matrix will be produced. In the case of the prefix (18) and (19) the learned goods are $\{y_1^0\}$ and $\{\overline{x}_1^0, \overline{x}_2^0, \overline{x}_1^1, x_2^1, y_1^0\}$ respectively. The good $\{y_1^0\}$ allows to assign $\overline{y}_1^0$ as a unit, while $\overline{y}_1^0$ can be assigned as a unit because of the good $\{\overline{x}_1^0, \overline{x}_2^0, \overline{x}_1^1, x_2^1, y_1^0\}$ only after all the other literals in the good are assigned to true. Analogous simplifications are performed by QUBE(PO) and not by QUBE(TO) if the sequence of assigned literals is one of the following:

$$\begin{gathered}\overline{x}_1^0, \overline{x}_2^0, x_1^1, \overline{x}_2^1, y_1^0, x; \\ \overline{x}_1^0, \overline{x}_2^0, \overline{x}_1^1, x_2^1, y_2^0, x; \\ \overline{x}_1^0, \overline{x}_2^0, x_1^1, \overline{x}_2^1, y_2^0, x.\end{gathered}$$

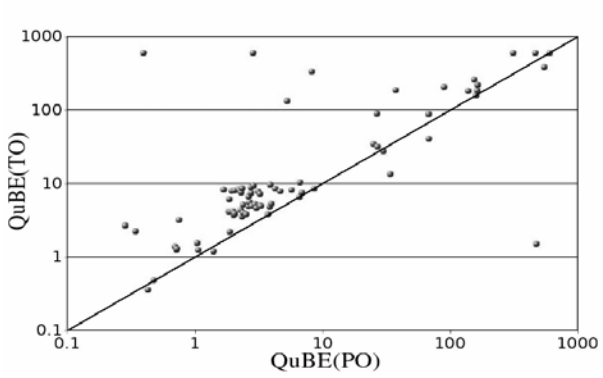

Fig. 7. QuBE(TO) **vs.** QuBE(PO) **on probabilistic (PROB) and fixed (FIXED) instances in the 2006 QBF evaluation**

More in general, for any sequence S of literals leading to the empty matrix,

- the good S_{po} learned by QuBE(PO) includes only the universal literals in S and the existential literals in $\{x_1^2, \overline{x}_1^2, x_2^2, \overline{x}_2^2\}$,
- the good S_{to} learned by QuBE(TO) includes all the literals in S except for those in $\{x, \overline{x}\}$.

Since, $S_{po} \subseteq S_{to}$, S_{po} allows for more pruning than S_{to}.

D. Prenex QBFs in QBF-Eval'06

The idea of solving QBFs in non prenex form is also useful for solving QBFs in prenex form. The basic idea is to minimize the scope of each quantifier in the hope that some pair of variables, once one in the scope of the other, will have different scopes in the final QBF. This idea has been already exploited in solvers like QUBOS [32], QUANTOR [33] and SKIZZO [34]. However, QUBOS', QUANTOR's and SKIZZO's approach is completely different from ours, and the techniques they use to minimize the scope of the variables are different from ours. In particular, taking into account that the matrix is in CNF, the only rules that we apply are

$$Qz(\varphi \wedge \psi) \mapsto (Qz\varphi \wedge \psi)$$

if z does not occur in ψ (this rule is implemented taking into account the associativity and commutativity properties of "$\wedge$"), and

$$Q_1 z_1 Q_2 z_2 \varphi \mapsto Q_2 z_2 Q_1 z_1 \varphi$$

if $Q_1 = Q_2$. These rules are recursively applied starting from the innermost quantifiers (i.e., from the ones bounding the variables with the greatest prefix level) and going outward. In the final QBF, if the scope of a variable z is a single clause C then

1) the clause is removed from the matrix, if z is existential, while
2) z and/or $\overline{z}$ are removed from C, otherwise.

Notice that, we do not apply the rule (used in [32], [33], [34])

$$\forall y(\varphi \wedge \psi) \mapsto (\forall y_1 \varphi[y_1/y] \wedge \forall y_2 \psi[y_2/y]) \qquad (20)$$

(where y_1, y_2 are new variables and $\varphi[t_1/t_2]$ is the expression obtained from φ by substituting t_1 for t_2). The rule (20) causes an increase in the number of variables and, according to some experiments we have performed, a degradation in the performances of the solver.

To test the effectiveness of the approach, we considered the QBF instances used in the 2006 QBF evaluation [28]. In the evaluation, instances are divided into 2 classes called "fixed" and "probabilistic". Intuitively, a class of instances is classified as "probabilistic" when at least one of the parameters that characterizes the class is a random variable. The class is "fixed", otherwise. Notice that according to the given definition, in the probabilistic class there are not only the instances which are randomly generated by generalizing the 3-SAT fixed clause length method from the SAT literature [35], but also structured problems like the conformant planning problems from [36].

The results are shown in Figure 7. In the figure, the first observation is that there are a few points. Indeed, for the vast majority of the 2460 probabilistic and 427 fixed problems in the QBF evaluation, the preliminary step of minimizing the quantifiers' scope did not produce any tangible[9] result in terms of the prefix of the formula and thus such QBFs have not been included in the test set. The last two rows of the table show the results for the probabilistic (first row) and fixed (last row) instances. The results are again positive for QuBE(PO) in most cases: for the probabilistic problems, this is always the case but no new problems get solved, while in the fixed class, 4 instances are solved by QuBE(PO) and not by QuBE(TO).

VIII. Conclusions and related work

The main points of the paper can be summarized as:

- The basic search algorithm of QBF solvers can be extended to take into account the quantifier structure.
- The conversion of QBF instances exhibiting quantifier structure into prenex form can have dramatic impacts on the effectiveness (i) of the branching heuristic, and (ii) of the learning mechanism.

[9]To be precise, we considered the percentage "PO/TO" of existential variables x and universal variables y such that (i) either $x \prec y$ or $y \prec x$ holds for the QBF in prenex form, and (ii) neither $x \prec y$ nor $y \prec x$ holds for the QBF in non prenex form: an instance has been included in the test set if and only if its "PO/TO" is bigger than 20%.

- Our experiments reveal that by taking into account the quantifier structure we can get dramatic improvements in the performance of the QBF solver, and, at least in the case of the diameter calculation problems, better scaling properties.

From a theoretical point of view, our work on non prenex QBFs can be seen as a special case of Henkin's branching quantifiers [37]. With Henkin's quantifiers it is possible to write expressions of the form

$$\begin{array}{l}\forall y_1 \exists x_1 \\ \forall y_2 \exists x_2\end{array} \Phi(x_1, x_2, y_1, y_2) \tag{21}$$

meaning that x_1 depends only on y_1 and x_2 depends only on y_2. Given that $\Phi(x_1, x_2, y_1, y_2)$ can be an arbitrary formula in the variables x_1, x_2, y_1, y_2, in general (21) cannot be expressed as a QBF as defined in Section II, even assuming $\Phi(x_1, x_2, y_1, y_2)$ is in CNF.

From a practical point of view, the works mostly related to ours are [32], [33], [34]. In these works, the authors try to re-construct the original non prenex structure of the formula starting from the instance in total order. The essential difference between [32], [33], [34] and our work is that the solver we use is based on search, while the solvers in [32], [33], [34] 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. Notice that the solver SKIZZO described in [34] is not entirely based on quantifier elimination, since it uses different strategies –including search– for trying to solve each problem. However, search is the last attempted and thus the least used strategy, and it is not clear how SKIZZO uses the quantifier structure during the search phase. Finally, to the best of our knowledge, this is the first paper clearly addressing the quantifier structure problem and giving clear evidence that keeping the original quantifier structure pays off, at least when using search based solvers.

Acknowledgments

We would like to thank Uwe Egly, Stefan Woltran and Michael Zolda for providing us the tools for generating the NCF instances and converting them in prenex form with the different prenex strategies. Marco Pistore, Marco Roveri and Paolo Traverso provided us the FPV benchmarks. Luca Pulina implemented the first version of the tool for converting the non prenex benchmarks into prenex form. The anonymous reviewers of this paper and also of [14] on which this paper is based are also thanked for their useful comments.

References

[1] C. Scholl and B. Becker, "Checking equivalence for partial implementations," in *Proceedings of the 38th Design Automation Conference*, June 2001, pp. 238–243.

[2] A. Abdelwaheb and D. Basin, "Bounded model construction for monadic second-order logics," in *Proceedings of the International Conference on Computer-Aided Verification*, July 2000, pp. 99–113.

[3] M. N. Mneimneh and K. A. Sakallah, "Computing vertex eccentricity in exponentially large graphs: QBF formulation and solution," in *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, May 2003, pp. 411–425.

[4] S. Roy, S. Das, P. Basu, P. Dasgupta, and P. Chakrabarti, "SAT based solutions for consistency problems in formal property specifications for open systems," in *International Conference on Computer-Aided Design*, Nov. 2005.

[5] J. Rintanen, "Constructing conditional plans by a theorem prover," *Journal of Artificial Intelligence Research*, vol. 10, pp. 323–352, 1999.

[6] U. Egly, T. Eiter, H. Tompits, and S. Woltran, "Solving advanced reasoning tasks using Quantified Boolean Formulas," in *Proceedings of the 7th Conference on Artificial Intelligence (AAAI-00) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-00)*, July 30– 3 2000, pp. 417–422.

[7] M. Narizzano, L. Pulina, and A. Tacchella, "The third QBF solvers comparative evaluation," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, pp. 145–164, 2006.

[8] D. Tang, Y. Yu, D. Ranjan, and S. Malik, "Analysis of Search Based Algorithms for Satisfiability of Quantified Boolean Formulas Arising from Circuit Space Diameter Problem," in *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, May 2004, pp. 292–305.

[9] A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, and P. Traverso, "Specifying and analyzing early requirements in Tropos," *Requirements Engineering*, vol. 9, no. 2, pp. 132–150, 2004.

[10] P. Jackson and D. Sheridan, "Clause form conversions for boolean circuits," in *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, May 2004, pp. 183–198.

[11] Z. Fu, Y. Yu, and S. Malik, "Considering circuit observability don't cares in CNF satisfiability." in *Proceedings of the Design, Automation and Test in Europe Conference*, March 2005, pp. 1108–1113.

[12] U. Egly, M. Seidl, H. Tompits, S. Woltran, and M. Zolda, "Comparing different prenexing strategies for quantified Boolean formulas." in *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, May 2003, pp. 214–228.

[13] E. Giunchiglia, M. Narizzano, and A. Tacchella, "QuBE++: An efficient QBF solver," in *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, November 2004, pp. 201–213.

[14] ——, "Quantifiers structure in search based procedures for QBF," in *Proc. Design, Automation and Test in Europe*, March 2006, pp. 812–817.

[15] M. Cadoli, A. Giovanardi, and M. Schaerf, "An algorithm to evaluate Quantified Boolean Formulae," in *Proceedings of the National Conference on Artificial Intelligence and of the Conference on Innovative Applications of Artificial Intelligence*, July 1998, pp. 262–267.

[16] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi, "An algorithm to evaluate quantified Boolean formulae and its experimental evaluation," *Journal of Automated Reasoning*, vol. 28, pp. 101–142, 2002.

[17] E. Giunchiglia, M. Narizzano, and A. Tacchella, "Backjumping for quantified Boolean logic satisfiability," in *Proc. of the International Joint Conference on Artificial Intelligence*, August 2001, pp. 275–281.

[18] L. Zhang and S. Malik, "Towards a symmetric treatment of satisfaction and conflicts in quantified Boolean formula evaluation," in *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, September 2002, pp. 200–215.

[19] R. Letz, "Lemma and model caching in decision procedures for quantified Boolean formulas," in *Proceedings of Tableaux*, July 2002, pp. 160–175.

[20] M. Davis, G. Logemann, and D. W. Loveland, "A machine program for theorem proving," *Communication of ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[21] E. Giunchiglia, M. Narizzano, and A. Tacchella, "Learning for Quantified Boolean Logic Satisfiability," in *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence*, July 2002, pp. 649–654.

[22] L. Zhang and S. Malik, "Conflict driven learning in a quantified Boolean satisfiability solver," in *Proceedings of International Conference on Computer Aided Design*, November 2002.

[23] E. Giunchiglia, M. Narizzano, and A. Tacchella, "Clause/term resolution and learning in the evaluation of quantified Boolean formulas," *Journal of Artificial Intelligence Research (JAIR)*, vol. 26, pp. 371–416, 2006.

[24] ——, "Monotone literals and learning in QBF reasoning," in *Tenth International Conference on Principles and Practice of Constraint Programming*, October 2004, pp. 260–273.

[25] H. Kleine-Büning, M. Karpinski, and A. Flögel, "Resolution for quantified Boolean formulas," *Information and Computation*, vol. 117, no. 1, pp. 12–18, 1995.

[26] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," in *Proceedings of the Design Automation Conference*, June 2001, pp. 530–535.

[27] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2001.

[28] M. Narizzano, L. Pulina, and A. Tacchella, "QBF evaluation 2006," www.qbflib.org/qbfeval [2006-8-2].

[29] E. Giunchiglia, M. Narizzano, M. Pistore, M. Roveri, and P. Traverso, "Using quantified Boolean logics to verify web service composition requirements," in *Proc. International Workshop on AI for Service Composition*, August 2006.

[30] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An opensource tool for symbolic model checking," in *Proceedings International Conference on Computer Aided Verification*, July 2002, pp. 359–364.

[31] A. Cimatti, E. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "Integrating BDD-based and SAT-based symbolic model checking," in *Proceedings of the 4rd International Workshop on Frontiers of Combining Systems*, April 2002, pp. 49–56.

[32] A. Ayari and D. A. Basin, "QUBOS: Deciding quantified boolean logic using propositional satisfiability solvers." in *Proceedings International Conference on Formal Methods in Computer-Aided Design*, November 2002, pp. 187–201.

[33] A. Biere, "Resolve and expand." in *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, May 2004, pp. 59–70.

[34] M. Benedetti, "Quantifier Trees for QBFs," in *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, June 2005, pp. 378–385.

[35] D. G. Mitchell, B. Selman, and H. J. Levesque, "Hard and easy distributions for SAT problems," in *Proceedings of the National Conference on Artificial Intelligence*, July 1992, pp. 459–465.

[36] C. Castellini, E. Giunchiglia, and A. Tacchella, "SAT-based planning in complex domains: Concurrency, constraints and nondeterminism," *Artificial Intelligence*, vol. 147, no. 1-2, pp. 85–117, 2003.

[37] L. A. Henkin, "Some remarks on infinitely long formulas," in *Infinitistic Methods: Proceedings of the Symposium on Foundations of Mathematics*. Pergamon Press and Państwowe Wydawnisctwo Naukowe, 1961, pp. 167–183.