

Antichain-based QBF Solving

T. Brihaye¹, V. Bruyère², L. Doyen³, M. Ducobu¹, and J.-F. Raskin⁴

¹ Institut de Mathématique – Université de Mons, Belgique

² Institut d’Informatique – Université de Mons, Belgique

³ LSV, ENS Cachan & CNRS, France

⁴ Département d’Informatique – Université Libre de Bruxelles, Belgique

Abstract. We consider the problem of QBF solving viewed as a reachability problem in an exponential And-Or graph. Antichain-based approaches to reachability analysis in large graphs leverage the inherent structure of the explored graph in order to reduce the effect of state explosion, with high performance in practice.

In this paper, we propose simple notions of subsumption induced by the structural properties of the And-Or graphs for QBF solving. Subsumption is used to reduce the size of the search tree, and to define compact representations of certificates (in the form of antichains) both for positive and negative instances of QBF. We show that efficient exploration of the reduced search tree essentially relies on solving variants of Max-SAT and Min-SAT. Preliminary stand-alone experiments of this algorithm and some optimizations show that the approach is promising.

1 Introduction

The problem of evaluating the truth value of a quantified Boolean formula (QBF) is one of the most popular PSPACE-complete problem, like SAT (the satisfiability problem for Boolean formulas) is the typical NP-complete problem. QBF is a simple and elegant formalism in which many problems of practical interest can be encoded, in a large number of areas such as automated planning, artificial intelligence, logic reasoning, and verification [11, 16]. The simple form of QBF formulas makes the problem understandable and accessible to a large community. However, despite its apparent simplicity, the design of efficient algorithmic solutions remains challenging. Recent progress has been observed in the practical approaches to this problem. In particular, generalizations of heuristics and optimizations used in SAT solving have been applied to QBF with some success [22, 25].

In verification and automata theory, a typical PSPACE-complete problem is the universality problem for nondeterministic finite automata. Despite its worst-case complexity, dramatic performance improvements have been obtained recently for this problem by *antichain algorithms* [8, 9]. One key idea of antichain algorithms is to exploit the underlying structure of automata constructions (classically, powerset-based constructions) to define *subsumption relations*, yielding compact symbolic representations, as well as sound pruning of the search space.

$$\psi = \underbrace{(x_1 \vee y_4 \vee y_7)}_1 \wedge \underbrace{(x_2 \vee \bar{y}_4 \vee x_6)}_2 \wedge \underbrace{(x_1 \vee x_3 \vee y_5 \vee \bar{y}_7)}_3 \wedge$$

$$\underbrace{(\bar{x}_3 \vee \bar{y}_5 \vee \bar{x}_6 \vee y_7)}_4 \wedge \underbrace{(\bar{x}_1 \vee x_2 \vee y_4)}_5 \wedge \underbrace{(\bar{x}_2 \vee \bar{y}_7)}_6 \wedge \underbrace{(x_1 \vee x_2 \vee x_3 \vee y_7)}_7$$

Fig. 1. The CNF formula ψ for the QBF formula $f = \forall x_1 x_2 x_3 \cdot \exists y_4 y_5 \cdot \forall x_6 \cdot \exists y_7 \cdot \psi$.

In this paper, we identify structural properties of QBF and we investigate pruning strategies for QBF in the spirit of antichain algorithms. We take the classical view of QBF as a reachability problem in an exponential And-Or graph, where the nodes represent subformulas (the And-nodes correspond to universal quantifications, and the Or-nodes to existential quantifications). We show that this And-Or graph enjoys a subsumption relation that can be used to substantially reduce the size of the search tree. This subsumption relation also provides compact certificates in the form of antichains of subformulas both for positive and negative instances of QBF. We show that efficient exploration of the reduced search tree essentially relies on solving variants of the Max-SAT and Min-SAT problems [17, 18]. We have implemented the main ideas presented in this paper in a stand-alone prototype (thus without the established heuristics commonly used in search-based QBF solvers) and the first results are encouraging. While absolute performance is not prominent, the experiments show that the search trees constructed by the subsumption-based algorithms are mostly much smaller (by orders of magnitude) as compared to the entire search space.

We illustrate the main ideas of the algorithm on the following running example. Let $f = \forall x_1 x_2 x_3 \cdot \exists y_4 y_5 \cdot \forall x_6 \cdot \exists y_7 \cdot \psi$ where ψ (shown in Fig. 1) is a CNF formula viewed as a set of clauses. The And-Or graph for f is a DAG where each level corresponds to a block of quantifiers (see a partial expansion in Fig. 2). Nodes of the DAG are subsets $\varphi \subseteq \psi$ of clauses which remain to be satisfied in the evaluation game. The root of the DAG is the set ψ of all clauses. The successors of a node at level i are the sets of clauses obtained by assigning the variables quantified in the i th block of the formula. In the game interpretation, two players choose the successor of the nodes (player P_\forall in And-nodes, and P_\exists in Or-nodes) by assigning the variables quantified in the block of the level of the node. The goal of player P_\exists is to reach a node \emptyset where all clauses are satisfied, and to avoid nodes where all literals of a clause are false (denoted by \perp). The QBF formula is true if and only if P_\exists has a winning strategy to reach \emptyset from the initial node ψ in the game. A key observation is that if player P_\exists has a winning strategy from a node ψ_1 at level i , then player P_\exists also has a winning strategy from all nodes $\psi_2 \subseteq \psi_1$ at level i , because in ψ_2 less clauses remain to be satisfied. This has two implications in the search through the DAG. First, player P_\exists should only consider valuations that make true a *maximal*¹ subset of the remaining clauses, while player P_\forall should make true a *minimal*¹ subset. Computing

¹ With respect to set inclusion.

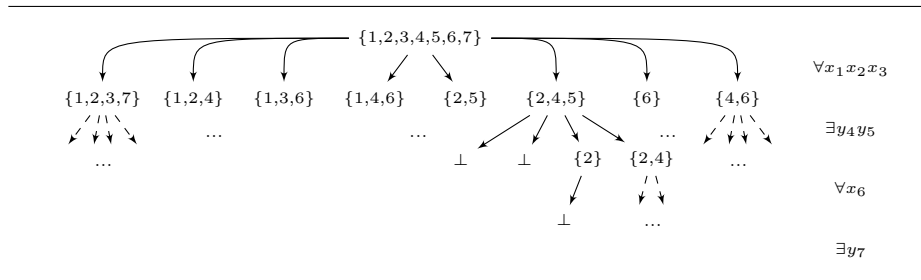


Fig. 2. Search tree for the formula of Fig. 1.

such variable assignments reduces to solving variants of Max-SAT and Min-SAT problems. Second, the set of winning nodes at level i is downward-closed¹, and the set of losing nodes at level i is upward-closed¹. Therefore, antichain of incomparable sets of clauses is the appropriate representation of winning and losing nodes. We exploit this structure when backward propagating the information collected during the exploration of the DAG, and we never explore a node which is smaller than a winning node (or greater than a losing node) at the same level. Finally, the information stored in the antichains at the end of the search allows to construct a winning strategy in the game (for whichever player is winning). Winning strategies transfer to compact certificates for the QBF formula.

Related Works. Many algorithms have been proposed in the literature to solve QBF and competitive events like QBFEVAL aim at assessing the advances in reasoning about QBF [25, 13]. Several leading QBF solvers are search-based. They typically use pruning techniques that extend the DPLL search strategy from SAT to QBF [6]. Common heuristics are unit propagation, conflict learning and backjumping, which are implemented in tools like QuBE [14, 15] and DepQBF [20, 21].

The antichain-based approach is also search-based but it reduces the search space using antichains of winning and losing nodes of the And-Or graph. Antichains can be viewed as compact symbolic representations which can be exponentially succinct, thus our approach also has the flavor of symbolic procedures. In contrast, traditional symbolic QBF solvers rely on binary decision diagrams (BDD) and they are based on quantifier elimination, such as Skolemization-based approaches [1] (with the aim at eliminating existentially quantified variables), or symbolic quantifier elimination by clause resolution or BDD algorithms [23]. The tool sKizzo falls in this category of solvers [4].

Recent works have focused on certifying (rather than just evaluating) QBF formulas. Indeed, certificates can help in extracting error traces in QBF-encoded problems. Tool suites such as ChEQ and sKizzo evaluate and certify QBF formulas [2, 22]. In this context, compact certificates represented by antichains is a new notion.

2 Preliminaries

2.1 Notations and QBF problem

Let $V = \{x_1, x_2, \dots, x_m\}$ be a set of m Boolean variables, we use X, X_1, X_2, \dots to denote subsets of V . A *literal* l is either a variable $x \in V$ or the negation \bar{x} of a variable $x \in V$, and a *clause* c is a disjunction of literals, or equivalently a set of literals. We use notations such as $l \in c$, $\bar{x} \in c$, etc. A CNF formula is a conjunction of clauses, or equivalently a set of clauses. The empty CNF formula is denoted **true**, and the empty clause is denoted **false**. In the figures we use the notations \emptyset and \perp instead of **true** and **false** respectively. Given a set $X \subseteq V$ and a CNF formula ψ over V , we denote by $\pi_X(\psi)$ the projection of ψ over X .

A *quantified Boolean formula* (QBF) is an expression $Q_1 X_1 \cdot Q_2 X_2 \cdots Q_n X_n \cdot \psi$ where each $Q_i \in \{\exists, \forall\}$ for $1 \leq i \leq n$, the sets X_1, \dots, X_n (called *blocks*) form a partition of V , and ψ is a CNF formula over V . We also write $Q_1 x_1 \cdot Q_2 x_2 \cdots Q_n x_n \cdot \psi$ when each block X_i contains one variable ($X_i = \{x_i\}$). Since ψ is in CNF we assume w.l.o.g. that the last block is *existential* (i.e., $Q_n = \exists$). The truth value of a QBF formula is defined as usual. The QBF *evaluation problem* is to decide whether a given QBF formula is true or false. This problem is PSPACE-complete [24].

A *valuation* for $X \subseteq V$ is a function $v : X \rightarrow \{0, 1\}$. The domain of v is $\text{dom}(v) = X$. If $X = \{x_1, \dots, x_k\}$, a valuation $v : X \rightarrow \{0, 1\}$ can be identified with a word $a_1 a_2 \cdots a_k \in \{0, 1\}^{|X|}$ such that $a_l = v(x_l)$ for all $1 \leq l \leq k$. The empty word ϵ corresponds to $\text{dom}(v) = \emptyset$. Given a partition $P = X_1 \cup X_2 \cup \cdots \cup X_n$ of V , let $X_{\leq i}$ be the set of variables $X_1 \cup X_2 \cdots \cup X_i$ (with $X_{\leq 0} = \emptyset$), and let $X_{\geq i} = V \setminus X_{\leq i-1}$. Given the valuations $v : X_{\leq i-1} \rightarrow \{0, 1\}$ and $w : X_i \rightarrow \{0, 1\}$, let vw be the valuation identified with the concatenation of the words representing v and w .

A clause c is *satisfied* by a valuation v (written $v \models c$) if there exists $x \in \text{dom}(v)$ such that either $x \in c$ such that $v(x) = 1$, or $\bar{x} \in c$ such that $v(x) = 0$. Given a CNF formula ψ , we denote by $\text{sat}_v(\psi)$ the set of clauses $c \in \psi$ such that $v \models c$. We denote by $\psi[v]$ the formula obtained by replacing in ψ each variable $x \in \text{dom}(v)$ by its value $v(x)$.

Let ψ be an unsatisfiable CNF formula. An *unsatisfiable core* ψ' of ψ is any subset of clauses of ψ , minimal for the inclusion, such that ψ' is still unsatisfiable.

2.2 QBF problem as a game

It is classical to view the QBF evaluation problem as reachability in an And-Or graph, or equivalently as a two-player reachability game [24]. For the formula $f = Q_1 x_1 \cdot Q_2 x_2 \cdots Q_m x_m \cdot \psi$ over $V = \{x_1, x_2, \dots, x_m\}$, the game is played in m rounds (numbered $1, \dots, m$) by the existential player P_{\exists} and the universal player P_{\forall} . In round i , the truth value of the variable x_i is chosen by player P_{Q_i} . After m rounds, the players have constructed a valuation $v : \{x_1, \dots, x_m\} \rightarrow \{0, 1\}$, and player P_{\exists} wins if $\psi[v] = \text{true}$ (all clauses are satisfied), otherwise player P_{\forall} wins. It is easy to see that player P_{\exists} has a winning strategy in this game iff the formula f is true. Note that instead of having one round for each variable, we

can also consider a game with one round for each block of variables, such that the blocks correspond to quantifier alternations in f . The players then choose a valuation for all the variables in the block at once, and the number of rounds is equal to the number of quantifier alternations. As our algorithm is based on this game metaphor, we present the And-Or graph on which the game is played.

Let $P = X_1 \cup X_2 \cup \dots \cup X_n$ be a partition of $V = \{x_1, x_2, \dots, x_m\}$, and let $f = Q_1 X_1 \cdot Q_2 X_2 \cdot \dots \cdot Q_n X_n \cdot \psi$ be a QBF formula over V . We define the And-Or graph $G_f = (S, S_\exists, S_\forall, s_0, E, F)$ where:

- $S = \{\psi[v] \mid \text{dom}(v) = X_{\leq i-1}, \text{ for } 1 \leq i \leq n+1\}$;
- $S_\exists = \{\psi[v] \mid \text{dom}(v) = X_{\leq i-1} \wedge Q_i = \exists, \text{ for some } i \text{ s.t. } 1 \leq i \leq n\}$ is the set of P_\exists nodes;
- $S_\forall = \{\psi[v] \mid \text{dom}(v) = X_{\leq i-1} \wedge Q_i = \forall, \text{ for some } i \text{ s.t. } 1 \leq i \leq n\}$ is the set of P_\forall nodes;
- $s_0 = \psi$ is the initial node;
- $E = \{(\psi[v], \psi[vw]) \mid \text{dom}(v) = X_{\leq i-1} \wedge \text{dom}(w) = X_i, \text{ for } 1 \leq i \leq n\}$;
- $F = \{\psi[v] \in S \mid \psi[v] = \text{true}\}$.

The set S is naturally partitioned into *levels* as follows: $S = \text{Level}_1 \cup \text{Level}_2 \cup \dots \cup \text{Level}_{n+1}$ where $\text{Level}_i = \{\psi[v] \mid \text{dom}(v) = X_{\leq i-1}\}$ for each $1 \leq i \leq n+1$. The objective of player P_\exists is to reach the set F of nodes $\psi[v]$ such that all clauses of ψ are satisfied by v . The game starts in node s_0 and player P_Q ($Q \in \{\exists, \forall\}$) chooses the successor of node s if $s \in S_Q$. Thus if $s = \psi[v] \in S_\exists$ and $\text{dom}(v) = X_{\leq i-1}$, then player P_\exists chooses one of the $2^{|X_i|}$ possible successors of s in E , corresponding to a valuation $w : X_i \rightarrow \{0, 1\}$. A node s is *winning* for player P_\exists if he has a strategy to force reaching a node in F from s , no matter the choices of P_\forall ; otherwise it is *losing*. We denote by W the set of winning nodes for player P_\exists , and by $L = S \setminus W$ the set of losing nodes for P_\exists . We say that P_\exists is *winning the game* if $s_0 \in W$. In the sequel, we use the notations W_i (resp. L_i) to denote $W \cap \text{Level}_i$ (resp. $L \cap \text{Level}_i$).

Proposition 1. *A QBF formula f is true iff player P_\exists is winning the game G_f .*

Note that in the graph G_f , each node $\psi[v]$ with $\text{dom}(v) = X_{\leq i-1}$ can be associated with the formula $\text{Formula}(\psi[v]) \equiv Q_i X_i \cdot \dots \cdot Q_n X_n \cdot \psi[v]$, and we can strengthen the previous proposition as follows.

Proposition 2. *Given a QBF formula f , the set of winning nodes in the graph G_f is $W = \{\psi[v] \in S \mid \text{Formula}(\psi[v]) \text{ is true}\}$, and the set of losing nodes is $L = \{\psi[v] \in S \mid \text{Formula}(\psi[v]) \text{ is false}\}$.*

2.3 Structure in the And-Or graph and antichains

We present in the next section an algorithm to solve the game played on G_f which exploits the following *subsumption* relation on QBF formulas. We write $f_1 \sqsubseteq f_2$ if $f_1 = Q_i X_i \cdot \dots \cdot Q_n X_n \cdot \psi_1$ and $f_2 = Q_i X_i \cdot \dots \cdot Q_n X_n \cdot \psi_2$ are two QBF formulas with the same quantifier prefix, and $\psi_1 \subseteq \psi_2$. Intuitively, f_1 is more promising than f_2 for player P_\exists because all strategies that are winning from f_2 are also winning from f_1 .

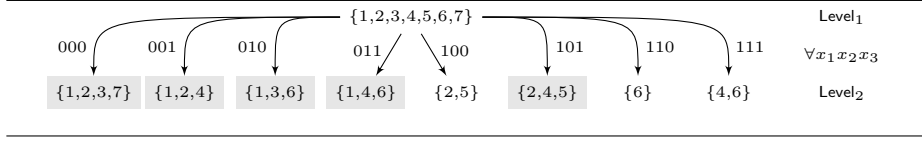


Fig. 3. Level₁ and Level₂ of G_f and the 5 minimal valuations of P_\forall .

Proposition 3. *Suppose that $f_1 \sqsubseteq f_2$. We have: if f_2 is true, then f_1 is true; and if f_1 is false, then f_2 is false.*

As a direct consequence of Proposition 2 and Proposition 3, we obtain the following corollary.

Corollary 1. *In the graph G_f , for all nodes $s_1, s_2 \in \text{Level}_i$ in the same level such that $\text{Formula}(s_1) \sqsubseteq \text{Formula}(s_2)$, we have: if $s_2 \in W_i$, then $s_1 \in W_i$; and if $s_1 \in L_i$, then $s_2 \in L_i$.*

Hence, W_i is \sqsubseteq -downward closed and L_i is \sqsubseteq -upward closed. The set of \sqsubseteq -maximal elements of W_i , noted $\lceil W_i \rceil$, is an *antichain* for the partial order \sqsubseteq that canonically (and compactly) represents W_i , and the \sqsubseteq -minimal elements of L_i , noted $\lfloor L_i \rfloor$, is an antichain that canonically (and compactly) represents L_i . Elements of these antichains are denoted α, β .

3 Algorithms

In Section 3.1, we discuss the computation of optimal valuations to explore only the most promising nodes, and in Section 3.2, we propose an antichain-based algorithm for solving the QBF evaluation game.

3.1 Maximal and minimal valuations

According to Corollary 1, when it is the turn for P_\exists to play in node $s = \varphi \in \text{Level}_i$, he can restrict his choices among valuations $w : X_i \rightarrow \{0, 1\}$ that *maximize* the set of clauses of φ that are satisfied. Symmetrically, player P_\forall can restrict his choices among valuations $w : X_i \rightarrow \{0, 1\}$ that *minimize* the set of clauses of φ that are satisfied.

We define the notion of maximal and minimal valuations as follows. Let φ be a CNF formula over $X_{\geq i}$. A valuation $w : X_i \rightarrow \{0, 1\}$ is φ -*maximal* if for all $w' : X_i \rightarrow \{0, 1\}$, $\text{sat}_w(\varphi) \subseteq \text{sat}_{w'}(\varphi)$ implies $\text{sat}_w(\varphi) = \text{sat}_{w'}(\varphi)$. Symmetrically, w is φ -*minimal* if for all $w' : X_i \rightarrow \{0, 1\}$, $\text{sat}_{w'}(\varphi) \subseteq \text{sat}_w(\varphi)$ implies $\text{sat}_w(\varphi) = \text{sat}_{w'}(\varphi)$.

Example 1. Consider the CNF formula ψ of Fig. 1. In level 1, we have $X_1 = \{x_1, x_2, x_3\}$ which are universal variables. Among the $2^3 = 8$ valuations, 5 are ψ -minimal (shaded in Fig. 3). Remember that the nodes in the And-Or graph are the clauses that *remain* to be satisfied, thus maximal such sets correspond to minimal valuations. Note also that we may need to compute *all* maximal (or minimal) valuations in a node.

Maximal and minimal valuations can be computed by multiple calls to a SAT solver. Let us give the intuition for maximal valuations. Let φ be a set of clauses over $X_{\geq i}$. First notice that a valuation $w : X_i \rightarrow \{0, 1\}$ is φ -maximal if and only if it is $\pi_{X_i}(\varphi)$ -maximal. Thus we can assume w.l.o.g. that φ is a set of clauses over X_i (instead of $X_{\geq i}$). Using a set of new variables $Y = \{y_c \mid c \in \varphi\}$, we transform the set of clauses φ into a set of clauses φ' over $X_i \cup Y$ such that any valuation $w : X_i \cup Y \rightarrow \{0, 1\}$ with $w(y_c) = 1$ implies that w satisfies c . By a first call to a SAT solver on φ' , we get a valuation w and a subset C of clauses of φ that are satisfied by w . Then we modify φ' into φ'' by imposing additional constraints on the variables of Y in a way that a second call to a SAT solver provides a subset of satisfied clauses of φ that strictly contains C . Iterating this procedure, we finally obtain a valuation that satisfies a maximal set of clauses in φ . Our reduction to SAT is classical and detailed in the appendix for the interested reader.

Computing maximal and minimal valuations can also be reduced to variants of the *Maximum Satisfiability* (Max-Sat) and *Minimum Satisfiability* (Min-SAT) problems [17, 18]. Given a CNF formula φ , the Max-Sat problem asks to compute a valuation that maximizes the *number* of satisfied clauses in φ (Min-Sat is defined symmetrically). Note that such a valuation is φ -maximal but the converse is not necessarily true. A variant of Max-SAT, called *partial Max-SAT*, can be used in our algorithm to generate φ -maximal valuations. Given a CNF formula $\varphi = \varphi_h \wedge \varphi_s$ where φ_h represents the *hard* clauses and φ_s represents the *soft* clauses, the partial Max-SAT problem consists in finding a valuation such that all hard clauses are satisfied and the number of satisfied soft clauses is maximized. To compute all φ -maximal valuations, we can use the hard clauses with clause-selectors to impose that for each already computed φ -maximal valuation w , at least one new clause $c \notin \text{sat}_w(\varphi)$ is satisfied.

3.2 Antichain-based algorithm

We present our antichain-based algorithm to evaluate a QBF formula $f = Q_1 X_1 \cdots Q_n X_n \cdot \psi$. It can be seen as a mix of forward and backward exploration of the And-Or graph G_f . Such a forward-backward exploration was also used with success in timed games [7].

The algorithm consists of two recursive procedures $\text{ATCDepthFirst}\exists(\varphi, i)$ and $\text{ATCDepthFirst}\forall(\varphi, i)$ where φ is a node of G_f and i is the recursion level (see Algorithms 1 and 2). Initially, we call $\text{ATCDepthFirst}\exists(\psi, 1)$ if $Q_1 = \exists$, and $\text{ATCDepthFirst}\forall(\psi, 1)$ if $Q_1 = \forall$. These procedures determine whether a node φ is winning or losing for P_{\exists} , i.e. whether $\varphi \in W_i$ or $\varphi \in L_i$. The sets W_i and L_i are updated as global variables and compactly stored by antichains $[W_i]$ and $[L_i]$ respectively. They are used to prune the search, by the *subsumption* checks (e.g., lines 8 and 11 in Algorithm 1).

The details of $\text{ATCDepthFirst}\exists(\varphi, i)$ are as follows. If φ is not even satisfiable, then it is a losing node; if φ is satisfiable and $i = n$, then it is winning since φ belongs to S_{\exists} ; otherwise, the procedure enumerates the φ -maximal valuations w (line 7) and checks if $\varphi[w]$ is winning at level $i + 1$. For player P_{\exists} ,

Algorithm 1 ATCDepthFirst $\exists(\varphi, i)$

Require: node $\varphi \in S_{\exists} \cap \text{Level}_i$, $i \leq n$.**Ensure:** Win if $\varphi \in W_i$, Lose if $\varphi \in L_i$.

```
1: if  $\neg \text{lsSat}(\varphi)$  then
2:   Add( $\varphi, \lfloor L_i \rfloor$ )
3:   return Lose
4: if  $i = n$  then
5:   Add( $\varphi, \lceil W_i \rceil$ )
6:   return Win
7: for each  $\varphi$ -maximal valuation  $w$  :
    $X_i \rightarrow \{0, 1\}$  do
8:   if  $\exists \alpha \in \lceil W_{i+1} \rceil$  s.t.  $\varphi[w] \subseteq \alpha$  then
9:     Add( $\varphi, \lceil W_i \rceil$ )
10:    return Win
11:   if  $\neg(\exists \alpha \in \lfloor L_{i+1} \rfloor$  s.t.  $\alpha \subseteq \varphi[w])$ 
   then
12:      $F \leftarrow \text{ATCDepthFirst}\forall(\varphi[w], i+1)$ 
13:     if  $F = \text{Win}$  then
14:       Add( $\varphi, \lceil W_i \rceil$ )
15:       return Win
16:   Add( $\varphi, \lfloor L_i \rfloor$ )
17: return Lose
```

Algorithm 2 ATCDepthFirst $\forall(\varphi, i)$

Require: node $\varphi \in S_{\forall} \cap \text{Level}_i$, $i < n$.**Ensure:** Win if $\varphi \in W_i$, Lose if $\varphi \in L_i$.

```
1: if  $\neg \text{lsSat}(\varphi)$  then
2:   Add( $\varphi, \lfloor L_i \rfloor$ )
3:   return Lose
4: for each  $\varphi$ -minimal valuation  $w$  :
    $X_i \rightarrow \{0, 1\}$  do
5:   if  $\exists \alpha \in \lfloor L_{i+1} \rfloor$  s.t.  $\alpha \subseteq \varphi[w]$  then
6:     Add( $\varphi, \lfloor L_i \rfloor$ )
7:     return Lose
8:   if  $\neg(\exists \alpha \in \lceil W_{i+1} \rceil$  s.t.  $\varphi[w] \subseteq \alpha)$ 
   then
9:      $F \leftarrow \text{ATCDepthFirst}\exists(\varphi[w], i+1)$ 
10:    if  $F = \text{Lose}$  then
11:      Add( $\varphi, \lfloor L_i \rfloor$ )
12:      return Lose
13:   Add( $\varphi, \lceil W_i \rceil$ )
14: return Win
```

maximal valuations are sufficient because the set of winning nodes is downward-closed (see Corollary 1). The recursive call to $\text{ATCDepthFirst}\forall(\varphi[w], i+1)$ can be avoided if $\varphi[w]$ is in the downward-closure of $\lceil W_{i+1} \rceil$ (line 8), or if $\varphi[w]$ is in the upward-closure of $\lfloor L_{i+1} \rfloor$ (line 11). Finally, if all φ -maximal valuations have been explored, then φ is losing (line 17).

The procedure $\text{ATCDepthFirst}\forall(\varphi, i)$ for nodes $\varphi \in S_{\forall}$ is dual. Note that the case $i = n$ is not relevant since $\mathbf{Q}_n = \exists$. By a symmetrical argument, \mathbf{P}_{\forall} needs to consider only the φ -minimal valuations.

The depth-first search is implemented by the above two procedures where the antichains $\lceil W_i \rceil$ and $\lfloor L_i \rfloor$ (for $1 \leq i \leq n$) are initially empty. The antichain structure is maintained by the procedure **Add** which computes $\lceil \{\varphi\} \cup W_i \rceil$ and $\lfloor \{\varphi\} \cup L_i \rfloor$. Maximal and minimal valuations are computed as discussed in Section 3.1.

Example 2. Consider the CNF formula ψ of Fig. 1. Since $\mathbf{Q}_1 = \forall$, the algorithm starts with $\text{ATCDepthFirst}\forall(\psi, 1)$ which needs to explore the 5 minimal valuations of Fig. 3. Assume that the first valuation is $(x_1 \mapsto 0, x_2 \mapsto 0, x_3 \mapsto 0)$, denoted 000, which satisfies clauses 4, 5, 6. Then, the game proceeds to the node $\psi[w] = \psi[000] = \{1, 2, 3, 7\}$ of remaining clauses where the turn is to player \mathbf{P}_{\exists} . The subgraph of G_f rooted at $\psi[000]$ is shown in Fig. 4. Among the 4 possible valuations for player \mathbf{P}_{\exists} , only 2 are maximal, namely 01 and 11. For valuation 01, only clauses 1 and 7 remain. At this point, all variables are instantiated except

x_6 and y_7 , and player P_{\exists} wins by choosing $y_7 \mapsto 1$ which satisfies ψ no matter the value of x_6 chosen by player P_{\forall} .

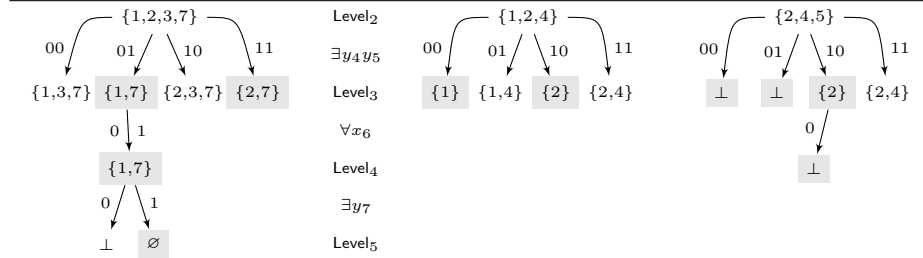


Fig. 4. Subgames rooted at $\psi[000]$, $\psi[001]$, and $\psi[101]$

Thus nodes $\{1, 7\}$ at Level₃, and $\{1, 2, 3, 7\}$ at Level₂ are winning, and the corresponding antichains are updated as follows: $\lceil W_3 \rceil = \{\{1, 7\}\}$ and $\lceil W_2 \rceil = \{\{1, 2, 3, 7\}\}$.

At the root node, the valuation 000 is not a good choice for player P_{\forall} , and no conclusion can be drawn yet for this node (Fig. 3). We need to explore another choice for player P_{\forall} . The second tree of Fig. 4 shows the subgame rooted at node $\psi[001]$. In this case, with minimal valuation 00, player P_{\exists} reaches node $\{1\}$ in Level₃. Since $\{1\} \in W_3$ (indeed $\{1\} \subseteq \{1, 7\} \in \lceil W_3 \rceil$, and W_3 is the downward-closure of $\lceil W_3 \rceil$), he knows immediately that he is winning without further exploring the graph. This situation illustrates the power of the subsumption which allows to prune the search for nodes smaller than previously visited winning ones. The antichain $\lceil W_2 \rceil$ is then updated to $\{\{1, 2, 3, 7\}, \{1, 2, 4\}\}$. Valuation 001 is again a bad choice for P_{\forall} .

One can check that valuations 010 and 011 are bad choices for P_{\forall} and their exploration leads to the following update of the antichains: $\lceil W_2 \rceil = \{\{1, 2, 3, 7\}, \{1, 2, 4\}, \{1, 3, 6\}, \{1, 4, 6\}\}$ and $\lceil W_3 \rceil = \{\{1, 7\}, \{6\}\}$. The last minimal valuation is 101 and for all choices of player P_{\exists} , there is a choice of player P_{\forall} to falsify the formula (see the last tree in Fig. 4). Therefore P_{\exists} is losing the game and the formula f is false.

The correctness of this algorithm is established using the notion of certificate presented in the next section.

Theorem 1. *Let f be a QBF formula. Applying Algorithms 1 and 2 on f returns Win if and only if f is true.*

4 Certificates

In the previous section we have described a depth-first search algorithm to evaluate a QBF formula f . This algorithm computes the sets of winning nodes and

losing nodes for each level of the graph G_f , and these sets are compactly represented by antichains.

Our algorithm gathers enough information in these antichains to easily build *compact certificates* for both true and false QBF formulas. The certificates for true formulas differ from the certificates for false formula. To the best of our knowledge they are different from the certificates considered in the literature [2, 22].

A *positive certificate* for a QBF formula $f \equiv Q_1 X_1 \cdots Q_n X_n \cdot \psi$ is a pair $\langle (C_i^+)_{1 \leq i \leq n}, \mathbf{w} \rangle$ such that:

- each C_i^+ is a set of nodes at the i th level of the graph G_f , that is, $C_i^+ \subseteq \text{Level}_i$;
- for each i such that $\text{Level}_i \subseteq S_{\exists}$, \mathbf{w} is a function that assigns a valuation $\mathbf{w}(\alpha) : X_i \rightarrow \{0, 1\}$ to each $\alpha \in C_i^+$;
- and the following properties are verified:
 1. $C_1^+ = \{\psi\}$.
 2. for each $i < n$ such that $\text{Level}_i \subseteq S_{\exists}$, for all $\alpha \in C_i^+$, there exists $\beta \in C_{i+1}^+$ such that $\alpha[\mathbf{w}(\alpha)] \subseteq \beta$.
 3. for each $i < n$ such that $\text{Level}_i \subseteq S_{\forall}$, for all $\alpha \in C_i^+$, for all $w : X_i \rightarrow \{0, 1\}$, there exists $\beta \in C_{i+1}^+$ such that $\alpha[w] \subseteq \beta$.
 4. for $i = n$, for all $\alpha \in C_i^+$, $\alpha[\mathbf{w}(\alpha)] = \text{true}$.

Clearly, there exists a nondeterministic polynomial time algorithm to recognize pairs $\langle (C_i^+)_{1 \leq i \leq n}, \mathbf{w} \rangle$ that are not positive certificate. All the verification related to Conditions 1, 2 and 4 can be done in deterministic polynomial time while Condition 3 requires nondeterminism. Therefore, verifying the validity of a positive certificate is a problem in coNP.

The next two lemmas are proved in the appendix.

Lemma 1. *If $\langle (C_i^+)_{1 \leq i \leq n}, \mathbf{w} \rangle$ is a positive certificate for a QBF formula f , then f is true.*

Lemma 2. *Let $\lceil W_i \rceil$, $1 \leq i \leq n$, be the antichains built by the execution of Algorithms 1 and 2 on formula f . For each i such that $\text{Level}_i \subseteq S_{\exists}$, for all $\alpha \in \lceil W_i \rceil$, let $\mathbf{w}(\alpha)$ be the valuation used by Algorithms 1 when α has been declared winning. If f is true, then $\langle (\lceil W_i \rceil)_{1 \leq i \leq n}, \mathbf{w} \rangle$ is a positive certificate for f .*

The next theorem directly follows from the two previous lemmas

Theorem 2. *Let f be a QBF formula. Then f is true if and only if there exists a positive certificate for f .*

Negative certificates are defined in a way similar to positive certificates; they are a witness for false formulas f . The interested reader is referred to the appendix for more details.

5 Optimizations

5.1 Guiding the search to promising valuations

We recall that in Algorithm 1, when φ is a satisfiable formula, and $i \leq n - 1$, player P_{\exists} traverses all the maximal valuations $w : X_i \rightarrow \{0, 1\}$ in the hope to find w such that $\varphi[w]$ is winning. The first optimization that we consider tries to guide the search to *promising* maximal valuations. The new algorithm for player P_{\exists} is described in the appendix. Symmetrical improvements also exist for player P_{\forall} .

Improving ATCDepthFirst \exists . When considering all the maximal valuations $w : X_i \rightarrow \{0, 1\}$, we observe that player P_{\exists} is winning as soon as he can find a valuation w such that $\varphi[w] \subseteq \alpha$ for some $\alpha \in [W_{i+1}]$ (see Corollary 1). So, it is wise for P_{\exists} to first try to find such a valuation w . Suppose now that player P_{\exists} cannot win by exploiting the elements of $[W_{i+1}]$. Then he should avoid considering (maximal) valuations w such that $\alpha \subseteq \varphi[w]$ for some $\alpha \in [L_{i+1}]$ as $\varphi[w]$ is losing (see Corollary 1). These two observations are exploited by the new algorithm.

Improving ATCDepthFirst \forall . We can improve the choice of minimal valuations for player P_{\forall} in a symmetric manner. Indeed, P_{\forall} should first look for a valuation w such that there exists $\alpha \in [L_{i+1}]$ and $\alpha \subseteq \varphi[w]$. If such a valuation does not exist, then he should only consider minimal valuations that avoid the set $[W_{i+1}]$ of winning nodes.

5.2 Improving the information about losing and winning nodes

We have proposed algorithms that can be seen as mixing a forward exploration of the And-Or graph with a backward propagation about the winning and losing nodes. As shown below, we can *improve the propagation* of this information in several ways.

At initialization. We recall that the antichain $[L_i]$ is initially empty for Algorithms 1 and 2. We can add some useful information to $[L_i]$ in the following case. Consider the initial node ψ of the graph G_f , and let i , $1 \leq i \leq n - 1$. Suppose that the projection $d = \pi_{X_{\geq i}}(c)$ of a clause c in ψ on $X_{\geq i}$ has all its literals *universally quantified*. The clause d must be satisfied by a valuation generated before reaching Level_i as it is controlled by player P_{\forall} . So, at initialization we can add $\{d\}$ to $[L_i]$ for any such d (all sets of clauses that contain d are losing at Level_i).

For example, consider clause $c_2 = x_2 \vee \bar{y}_4 \vee x_6$ of our running example (see Fig. 1). When projected on the variables $X_{\geq 3} = \{x_6, y_7\}$, this clause reduces to x_6 which is a universally quantified variable. If this clause has not been satisfied by a valuation w generated before reaching Level_3 , then clearly player P_{\forall} has a strategy to falsify it.

When updating $[W_i]$ and $[L_i]$. Now, consider an optimization that can be applied when updating the sets of winning and losing nodes during the search. We consider two scenarios. First, assume that node φ is declared winning by the algorithm `ATCDepthFirst \exists` in `Level $_i$` . This means that there exists a valuation $w : X_i \rightarrow \{0, 1\}$ such that either $\varphi[w] \subseteq \alpha$ for some $\alpha \in [W_{i+1}]$, or $\varphi[w]$ is declared winning by the recursive call to `ATCDepthFirst \forall` . Notice that φ is a subset of the set of clauses in the root ψ (projected on variables $X_{\geq i}$). It may happen that other clauses $c \in \psi$ are also satisfied by valuation w . Therefore, in a way to have bigger elements in $[W_i]$, it is preferable to add the set $\varphi' = \varphi \cup \{c \mid c \in \text{sat}_w(\psi)\}$ instead of φ to $[W_i]$.

As an example, consider the first tree in Fig. 4 of our running example. We see that at node $\{1, 7\}$ in `Level $_4$` , player P_{\exists} has a winning strategy by choosing value 1 for variable y_7 . With this choice, clause 1 and clause 7 are satisfied but also clause 4. It means that in `Level $_4$` , both $\{1, 7\}$ and $\{1, 4, 7\}$ are winning. So instead of adding $\{1, 7\}$ to $[W_4]$, we can add $\{1, 4, 7\}$.

Second, assume that φ is declared losing by Algorithm `ATCDepthFirst \exists` in `Level $_i$` with $i = n$. This means that φ is unsatisfiable. Instead of adding φ to $[L_i]$, we can add any unsatisfiable core $\varphi' \subseteq \varphi$ instead.

6 Experimental Results

We have implemented the algorithm of Section 3 including the optimizations described in Section 5. The code is written in Python and C, and makes calls to a SAT solver. Python is used to implement high level operations like the exploration of the And-Or graph, and for the construction of CNF formulas submitted to the SAT solver. A small part of the code is written in C for some low level operations on the data structures. We use two SAT solvers, mainly MiniSat [10], and PicoSAT [5] (for computing an unsatisfiable core as described at the end of Section 5). The code begins with a preprocessing: the formula is first simplified with PreQuel [26] and then presented in a tree-like structure [3]. These are standard practices in QBF solving.

Our implementation is preliminary and not optimized. For example, we compute maximal and minimal valuations with multiple calls to the SAT solver MiniSat (see Section 3.1 and the appendix); we do not use a reduction to variants of Max-SAT and Min-SAT. We adopted this strategy (and Python) to facilitate the fast evaluation of different ideas. As we explain below, using dedicated solvers for the partial Max-SAT and Min-SAT problems and efficient programming language such as C should bring huge performance speed-up. The experiments were run on a PC equipped with a Intel i7 2.8GHz processor, 6 GB of RAM and running Linux Ubuntu 2.6.

Table 1 shows the experimental results obtained with our solver on several instances proposed during the seventh QBF solvers evaluation (QBFEVAL'10) [25]. They are compared with the results obtained with three state-of-the-art QBF solvers: QuBE-7.0, sKizzo-v0.8.2-beta and DepQBF-0.1 (winner of QBFEVAL'10). **Var** (resp. **Cl**, **Blocs**) gives the number of variables (resp. clauses, blocs) of the instance. **Nodes** is the number of nodes of the And-Or

graph that have been explored. **SATc** gives the number of calls to the SAT solver for computing the minimal and maximal valuations. Column **ATC** (resp. **QuBE**, **sKizzo**, **DepQBF**) presents the execution times (in seconds) of our solver (resp. **QuBE**, **sKizzo**, **DepQBF**).

According to the execution times, our approach does not perform well in general as compared the other solvers. However, the size of the constructed search tree is amazingly small as compared to the size of the entire search space. It is also very interesting to observe that as the antichains are composed of nodes of the search tree, they are also small. For instance, the search tree for the third instance in Table 1 contains 41 nodes only (against $2^{377} - 1$ potential nodes). On the other hand, we notice that 3618 calls to a SAT solver were performed to build these 41 nodes. Roughly speaking, this means (for this instance) that our solver needs to perform an average of 88 calls per node to a SAT solver. If the maximal and minimal valuations were computed with a partial Max-SAT or Min-SAT solver, only 41 (instead of 3618) calls would be performed.

Instance	Var	Cl	Blocs	Nodes	SATc	ATC	QuBE	sKizzo	DepQBF
aim-50-2-0-yes1-1-90-shuffled	210	415	3	62	1892	1.31	0.02	0.09	0.01
aim-50-3-4-yes1-2-90-shuffled	278	693	3	78	2429	2.42	0.03	0.12	0.01
aim-100-1-6-yes1-3-90-shuffled	376	665	3	41	3618	3.14	0.04	0.15	0.01
comp.blif-0.10-0.20-0-0-inp-exact-shuffled	310	831	7	50	915	0.66	0.02	0.04	0.01
comp.blif-0.10-1.00-0-0-inp-exact-shuffled	306	842	3	8	23	0.09	0.03	0.01	0.01
ii8a1-90-shuffled	328	652	3	121	4118	4.95	0.12	0.72	0.01
toilet-c-08-05.9-shuffled	597	4441	3	44	15329	140.56	0.03	0.03	0.02
toilet-a-10-01.5-shuffled	170	10902	3	16	1362	14.02	8.29	0.49	0.03
toilet-a-04.10.2-shuffled	140	894	3	13	359	0.44	0.01	0.01	0.01
z4ml.blif-0.10-1.00-0-0-out-exact-shuffled	63	194	3	101	766	0.61	0.01	0.01	0.01

Table 1. Several instances of QBFEVAL'10

Moreover, there are benchmarks from the QBFLIB library [13] on which our solver already performs better than state-of-the-art solvers. They correspond to the encodings of modal K formulas satisfiability into QBF. The execution times of our solver is compared with three solvers (with a timeout of 300 seconds), for the family k-path-n in Fig. 5, and for the family k-path-p in Fig. 6.

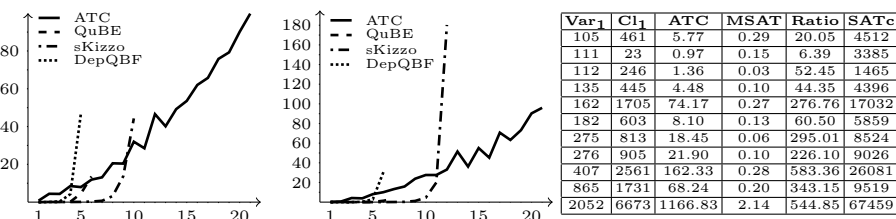


Fig. 5. Family k-path-n. Fig. 6. Family k-path-p. Fig. 7. Computation of 10 maximal valuations.

We are convinced that we can strongly increase the performance of our solver by computing minimal and maximal valuations using a reduction to partial Max-

SAT and Min-SAT, instead of the current reduction to SAT. To substantiate this claim, we compare the time needed to compute 10 distinct maximal valuations using the partial Max-SAT solver IncWMaxsatz [19] versus multiple calls to the SAT solver MiniSat. The comparison results are collected in Table 7. The selected instances are taken from QBFEVAL’10. They are all of the form $\exists X_1 \cdot \forall X_2 \cdots \exists X_n \cdot \psi$ with a first block of existential variables. We recall that a valuation $w : X_1 \rightarrow \{0, 1\}$ is ψ -maximal if and only if it is $\pi_{X_1}(\psi)$ -maximal. Column **Var₁** in Table 7 gives the size of X_1 , and column **Cl₁** gives the number of non equivalent clauses of $\pi_{X_1}(\psi)$. Column **ATC** provides the execution times (in seconds) for the computation of 10 maximal valuations as done in our current solver, and column **SATc** gives as before the number of calls to a SAT solver. Column **MSAT** provides the execution times to compute 10 maximal valuations via calls to the partial Max-SAT solver INCWMAXSATZ [19]. Column **Ratio** provides the ratio obtained by dividing **ATC** by **MSAT**. In other words it gives a rough idea of the speed-up factor that could be achieved if a partial Max-SAT solver was used in our solver.

As we can observe, these ratios are mostly above 200. Moreover, they are often larger than the ratios between the execution times of our solver and the other solvers as given in Table 1. Therefore we believe that our exploration technique is promising and that it could be much more effective using partial Max-SAT and Min-SAT solvers.

7 Conclusion and Perspectives

The approach presented in this paper for QBF solving is inspired by previous works on effective antichain algorithms for PSPACE-complete problems in automata theory which are based on simple subsumption relations [8, 9]. While the And-Or graph of QBF formulas enjoys such a subsumption relation, this idea has not been exploited in search-based QBF solvers. In a prototypical implementation, we have evaluated the feasibility of antichain-based algorithms for QBF. Experimental results show that on several benchmarks the size of the search tree is drastically reduced, and that some instances are solved more efficiently than by the leading QBF solvers. This shows that the approach is promising and it provides a new research direction in the area.

We identify some perspectives in the development of competitive antichain algorithms and their integration in QBF solvers.

First, our current Python implementation computes minimal and maximal valuations by multiple calls to a SAT solver. As we have shown in Section 6, the integrated use of efficient and dedicated partial Max-Sat/Min-Sat solvers would reduce the execution times by a large factor. The use of a programming language like C would also increase the performance of our solver. Further optimizations and fine-tuning of the code would definitely bring favorable performance comparison with the state-of-the-art QBF solvers on a larger set of benchmarks.

Second, suitable trade-offs need to be found for collecting relatively cheap and useful information along the search. Several optimizations in the spirit of Section 5.2 can be considered. For example, in the initialization phase we can

pre-compute richer information about winning and losing positions (to be stored in $[W_i]$ and $[L_i]$) to better guide the search. As an illustration, consider a QBF formula with a set of clauses ψ . At level i , let φ be the projection of the clauses in ψ to the variables $X_{\geq i}$ occurring at level i and after. If φ is unsatisfiable, then we can extract any unsatisfiable core of φ and add it to the losing positions $[L_i]$ at level i . On the other hand, if φ is satisfiable and we further restrict φ to the existentially quantified variables, then we can try to satisfy maximal subsets of clauses φ' within φ . If the set of clauses not in φ' are satisfied when the game enters level i , then player P_{\exists} has a winning strategy from there. Thus we can add to $[W_i]$ the set φ' , etc.

Third, in order to evaluate absolute performance of an antichain-based QBF solver, it would be necessary to consider other optimizations that have been proposed in the literature. Most of them are compatible with our approach, and a realistic search-based solver needs to implement them. For example, standard backtracking could be replaced by various backtracking strategies, like back-jumping that has been integrated in many state-to-the-art QBF solvers [12]. Other classical optimizations that we have not implemented are unit propagation, and monotone literals elimination [6].

Acknowledgements We thank Marco Benedetti (the author of sKizzo) for his useful answers and great help, and Nicolas Maquet for his guidance in the implementation.

References

1. M. Benedetti. Evaluating QBFs via Symbolic Skolemization. In F. Baader and A. Voronkov, editors, *LPAR*, volume 3452 of *LNCS*, pages 285–300. Springer, 2004.
2. M. Benedetti. Extracting Certificates from Quantified Boolean Formulas. In L. P. Kaelbling and A. Saffiotti, editors, *IJCAI*, pages 47–53. Professional Book Center, 2005.
3. M. Benedetti. Quantifier Trees for QBFs. In F. Bacchus and T. Walsh, editors, *SAT*, volume 3569 of *LNCS*, pages 378–385. Springer, 2005.
4. M. Benedetti. sKizzo: A Suite to Evaluate and Certify QBFs. In R. Nieuwenhuis, editor, *CADE*, volume 3632 of *LNCS*, pages 369–376. Springer, 2005.
5. A. Biere. PicoSAT Essentials. *JSAT*, 4(2-4):75–97, 2008.
6. M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified Boolean formulae. In *AAAI-98/IAAI-98 Proceedings (Madison, WI, 1998)*, pages 262–267. MIT Press, Cambridge, MA, 1998.
7. F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In M. Abadi and L. de Alfaro, editors, *CONCUR*, volume 3653 of *LNCS*, pages 66–80. Springer, 2005.
8. M. De Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A New Algorithm for Checking Universality of Finite Automata. In T. Ball and R. B. Jones, editors, *CAV*, volume 4144 of *LNCS*, pages 17–30. Springer, 2006.
9. L. Doyen and J.-F. Raskin. Antichain Algorithms for Finite Automata. In J. Esparza and R. Majumdar, editors, *TACAS*, volume 6015 of *LNCS*, pages 2–22. Springer, 2010.

10. N. Eén and N. Sörensson. An extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *SAT*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.
11. U. Egly, T. Eiter, H. Tompits, and S. Woltran. Solving Advanced Reasoning Tasks Using Quantified Boolean Formulas. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 417–422. AAAI Press, 2000.
12. E. Giunchiglia, M. Narizzano, and A. Tacchella. An analysis of backjumping and trivial truth in quantified boolean formulas satisfiability. In F. Esposito, editor, *AI*IA*, volume 2175 of *LNCS*, pages 111–122. Springer, 2001.
13. E. Giunchiglia, M. Narizzano, and A. Tacchella. Quantified Boolean Formulas satisfiability library (QBFLIB), 2001. www.qbflib.org.
14. E. Giunchiglia, M. Narizzano, and A. Tacchella. QuBE++: An Efficient QBF Solver. In A. J. Hu and A. K. Martin, editors, *FMCAD*, volume 3312 of *LNCS*, pages 201–213. Springer, 2004.
15. E. Giunchiglia, M. Narizzano, and A. Tacchella. Clause/Term Resolution and Learning in the Evaluation of Quantified Boolean Formulas. *J. Artif. Intell. Res. (JAIR)*, 26:371–416, 2006.
16. T. Jussila and A. Biere. Compressing BMC Encodings with QBF. *Electron. Notes Theor. Comput. Sci.*, 174:45–56, May 2007.
17. R. Kohli, R. Krishnamurti, and P. Mirchandani. The Minimum Satisfiability Problem. *SIAM J. Discrete Math.*, 7(2):275–283, 1994.
18. C. M. Li and F. Manyà. MaxSAT, Hard and Soft Constraints. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 613–631. IOS Press, 2009.
19. H. Lin and K. Su. Exploiting inference rules to compute lower bounds for max-sat solving. In *IJCAI*, pages 2334–2339, 2007.
20. F. Lonsing and A. Biere. DepQBF: A dependency-aware QBF solver (System Description). *Journal on Satisfiability, Boolean Modeling and Computation*, 7:71–76, 2010.
21. F. Lonsing and A. Biere. Integrating Dependency Schemes in Search-Based QBF Solvers. In Strichman and Szeider [27], pages 158–171.
22. M. Narizzano, C. Peschiera, L. Pulina, and A. Tacchella. Evaluating and certifying QBFs: A comparison of state-of-the-art tools. *AI Commun.*, 22:191–210, December 2009.
23. G. Pan and M. Y. Vardi. Symbolic Decision Procedures for QBF. In M. Wallace, editor, *CP*, volume 3258 of *LNCS*, pages 453–467. Springer, 2004.
24. C. H. Papadimitriou. *Computational complexity*. Addison-Wesley Publishing Company, Reading, MA, 1994.
25. C. Peschiera, L. Pulina, A. Tacchella, U. Bubeck, O. Kullmann, and I. Lynce. The Seventh QBF Solvers Evaluation (QBFEVAL’10). In Strichman and Szeider [27], pages 237–250.
26. H. Samulowitz, J. Davies, and F. Bacchus. Preprocessing QBF. In F. Benhamou, editor, *CP*, volume 4204 of *LNCS*, pages 514–529. Springer, 2006.
27. O. Strichman and S. Szeider, editors. *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6175 of *LNCS*. Springer, 2010.

Appendix

Details concerning Subsection 3.1

Maximal valuations. To compute φ -maximal valuations $w : X_i \rightarrow \{0, 1\}$ for a CNF formula φ over $X_{\geq i}$, we adopt a classical approach that performs multiple calls to a SAT solver, and can be summarized as follows.

First recall that w.l.o.g. we can suppose that $\varphi = \{c_1, c_2, \dots, c_k\}$ is a formula over X_i (instead of $X_{\geq i}$). For each clause $c_i \in \varphi$, we introduce a new variable y_i called the *selector* of c_i . We denote by Y this set of new variables. We modify each clause $c_i \in \varphi$ as $c'_i = \bar{y}_i \vee c_i$. Notice that if c'_i is satisfied by some valuation $v : X_i \cup Y \rightarrow \{0, 1\}$ and $v(y_i) = 1$, then c_i must be satisfied by v . Then we construct the CNF formula

$$\varphi' = \bigwedge_{c_i \in \varphi} c'_i \wedge \bigvee_{y_i \in Y} y_i.$$

If φ' is unsatisfiable, then there is no clause c_i in φ that can be made true, and all valuations $w : X_i \rightarrow \{0, 1\}$ are φ -maximal (because $\text{sat}_w(\varphi) = \emptyset$). Otherwise, let $v_1 : X_i \cup Y \rightarrow \{0, 1\}$ be a valuation such that $\varphi'[v_1] = \text{true}$. From v_1 , we know that all clauses c_i such that $v_1(y_i) = 1$ are satisfied by v_1 . Let $C_{v_1}(\varphi) = \{c_i \in \varphi \mid v_1(y_i) = 1\}$. This set gives a first subset of clauses in φ that can be satisfied *all together*. Now, we want to know if there exists a *superset* of this set that can be still satisfied. In this aim we construct a new formula

$$\varphi'(v_1) = \bigwedge_{c_i \in \varphi} c'_i \wedge \bigwedge_{y_i \in Y, v_1(y_i)=1} y_i \wedge \left(\bigvee_{y_i \in Y, v_1(y_i)=0} y_i \right).$$

Suppose that $\varphi'(v_1)$ is satisfiable and let v_2 be a valuation that evaluates it to true. The second part of $\varphi'(v_1)$ forces that all the clauses c_i satisfied by v_1 are still satisfied by v_2 . The last part of the formula imposes that at least one additional clause c_i is satisfied by v_2 . It follows that $C_{v_1}(\varphi) \subset C_{v_2}(\varphi)$, and we can iterate the process until formula $\varphi'(v_l)$ becomes unsatisfiable for some $l \geq 2$. Valuation $w = v_{l-1}$ identifies a maximal subset of satisfiable clauses c_i , and w is thus a φ -maximal valuation.

Minimal valuations. The approach to find minimal valuations uses an iteration schema similar to the one for maximal valuations. Let $\varphi = \{c_1, c_2, \dots, c_k\}$ be a formula and let Y be the set of selectors as above. To compute a φ -minimal valuation $w : X_i \rightarrow \{0, 1\}$, we first transform each clause of $c_i \in \varphi$ into the formula $\bar{c}_i \vee y_i$ which is expressed in CNF as the clause $c'_i = \bigwedge_{\ell \in c_i} \neg \ell \vee y_i$. Notice that if c'_i is satisfied by some valuation $v : X_i \cup Y \rightarrow \{0, 1\}$ and $v(y_i) = 0$, then c_i cannot be satisfied by v . Then, we consider the formula

$$\varphi' = \bigwedge_{c_i \in \varphi} c'_i \wedge \bigvee_{y_i \in Y} \neg y_i.$$

If φ' is unsatisfiable, then no clause in φ can be made false and any valuation $w : X_i \rightarrow \{0, 1\}$ is φ -minimal. Otherwise, let $v_1 : X_i \cup Y \rightarrow \{0, 1\}$ be a valuation such that $\varphi'[v_1] = \text{true}$. Valuation v_1 identifies the set of clauses $C_{v_1}(\varphi) = \{c_i \in \varphi \mid v_1(y_i) = 0\}$ such that these clauses are unsatisfied by v_1 all together. With an iteration schema similar to the one defined above, we can find a set of valuations $\{v_1, v_2, \dots, v_l\}$ such that $\varphi'(v_l)$ is unsatisfiable and $C_{v_1}(\varphi) \subset C_{v_2}(\varphi) \subset \dots \subset C_{v_{l-1}}(\varphi)$. Valuation $w = v_{l-1}$ identifies a minimal subset of satisfiable clauses c_i in φ (given by the complement of $C_{v_{l-1}}(\varphi)$) and it is thus a φ -minimal valuation.

Details concerning Section 4

Proof (of Lemma 1). By induction we show that for all i , $1 \leq i \leq n$, for all $\alpha \in C_i^+$, $\text{Formula}(\alpha)$ is true. This is sufficient as $f = \text{Formula}(\psi)$, and $\psi \in C_1^+$.

We start with the base case $i = n$. Let $\alpha \in C_n^+$, so $\text{Formula}(\alpha) = \exists X_n \cdot \alpha$. By definition of positive certificates, the valuation $w(\alpha) : X_n \rightarrow \{0, 1\}$ is such that $\alpha[w(\alpha)] = \text{true}$. So the valuation $w(\alpha)$ satisfies all the clauses of α . It is thus a witness that the formula $\exists X_n \cdot \alpha$ is true.

We now proceed with the induction case, under the hypothesis that for all for all $\alpha \in C_{i+1}^+$, $\text{Formula}(\alpha)$ is true. We consider two cases.

Suppose that $\text{Level}_i \subseteq S_{\exists}$, and let $\alpha \in C_i^+$. By definition of positive certificates, there exists $\beta \in C_{i+1}^+$ such that $\alpha' = \alpha[w(\alpha)]$ and $\alpha' \subseteq \beta$. So $\text{Formula}(\alpha') \sqsubseteq \text{Formula}(\beta)$. By induction hypothesis $\text{Formula}(\beta)$ is true and by Proposition 3, $\text{Formula}(\alpha')$ is also true. Therefore, as $Q_i = \exists$, it follows that $\text{Formula}(\alpha)$ is true.

Suppose that $\text{Level}_i \subseteq S_{\forall}$, and let $\alpha \in C_i^+$. By definition of positive certificates, for all valuations $w : X_i \rightarrow \{0, 1\}$, there exists $\beta \in C_{i+1}^+$ such that $\alpha[w] \subseteq \beta$, and so $\text{Formula}(\alpha[w]) \sqsubseteq \text{Formula}(\beta)$. By induction hypothesis, we know that $\text{Formula}(\beta)$ is true, and by Proposition 3, $\text{Formula}(\alpha[w])$ is also true. Since this holds for all valuations $w : X_i \rightarrow \{0, 1\}$ and $Q_i = \forall$, we have that $\text{Formula}(\alpha)$ is true.

Proof (of Lemma 2). We reason by induction on the number of blocks in formula f . Let i , $1 \leq i \leq n$, we show that $\langle (\{\alpha\}, \lceil W_{i+1} \rceil, \dots, \lceil W_n \rceil), w \rangle$ is a positive certificate for $\text{Formula}(\alpha)$, for all $\alpha \in \text{Level}_i$ treated by the algorithm and such that $\text{Formula}(\alpha)$ is true. In this certificate, $\lceil W_{i+1} \rceil, \dots, \lceil W_n \rceil$ are the current antichains when α has just been treated. Notice that at the end of the execution, the last treated node α is the root $\psi \in \text{Level}_1$ and $\lceil W_1 \rceil = \{\alpha\}$.

For the base case $i = n$, we have $\alpha \in \text{Level}_n$ and $\text{Formula}(\alpha) = \exists X_n \cdot \alpha$. If $\text{Formula}(\alpha)$ is true then there exists $w : X_n \rightarrow \{0, 1\}$ such that $w \models \alpha$. Let $w(\alpha) = w$ with w being the valuation constructed by Algorithm 1. Clearly $\langle (\{\alpha\}, w) \rangle$ is a positive certificate for $\text{Formula}(\alpha)$.

For the induction case, we assume that the property holds for level $i + 1$. Let $\alpha \in \text{Level}_i$ such that $\text{Formula}(\alpha)$ is true. Assume that the first quantifier of $\text{Formula}(\alpha)$ is existential, that is, $\text{Formula}(\alpha) = \exists X_i \cdot \forall X_{i+1} \cdot \dots \exists X_n \cdot \alpha$ (the proof is similar for the universal case). As $\text{Formula}(\alpha)$ is true, there exists a valuation $w : X_i \rightarrow \{0, 1\}$ such that $\text{Formula}(\alpha[w])$ is true, and in particular there exists a maximal valuation w with this property. Algorithm 1 considers

such a maximal valuation w and makes a recursive call to Algorithm 2 on node $\alpha[w]$. By induction hypothesis, the algorithm will construct a positive certificate $\langle (\{\alpha[w]\}, \lceil W_{i+2} \rceil, \dots, \lceil W_n \rceil), \mathbf{w} \rangle$ for $\text{Formula}(\alpha[w])$. As $\alpha[w]$ is not necessarily a maximal winning node in Level_{i+1} , this node may have been replaced by another winning node β in $\lceil W_{i+1} \rceil$ such that $\alpha[w] \subseteq \beta$, this is compatible with Property 3 of the definition of positive certificates. So, $\langle (\{\alpha\}, \lceil W_{i+1} \rceil, \dots, \lceil W_n \rceil), \mathbf{w}' \rangle$ where \mathbf{w}' is extending \mathbf{w} for α by $\mathbf{w}'(\alpha) = w$, is a positive certificate for $\text{Formula}(\alpha)$.

Negative certificates are defined as follows and they enjoy properties similar to those for positive certificates. A *negative certificate* for a QBF formula $f = \text{Q}_1 X_1 \cdots \text{Q}_n X_n \cdot \psi$ is a pair $\langle (C_i^-)_{1 \leq i \leq n}, \mathbf{w} \rangle$ such that:

- each C_i^- is a set of nodes such that $C_i^- \subseteq \text{Level}_i$
- for each i such that $\text{Level}_i \subseteq S_\forall$, \mathbf{w} is a function that assigns a valuation $\mathbf{w}(\alpha) : X_i \rightarrow \{0, 1\}$ to each $\alpha \in C_i^-$
- and the following properties are verified:
 1. $C_1^- = \{\psi'\}$ with $\psi' \subseteq \psi$
 2. for each $i < n$ such that $\text{Level}_i \subseteq S_\forall$, for all $\alpha \in C_i^-$, there exists $\beta \in C_{i+1}^-$ such that $\alpha[\mathbf{w}(\alpha)] \supseteq \beta$
 3. for each $i < n$ such that $\text{Level}_i \subseteq S_\exists$, for all $\alpha \in C_i^-$, for all $w : X_i \rightarrow \{0, 1\}$, there exists $\beta \in C_{i+1}^-$ such that $\alpha[w] \supseteq \beta$
 4. for $i = n$, for all $\alpha \in C_i^-$, for all $w : X_i \rightarrow \{0, 1\}$, $\alpha[w] = \text{false}$.

Example 3. Consider the running example of Fig. 1. As f is false, the following negative certificate $\langle (C_i^-)_{1 \leq i \leq 4}, \mathbf{w} \rangle$ is built by the execution of Algorithms 1 and 2 (see also Example 2):

$$\begin{aligned}
C_1^- &= \lceil L_1 \rceil = \{\psi\}, & \mathbf{w}(\psi) &= 101, \\
C_2^- &= \lceil L_2 \rceil = \{\{2, 4, 5\}\}, \\
C_3^- &= \lceil L_3 \rceil = \{\{2\}, \{5\}\}, & \mathbf{w}(\{2\}) &= 0, \mathbf{w}(\{5\}) = 0 \text{ or } 1, \\
C_4^- &= \lceil L_4 \rceil = \{\{2\}\}.
\end{aligned}$$

Details concerning Subsection 5.1

Improving ATCDepthFirst \exists . When considering all the maximal valuations $w : X_i \rightarrow \{0, 1\}$, we observe that player P_\exists is winning as soon as he can find a valuation w such that $\varphi[w] \subseteq \alpha$ for some $\alpha \in \lceil W_{i+1} \rceil$ (see Corollary 1). So, it is wise for P_\exists to first try to find such a valuation w . This approach is followed by the new algorithm (line 7). Procedure $\text{TestWinVal}(\varphi, \lceil W_i \rceil)$ tests, using a SAT solver, if player P_\exists can choose a valuation w such that $\varphi[w] \subseteq \alpha$ for some $\alpha \in \lceil W_i \rceil$. In that case it returns (true, w) , otherwise (false, \cdot) .

Now, suppose that player P_\exists cannot win by exploiting the elements of $\lceil W_{i+1} \rceil$. Then he should avoid considering (maximal) valuations w such that $\alpha \subseteq \varphi[w]$ for some $\alpha \in \lceil L_{i+1} \rceil$ as $\varphi[w]$ is losing (see Corollary 1). In this aim, he must ensure to only consider valuations w that satisfy at least one clause of each $\alpha \in \lceil L_{i+1} \rceil$. Procedure $\text{MaxVal}(\varphi, \lceil L_{i+1} \rceil)$ checks in line 11, using a SAT solver, if such a valuation w exists. If yes it returns (true, w) , otherwise (false, \cdot) . Assume that a valuation w generated by Procedure MaxVal leads to line 17 of the algorithm, this

Algorithm 3 ATCDepthImproved $\exists(\varphi, i)$

Require: node $\varphi \in S_{\exists} \cap \text{Level}_i, i \leq n$.**Ensure:** Win if $\varphi \in W_i$, Lose if $\varphi \in L_i$.

```
1: if  $\neg \text{IsSat}(\varphi)$  then
2:   Add( $\varphi, \lfloor L_i \rfloor$ )
3:   return Lose
4: if  $i = n$  then
5:   Add( $\varphi, \lceil W_i \rceil$ )
6:   return Win
7: (ExistVal,  $w$ )  $\leftarrow$  TestWinVal( $\varphi, \lceil W_{i+1} \rceil$ )
8: if ExistVal then
9:   Add( $\varphi, \lceil W_i \rceil$ )
10:  return Win
11: (ExistVal,  $w$ )  $\leftarrow$  MaxVal( $\varphi, \lfloor L_{i+1} \rfloor$ )
12: while ExistVal do
13:   F  $\leftarrow$  ATCDepthImproved $\forall(\varphi[w], i + 1)$ 
14:   if F = Win then
15:     Add( $\varphi, \lceil W_i \rceil$ )
16:     return Win
17:   (ExistVal,  $w$ )  $\leftarrow$  MaxVal( $\varphi, \lfloor L_{i+1} \rfloor$ )
18: Add( $\varphi, \lfloor L_i \rfloor$ )
19: return Lose
```

means that $\varphi[w]$ is losing and has been added to $\lfloor L_{i+1} \rfloor$. Thus a call to MaxVal in line 17 generates a new maximal valuation (if it exists) which is incomparable with w as it avoids $\lfloor L_{i+1} \rfloor$.

The rest of the new algorithm is the same as in Algorithm 1.

Details concerning Section 6

Table 2 presents the experimental results obtained for instances of the families k-path-n, with a timeout of 300 seconds.

Instance	Var	Cl	Blocs	Nodes	SATc	ATC	QuBE	sKizzo	DepQBF
k-path-n-01	108	275	7	18	461	0.77	0.01	0.01	0.01
k-path-n-02	180	481	9	62	3698	4.41	0.03	0.08	0.05
k-path-n-05	384	1051	15	137	6481	7.95	4.54	0.11	46.91
k-path-n-06	456	1257	17	160	8890	11.9	13.57	0.29	/
k-path-n-10	732	2033	25	299	17797	31.99	/	44.31	/
k-path-n-11	804	2234	27	302	15307	28.5	/	/	/
k-path-n-20	1428	3992	45	571	36397	90.12	/	/	/
k-path-n-21	1488	4155	47	551	31808	99.76	/	/	/

Table 2. Instances of k-path-n.