Quantifier Trees for QBFs

Marco Benedetti[§]

Istituto per la Ricerca Scientifica e Tecnologica (IRST)
Via Sommarive 18, 38055 Povo, Trento, Italy
benedetti@itc.it

Abstract. We present a method—called quantifier tree reconstruction—that allows to efficiently recover *ex-post* a portion of the internal structure of QBF instances which was hidden as a consequence of the cast to prenex normal form. Means to profit from a quantifier tree are presented for all the main families of QBF solvers. Surprising experiments on QBFLIB instances are also reported.

1 Introduction

Some standard specification language lays at the intersection between researchers developing theorem provers and those interested in using such tools. For the case of propositional theorem proving, *Conjunctive Normal Form* (CNF) is a *de facto* standard. Entire libraries of CNF instances exist in publically available repositories (such as [17]), and this makes it possible a share of knowledge which is advantageous to both sides.

For the case of Quantified Boolean Formulas (QBFs) the standard form for instances is *prenex* conjunctive normal form. Each instance is comprised of a linear *prefix* specifying an arbitrary number of alternations between universally and existentially quantified variables, followed by a quantifier-free *matrix* which is a CNF formula.

Standardization enables a complete decoupling between the production of instances and their solution, with countless benefits. There are some disadvantages thought. Most of them stem from the loose of structural information that happens when instances are flattened onto a CNF representation. Experiments show that the additional structure present in the original formula could be exploited to achieve gains in solving performances [26], and this is coherent with considerations on the increase in proof complexity that a cast to CNF might produce. A radical way to overcome these issues would be to give up the clause-based representation. This would require (1) to be able to control the encoding process so to produce non-standard encodings, and (2) to construct a solver that is able to understand the enriched specification language and to take advantage of it. Attempts to follow this direction exist for both SAT and QBF solvers (see below).

In this paper, we propose a less demanding and more broadly applicable approach. We preserve the standard encoding and attempt to recover part of the lost information *ex-post*. We focus on QBF instances, which are converted to prenex CNF in two steps:

a. A prenex form is generated by moving all the quantifiers outside an (arbitrarily shaped) quantifier-free matrix.
b. The matrix is converted into conjunctive normal form.

[§] This work is funded by PAT (*Provincia Autonoma di Trento*, Italy), under grant n. 3248/2003.

The latter step is analogous to the one performed to obtain CNF SAT encodings. The former is concerned with the specificity of quantified formulas, i.e. the existence of alternated universal and existential quantifications. A direct strategy to recover the lost structure would be to invert both steps in (reverse) order: ($\mathbf{b}^{-1}$) reconstruct some internal structure out of a flat CNF instance, then ($\mathbf{a}^{-1}$) re-position quantifiers in as deep as possible positions via *anti-prenexing* steps.

We are interested in reverting Step **a**, to some extent, even if the matrix is left in CNF form. In essence, it is a matter of extracting a tree-shaped syntactic structure—which we call *quantifier tree*—out of a flat prenex conjunctive normal form instance.

The prefix of a QBF instance conveys a relevant part of its semantics: It states which existential variables are in the scope of which universal variables. In a linear prefix, an existential variable is in the scope of all the universal variables to its left. So, if an existential variable e_2 follows another existential variable e_1 in the left-to-right prefix order, then necessarily e_2 is in the scope of at least all the universal variables dominating e_1. This condition may impose unnecessarily strong restrictions w.r.t. the intrinsic dependencies between existential and universal variables in a given matrix. In Section 3 we show how to reconstruct tree-shaped dependencies starting form a linear prefix.

Our conservative method overcomes one of the two problems that would arise if we were to renounce to CNF: It is no longer necessary to define a richer intermediate language and to guide the encoding process towards producing such format. As a consequence, the whole set of QBF benchmarks is at our disposal for experimentation.

Yet, the other side of the problem stays inescapable: We are in need for a solving strategy able to profit from quantifier trees. Fortunately, tree-shaped CNFs essentially retain a clause-based structure. Hence, we can keep all the techniques developed to efficiently represent clause sets and to reason about them. Three major families of CNF-based approaches to QBF satisfiability are analyzed in Section 4 to show how each family can profit from a quantifier tree. Programmatically, experimentation can be performed on publically available benchmarks. Section 5 is indeed devoted to the presentation of some surprising experiments over instances in the QBFLIB. The paper is closed with a few conclusive remarks in Section 6.

Related works. In first order logic, a related technique called *miniscoping* is adopted to simplify the conjunctive normal forms produced out of arbitrary FOL formulas for resolution-based theorem provers [20]. The aim of this technique is to reduce the arity of Skolem functions by moving quantifiers as inwards as possible via *anti-prenexing* steps [8]. *Skolemization* is indeed a key step towards obtaining an universally quantified clausal forms out of FOL formulas with existential quantifications. In [9] a related method is presented to generate QBFs in prenex form with a small number of quantifier alternations. A form of *quantifier shifting* is leveraged to put similar quantifiers close, in so as to minimize the number of alternations. The objective is to construct a QBF that belongs to an as simple as possible complexity class of the polynomial hierarchy [25].

Interesting connections do exist with query optimization in DBMS. QBF can be stated as relational algebra by representing existential quantifiers by projection operators, conjunctions by join, and negation by complementation (universal quantifiers may then be rewritten). Some forms of query optimization minimize the dimension of the relations to be handled by manipulating the *query tree* so that projections appear deep in the tree[10] (notice that DBMSs are concerned with the *evaluation* of formulas for a given interpretation—the content of the database—not with its satisfiability).

It is worth noticing that the perspective we adopt here is reversed w.r.t. all the above approaches. In particular, [20] focuses on constructing a CNF matrix with certain properties (small number of clauses) given a structured FOL formula. The authors of [9] start from a non-clausal QBF form, and focus on producing a linear prefix with certain properties (small number of alternations). Query optimization moves from an existing non-linear query tree. Conversely, here we start with a linear prefix and a conjunctive normal form matrix, and aim to reconstruct a part of the lost/hidden structure of the original formula. Many works on recovering (and exploiting) hidden structure from CNFs have been proposed for the non-quantified case. For example, [21] focuses on recovering functional dependencies between variables, while [23] concentrates on using (external) higher level problem description to generate a good branching sequence for DPLL-like solvers. A related form of structure uncovering for QBFs is the exploitation of the *Gaifman graph* of the matrix, used in the QBDD and QMRES algorithms [22], to decide a good variable ordering for the decision diagrams used in those solvers to perform quantifier elimination. Interesting notes on tree-like prefixes are reported in [4].

Attempts to supersede the CNF format by means of solvers able to manage richer input languages exist both in the SAT community [26,16], and in the QBF setting [13,15]. As expected, experimentations suffer from the scarcity of available instances.

2 QBFs and their structure

We consider prenex CNF formulas in the form $F = \mathcal{Q}_1 V_1 \mathcal{Q}_2 V_2 \ldots \mathcal{Q}_n V_n\ \widetilde{F}$, where the matrix $\widetilde{F}$ is a CNF propositional formula on variables $var(F)$, and the prefix $\mathcal{Q}_1 V_1 \mathcal{Q}_2 V_2 \ldots \mathcal{Q}_n V_n$ is such that $\mathcal{Q}_i \in \{\forall, \exists\}, i = 1, \ldots, n$ and $\mathcal{Q}_i \neq \mathcal{Q}_{i+1}, i = 1, \ldots, n-1$, while $\{V_i\}$ is a partition of $var(F)$ (i.e.: $\cup_{i=1}^{n} V_i = var(F)$ and $V_i \cap V_j = \emptyset$ for $i \neq j$). We suppose that each V_i is non-empty. Each V_i is called *scope*. A scope V_i is *existential* (*universal*) if $\mathcal{Q}_i = \exists$ ($\mathcal{Q}_i = \forall$). The scope $\sigma(v)$ of a variable is the index i such that $v \in V_i$. Variables $v \in V_i$ are said to be existentially (universally) quantified if $\mathcal{Q}_i = \exists$ ($\mathcal{Q}_i = \forall$). The set of existentially (universally) quantified variables in F is denoted by $var_\exists(F)$ ($var_\forall(F)$, respectively). We write $\mathcal{Q}(v)$ to mean $\mathcal{Q}_{\sigma(v)}$.

The total ordering $V_1 < V_2 < \cdots < V_n$ among scopes induces a relation of *dominance* $\mathcal{R}^F_\preceq \subseteq var(F) \times var(F)$ which is a partial order relation defined as

$$\langle w, v \rangle \in \mathcal{R}^F_\preceq \quad \text{iff} \quad w = v \text{ or } \sigma(w) < \sigma(v)$$

When no ambiguity arises, we simply write $w \preceq v$ (read "w dominates v"). The order defined by any $\mathcal{R}^F$ is not total, but *sequentially* total: $v \preceq w$ or $w \preceq v$ whenever $\sigma(v) \neq \sigma(w)$. Given a subset S of the variables and the partial ordering induced by the prefix, we define $Sup(S) = \{v \in S | \nexists v' \in S . v \preceq v'\}$. The Sup function is extended to clauses, in so as $Sup(\Gamma) = \{v \in var(\Gamma) | \nexists v' \in var(\Gamma) . v \preceq v'\}$. The universal depth $\delta(v)$ of an existential variable v is the number of dominating universal variables for v: $\delta(v) = |var_\forall(F) \cap \{w | w \preceq v\}|$. Given two binary relations $\preceq_1$ and $\preceq_2$, we say that $\preceq_2$ is a *restriction* of $\preceq_1$ when $a \preceq_1 b$ implies $a \preceq_2 b$. By fixing an arbitrary order for the variables in each scope, we restrict the relation $\preceq$ and obtain a total ordering. Though the results presented here can be lifted to work with a sequentially total order, we limit to consider the simpler total order case.

3 Building a quantifier tree

In this section we characterize a class of QBF instances which lays in between prenex CNF instances and general, unrestricted quantified formulas. In particular, we retain a clause-based negated normal form, but relax the restriction on linearly shaped prefixes. The formulas we consider have a tree-like structure with clauses attached to the leaves.

Our aim is to show that a tree-like formula F^{tree} with certain interesting properties can be extracted efficiently out of any given flat QBF instance F, guaranteeing the key property $F^{tree} \equiv F$. Our starting point is the notion of quantifier tree.

Definition 1 (Quantifier Tree). *A quantifier tree t for a QBF F (whose prefix induces the partial order $\preceq$ among variables) is a labeled tree with the following properties:*

1. *The* root *is labeled by an "and" connective.*
2. *The* internal nodes *are labeled by some variable in F. Existential variables appear once in the tree, while any two internal nodes not laying on the same branch can be labeled by the same universal variable. The labeling of nodes is such that $\preceq$ is a restriction of the relation $\preceq_t$ induced by the tree, where $v \preceq_t w$ iff a descending path from v to w exists in t involving at least one universal and one existential node.*
3. *Each* leaf node *n is labeled with a non-empty set of clauses $G(n) \subseteq F$, and is such that the set of variables encountered along the path from the root to n always contains $var(G(n))$. Every clause in F appears exactly once in the whole tree.*

Let us denote by $trees(F)$ the set of quantifier trees for F. The set $trees(F)$ is never empty: It is easy to verify that at least the tree made up of one single branch linearly replicating the sequence of variables in the prefix of F belongs to $trees(F)$ for every F. In general, further (non-trivial) quantifier trees exist. We focus on such "structured" elements of $trees(F)$. As an example, let us consider the following prenex QBF.

$$\forall a \forall b \exists c \forall d \forall e \exists f \exists g \exists h.(a \vee \neg c) \wedge (a \vee h) \wedge (c \vee \neg d \vee g) \wedge (\neg a \vee b \vee f) \wedge (c \vee e \vee \neg h) \wedge (\neg b \vee \neg f) \wedge (a \vee c \vee \neg g) \tag{1}$$

A quantifier tree (with more than one branch) for (1) is depicted in Figure 1. It is straightforward to interpret a quantifier tree t as the syntactic tree of a non-prenex quantified boolean formula $qbf(t)$ inductively defined as follows:

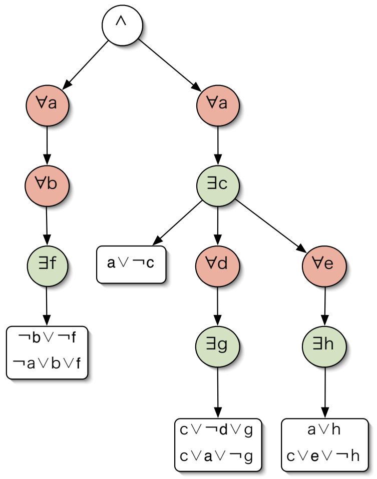

Fig. 1. A quantifier tree for (1).

$$qbf(n) = \begin{cases} qbf(c_1) \wedge \ldots \wedge qbf(c_n) & \text{for the root} \\ Q_{\sigma(v)}v.\ (qbf(c_1) \wedge \ldots \wedge qbf(c_n)) & \text{for a node labeled by } v \\ \Gamma_1 \wedge \ldots \wedge \Gamma_n & \text{for a leaf labeled by } \{\Gamma_1, \ldots, \Gamma_n\} \end{cases}$$

where $c_1, \ldots, c_n$ are the children of n.

As an example, by applying the qbf function to the tree t in Figure 1 we obtain:

$$\begin{aligned} qbf(t) = \forall a \forall b \exists f.\ & ((\neg a \lor b \lor f) \land (\neg b \lor \neg f)) \land \\ \forall a \exists c.\ & (a \lor \neg c) \land (\forall d \exists g.\ (c \lor \neg d \lor g) \land (a \lor c \lor \neg g)) \land \\ & (\forall e \exists h.\ (a \lor h) \land (c \lor e \lor \neg h)) \end{aligned} \tag{2}$$

A quantifier tree is interesting in that the formula it represents is logically equivalent to the original QBF it has been extracted from. For example, (2) is equivalent to (1).

Algorithm 1: An algorithm to construct a quantifier tree for a QBF formula

```
input  : A prenex QBF formula $f$
output: A quantifier tree for $f$

// First, we create the root;
1  $r \leftarrow$ the root node for the tree;
2  $label(r) \leftarrow$ "$\land$";

// Then, we create the leaves together with their lists of attached clauses;
3  $openNodes \leftarrow \emptyset$;
4  foreach $v \in var_\exists(f)$ do
5      $n \leftarrow$ new node;
6      $label(n) \leftarrow v$;
7      $clauses(n) \leftarrow \{\Gamma \in \widetilde{F} | v \in Sup(\Gamma)\}$;
8      $free(n) \leftarrow \emptyset$;
9      foreach $\Gamma \in clauses(n)$ do
10         foreach $\gamma \in \Gamma$ do
11             $free(n) \leftarrow free(n) \cup var(\gamma)$;
12     $free(n) \leftarrow free(n) \setminus \{v\}$;
13     $\widetilde{F} \leftarrow \widetilde{F} \setminus clauses(n)$;
14     $openNodes \leftarrow openNodes \cup \{n\}$;

// Finally, the rest of the tree in a bottom-up way;
15 while $openNodes \neq \emptyset$ do
16     $n \leftarrow$ pick one variable from $Sup(openNodes)$;
17     if $free(n) = \emptyset$ then
18         $father \leftarrow r$;
       else
19         $v \leftarrow$ pick one from $Sup(free(n))$;
20         if $isUniversal(v)$ then
21             $father \leftarrow$ new node;
22             $label(father) \leftarrow v$;
23             $openNodes \leftarrow openNodes \cup \{father\}$;
24             $free(father) \leftarrow free(n) \setminus \{label(n)\}$;
           else
25             $father \leftarrow$ the node $n$ with $label(n) = v$ ;
26             $free(father) \leftarrow free(father) \cup free(n) \setminus \{label(n)\}$;
27     $father(n) \leftarrow father$;
28     $openNodes \leftarrow openNodes \setminus \{n\}$;
```

Lemma 1. *For each quantifier tree $t \in trees(F)$, it is $F \equiv qbf(t)$.*

We are interested in efficiently extracting quantifiers trees with certain properties. To this end, we employ Algorithm 1. It works in a bottom-up way, growing the tree from the leaves to the root. For the sake of simplicity, we assume that to describe a tree it is sufficient to know the father node $father(n)$ of each node n. Each node n is labeled by a variable $label(n)$ as soon as it is created, and has attached a set of variables $free(n)$ updated at each step to represent the variables that occur free in the current $qbf(n)$. A leaf node n is also labeled by a set of clauses $clauses(n)$.

As an example, let us show how the tree in Figure 1 is constructed out of the prenex QBF (1). The cycle starting at line 4 creates one existential leaf node l for each variable in $var_\exists(F)$ and properly distributes the clauses in F among the leaves. The exact mapping between leaves and clauses depends on the sequence of variables selected at line 4, but, whichever the order, each clause is attached to one of its deepest variable according to $\preceq$. In the same cycle the set $free(l)$ of the free variables in $qbf(l)$ is attached to each leaf l. The resulting situation for our example is depicted in the first (left-most) column of Figure 2. The set of nodes $openNodes$ collects all (and only) the nodes whose father

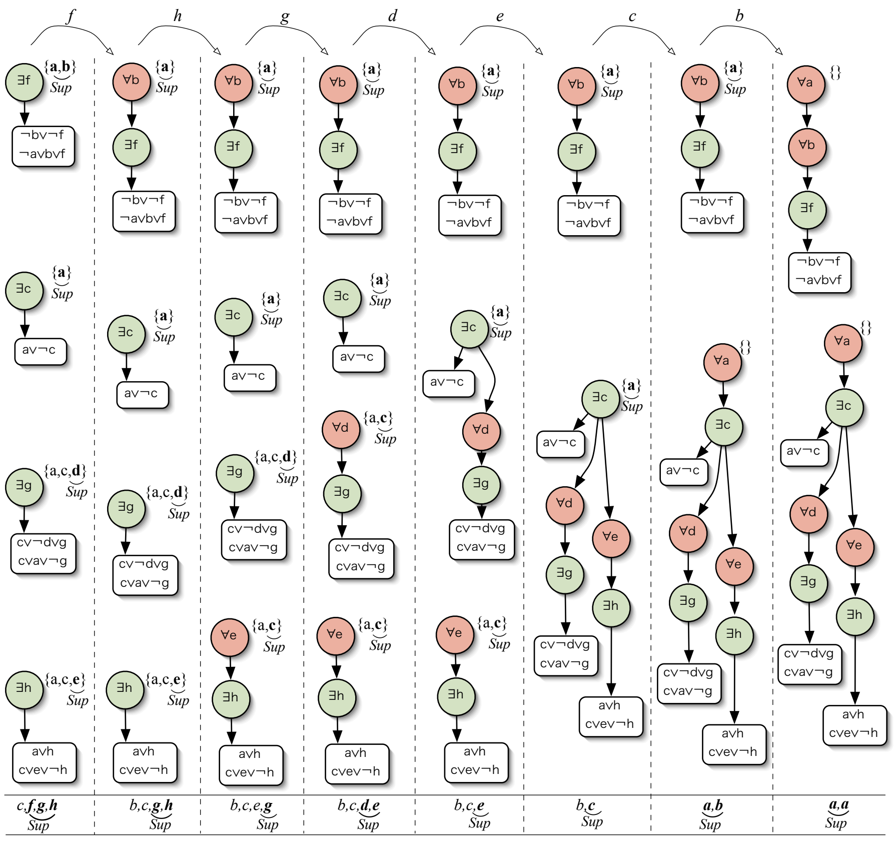

Fig. 2. Step-by-step extraction of a quantifier tree for (1). The last row reports $openNodes$.

Logical equivalences		Condition
1_a. $\exists x.(\neg\phi) \equiv \neg(\forall x.\phi)$	1_b. $\forall x.(\neg\phi) \equiv \neg(\exists x.\phi)$	
2_a. $\forall x.\phi \equiv \phi$	2_b. $\exists x.\phi \equiv \phi$	$x \notin free(\phi)$
3_a. $\forall x.(\phi \wedge \psi) \equiv (\forall x.\phi) \wedge \psi$	3_b. $\exists x.(\phi \wedge \psi) \equiv (\exists x.\phi) \wedge \psi$	$x \notin free(\psi)$
4_a. $\forall x.(\phi \vee \psi) \equiv (\forall x.\phi) \vee \psi$	4_b. $\exists x.(\phi \vee \psi) \equiv (\exists x.\phi) \vee \psi$	
5_a. $\forall x.(\phi \wedge \psi) \equiv (\forall x.\phi) \wedge (\forall x.\psi)$	5_b. $\exists x.(\phi \vee \psi) \equiv (\exists x.\phi) \vee (\exists x.\psi)$	

Table 1. Equivalence-preserving FOL rules that move quantifiers.

has not be decided yet, so each existential leaf belongs to $openNodes$ at the beginning. The subset $Sup(openNodes)$ of $openNodes$ contains those nodes that are roots of *completed* subtrees. Each step of the algorithm (each iteration of the cycle starting at line 15) consists in deciding (line 19) and possibly creating the father node of one finished subtree rooted at the node selected at line 16, also computing the set of free variables associated to such node. Figure 2 depicts a sequence of seven steps that leads to two subtrees with no free variables. They are attached (line 18) to the "and" root (not depicted) so to obtain the final tree in Figure 1.

Lemma 2. *Given a prenex QBF F, Algorithm 1 generates a quantifier tree in $trees(F)$.*

To our end, some quantifiers trees are more interesting than others. We characterize trees on the basis of Table 1, where a number of FOL equivalence-preserving rules concerned with quantifiers are reported. Such rules have been used, for example, in [9] to minimize the number of alternations in linear QBF prefixes, and in [20] to obtain FOL formulas amenable to skolemization. When applied in a left-to-right direction, each rule indeed pushes a selected quantifier one step inward in the syntactic tree of the formula (retaining logical equivalence). Let us call *miniscoped* a quantifier tree t such that none of the rules in Table 1 can be applied to $qbf(t)$ in a left-to-right direction.

Lemma 3. *Algorithm 1 produces miniscoped quantifier trees. It runs in $O(max(|F| \cdot |var(F)|, |var_\exists(F)| \cdot |var(F)|^2))$ requiring $O(|var(F)| \cdot |var_\exists(F)|)$ memory.*

4 How to profit from a quantifier tree

The primary objective of quantifier tree extraction is to help solvers in deciding QBF instances, according to a semantics which we give informally for the sample formula (1). That formula is decided by checking whether for any truth value on a and b a truth value for c exists such that for all possible truth values for d and e, some truth values can be assigned to f, g and h in so as the matrix always evaluates to *true*.

QBF solvers are algorithms designed to answer such questions. Most solvers fall into one of three classes, depending on the strategy they follow to attack the problem. We briefly review these three approaches and show how each one benefits from a quantifier tree with almost no modification to its core reasoning mechanisms.

4.1 Search-based solvers

Search-based solvers extend the DPLL-approach [7] to the quantified case [6] according to Algorithm 2 (many enhancements to this basic scheme have been proposed). Examples of solvers in this class are `Qsolve` [11], `Quaffle`[27], `QuBE` [15], `ZQSAT` [13], and `semprop` [19]. These algorithms look for models in the most natural way: They follow the left-to-right order of the variables in the prefix during a top-down, depth-first visit of the semantic evaluation tree of the formula. Each node n in this tree is labeled

Algorithm 2: eval

```
input  : A prenex QBF: $F = Q_1 v_1 . Q_2 v_2 . \cdots Q_n v_n . \widetilde{F}$
output: A TRUE/FALSE answer
begin
    if $\widetilde{F} = \emptyset$ then result $\leftarrow$ TRUE;
    else if $\bot \in \widetilde{F}$ then result $\leftarrow$ FALSE;
    else
        leftEval $\leftarrow$ eval ($Q_2 v_2 . Q_3 v_3 . \cdots Q_n v_n . (\widetilde{F} * v_1)$);
        rightEval $\leftarrow$ eval ($Q_2 v_2 . Q_3 v_3 . \cdots Q_n v_n . (\widetilde{F} * \neg v_1)$);
        if $Q_1 == \forall$ then
            result $\leftarrow$ leftEval and rightEval;
        else
            result $\leftarrow$ leftEval or rightEval;
    return result ;
end
```

Algorithm 3: evalTree

```
input  : A quantifier tree $t$
output: A TRUE/FALSE answer for $qbf(t)$
begin
    if $clauses(t) = \emptyset$ then eval $\leftarrow$ TRUE;
    else if $\bot \in clauses(t)$ then eval $\leftarrow$ FALSE;
    else
        $l \leftarrow$ connective at the root of $t$;
        if ($l$ = " $\wedge$ ") then
            eval $\leftarrow$ TRUE;
            foreach child subtree $t'$ of $t$ do
                eval $\leftarrow$ eval and evalTree ($t'$);
        else
            leftEval $\leftarrow$ TRUE;
            foreach $t' \in split_0(t)$ do
                leftEval $\leftarrow$ leftEval and evalTree ($t'$);
            rightEval $\leftarrow$ TRUE;
            foreach $t' \in split_1(t)$ do
                rightEval $\leftarrow$ rightEval and evalTree ($t'$);
            if $l == \forall$ then
                eval $\leftarrow$ leftEval and rightEval;
            else
                eval $\leftarrow$ leftEval or rightEval;
    return eval ;
end
```

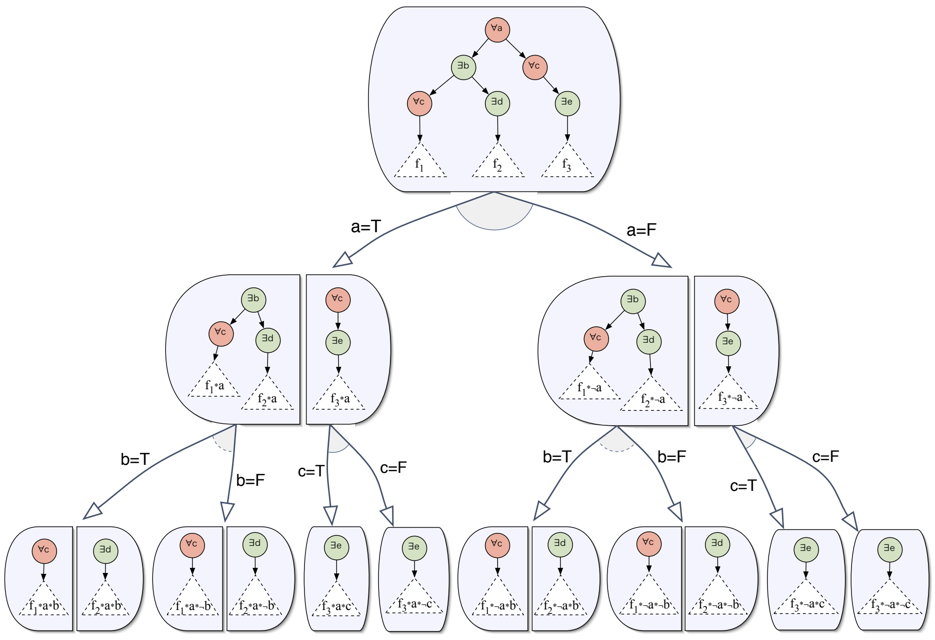

Fig. 3. The top-most part of an AND/OR, divide-et-impera search tree for the quantifier tree $t \in trees(F)$ such that $qbf(t) = \forall a((\exists b(\forall c\, f_1(a, b, c)) \wedge \forall d\, f_2(a, b, d)) \wedge \forall c \forall e\, f_3(a, c, e))$.

by the cofactored matrix $M * \varDelta$ where $\varDelta$ is the assignment on the path to n, while the root is labeled by the original matrix M. Leaves are labeled by either the empty formula $\top$ (the assignment from the root to the current leaf satisfies the original matrix) or the empty clause $\perp$ (the assignment is inconsistent). According to the semantics of quantifiers, an existential variable generates an *or* node that disjunctively splits its branch, while universal quantifiers are associated to *and* nodes that split branches conjunctively. A model, if one exists, is a subtree with all the leaves labeled by $\top$, extracted by choosing one child for each existential node, and both children for conjunctive nodes.

How quantifier trees help search-based solvers. Rather than following the order of variables in the prefix, search-based solvers can work with the partial order induced by the internal structure of the quantifier tree. The advantage is that nodes of the tree with more than one child induce sets of *disjoint* sub-instances that can be solved in isolation of one another. For example, suppose we have reached an internal existential node n with two children, having collected the assignment $\varDelta$ along the path from the root to n. The formula to be decided has the following shape: $\exists v.(\phi_1 \wedge \phi_2)$. Once v has got a truth value—say positive—the two instances $\psi_1 * v$ and $\psi_2 * v$ share only universal variables (if any at all): all the common existential variables have been assigned in $\varDelta$, while the clauses in ϕ_1 and ϕ_2 cannot share existential quantifiers by construction. Thus, $\psi_1 * v$ and $\psi_2 * v$ can be solved independently of one another. The whole procedure follows a *divide-et-impera* scheme, according to Algorithm 3, where $split_p(t)$ is a set

$\{t'_1, \ldots, t'_n\}$ of quantifier trees obtained from the set $\{t_1, \ldots, t_n\}$ of child subtrees of t as follows: if v is the variable labeling the root of t, t'_i is obtained from t_i by replacing each clause C in each leaf node of t_i by $C * v$ if $p = 0$, and by $C * \neg v$ if $p = 1$.

The inductive case now deals with two conceptually different kinds of trees: the (syntactic) quantifier tree and the (semantic) evaluation tree. The former is explicitly manipulated as a parameter, the latter is implicitly explored via the recursive structure of the decision procedure. The two trees are related as each node of the quantifier tree is to be decided by checking both truth values for the labeling variable (or just one, should lazy evaluation suffice), while each truth value in turn generates a set of quantifier subtrees to be decided. The resulting situation is depicted for a sample case in Figure 3.

4.2 Resolution-based solvers

Rather than search for a model, it is possible to *solve* the formula by applying a *refutationally complete procedure* (`quantor` [4], `QMRES` [22], `QBDD` [22]). Such strategy aims to derive necessary consequences from the given formula, ending up with the empty clause if and only if the original formula is unsatisfiable. These methods build upon generalizations of the resolution approach to existential satisfiability, such as *q-resolution* [18,5]. There are several possible complete strategies for applying resolution. For example, we can focus on *eliminating quantifiers* in a right-to-left order (w.r.t. the order in the prefix). We get rid of existential quantifiers by q-resolution, and *expand* universal quantifiers to the two cases they represent. In the sample case of the instance

$$\exists a \forall b \exists c \; (a \vee b \vee c) \wedge (b \vee \neg c) \wedge (a \vee \neg b \vee \neg c) \wedge (\neg a \vee b) \tag{3}$$

we would start by resolving each clause containing c against each clause containing $\neg c$, thus obtaining $\exists a \forall b (a \vee b) \wedge (\neg a \vee b)$ (where c vanished). The universal quantifier b can be eliminated by constructing the conjunction of a copy $(a' \vee b') \wedge (\neg a' \vee b')$ of the matrix where b' has to be assigned to *true* with a different copy $(a'' \vee b'') \wedge (\neg a'' \vee b'')$ where b'' is assigned to *false*. We obtain $\exists a''(a'') \wedge (\neg a'')$, which by resolution finally yields the empty clause: The instance is unsatisfiable.

How quantifier trees help resolution-based solvers. If resolution-based solvers were able to exercise brute force against any instance, there would be no point in quantifier trees and, more in general, in heuristic reasoning: it would suffice to apply until fixpoint some complete rule (such as q-resolution). Unfortunately, time and (more often) space limits prevent most inference sequences form being feasible (intermediate clause sets blow up), in so as finding a viable solution is the key problem the solver has to face.

Quantifier trees help resolution-based solvers in finding better ways to carry on. Let us consider, for example, the solver `quantor` [4]. It follows the *right-to-left* order of variables in the prefix: At each step, it chooses whether to eliminate by q-resolution one of the variables in the deepest existential scope $S_\exists$ or to eliminate by *expansion* one of the variables in the deepest existential scope $S_\forall$. Variables in $S_\forall \cup S_\exists$ are ranked according to a heuristic cost function and the cheapest one is greedily eliminated.

With a linear prefix there is only one deepest existential scope and one deepest universal scope. Conversely, within a quantifier tree there are up to one existential/universal scope per branch. This produces two potential benefits: (1) a possibly wider pool of quantifiers to choose from, and (2) one more degree of freedom in the elimination strategy (for example: prefer a quantifier with minimal/maximal depth in the quantifier tree, or focus on one single branch). For example, b is not eligible for expansion according to (1), while it is eliminated working bottom-up in the tree of Figure 1.

4.3 Skolemization-based solvers.

The Skolem theorem [24] shows how to transform any given *First Order Logic* (FOL) statement F into a *skolemized* formula $Sk(F)$ that has two properties: (1) $Sk(F)$ contains no existential quantifier, and (2) $Sk(F)$ is satisfiable iff F is satisfiable (see [12] for details). Existential quantifiers are eliminated by replacing the variables they bind with *Skolem functions* whose definition domains are appositely chosen to preserve satisfiability. Several ways exist to construct such domains, depending on the form of skolemization we employ (inner, outer, strong, optimized, etc.).

We focus on *outer* skolemization [20], in which the function introduced for $e \in var_{\exists}(F)$ depends on all the universal variables that have e in their scope, i.e. (for prenex formulas) all the universal variables to the left of e in the prefix. While FOL is *closed* w.r.t. skolemization (skolemizing a FOL formula yields a FOL formula), QBF is not. However, skolemization can still be leveraged in the QBF framework [3]. For example, by outer skolemizing the QBF instance (1) we obtain

$$\begin{array}{l}(a \vee \neg s^c(a,b)) \ \wedge \ (\neg a \vee b \vee s^f(a,b,d,e)) \ \wedge \ (\neg b \vee \neg s^f(a,b,d,e)) \ \wedge \\ \wedge \ (a \vee s^h(a,b,d,e)) \ \wedge \ (s^c(a,b) \vee e \vee \neg s^h(a,b,d,e)) \ \wedge \\ \wedge \ (s^c(a,b) \vee \neg d \vee s^g(a,b,d,e)) \ \wedge \ (a \vee s^c(a,b) \vee \neg s^g(a,b,d,e))\end{array} \tag{4}$$

where s^x is the function with arity $\delta(x)$ introduced to skolemize x, and all the variables are meant to be universally quantified. Hence, skolemization-based solvers replace the original QBF satisfiability problem with the satisfiability problem on the skolemized instance. Once (4) is obtained from (1), methods from FOL automated theorem proving or ad-hoc strategies such as the ones presented in [1,3] can be employed.

How quantifier trees help skolemization-based solvers. Quantifier trees allows us to reduce the arity of the skolem functions. Certain dependencies are indeed artificially forced by the linear shape of the prefix. Given that $qbf(t) \equiv F$ for $t \in trees(F)$, we work on $Sk(qbf(t))$ rather than on $Sk(F)$. For example, we obtain for the instance (1):

$$\begin{array}{l}(a \vee \neg s^c(a)) \ \wedge \ (\neg a \vee b \vee s^f(a,b)) \ \wedge \ (\neg b \vee \neg s^f(a,b)) \ \wedge \\ \wedge \ (a \vee s^h(a,e)) \ \wedge \ (s^c(a) \vee e \vee \neg s^h(a,e)) \ \wedge \\ \wedge \ (s^c(a) \vee \neg d \vee s^g(a,d)) \ \wedge \ (a \vee s^c(a) \vee \neg s^g(a,d))\end{array} \tag{5}$$

Simpler skolem functions help the solver in a way that depends on the approach to evaluation it takes. For example, in the case of `sKizzo` [2], each skolemized clause C with m universal variables is translated into (the symbolic representation of) $2^{\delta-m}$ propositional clauses, with $\delta = \delta(C)$ for prenex formulas, and $\delta = \delta_t(C) \leq \delta(C)$ for a quantifier tree t. This produce a linear shrinking of the symbolic representation, and up to an exponential advantage over linear prefixes when ground reasoning is attempted.

5 Experimental results

We may think of a prenex QBF as a tree-like instance with one single branch, having a depth equal to the number of variables in the instance. In the worst case, such structure stays untouched after tree reconstruction. But, we hope to obtain non-collapsed structures, with more than one branch, and a maximal depth smaller than the number of variables in the instance. Consequently, we also expect that both the average and the maximal universal depth of existential variables decreases. According to Section 4, these effects would map onto advantages for a variety of solving procedures. Does tree reconstruction actually re-shape real-world instances? And is the time taken to grow such a tree worth its benefits? We refer to the QBFLIB's archive [14] (maintained by the STAR-lab group at the University of Genova), which is a growing set of benchmarks currently comprised of more than 4000 instances. Table 2 gives a first answer on some families of instances. It compares the depth, average universal depth, maximal universal depth, and number of branches computed over the linear prefix, against the same values computed on the reconstructed syntactic tree. The time taken to build the tree is also reported. On these instances, the impact of our reconstruction algorithm is strong, and in some cases even surprisingly strong (see for example the instance-independent depth of the reconstructed instances in the "*tree*" family).

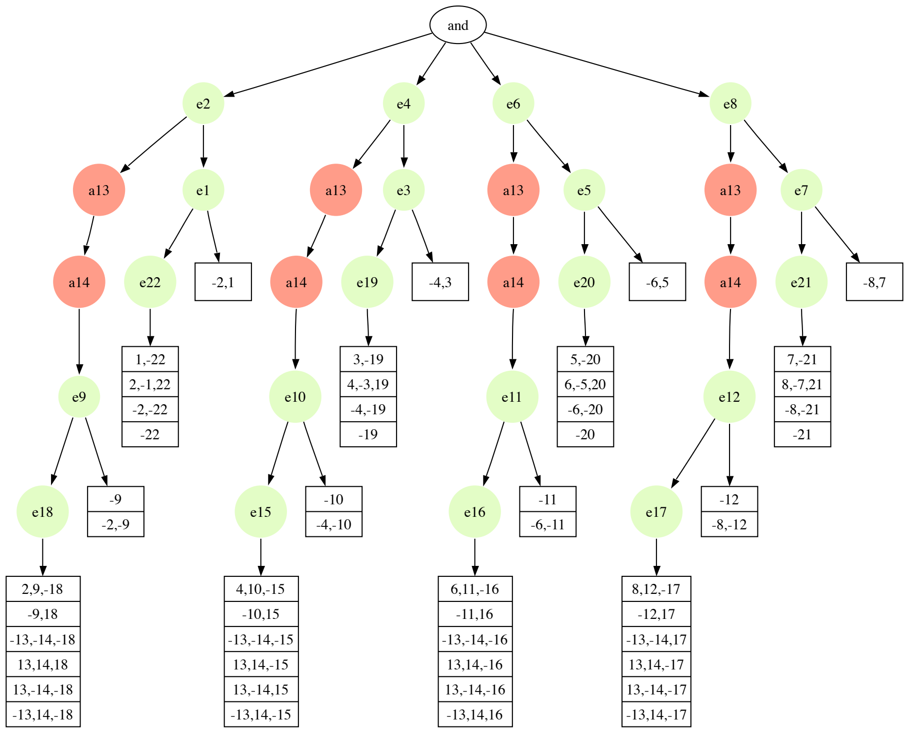

Fig. 4. The reconstructed tree for the instance "*toilet-g-04-01*"

Instance	*Before reconstruction*				Time	*After reconstruction*			
	Depth	*Max & Avg ∀-depth*		*Br.*	(ms)	*Depth*	*Max & Avg ∀-depth*		*Branches*
adder-12-sat	2,665	942	804.8	1	10	227	80	43.1	24
adder-12-unsat	2,687	486	277.9	1	10	2189	354	242.8	1
adder-14-sat	3,641	1,281	1,093.5	1	20	267	94	50.2	28
adder-14-unsat	3,667	665	381.1	1	20	2,988	483	331.8	1
adder-16-sat	4,769	1,672	1,426.4	1	30	307	108	57.4	32
adder-16-unsat	4,799	872	500.6	1	30	3,911	632	434.6	1
Adder2-10-c	7,970	445	417.8	1	40	670	300	287.8	2500
Adder2-10-s	7,970	545	524.8	1	40	98	56	29.5	2500
Adder2-12-c	11,580	642	603.0	1	60	957	432	414.5	3624
Adder2-12-s	11,580	786	756.9	1	50	116	68	35.3	3624
Adder2-14-c	15,862	875	822.0	1	70	1,296	588	564.2	4956
Adder2-14-s	15,862	1,071	1,031.5	1	70	134	80	41.2	4956
flipflop-6-c	6864	30	29.9	1	30	560	18	17.6	2,364
flipflop-7-c	15,213	35	35.0	1	50	1,330	21	20.7	5,121
flipflop-8-c	30,427	40	40.0	1	120	2,824	24	23.8	10,043
flipflop-9-c	56,175	45	45.0	1	220	5,466	27	26.9	18,246
flipflop-10-c	97,272	50	50.0	1	410	9,820	30	29.9	31,186
flipflop-11-c	159,837	55	55.0	1	720	16,610	33	32.9	50,705
k-branch-n-20	13,822	127	97.9	1	150	5568	127	64.3	2646
k-branch-p-19	12,544	121	93.2	1	130	5063	121	61.3	2400
k-d4-n-16	1,438	69	51.7	1	10	755	69	35.3	310
k-d4-p-19	1,176	62	45.9	1	10	638	62	31.6	250
k-dum-n-18	885	44	32.2	1	5	495	44	22.4	198
k-dum-p-20	854	41	30.5	1	5	469	41	21.4	190
k-grz-n-18	792	24	17.4	1	10	393	24	11.2	175
k-grz-p-19	767	24	17.7	1	10	379	24	11.5	169
k-lin-n-19	4,103	18	11.8	1	80	2248	18	8.2	859
k-lin-p-18	932	12	9.9	1	10	430	12	8.4	189
k-path-n-13	937	43	32.1	1	10	450	43	22.9	138
k-path-p-20	1358	61	45.3	1	10	645	61	32.0	199
k-ph-n-21	11,131	12	9.7	1	450	5347	12	6.5	1,643
k-ph-p-20	10,444	12	9.7	1	460	5067	12	6.4	1,524
k-poly-n-18	1,465	110	84.0	1	10	926	110	69.1	323
k-poly-p-17	1,384	104	79.4	1	10	875	104	65.4	305
k-t4p-n-19	2,725	123	90.9	1	20	1446	122	61.4	620
k-t4p-p-19	1,470	69	50.6	1	10	782	68	34.5	333
toilet-a-06-01.11	227	6	3.9	1	<1	84	6	1.8	27
toilet-a-06-01.12	247	6	3.9	1	<1	92	6	1.8	27
toilet-c-10-05.10	805	4	1.2	1	10	498	4	0.5	55
toilet-c-10-05.12	965	4	1.2	1	20	610	4	0.5	55
toilet-g-15-01.2	80	4	3.2	1	<1	7	4	1.3	30
toilet-g-20-01.2	106	5	4.0	1	<1	8	5	1.7	40
TOILET7.1.iv.13	400	3	2.2	1	10	216	3	1.5	32
TOILET7.1.iv.14	431	3	2.2	1	10	234	3	1.5	32
TOILET10.1.iv.20	855	4	3.0	1	10	457	4	2.0	44
TOILET16.1.iv.32	2,133	4	3.0	1	30	1,117	4	2.0	68
tree-exa10-10	21	10	10.0	1	<1	4	2	2.0	9
tree-exa10-15	31	15	15.0	1	<1	4	2	2.0	14
tree-exa10-20	41	20	20.0	1	<1	4	2	2.0	19
tree-exa10-25	51	25	25.0	1	<1	4	2	2.0	24
tree-exa10-30	61	30	30.0	1	<1	4	2	2.0	29

Table 2. The effect of tree-reconstruction over the structure of the syntactic tree

		Solving time (s)	
Instance	*Personality*	*Linear prefix*	*qTree*
tree-exa10-10	B	7.7	0.1
tree-exa10-20	B	timeout	0.2
tree-exa10-30	B	timeout	0.3
TOILET2.1.iv.3	BG	0.3	0.2
TOILET6.1.iv.11	BG	4.0	3.4
TOILET7.1.iv.13	BG	26.4	5.3
k-dup-p-8	QBGS	50.4	0.2
k-dup-p-11	QBGS	54.5	9.7
k-dup-p-15	QBGS	timeout	74.1
k-dup-p-16	QBGS	timeout	75.5
k-dup-p-17	QBGS	timeout	327.0
adder-2-sat	B	timeout	0.3
adder-2-sat	BG	timeout	0.9
adder-4-sat	B	timeout	3.0
adder-4-sat	BG	timeout	26.8
adder-6-sat	RS	13.9	10.7
adder-8-sat	RS	57.1	37.4
adder-10-sat	RS	286.5	198.1

Table 3. Some instances solved on a 900MHz G3 with a 400s timeout.

Conversely, on some classes of instances (such as *mutex* or *chain*) the impact of reconstruction is much weaker. The reader may experiment with the downloadable tool `qTree`[2] which takes a QBF as input and produces statistics on the reconstructed tree.

An intuitive way to get the feeling of what reconstruction does is to take a look at some tree extracted from real-world problems. `qTree` indeed produces on request a textual representation for such trees that can be rendered by suited programs (such as *dot/graphviz*). Results are often surprising. Unfortunately, the trees of all non-trivial instances are too big to be represented while keeping readable fonts for clauses and variables. However, even the smallest instances retain interesting features. For example, Figure 4 depicts the reconstructed tree for one of the smaller "*toilet*" instances. In Figure 5 we give up the requirement on readable fonts and just get the overall picture: the top-most part of a much bigger tree is depicted for a "*flipflop*" instance.

Advantages coming at solving time from quantifier trees cannot be assessed in general, as they heavily depend on the solving strategy and on the specific implementation. We here consider `sKizzo` [2,3,1], a skolemization-based, hybrid QBF solver (incorporating tree-reconstruction) that can be configured to exercise the following strategies[1]: symbolic BDD-based incomplete reasoning (abbreviated in "S"), symbolic resolution-based solving ("R"), branching reasoning with backjumping and learning ("B"), SAT-based solution of ground sub-problems ("G"), and q-resolution reasoning ("Q"). The preliminary results in Table 3 concern different reasoning styles, and show that advantages are expected to cover a broad family of QBF solvers.

[1] Different strategies can be *combined* together to obtain solving "personalities". Their combination is neither sequential nor parallel: They are fully integrated within one another [1].

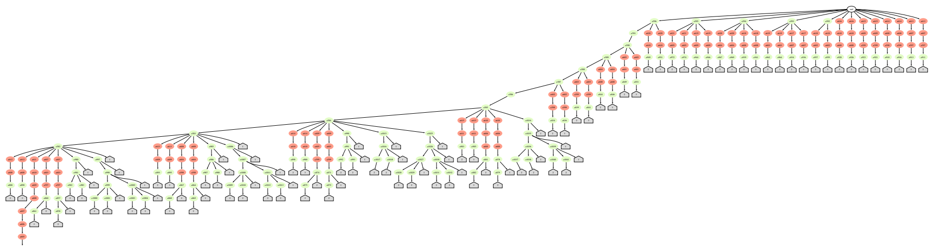

Fig. 5. The compressed, top-most part of the reconstructed tree for the instance "*flipflop-5-c*"

6 Conclusions

We presented a method—called quantifier tree reconstruction—that allows to recover *ex-post* a portion of the internal structure of QBF instances which was hidden as a consequence of the cast to prenex normal form. Means to profit from this reconstruction have been presented, together with experimental results on public-domain instances. As a side effect of our work, we have been able to *visualize* the surprisingly articulated internal shape of certain instances. For example, all the instances in the *adder-sat* family (Ayari's benchamarks in QBFLIB), though regarded as a very challenging ones, materialized in a *disconnectable* form. Future work on this topic relates to what makes some instances much more sensible to tree reconstruction than others, and to the generalization of our algorithm towards trees with depth minimality properties.

Acknowledgments. We thank Lucas Bordeaux for pointing out connections to query optimization, and Luciano Serafini for hints on previous versions of this paper.

References

1. M. Benedetti. sKizzo: a QBF Decision Procedure based on Propositional Skolemization and Symbolic Reasoning, Tech.Rep. 04-11-03, ITC-irst, 2004.
2. M. Benedetti. `sKizzo`'s web site, `sra.itc.it/people/benedetti/sKizzo`, 2004.
3. M. Benedetti. Evaluating QBFs via Symbolic Skolemization. In *Proc. of the 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR04)*, number 3452 in LNCS. Springer, 2005.
4. A. Biere. Resolve and Expand. In *Proc. of SAT'04*, pages 238–246, 2004.
5. H. K. Büning and T. Lettmann. *Propositional Logic: Deduction and Algorithms*. Cambridge University Press, 1999.
6. M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified boolean formulae. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 262–267. American Association for Artificial Intelligence, 1998.
7. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5:394–397, 1962.
8. U. Egly. On the Value of Antiprenexing. In F. Pfennig, editor, *Logic Programming and Automated Reasoning, Proc. of the 5th International Conference, LPAR'94*, pages 69–83. Springer, Berlin, Heidelberg, 1994.
9. U. Egly, H. Tompits, and S. Woltran. On Quantifier Shifting for Quantified Boolean Formulas. In *Proceedings of the SAT-02 Workshop on Theory and Applications of Quantified Boolean Formulas (QBF-02)*, pages 48–61, 2002.
10. R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, 2003.
11. R. Feldmann, B. Monien, and S. Schamberger. A Distributed Algorithm to Evaluate Quantified Boolean Formulas. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 285–290, 2000.
12. M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer Verlag, 1996.
13. M. GhasemZadeh, V. Klotz, and C. Meinel. ZQSAT: A QSAT Solver based on Zero-suppressed Binary Decision Diagrams, available at `www.informatik.uni-trier.de/TI/bdd-research/zqsat/zqsat.html`, 2004.
14. E. Giunchiglia, M. Narizzano, and A. Tacchella. QuBE: A system for deciding Quantified Boolean Formulas Satisfiability. In *Proc. of the International Joint Conference on Automated Reasoning (IJCAR'2001)*, 2001.
15. E. Giunchiglia, M. Narizzano, and A. Tacchella. QuBE++: an Efficient QBF Solver. In *Proc. of the 5th Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, 2004.
16. E. Giunchiglia and R. Sebastiani. Applying the Davis-Putnam Procedure to Non-clausal Formulas. In *Proc. of the 6th Congress of the Italian Association for Artificial Intelligence (AI*IA)*, number 1792 in LNAI. Springer, 1999.
17. H. Hoos and T. Stützle. Satlib - the satisfiability library. http://www.informatik.tu-darmstadt.de/AI/SATLIB, 1998.
18. H. Kleine-Buning, M. Karpinski, and A. Flogel. Resolution for quantified Boolean formulas. *Information and Computation*, 117(1):12–18, 1995.
19. R. Letz. Advances in Decision Procedures for Quantified Boolean Formulas. In *Proceedings of the First International Workshop on Quantified Boolean Formulae (QBF'01)*, pages 55–64, 2001.
20. A. Nonnengart and C. Weidenbach. Computing Small Clause Normal Forms. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, chapter 6, pages 335 – 367. Elsevier, Amsterdam, Netherlands, 2001.
21. R. Ostrowski, É. Grégoire, and S. Lakhdar. Recovering and Exploiting Structural Knowledge from CNF Formulas. In *Principles and Practice of Constraint Programming*, number 2470 in LNCS. Springer, 2003.
22. G. Pan and M.Y. Vardi. Symbolic Decision Procedures for QBF. In *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP04)*, 2004.
23. A. Sabharwal, P. Beame, and H. A. Kautz. Using Problem Structure for Efficient Clause Learning. In *Proc. of SAT03*, number 2919 in LNCS, pages 242–256. Springer, 2003.
24. T. Skolem. Logico-combinatorial investigations in the satisfiability or provability of mathematical propositions: a simplified proof of a theorem by L. Löwenheim and generalizations of the theorem. In *From Frege to Gödel. A Source Book in Mathematical Logic, 1879-1931*, pages 252–263. Harvard University Press, Cambridge, 1967 (1920).
25. L. J. Stockmeyer. The Polynomial-Time Hierarchy. *Theorical Computer Science*, (3):1–22, 1977.
26. C. Thiffault, F. Bacchus, and T. Walsh. Solving Non-clausal Formulas with DPLL search. In *Proc. of SAT04*, 2004.
27. L. Zhang and S. Malik. Towards Symmetric Treatment of Conflicts And Satisfaction in Quantified Boolean Satisfiability Solver. In *Proc. of CP'02*, 2002.