Handbook of Satisfiability
Armin Biere, Marijn Heule, Hans van Maaren and Toby Walsch
IOS Press, 2008

Chapter 10

Reasoning with Quantified Boolean Formulas

Enrico Giunchiglia, Paolo Marin and Massimo Narizzano

10.1. Introduction

The implementation of effective reasoning tools for deciding the satisfiability of Quantified Boolean Formulas (QBFs) is an important research issue in Artificial Intelligence and Computer Science. Indeed, QBF solvers have already been proposed for many reasoning tasks in knowledge representation and reasoning, in automated planning and in formal methods for computer aided design. Even more, since QBF reasoning is the prototypical PSPACE problem, the reduction of many other decision problems in PSPACE are readily available (see, e.g., [Pap94]). For these reasons, in the last few years several decision procedures for QBFs have been proposed and implemented, mostly based either on search or on variable elimination, or on a combination of the two.

In this chapter, after a brief recap of the basic terminology and notation about QBFs (Sec. 10.2, but see Part 2, Chapter 9 for a more extensive presentation), we briefly review various applications of QBF reasoning that have been recently proposed (Section 10.3), and then we focus on the description of the main approaches which are at the basis of currently available solvers for prenex QBFs in conjunctive normal form (CNF) (Section 10.4). Other approaches and extensions to non prenex, non CNF QBFs are briefly reviewed at the end of the chapter.

10.2. Quantified Boolean Logic

Consider a set P of symbols. A *variable* is an element of P. The set of *Quantified Boolean Formulas (QBFs)* is defined to be the smallest set such that

1. if z is a variable, then z is a QBF;
2. if $\varphi_1, \ldots, \varphi_n$ are QBFs then also $(\varphi_1 \wedge \ldots \wedge \varphi_n)$ and $(\varphi_1 \vee \ldots \vee \varphi_n)$ are QBFs $(n \geq 0)$;
3. if φ is a QBF then also $\neg\varphi$ is a QBF;
4. if φ is a QBF and z is a variable, then also $\forall z\varphi$ and $\exists z\varphi$ are QBFs.

Other popular propositional connectives, like implication and equivalence, can be defined on the basis of the given ones. In the following, we use TRUE and FALSE as abbreviations for the empty conjunction and the empty disjunction respectively.

In a QBF $Qz\varphi$ with $Q \in \{\forall, \exists\}$, φ is called the *scope* of Qz and Q is a quantifier *binding* z. An *occurrence of a variable z is free in a QBF φ* if it is not in the scope of a quantifier Q binding z. A *variable z is free in a QBF φ* if it has free occurrences. A QBF is *closed* if it has no free occurrences. For example, the QBF

$$\forall y \exists x_2((\neg x_1 \vee \neg y \vee x_2) \wedge (\neg y \vee \neg x_2) \wedge (x_2 \vee ((x_1 \vee \neg y) \wedge (y \vee x_2)))) \qquad (10.1)$$

is not closed, since x_1 is free in it. From the above definitions, it is also clear that a QBF without quantifiers is a propositional formula.

A *valuation* is a mapping $\mathcal{I}$ from the set P of variables to $\{\text{TRUE}, \text{FALSE}\}$. $\mathcal{I}$ can be extended to an arbitrary QBF φ as follows:

1. If φ is a variable z, $\mathcal{I}(\varphi) = \mathcal{I}(z)$;
2. If φ is $\neg\psi$, $\mathcal{I}(\varphi) = \text{TRUE}$ iff $\mathcal{I}(\psi) = \text{FALSE}$;
3. If φ is $(\varphi_1 \wedge \ldots \wedge \varphi_n)$, $\mathcal{I}(\varphi) = \text{TRUE}$ iff $\forall i : 1 \leq i \leq n$, $\mathcal{I}(\varphi_i) = \text{TRUE}$;
4. If φ is $(\varphi_1 \vee \ldots \vee \varphi_n)$, $\mathcal{I}(\varphi) = \text{TRUE}$ iff $\exists i : 1 \leq i \leq n$, $\mathcal{I}(\varphi_i) = \text{TRUE}$;
5. If φ is $\exists x\psi$, $\mathcal{I}(\varphi) = \text{TRUE}$ iff $\mathcal{I}(\varphi_x) = \text{TRUE}$ or $\mathcal{I}(\varphi_{\neg x}) = \text{TRUE}$;
6. If φ is $\forall y\psi$, $\mathcal{I}(\varphi) = \text{TRUE}$ iff $\mathcal{I}(\psi_y) = \text{TRUE}$ and $\mathcal{I}(\psi_{\neg y}) = \text{TRUE}$.

If φ is a QBF and z is a variable, φ_z (resp. $\varphi_{\neg z}$) is the QBF obtained by substituting all the free occurrences of z in φ with TRUE (resp. FALSE).

A valuation $\mathcal{I}$ *satisfies* a QBF φ if $\mathcal{I}(\varphi) = \text{TRUE}$. A QBF is *satisfiable* if there exists a valuation satisfying it. Two QBFs φ_1, φ_2 are *(logically) equivalent* if every valuation satisfies $((\neg\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg\varphi_2))$, and are *equisatisfiable* if they are both satisfiable or both unsatisfiable.

From the above definitions, it is clear that if φ is closed, any two valuations give the same value to φ. Thus, if φ_1 and φ_2 are both closed QBFs

1. φ_1 and φ_2 are satisfiable if and only if they are given the value TRUE by any valuation, and
2. φ_1 and φ_2 are equivalent if and only if they are equisatisfiable.

10.3. Applications of QBFs and QBF reasoning

Deciding the value of a QBF is the prototypical PSPACE complete problem. As such, many decision problems in PSPACE have been shown to be reducible to the task of deciding the satisfiability of a QBF (see, e.g., [Pap94]). Further, the availability of progressively more efficient QBF solvers has recently fostered the definition of effective encodings of various problems as QBFs, and the use of QBF solvers. In the following, we briefly review some of these recent proposals in the areas of automated planning, knowledge representation and reasoning, formal methods.

In the area of automated planning, QBF solvers are used to solve conformant and conditional planning problems in [Rin99a]. The work of [FG00] defines several encodings of conformant planning problems and presents a procedure mimicking

the computation of a search based QBF solver (see also [CGT03]). Turner [Tur02] shows how many other planning reasoning tasks can be encoded as QBFs, but does not evaluate the effectiveness of the approach. In [GR04, AGS05], various encodings in QBFs of the famous "Connect4" and evader-pursuer problems on a fixed size checkerboard are presented. See also Part 2, Chapter 1 for details (including the presentation of some encodings) about how conditional planning problems can be casted in Quantified Boolean Logic.

In the area of knowledge representation and reasoning, [EETW00] defines the encoding of several reasoning tasks (including autoepistemic, default logic, disjunctive logic programming, and circumscription problems) in QBFs. [PV03] defines a translation of the modal logic K to QBF (see Part 2, Chapter 11 for more on this topic). [EST$^+$03] shows how the problem of the evaluation of nested counterfactuals is reducible to a QBF. [DSTW04] shows that the task of modifying a knowledge base K in order to make the result satisfying and consistent with a set of formulas R and C respectively, can be encoded as a QBF φ such that the valuations satisfying φ correspond to belief change extensions in the original approach. In [BSTW05], the authors describe polynomial-time constructible encodings in QBFs of paraconsistent reasoning principles, and advocates the usage of QBF solvers. [Tom03] gives polynomial reductions mapping a given abduction problem into a QBF φ such that the valuations satisfying φ correspond to solutions of the original problem. [OSTW06] uses QBF solvers to check the equivalence of answer-set programs.

In the area of formal methods, the use of QBF for checking the equivalence of partial implementations has been proposed in [SB01, HBS06]. In [AB00] a similar technique is applied to check whether the structural and behavioral descriptions of protocols agree with each other. In [MS04, GNT07] QBF solvers are used to solve vertex eccentricity problems, while [KHD05, DHK05, JB07, MVB07] show how QBF could improve over SAT in various model checking and formal verification tasks. In [BLS03] QBF solvers are applied to verify pipeline processors. In [CKS05] QBF solvers are proposed for the formal verification of interleaving nondeterministic Boolean programs (see also Part 2, Chapter 2). In the area of Field Programmable Gate Array (FPGA) logic synthesis, [LSB05] uses QBFs for determining if a logic function can be implemented in a given programmable circuit.

10.4. QBF solvers

The development of solvers dedicated to the satisfiability problem of QBFs started with the seminal works of [KBKF95, CGS98] and has, since then, attracted more and more attention. Considering the currently available QBF solvers, the vast majority of them focuses on closed QBFs in prenex conjunctive normal form.

A QBF is in *prenex form* if it has the form

$$Q_1 z_1 Q_2 z_2 \ldots Q_n z_n \Phi \qquad (n \geq 0) \tag{10.2}$$

where

- every Q_i $(1 \leq i \leq n)$ is a quantifier,

- $z_1, \ldots, z_n$ are distinct variables, and
- Φ is a propositional formula.

In (10.2), $Q_1 z_1 \ldots Q_n z_n$ is the *prefix* and Φ is the *matrix*. (10.2) is in *Conjunctive Normal Form (CNF)* if Φ is a conjunction of clauses, where a *clause* is a disjunction of literals. A *literal* is a variable or its negation.

The motivation of this restriction is that any QBF φ can be in linear time transformed into a closed QBF φ' in CNF such that φ and φ' are equisatisfiable. A simple such procedure

1. renames variables in order to make sure that φ does not contain variables bounded by two distinct quantifier occurrences;
2. converts φ in prenex form by moving quantifiers in front of the formula: in the process, care has to be taken in order to ensure that if Qz occurs in the scope of $Q'z'$ in φ, then Qz occurs in the scope of $Q'z'$ also in the scope of the resulting formula (Q and Q' are quantifiers, z and z' are variables). In [EST+03], the authors define several polynomial prenexing strategies such that the resulting QBF is guaranteed to belong to the lowest possible complexity class in the polynomial hierarchy;
3. converts the matrix in CNF by using a procedure based on renaming, such as those described in [Tse70, PG86, JS04]: the additional variables introduced by the CNF transformation can be existentially quantified in the scope of all the other quantifiers in the formula;
4. assuming $\{x_1, \ldots, x_n\}$ ($n \geq 0$) are the free variables in the QBF φ built so far, $\exists x_1 \ldots \exists x_n \varphi$ is the final closed, prenex, CNF QBF.

Each of the above three steps can be performed in linear time, and an efficient procedure performs all the above steps with a single traversal of the input QBF.

For example, using the procedure described in [JS04] for converting the matrix in CNF, it is easy to see that (10.1) is equisatisfiable to the QBF

$$\exists x_1 \forall y \exists x_2 \exists x_3 \{\{\overline{x}_1, \overline{y}, x_2\}, \{\overline{y}, \overline{x}_2\}, \{x_2, x_3\}, \{x_1, \overline{y}, \overline{x}_3\}, \{y, x_2, \overline{x}_3\}\}, \qquad (10.3)$$

in which

- the matrix is represented as a set of clauses (to be interpreted conjunctively),
- each clause is represented as a set of literals (to be interpreted disjunctively),
- $\overline{z}$ stands for $\neg z$.

However, we have to mention that in the process of converting the formula in prenex and CNF form, some relevant information about the structure of both the matrix and the prefix of in input QBF is lost. Using such structural information (of the matrix and of the prefix) can greatly improve performances of the solvers, as shown in [Bie04, Ben05a, GNT07].

For sake of simplicity, in the following, we will restrict our attention to closed QBFs in CNF and prenex form, and keep the above conventions for representing matrices, clauses and literals. We further assume that each clause in the matrix is non tautological, i.e., that a clause does not contain both a literal l and $\overline{l}$: Indeed,

such clauses can be safely removed from the matrix without affecting the value of the QBF.

For a literal l,

- $|l|$ is the variable occurring in l; and
- $\bar{l}$ is the negation of l if l is a variable, and it is $|l|$ otherwise.

We also say that a literal l is *existential* if $\exists|l|$ belongs to the prefix, and it is *universal* otherwise. With these assumptions, if φ is (10.2) and l is a literal with $|l| = z_i$, we redefine φ_l to be the QBF

- whose matrix is obtained from Φ by removing the clauses C with $l \in C$, and by removing $\bar{l}$ from the other clauses; and
- whose prefix is $Q_1 z_1 Q_2 z_2 \dots Q_{i-1} z_{i-1} Q_{i+1} z_{i+1} \dots Q_n z_n$.

Further, we extend the notation to sequences of literals: If $\mu = l_1; l_2; \dots; l_m$ ($m \geq 0$), φ_μ is defined as $(\dots((\varphi_{l_1})_{l_2})\dots)_{l_m}$. For instance, if φ is (10.3), $\varphi_{x_1;x_3}$ is

$$\forall y \exists x_2 \{\{\overline{y}, x_2\}, \{\overline{y}, \overline{x}_2\}, \{y, x_2\}\}.$$

10.4.1. Solvers based on search

A simple recursive procedure for determining the satisfiability of a QBF φ, simplifies φ to φ_z and/or $\varphi_{\overline{z}}$ if z is the leftmost variable in the prefix, till either an empty clause or the empty set of clauses are produced: On the basis of the satisfiability of φ_z and $\varphi_{\overline{z}}$, the satisfiability of φ can be determined according to the semantics of QBFs.

There are some simple improvements to this basic procedure.

Let φ be a QBF (10.2). Consider φ.

The first improvement is that we can directly conclude that φ is unsatisfiable if the matrix of φ contains a contradictory clause. A clause C is *contradictory* if it contains no existential literal. An example of a contradictory clause is the empty clause.

The second improvement is based on the fact that in a QBF we can swap two variables in the prefix if they have the same level. In (10.2), the *level of a variable* z_i is 1 + the number of expressions $Q_j z_j Q_{j+1} z_{j+1}$ in the prefix with $j \geq i$ and $Q_j \neq Q_{j+1}$. For example, in (10.3), x_2 and x_3 have level 1, $\overline{y}$ has level 2, $\overline{x}_1$ has level 3. Thus, assuming that z_i and z_1 have the same level in (10.2), (10.2) is logically equivalent to

$$Q_i z_i Q_2 z_2 \dots Q_{i-1} z_{i-1} Q_1 z_1 Q_{i+1} z_{i+1} \dots Q_n z_n \Phi$$

and we can determine the satisfiability of φ on the basis of φ_{z_i} and/or $\varphi_{\overline{z}_i}$. This allows to introduce some heuristics in the choice of the literal for branching.

Finally, if a literal l is unit or monotone in φ, then φ is logically equivalent to φ_l. In (10.2), a literal l is

- *Unit* if l is existential, and, for some $m \geq 0$,
  - a clause $(l, l_1, \dots, l_m)$ belongs to Φ, and
  - each literal l_i ($1 \leq i \leq m$) is universal and has a lower level than l. The *level of a literal* l is the level of $|l|$.

```
0 function Q-DLL(φ, μ)
1   if (⟨a contradictory clause is in the matrix of φ_μ⟩) return FALSE;
2   if (⟨the matrix of φ_μ is empty⟩) return TRUE;
3   if (⟨l is unit in φ_μ⟩) return Q-DLL(φ, μ; l);
4   if (⟨l is monotone in φ_μ⟩) return Q-DLL(φ, μ; l);
5   l := ⟨a literal at the highest level in φ_μ⟩;
6   if (⟨l is existential⟩) return Q-DLL(φ, μ; l) or Q-DLL(φ, μ; l̄);
7   else return Q-DLL(φ, μ; l) and Q-DLL(φ, μ; l̄).
```

Figure 10.1. The algorithm of *Q-DLL*.

For example, in a QBF of the form

$$\ldots \exists x_1 \forall y_1 \exists x_2 \ldots \{\{x_1, y_1\}, \{x_2\}, \ldots\},$$

both x_1 and x_2 are unit.

- *Monotone* or *pure* if
  - either l is existential, $\bar{l}$ does not belong to any clause in Φ, and l occurs in Φ;
  - or l is universal, l does not belong to any clause in Φ, and $\bar{l}$ occurs in Φ.

  For example, in the QBF

$$\forall y_1 \exists x_1 \forall y_2 \exists x_2 \{\{\neg y_1, y_2, x_2\}, \{x_1, \neg y_2 \neg x_2\}\},$$

  the only monotone literals are y_1 and x_1.

With such improvements, the resulting procedure, called *Q-DLL*, is essentially the one presented in the work of Cadoli, Giovanardi, and Schaerf [CGS98], which extends *DLL* in order to deal with QBFs. Figure 10.1 is a simple, recursive presentation of it. In the figure, given a QBF φ,

1. FALSE is returned if a contradictory clause is in the matrix of φ_μ (line 1); otherwise
2. TRUE is returned if the matrix of φ_μ is empty (line 2); otherwise
3. at line 3, μ is recursively extended to $\mu; l$ if l is unit (and we say that l has been *assigned as unit*); otherwise
4. at line 4, μ is recursively extended to $\mu; l$ if l is monotone (and we say that l has been *assigned as monotone*); otherwise
5. a literal l at the highest level is chosen and
   - If l is existential (line 6), μ is extended to $\mu; l$ first (and we say that l has been *assigned as left split*). If the result is FALSE, $\mu; \bar{l}$ is tried and returned (and in this case we say that $\bar{l}$ has been *assigned as right split*).
   - Otherwise (line 7), l is universal, μ is extended to $\mu; l$ first (and we say that l has been *assigned as left split*). If the result is TRUE, $\mu; \bar{l}$ is tried and returned (and in this case we say that $\bar{l}$ has been *assigned as right split*).

Initially, φ is the input formula, and μ is the empty sequence of literals ϵ.

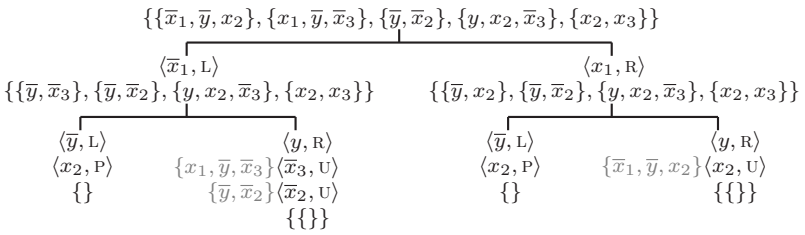

Figure 10.2. The tree generated by *Q-DLL* for (10.3). The matrix of (10.3) is shown at the root node, and the prefix is $\exists x_1 \forall y \exists x_2 \exists x_3$. U, P, L, R stand for "unit", "pure", "left split", "right split" respectively, and have the obvious meaning.

Theorem 1. *Q-DLL*(φ, ϵ) returns TRUE if φ is true, and FALSE otherwise.

Given what we have said so far, it is clear that *Q-DLL* evaluates φ by generating a semantic tree [Rob68] in which each node corresponds to an invocation of *Q-DLL* and thus to an assignment μ. For us,

- an *assignment (for a QBF φ)* is a possibly empty sequence $\mu = l_1; l_2; \ldots; l_m$ ($m \geq 0$) of literals such that for each l_i in μ, l_i is unit, or monotone, or at the highest level in $\varphi_{l_1;l_2;\ldots;l_{i-1}}$;
- the *(semantic) tree* representing a run of *Q-DLL* on φ is the tree
  - having a node μ for each call to *Q-DLL*(φ, μ); and
  - an edge connecting any two nodes μ and $\mu; l$, where l is a literal.

Any tree representing a run of *Q-DLL* has at least the node corresponding to the empty assignment ϵ.

As an example of a run of *Q-DLL*, consider the QBF (10.3). For simplicity, assume that the literal returned at line 5 in Figure 10.1 is the negation of the first variable in the prefix which occurs in the matrix of the QBF under consideration. Then, the tree searched by *Q-DLL* when φ is (10.3) can be represented as in Figure 10.2. In the figure:

- Each node is labeled with the literal assigned by *Q-DLL* in order to extend the assignment built so far. Thus, the assignment corresponding to a node is the sequence of labels in the path from the root to the node. For instance, the assignment corresponding to the node with label $\overline{x}_3$ is $\overline{x}_1; y; \overline{x}_3$.
- When literals are assigned as unit or monotone, the corresponding nodes are aligned one below the other. Further for each assigned literal l, we also show whether l has been assigned as unit, monotone, left or right split by marking it as U, P, L, R respectively.
- When l has been assigned as a left or right split, we also show the matrix of $\varphi_{\mu;l}$, where μ is the sequence of literals assigned before l.
- When the node μ is a leaf, then the matrix of φ_μ is either empty (in which case we write "{}" below the node), or it contains a contradictory clause (in which case we write "{{}}" below the node).

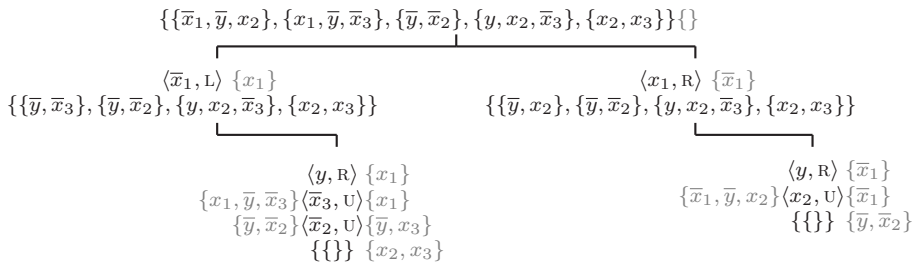

Figure 10.3. The clause resolution corresponding to the tree generated by *Q-DLL* for (10.3). The prefix is $\exists x_1 \forall y \exists x_2 \exists x_3$.

Considering Figure 10.2, it is easy to see that *Q-DLL* would correctly return FALSE, meaning that (10.3) is false.

As in SAT, there is a close correspondence between the search tree explored by *Q-DLL* and a resolution proof showing the (un)satisfiability of the input formula, and this correspondence lays down the foundations for incorporating nogood and good learning in *Q-DLL* [GNT06].

Consider a QBF φ. Assume φ is unsatisfiable and let Π be the search tree explored by *Q-DLL*(φ, ϵ). Then, we can restrict our attention to the *minimal false subtree* of Π, i.e., to the tree obtained from Π by deleting the subtrees starting with a left split on a universal literal: These subtrees are originated from "wrong choices" when deciding which branch to explore first. In the minimal false subtree Π' of Π, all the leaves terminate with the empty clause, and we can associate with each node of Π' a clause pretty much in the same way as in SAT. For instance, if φ is (10.3), Figure 10.3 shows the minimal false subtree of *Q-DLL*'s computation, and the clause associated to each node. In the figure,

- the clause associated with each node is written in gray and to the right of the node itself;
- when a node corresponds to the assignment of a unit literal l, a clause of φ which causes l to be unit at that node (used in the corresponding clause resolution) is written in gray and to the left of the node.

Indeed, the clause associated to the root node ϵ of the figure, is the empty one, meaning that (10.3) is unsatisfiable.

As in SAT, the leaves of the proof tree associated to the search tree are clauses that generated the empty clause, and the clauses associated to the internal nodes are obtained by resolving the clauses associated to children nodes in the search tree (a literal l assigned as unit can be considered as assigned as right split: performing a left split on $\bar{l}$ would immediately generate the empty clause and this node would be a leaf of the search tree).

There are however two differences between the QBF and the SAT cases. The first one, is that Q (clause) resolution [KBKF95] is used instead of plain resolution.

Q (clause) resolution (on a literal l) is the rule

$$\frac{C_1 \qquad C_2}{min(C)} \tag{10.4}$$

where

- l is an existential literal;
- C_1, C_2 are two clauses such that $\{l, \bar{l}\} \subseteq (C_1 \cup C_2)$, and for no literal $l' \neq l$, $\{l', \overline{l'}\} \subseteq (C_1 \cup C_2)$;
- C is $(C_1 \cup C_2) \setminus \{l, \bar{l}\}$;
- $min(C)$ is the clause obtained from C by removing the universal literals whose level is lower than the level of all the existential literals in C.

C_1 and C_2 are the *antecedents*, and $min(C)$ is the *resolvent* of the rule.

The second difference is that the process of associating a clause to each node may require more than a single Q-resolution. Indeed, some clause resolutions can be blocked because of universal variables occurring both as y and $\overline{y}$ in the clauses to be used for the resolution. Consider for instance the QBF:

$$\exists x_1 \exists x_2 \forall y \exists x_3 \{\{x_1, \overline{x}_3\}, \{\overline{x}_2, \overline{y}, x_3\}, \{x_2, y, x_3\}, \{\overline{x}_1, x_3\}\}. \tag{10.5}$$

Then, *Q-DLL* may explore the following branch:

$$\begin{array}{rl} & \langle \overline{x}_1, \text{L} \rangle \\ \{x_1, \overline{x}_3\} & \langle \overline{x}_3, \text{U} \rangle \\ \{\overline{x}_2, \overline{y}, x_3\} & \langle \overline{x}_2, \text{U} \rangle \ \ldots \\ & \{\{\}\} \quad \{x_2, y, x_3\} \end{array} \tag{10.6}$$

where it is not possible to perform the clause resolution associated with the node having label $\langle \overline{x}_2, \text{U} \rangle$. As in the example, a clause resolution (10.4) may be blocked only because of some "blocking" universal literal l

- with both l and $\bar{l}$ not in μ, and
- with $l \in C_1$ and $\bar{l} \in C_2$.

Since both C_1 and C_2 are in minimal form, this is only possible if both C_1 and C_2 contain an existential literal l'

- having level less than or equal to the level of all the other literals in the clause; and
- assigned as unit.

Then, the obvious solution is to get rid of the blocking literals l in C_1 (or in C_2) by resolving away from C_1 (or from C_2) the existential literals with a level lower than the level of l.

In our example, if φ is (10.5) and with reference to the deduction in (10.6), the blocked clause resolution is the one associated with the node $\overline{x}_1; \overline{x}_3; \overline{x}_2$. Indeed, the two clauses $C_1 = \{x_2, y, x_3\}$ and $C_2 = \{\overline{x}_2, \overline{y}, x_3\}$ cannot be resolved on x_2 because of the occurrences of y and $\overline{y}$ in the two clauses. Considering the first clause $\{x_2, y, x_3\}$, since the level of y is less than the level of x_2 and the clause is in minimal form, there must be another existential literal (in our case x_3) having

level less than the level of y and assigned as unit: By resolving $\{x_2, y, x_3\}$ with the clause $\{x_1, \overline{x}_3\}$ (which causes the assignment of $\overline{x}_3$ as unit) we get the resolvent $C_3 = min(\{x_1, x_2, y\}) = \{x_1, x_2\}$ which can be now resolved with C_2 on x_2. Thus, the clause associated with each node is:

$$\begin{array}{rll} & \langle \overline{x}_1, \mathrm{L} \rangle & \{x_1\} \\ \{x_1, \overline{x}_3\} & \langle \overline{x}_3, \mathrm{U} \rangle & \{x_1\} \\ \{\overline{x}_2, \overline{y}, x_3\} & \langle \overline{x}_2, \mathrm{U} \rangle & \{x_1, \overline{y}, x_3\} \\ & \{\{\}\} & \{x_1, x_2\} \text{ (From } \{x_2, y, x_3\} \ \{x_1, \overline{x}_3\}\text{).} \end{array}$$

Notice that the choice of eliminating the blocking literal y in C_1 while keeping C_2 the same, is arbitrary. Indeed, we could eliminate the blocking literal $\overline{y}$ in C_2 and keep C_1. In the case of the deduction in (10.6), this amounts to eliminate the universal literal $\overline{y}$ in $\{\overline{x}_2, \overline{y}, x_3\}$: By resolving this clause with $\{x_1, \overline{x}_3\}$ on x_3, we get the resolvent $\{x_1, \overline{x}_2\}$, which leads to the following legal deduction:

$$\begin{array}{rll} & \langle \overline{x}_1, \mathrm{L} \rangle & \{x_1\} \\ \{x_1, \overline{x}_3\} & \langle \overline{x}_3, \mathrm{U} \rangle & \{x_1\} \\ (\text{From } \{\overline{x}_2, \overline{y}, x_3\}, \{x_1, \overline{x}_3\}) \ \{x_1, \overline{x}_2\} & \langle \overline{x}_2, \mathrm{U} \rangle & \{x_1, y, x_3\} \\ & \{\{\}\} & \{x_2, y, x_3\}. \end{array}$$

Variants of the above procedures are described in [GNT02, Let02, ZM02a]. In [GNT06], it is proved that we can always associate a clause C (resulting from a sequence of clause resolutions) to the node φ_μ of the minimal false subtree explored by *Q-DLL*: The clause C is such that for each existential literal $l \in C$, $\overline{l}$ is in μ. Such clauses can be learned as nogoods, pretty much as in SAT.

If φ is satisfiable the situation is the dual one, except that we have to consider terms or cubes instead of clauses. A *term* or *cube* is a conjunction of literals. Terms are associated to the *minimal true subtree* Π' explored by *Q-DLL*, i.e., to the tree obtained from Π by deleting the subtrees starting with a left split on an existential literal. In more details, to each node φ_μ of Π' we associate a term T such that for each universal literal $l \in T$, $\overline{l}$ is in μ. Such terms can be learned as goods, to be treated as in disjunction with the matrix of φ. Because of the learned goods, also universal literals can be assigned as unit at any point of the search tree. Assuming the current assignment is μ, a universal literal l can be assigned as *unit* if there exists a learned term T such that for each literal $l \in T$, $\overline{l}$ is not in μ; whose only unassigned universal literal is l while all the other unassigned literals have level lower than l. The process of associating terms to each node of Π' starts by associating the conjunction of the literals in μ when the matrix of φ_μ becomes empty. Then, the situation is analogous to the previous case, except that Q term resolution has to be used. *Term resolution (on a literal l)* is the rule

$$\frac{T_1 \qquad T_2}{min(T)}$$

where

- l is an universal literal;
- T_1, T_2 are two terms such that $\{l, \overline{l}\} \subseteq (T_1 \cup T_2)$, and for no literal $l' \neq l$, $\{l', \overline{l'}\} \subseteq (T_1 \cup T_2)$;

- T is $(T_1 \cup T_2) \setminus \{l, \bar{l}\}$;
- $min(T)$ is the term obtained from T by removing the existential literals whose level is lower than the level of all the universal literals in T.

As before, some term resolution may be blocked, but this situation can be solved by getting rid of the existential blocking literals by performing term resolutions on one of the antecedents. See [GNT02, Let02, ZM02b, GNT06] for more details, and [GNT04] for a discussion about the interactions and problems related to the implementation of learning in the presence of monotone literal fixing.

Given the above, it is possible to associate a clause/term resolution deduction to each branch of the search tree explored by *Q-DLL*. The resolvents of such deductions can be learned as in SAT. Further, the Unique Implication Point (UIP) mechanism that has shown to be very effective in SAT (see [MSS96] for a presentation of UIP based learning and [ZMMM01] for a comparison of different UIP based learning mechanisms), can be generalized to the QBF case in a simple way. Assume that we are backtracking on a literal l assigned at decision level n, where the *decision level* of a literal is the number of branching nodes before l. The clause/term corresponding to the reason for the current conflict (resp. solution) is learned if and only if:

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

Under the above conditions, it is possible to backjump to the node at the maximum decision level among the literals in the reason, excluding l, and to assign l (resp. $\bar{l}$) as unit.

Also the ideas underlying the heuristic to be used for selecting the next literal to branch one, can be generalized from the SAT case. However, it is also clear that the effectiveness of any devised heuristic not only depends on the structure of the matrix of the QBFs, but also and in a crucial way, on the structure of the prefix. Indeed, QBFs range from formulas like

$$\exists x_1 \forall x_2 \exists x_3 ... \forall x_{n-1} \exists x_n \Phi \tag{10.7}$$

to formulas like

$$\exists x_1 \exists x_2 \ldots \exists x_m \Phi, \tag{10.8}$$

i.e., to SAT instance. If we consider QBFs of the type (10.7) then it is likely that the heuristic is almost useless: unless an atom $|l|$ is removed from the prefix because l is either unit or monotone, the atom to pick at each node is fixed. On the other hand, considering QBFs of the sort (10.8), we know from the SAT literature that nontrivial heuristics are essential to reduce the search space. In practice, QBF instances lay between the extremes marked by (10.7) and (10.8), and instances like (10.7) are fairly uncommon, particularly on QBFs encoding real-world problems. Because of this, the scoring mechanisms devised SAT in order to decide which literal is best to branch on, are applied also to the QBF case. However, it is clear that the literal with the highest score can be selected

only if it has also the highest level among the unassigned literals. These ideas can be effectively implemented by arranging literals in a priority queue according to (i) the prefix level of the corresponding atom, and (ii) their score. In this way, atoms at prefix level i are always assigned before atoms at prefix level $j > i$ no matter the score, and atoms with the same prefix level are assigned according to their score.

The limitation to branch on literals at the highest level is one of the drawbacks of search based procedures. [Rin99b] proposed techniques based on the inversion of quantifiers, e.g., on assigning universal literals not at the highest level. Indeed, assigning a universal (resp. existential) literal not at the highest level corresponds to weakening (resp. strengthening) the satisfiability of the QBF: If the resulting QBF is unsatisfiable (resp. satisfiable), so it is the original one. These ideas have been proposed and used also in [CSGG02, SB05]. In [CSGG02] a SAT solver is called at each recursive call on the SAT formula obtained by removing all the universal literals from the matrix (this corresponds to consider the QBF in which all the universal quantifiers are pushed down the prefix till level 1): If the SAT formula is satisfiable, this is also the case for the original QBF. Along the same lines, [SB05] calls a SAT solver on the matrix of the QBF (this corresponds to consider the QBF in which all the universal quantifiers are changed to existential ones): If the SAT formula is unsatisfiable, so is also the QBF; in the case an assignment satisfying the matrix is found, such assignment is used to guide the search in the QBF solver.

Beside the above, [GGN$^+$04] presents lazy data structures for unit and monotone literal detection. [GNT01, GNT03] show how *Q-DLL* can be extended with conflict and solution backjumping. [GHRS03] combines Conflict and Solution Directed Backjumping (CSBJ) with a Stochastic Local Search procedure: The result is a procedure which in some cases is unable to determine the (un)satisfiability of the QBF (in which cases it returns "Unknown"), but otherwise the result is guaranteed to be correct. [CMBL05] presents a branching heuristics promoting renamable Horn formulas. [SB06] proposes the use of binary clause reasoning and hyper-resolution.

10.4.2. Solvers based on variable elimination

An alternative approach consists in eliminating variables till the formula contains the empty clause or becomes empty. Elimination of variables can be performed in QBF on the basis that for any QBF (non necessarily in prenex CNF form) φ, $\exists x\varphi$ and $\forall y\varphi$ are logically equivalent to $(\varphi_x \vee \varphi_{\overline{x}})$ and $(\varphi_y \wedge \varphi_{\overline{y}})$ respectively.

The main problem of this approach is that at each step the formula can double its size. There are however several ways to address these issues. The first one, is that if a variable is unit or monotone then it can be simplified as we have seen in the previous section.

Further, in the case of universal variables, each clause C can be replaced with $min(C)$, and we can avoid the duplication of the clauses not in their minimal scope. Given a universal variable y, a clause C is in the *minimal scope of* y if y occurs in C, or if C is y-connected to a clause already determined to be in the minimal scope of y. Two clauses are y-connected if there exists an existential

variable with level lower than y occurring in both clauses. For example, in (10.3) the minimal scope of y consists of all the clauses in the matrix. In other cases however, the minimal scope can be (significantly) smaller, as in

$$\forall y \exists x_1 \exists x_2 \exists x_3 \{\{y, x_1\}, \{\overline{y}, \overline{x}_1\}, \{x_2, x_3\}, \{\overline{x}_2, \overline{x}_3\}\}$$

in which the minimal scope of y are just the first two clauses in the matrix. The expansion of a variable y is obtained by

1. adding a variable z' for each variable z with level lower than the level of y and occurring in minimal scope of y,
2. quantifying each variable z' in the same way and at the same level of the variable z,
3. for each C in the minimal scope of y, adding a new clause C' obtained from C by substituting the newly introduced variables z' to z,
4. considering the newly introduced clauses, those containing $\overline{y}$ are eliminated and y is eliminated from the others.
5. considering the clauses in the original minimal scope of y, those containing y are eliminated and $\overline{y}$ is eliminated from the others.

For example, the expansion of the universal variable y in (10.3) yields the SAT formula:

$$\exists x_1 \exists x_2 \exists x_3 \exists x'_2 \exists x'_3 \{\{x_2, x_3\}, \{x_2, \overline{x}_3\}, \{\overline{x}_1, x'_2\}, \{\overline{x}'_2\}, \{x'_2, x'_3\}, \{x_1, \overline{x}'_3\}\}.$$

which can be determined to be unsatisfiable by a call to a SAT solver. In order to determine the cost of such expansion, for a literal l let S_l be the set of clauses C with $l \in C$, $o(l)$ be $|S_l|$ and $s(l)$ be the sum of the sizes of the clauses in S_l (i.e., $\sum_{C \in S_l} |C|$). Then, if $minscope(y)$ is the size of the clauses in the minimal scope of y, after y expansion, the size of the matrix of the QBF increases or decreases by

$$minscope(y) - (s(y) + s(\overline{y}) + o(y) + o(\overline{y})). \tag{10.9}$$

Notice that when the variable y occurs in all the clauses in its minimal scope, then $minscope(y) = s(y) + s(\overline{y})$ and (10.9) is negative, i.e., the matrix of the QBF shrinks after the expansion.

Alternatively to the expansion of a universal variable y, we can eliminate the existential variables in its minimal scope. In general, the elimination of an existential variable x is performed only when x occurs in clauses in which all the other literals have level greater than or equal to the level of x. If this is the case, we say that x is an *innermost* variable (notice that all the variables with level 1 are innermost). If x is an innermost variable, we can eliminate x by replacing S_x and $S_{\overline{x}}$ with the clauses obtained by resolving each clause in S_x with each clause in $S_{\overline{x}}$ on x, i.e., with the clauses in

$$\{C \cup C' : C \cup \{x\} \in S_x, x \notin C, C' \cup \{\overline{x}\} \in S_{\overline{x}}, \overline{x} \notin C'\}.$$

For example, the elimination of the innermost variable x_3 from (10.3) yields the QBF:

$$\exists x_1 \forall y \exists x_2 \{\{\overline{x}_1, \overline{y}, x_2\}, \{\overline{y}, \overline{x}_2\}, \{x_1, \overline{y}, x_2\}, \{y, x_2\}\}. \tag{10.10}$$

```
0 function Q-DP(φ)
1    if (⟨a contradictory clause is in the matrix of φ⟩) return FALSE;
2    if (⟨the matrix of φ is empty⟩) return TRUE;
3    if (⟨l is unit in φ⟩) return Q-DP(φ_l);
4    if (⟨l is monotone in φ⟩) return Q-DP(φ_l);
5    z := ⟨a variable in φ having level ≤ 2⟩;
6    if (⟨z is existential⟩) return Q-DP(resolve(z, φ));
7    else return Q-DP(expand(z, φ)).
```

Figure 10.4. The algorithm of *Q-DP*.

After the elimination of a variable x, the size of the resulting formula can increase or decrease by

$$o(x) \times (s(\overline{x}) - o(\overline{x})) + o(\overline{x}) \times (s(x) - o(x)) - (s(x) + s(\overline{x})), \tag{10.11}$$

where

1. the first (resp. second) addendum takes into account the fact that each clause containing x (resp. $\overline{x}$) has to be resolved with all the clauses in $S_{\overline{x}}$ (resp. S_x); and
2. the last addendum takes into account the fact that all the clauses in S_x and $S_{\overline{c}}$ can be removed.

From (10.11) it is easy to see that there are cases in which the formula can shrink, i.e., in which (10.11) is negative. This is always the case, e.g., when $o(x) = 1$ and $s(x) = 2$, as it is the case for x_3 in (10.3). Further, in practice the size of the resulting formula may result to be smaller than the one predicted by (10.11). For instance, the size of matrix of (10.10) is 10, while the one predicted by (10.11) is 11. It is however easy to have cases in which (10.11) is positive and the size of the resulting formula is exactly the one predicted by (10.11), e.g., when $o(x) > 1$, $s(x) > 2$ and the variables in S_x are distinct from the variables in $S_{\overline{x}}$.

For example, the elimination of x_3 from (10.3) yields the QBF:

$$\exists x_1 \forall y \exists x_2 \{\{\overline{x}_1, \overline{y}, x_2\}, \{\overline{y}, \overline{x}_2\}, \{x_1, \overline{y}, x_2\}, \{y, x_2\}\},$$

in which the elimination of x_2 leads to

$$\exists x_1 \forall y \{\{\overline{x}_1, \overline{y}\}, \{\overline{y}\}, \{x_1, \overline{y}\}\},$$

which can be immediately concluded to be unsatisfiable because of the contradictory clause $\{\overline{y}\}$.

Alternatively, the expansion of y in (10.3) yields the SAT formula:

$$\exists x_1 \exists x_2 \exists x_3 \exists x'_2 \exists x'_3 \{\{x_2, x_3\}, \{x_2, \overline{x}_3\}, \{\overline{x}_1, x'_2\}, \{\overline{x}'_2\}, \{x'_2, x'_3\}, \{x_1, \overline{x}'_3\}\}$$

which can be determined to be unsatisfiable by a call to a SAT solver, or by repeated applications of variable eliminations till the empty clause is produced.

Figure 10.4 presents a high level description of a recursive procedure incorporating the above ideas. In the figure, given a QBF φ, lines (1)-(4) are analogous to the ones in Figure 10.1, while at line (5)

1. either an existential variable z at level 1 is chosen and then the call $\mathit{resolve}(z, \varphi)$ eliminates z by resolution (line (6)).
2. or a universal variable z at level 2 is chosen and then the call $\mathit{expand}(z, \varphi)$ eliminates z by expansion (line (7)).

If φ is a SAT formula, *Q-DP* behaves as the Davis Putnam procedure [DP60].

Theorem 2. *Q-DP*(φ) returns TRUE if φ is true, and FALSE otherwise.

As for *Q-DLL*, there are many variable elimination procedures which are variations to the above presented procedure. For example, in [Ben04, Bie04] a SAT solver is called as soon as the formula does not contain universal variables. In [Ben04, Bie04], existential variables get first eliminated if they are detected to appear in a binary equivalence: Indeed, if two clauses $\{\bar{l}, x\}$ and $\{l, \overline{x}\}$ are in the matrix of QBF, assuming x has a level not greater than the level of l, x can be safely substituted by l. In [Ben04, Bie04, PV04], simplification mechanisms are defined and used in order to remove subsumed clauses: a clause C is *subsumed* if there exists another clause C' with $C' \subset C$. Considering the way variables are selected for elimination, in [Ben04, Bie04] the choice takes into account the size of the resulting formula as estimated by equations (10.9) and (10.11); in [PV04], universal variables are never expanded since the variables being eliminated are always at level 1. Further, these procedures differ also for the way they represent the matrix. In [Ben04] the QBF is first Skolemized: Each clause corresponds to a disjunction of Boolean functions and is represented via Binary Decision Diagrams (BDDs) [Bry92]. In [PV04] clauses are represented and manipulated as BDDs and ZDDs. In [Bie04] a clause is represented as a set of literals.

The procedure described in [PBZ03] is somehow different from *Q-DP* though it is based on the elimination of variables from the matrix of the QBF. Given a QBF φ, the basic idea of the procedure is to replace a quantified subformula φ' of φ with a logically equivalent formula φ'' without quantifiers: φ''

1. is in CNF since it corresponds to the conjunction of the negation of the valuations falsifying φ'; and
2. can be computed using a SAT solver as back-engine if all the quantifiers in φ' are of the same type, or using a QBF solver as back-engine in the general case.

Thus, in [PBZ03], more than a variable can be eliminated in a single step, and these variables can be of different type. Indeed, it is crucial the selection of the subformula φ' which corresponds to the set V of variables to be eliminated: If S is the union of the minimal scopes of the variables in V, the intuition for the selection of V is to

1. try to minimize the set of variables in S and not in V, and, at the same time,
2. try to maintain V reasonably small in order to be able to enumerate the valuations falsifying φ'.

10.5. Other approaches, extensions and conclusions

Given a QBF, it is indeed possible to alternate the techniques described in the previous sections for solving it. This is what it has been proposed in [Ben05b], where variable elimination and search techniques can be alternated inside one solver. In [PT07] a portfolio of solvers is considered, and the best one is selected using machine learning techniques. [DLMS02] shows how it is possible to encode QBFs in the language of the model checker NuSMV [CCG+02], and then use it as engine.

Extensions to the above procedures in order to deal with non prenex, non CNF QBFs have been proposed. When dealing with solvers based on variable elimination, we already mentioned that it is useful to compute the minimal scope of variables and this indeed corresponds to deal with non prenex QBFs, see [Bie04, Ben05a, PV04, PBZ03]. In [GNT07] it is shown that basic *Q-DLL* search procedures can be extended to take into account the quantifier structure, and that substantial speed ups can be obtained. Finally, non prenex QBFs are naturally handled by solvers based on Skolemization of the input QBF, which naturally allow for handling the more general Henkin's branching quantifiers [Hen61].

Non clausal QBF solvers have been proposed in [Zha06, SAG+06, ESW06]. In more details, the first two papers show that non clausal formulas can be more naturally encoded in both conjunctive and disjunctive normal forms allowing at the same time for substantial speed ups. The last paper presents a procedure which can be directly applied to formulas in negation normal form.

Acknowledgments

This work is partially supported by the Italian Ministero dell'Universitá e della Ricerca.

References

[AB00] Ayari Abdelwaheb and David Basin. Bounded model construction for monadic second-order logics. In *12th International Conference on Computer-Aided Verification (CAV'00)*, number 1855 in Lecture Notes in Computer Science, pages 99–113, Chicago, USA, July 2000. Springer-Verlag.

[AGS05] Carlos Ansótegui, Carla P. Gomes, and Bart Selman. The achilles' heel of qbf. In *Proc. AAAI*, pages 275–281, 2005.

[Ben04] Marco Benedetti. Evaluating qbfs via symbolic skolemization. In *Proc. LPAR*, pages 285–300, 2004.

[Ben05a] Marco Benedetti. Quantifier Trees for QBFs. In *8th International Conference on Theory and Applications of Satisfiability Testing (SAT 2005)*, volume 3569 of *Lecture Notes in Computer Science*. Springer Verlag, 2005.

[Ben05b] Marco Benedetti. skizzo: A suite to evaluate and certify qbfs. In *Proc. CADE*, pages 369–376, 2005.

[Bie04] Armin Biere. Resolve and expand. In *Proc. SAT*, pages 59–70, 2004.

[BLS03] Randal E. Bryant, Shuvendu K. Lahiri, and Sanjit A. Seshia. Convergence testing in term-level bounded model checking. In *Proc. CHARME*, pages 348–362, 2003.

[Bry92] Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[BSTW05] Philippe Besnard, Torsten Schaub, Hans Tompits, and Stefan Woltran. Representing paraconsistent reasoning via quantified propositional logic. In Leopoldo E. Bertossi, Anthony Hunter, and Torsten Schaub, editors, *Inconsistency Tolerance*, pages 84–118, 2005.

[CCG+02] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An open-source tool for symbolic model checking. In *Proc. CAV*, 2002.

[CGS98] M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified Boolean formulae. In *Proc. AAAI*, 1998.

[CGT03] Claudio Castellini, Enrico Giunchiglia, and Armando Tacchella. SAT-based planning in complex domains: Concurrency, constraints and nondeterminism. *Artificial Intelligence*, 147(1-2):85–117, july 2003.

[CKS05] Byron Cook, Daniel Kroening, and Natasha Sharygina. Symbolic model checking for asynchronous Boolean programs. In *Proc. SPIN*, pages 75–90, 2005.

[CMBL05] Sylvie Coste-Marquis, Daniel Le Berre, and Florian Letombe. A branching heuristics for quantified renamable horn formulas. In *SAT*, pages 393–399, 2005.

[CSGG02] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An algorithm to evaluate quantified Boolean formulae and its experimental evaluation. *Journal of Automated Reasoning*, 28:101–142, 2002.

[DHK05] Nachum Dershowitz, Ziyad Hanna, and Jacob Katz. Bounded model checking with qbf. In *SAT*, pages 408–414, 2005.

[DLMS02] Francesco M. Donini, Paolo Liberatore, Fabio Massacci, and Marco Schaerf. Solving QBF by SMV. In *Proc. KR*, pages 578–592, 2002.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[DSTW04] James P. Delgrande, Torsten Schaub, Hans Tompits, and Stefan Woltran. On computing belief change operations using quantified boolean formulas. *Journal of Logic and Computation*, 14(6):801–826, 2004.

[EETW00] Uwe Egly, Thomas Eiter, Hans Tompits, and Stefan 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)*, pages 417–422, Menlo Park, CA, July 30– 3 2000. AAAI Press.

[EST+03] Uwe Egly, Martina Seidl, Hans Tompits, Stefan Woltran, and Michael Zolda. Comparing different prenexing strategies for quantified Boolean formulas. In *SAT*, pages 214–228, 2003.

[ESW06] Uwe Egly, Martina Seidl, and Stefan Woltran. A solver for qbfs in nonprenex form. In *ECAI*, pages 477–481, 2006.

[FG00] Paolo Ferraris and Enrico Giunchiglia. Planning as satisfiability in simple nondeterministic domains. In *AIPS'2000 Workshop on Model Theoretic Approaches to Planning*, pages 55–61, 2000.

[GGN+04] I. Gent, E. Giunchiglia, M. Narizzano, A. Rowley, and A. Tacchella. Watched data structures for QBF solvers. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, (SAT)*, volume 2919 of *LNCS*, pages 25–36. Springer, 2004.

[GHRS03] Ian P. Gent, Holger H. Hoos, Andrew G. D. Rowley, and Kevin Smyth. Using stochastic local search to solve quantified Boolean formulae. In *Proc. CP*, pages 348–362, 2003.

[GNT01] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Backjumping for quantified Boolean logic satisfiability. In *Proc. IJCAI*, pages 275–281, 2001.

[GNT02] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Learning for Quantified Boolean Logic Satisfiability. In *Proc. 18th National Conference on Artificial Intelligence (AAAI) (AAAI'2002)*, pages 649–654, 2002.

[GNT03] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Backjumping for Quantified Boolean Logic Satisfiability. *Artificial Intelligence*, 145:99–120, 2003.

[GNT04] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Monotone literals and learning in QBF reasoning. In *Tenth International Conference on Principles and Practice of Constraint Programming, CP 2004*, pages 260–273, 2004.

[GNT06] 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)*, 26:371–416, 2006.

[GNT07] E. Giunchiglia, M. Narizzano, and A. Tacchella. Quantifiers structure in search based procedures for QBF. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(3):497–507, March 2007.

[GR04] Ian P. Gent and Andrew G. D. Rowley. Encoding connect-4 using quantified boolean formulae. In *Proc. 2nd International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, pages 78–93, February 2004.

[HBS06] Marc Herbstritt, Bernd Becker, and Christoph Scholl. Advanced sat-techniques for bounded model checking of blackbox designs. In *Proc. MTV*, pages 37–44, 2006.

[Hen61] L. A. Henkin. Some remarks on infinitely long formulas. In *Infinitistic Methods: Proceedings of the Symposium on Foundations of Mathematics*, pages 167–183. Pergamon Press and Państwowe Wydawnisctwo Naukowe, 1961.

[JB07] Toni Jussila and Armin Biere. Compressing bmc encodings with qbf. *Electr. Notes Theor. Comput. Sci.*, 174(3):45–56, 2007.

[JS04] Paul Jackson and Daniel Sheridan. Clause form conversions for Boolean circuits. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, pages 183–198, May 2004.

[KBKF95] H. Kleine-Büning, M. Karpinski, and A. Flögel. Resolution for quantified Boolean formulas. *Information and Computation*, 117(1):12–18, 1995.

[KHD05] Jacob Katz, Ziyad Hanna, and Nachum Dershowitz. Space-efficient bounded model checking. In *Proc. DATE*, pages 686–687, 2005.

[Let02] R. Letz. Lemma and model caching in decision procedures for quantified Boolean formulas. In *Proceedings of Tableaux 2002*, LNAI 2381, pages 160–175. Springer, 2002.

[LSB05] Andrew C. Ling, Deshanand P. Singh, and Stephen Dean Brown. FPGA logic synthesis using quantified Boolean satisfiability. In *Proc. SAT*, pages 444–450, 2005.

[MS04] Maher N. Mneimneh and Karem A. Sakallah. Computing vertex eccentricity in exponentially large graphs: QBF formulation and solution. In *Theory and Applications of Satisfiability Testing, 6th International Conference, (SAT)*, volume 2919 of *LNCS*, pages 411–425. Springer, 2004.

[MSS96] J. P. Marques-Silva and K. A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, November 1996.

[MVB07] H. Mangassarian, A. Veneris, and M. Benedetti. Fault diagnosis using quantified Boolean formulas. In *Proc. IEEE Silicon Debug and Diagnosis Workshop (SDD)*, May 2007.

[OSTW06] Johannes Oetsch, Martina Seidl, Hans Tompits, and Stefan Woltran. ccT: A correspondence-checking tool for logic programs under the answer-set semantics. In *Proc. JELIA*, pages 502–505, 2006.

[Pap94] Christos H. Papadimitriou. *Computatational Complexity*. Addison-Wesley, Reading, Mass., 1994.

[PBZ03] David A. Plaisted, Armin Biere, and Yunshan Zhu. A satisfiability procedure for quantified Boolean formulae. *Discrete Applied Mathematics*, 130(2):291–328, 2003.

[PG86] D. A. Plaisted and S. Greenbaum. A Structure-preserving Clause Form Translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[PT07] Luca Pulina and Armando Tacchella. A multi-engine solver for quantified Boolean formulas. In *Proc. CP*, pages 494–497, 2007.

[PV03] Guoqiang Pan and Moshe Y. Vardi. Optimizing a BDD-based modal solver. In *Proc. CADE-19*, pages 75–89, 2003.

[PV04] Guoqiang Pan and Moshe Y. Vardi. Symbolic decision procedures for qbf. In *Proc. CP*, pages 453–467, 2004.

[Rin99a] Jussi Rintanen. Constructing conditional plans by a theorem prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.

[Rin99b] Jussi Rintanen. Improvements to the evaluation of Quantified

Boolean Formulae. In Dean Thomas, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol2)*, pages 1192–1197, S.F., July 31–August 6 1999. Morgan Kaufmann Publishers.

[Rob68] Alan Robinson. The generalized resolution principle. In *Machine Intelligence*, volume 3, pages 77–93. Oliver and Boyd, Edinburgh, 1968. Reprinted in [SW83].

[SAG+06] Ashish Sabharwal, Carlos Ansótegui, Carla P. Gomes, Justin W. Hart, and Bart Selman. QBF modeling: Exploiting player symmetry for simplicity and efficiency. In *Proc. SAT*, pages 382–395, 2006.

[SB01] C. Scholl and B. Becker. Checking equivalence for partial implementations. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 238–243, 2001.

[SB05] Horst Samulowitz and Fahiem Bacchus. Using SAT in QBF. In *CP*, pages 578–592, 2005.

[SB06] Horst Samulowitz and Fahiem Bacchus. Binary clause reasoning in QBF. In *SAT*, pages 353–367, 2006.

[SW83] Jörg Siekmann and Graham Wrightson, editors. *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, volume 1-2. Springer-Verlag, 1983.

[Tom03] Hans Tompits. Expressing default abduction problems as quantified boolean formulas. *AI Commununications*, 16(2):89–105, 2003.

[Tse70] G. Tseitin. On the complexity of proofs in propositional logics. *Seminars in Mathematics*, 8, 1970. Reprinted in [SW83].

[Tur02] Hudson Turner. Polynomial-length planning spans the polynomial hierarchy. In *Proc. JELIA*, pages 111–124, 2002.

[Zha06] Lintao Zhang. Solving QBF by combining conjunctive and disjunctive normal forms. In *Proc. AAAI*, 2006.

[ZM02a] L. Zhang and S. Malik. Conflict driven learning in a quantified Boolean satisfiability solver. In *Proceedings of International Conference on Computer Aided Design (ICCAD'02)*, 2002.

[ZM02b] Lintao Zhang and Sharad Malik. Towards a symmetric treatment of satisfaction and conflicts in quantified Boolean formula evaluation. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming*, pages 200–215, 2002.

[ZMMM01] L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *International Conference on Computer-Aided Design (ICCAD'01)*, pages 279–285, November 2001.