SAT based BDD solver for Quantified Boolean Formulas

Gilles Audemard and Lakhdar Saïs*
CRIL CNRS – Université d'Artois
rue Jean Souvraz SP-18
F-62307 Lens Cedex France
{audemard,sais}@cril.univ-artois.fr

Abstract

Solving Quantified Boolean Formulas (QBF) has become an attractive research area in Artificial intelligence. Many important artificial intelligence problems (planning, non monotonic reasoning, formal verification, etc.) can be reduced to QBFs. In this paper, a new DLL-based method is proposed that integrates binary decision diagram (BDD) to set free the variable ordering heuristics that are traditionally constrained by the static order of the QBF quantifiers. BDD is used to represent in a compact form the set of models of the boolean formula. Interesting reduction operators are proposed in order to dynamically reduce the BDD size and to answer the validity of the QBF. Experimental results on instances from the QBF'03 evaluation show that our approach can efficiently solve instances that are very hard for current QBF solvers.

Keywords: *Quantified boolean formula, Binary decision diagram, Satisfiability.*

1. Introduction

Solving Quantified Boolean Formulas has become an attractive and important research area over the last years. Such increasing interest might be related to different factors including the fact that many important artificial intelligence problems (planning, non monotonic reasoning, formal verification, etc.) can be reduced to QBFs which is considered as the canonical problem of the PSPACE complexity class. Another important reason comes from the recent impressive progress in the practical resolution of the satisfiability problem.

Many solvers for QBFs have been proposed recently (e.g. [10, 18, 13, 9]), most of them are obtained by extending satisfiability results. This is not surprising since QBFs is a natural extension of SAT where the boolean variables are universally or existentially quantified. Most of these solvers consider an input formula in the prenex clausal form and are variant of Davis Logemann and Loveland procedures (DLL) [7]. However, one of the main drawback of such proposed approaches is that variables are instantiated according to the quantifier order. Such preset ordering limits the efficiency of the obtained solver. Indeed, the size of the search tree depends heavily on the order in which the variables are instantiated during search.

The main goal of this paper is to set free the solver from the preset ordering of the QBF and to facilitate the extension of useful and efficient satisfiability results. To this end, we propose an extension of binary decision diagram and use it to represent in a compact form the set of models of the boolean formula obtained by the satisfiability solver. New reduction operators are proposed to reduce at least to some extent the size of the constructed binary decision diagram and to answer the validity of the QBF. Then, it is integrated in a DLL like techniques providing a new QBF solver named QBFBDD.

The paper is organized as follows. After some preliminaries and technical background on quantified boolean formulas and binary decision diagram, it is shown how binary decision diagram can be naturally integrated with DLL-like techniques to handle QBFs. Experiments on instances of the last QBFs evaluation are presented and show that our approach is very promising since it solves for the first time many hard QBF instances.

2. Preliminaries and technical background

Before introducing our approach, we briefly review some necessary definitions and notation about quantified boolean formulas and binary decision diagram.

* This work has been supported in part by the IUT de Lens, the CNRS and the Region Nord/Pas-de-Calais under the TAC Programme

2.1. Quantified boolean formulas

Let $\mathcal{P}$ be a finite set of propositional variables. Then, $\mathcal{L}_{\mathcal{P}}$ is the language of quantified boolean formulas built over $\mathcal{P}$ using ordinary boolean formulas (including propositional constants $\top$ and $\perp$) plus the additional quantification ($\exists$ and $\forall$) over propositional variables.
We consider quantified boolean formula in the prenex form: $\Phi = Q_kX_k, \ldots, Q_1X_1\Psi$ (in short $QX\Psi$, QX is called the prefix of Φ and Ψ the matrix of Φ) where $Q_i \in \{\exists, \forall\}$, $X_k, \ldots, X_1$ are disjoint sets of variables and Ψ a boolean formula. Consecutive variables with the same quantifier are grouped. The rank of a variable $x \in X_i$ is equal to i (noted $rank(x)$). Variables in the same quantifier group have the same rank value. We define a prefix ordering of QBF formula $\Phi = Q_kX_k, \ldots, Q_1X_1\Psi$ as the partial ordering obtained according to the decreasing rank of the variables, noted $X_k < X_{k-1} < \cdots < X_1$. A QBF formula Φ is said to be in clausal form if Φ is in prenex form and Ψ is in Conjunctive Normal Form (CNF). Note that we can consider QBFs with inner quantifier Q_1 as existential. Indeed, when Q_1 is a universal quantifier, suppressing $\forall X_1$ from the prefix and all occurrences of $x \in X_1$ from the matrix lead to an equivalent QBF. We define $Var(\Phi) = \bigcup_{i\in\{1,\ldots,k\}} X_i$ the set of variables of Φ. A literal is the occurrence of propositional variable in either positive (l) or negative form ($\neg l$). $Lit(\Phi) = \bigcup_{i\in\{1,\ldots,k\}} Lit(X_i)$ the set of complete literals of Φ, where $Lit(X_i) = \{x_i, \neg x_i | x_i \in X_i\}$. We note $var(l)$ the variable associated to a literal l. A literal $l \in \Phi$ is a unit literal iff l is existentially quantified and $\exists c = \{l, l_1, \ldots, l_i\} \in \Psi$ s.t. $\forall l_j, 1 \leq j \leq i$, $var(l_j)$ is universally quantified and $rank(l_j) < rank(l)$. A monotone literal is defined in the usual way as in the pure boolean case (i.e. l is monotone in Φ iff it appears either positively or negatively).

To define the semantic of quantified boolean formulas, let us introduce some necessary notations. Let S be the set of assignments over the set of variables V. The Up-projection (resp. Down-projection) of a set of assignments S on a set of variables $X \subset V$, denoted $S \uparrow X$ (resp. $S \downarrow X$), is obtained by restricting each assignment to literals in X (resp. in $V \backslash X$). The set of all possible assignments over X is denoted by 2^X. An assignment over X is denoted by a vector of literals $\overrightarrow{x}$. In the same way, Up-projection and Down-projection also apply on vector of literals $\overrightarrow{x}$. If $\overrightarrow{y}$ is an assignment over Y s.t. $Y \cap X = \emptyset$, then $\overrightarrow{y}.S$ denotes the set of interpretations obtained by concatenating $\overrightarrow{y}$ with each interpretation of S. Finally, $\Psi(\overrightarrow{x})$ denotes the boolean formula Ψ simplified with the partial assignment $\overrightarrow{x}$.
A QBF formula is valid (is true) if there exists a solution (called a total policy) defined as follows. It is a simplified version of the definition by Sylvie Coste-Marquis *et al.* [14].

Definition 1 *Let $\Phi = Q_kX_k, \ldots, Q_1X_1\Psi$ a quantified boolean formula and $\pi = \{\overrightarrow{x}_1, \ldots, \overrightarrow{x}_n\}$ a set of models of the boolean formula Ψ. π is a total policy of the quantified boolean formula Φ iff π recursively verifies the following conditions:*

1. *$k = 0$, and $\Psi = \top$*
2. *if $Q_k = \forall$, then $\pi \uparrow X_k = 2^{X_k}$, and $\forall \overrightarrow{x}_k \in 2^{X_k}$, $\pi \downarrow \overrightarrow{x}_k$ is a total policy of $Q_{k-1}X_{k-1}, \ldots, Q_1X_1\Psi(\overrightarrow{x}_k)$*
3. *if $Q_k = \exists$, then $\pi \uparrow X_k = \{\overrightarrow{x}_k\}$ and $\pi \downarrow \overrightarrow{x}_k$ is a total policy of $Q_{k-1}X_{k-1}, \ldots, Q_1X_1\Psi(\overrightarrow{x}_k)$*

Remark 1 *Let π be a total policy of $\Phi = Q_kX_k, \ldots, Q_1X_1\Psi$. If $Q_k = \forall$ then we can rewrite π as $\bigcup_{\overrightarrow{x}_k \in 2^{X_k}} \{\overrightarrow{x}_k.(\pi \downarrow \overrightarrow{x}_k)\}$ and if $Q_k = \exists$, then $\pi \uparrow X_k = \{\overrightarrow{x}_k\}$ and π can be rewritten as $\{\overrightarrow{x}_k.(\pi \downarrow \overrightarrow{x}_k)\}$*

Example 1 *Let $\Phi = \exists x_5x_6\forall x_2x_4\exists x_1x_3\Psi$ be a QBF formula, where $\Psi = (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_4 \vee x_3) \wedge (x_5 \vee x_6)$. Φ is a valid QBF, since the set of models $\pi = \{(\neg x_5, x_6, x_2, x_4, \neg x_1, x_3), (\neg x_5, x_6, x_2, \neg x_4, \neg x_1, x_3), (\neg x_5, x_6, \neg x_2, \neg x_4, x_1, \neg x_3), (\neg x_5, x_6, \neg x_2, x_4, x_1, x_3)\}$ is a total policy of Φ (Figure 1). The different projection operations are illustrated as follow :*

$\pi \uparrow \{x_5, x_6\} = \{(\neg x_5, x_6)\}$

$$\pi' = \pi \downarrow \{x_5, x_6\} = \{(x_2, x_4, \neg x_1, x_3), (x_2, \neg x_4, \neg x_1, x_3), (\neg x_2, \neg x_4, x_1, \neg x_3), (\neg x_2, x_4, x_1, x_3)\}$$

$$\pi' \uparrow \{x_2, x_4\} = \{(\neg x_2, \neg x_4), (\neg x_2, x_4), (x_2, \neg x_4), (x_2, x_4)\} = 2^{\{x_2, x_4\}}$$

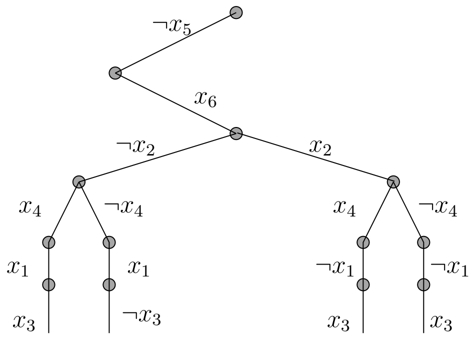


Figure 1. Policy decision tree representation (example 1)

Motivated by the impressive results obtained in practical solving of the satisfiability problem, several QBF solvers have been developed recently. Most of them are extensions of the well known DLL procedure including many effective

SAT results such as learning, heuristics and constraint propagation (QUBE [10], QUAFFLE [18], EVALUATE [5], DECIDE[16]). For examples, QUBE [10] and QUAFFLE [18] extend backjumping and learning techniques, EVAUATE[5] and DECIDE [16] extend some SAT pruning techniques such as unit propagation.

Algorithm 1 gives a general scheme of a basic DLL procedure for checking the validity of QBFs. It takes as input variables of the QBF prefix $< X_k, \ldots, X_1 >$ associated to quantifiers $Q_k, \ldots, Q_1$ and a matrix Ψ in clausal form. It returns true if the QBF formula $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$ is valid and false otherwise. The algorithm starts by simplifying the formula by propagating unit and monotone literals. Then, if the current simplified matrix contains the empty clause then the current QBF is not valid (value false is returned); otherwise, if the current matrix is empty then the QBF formula is valid (value true is returned). The next step consists in choosing the next variable to instanciate (splitting rule) in the most external non empty set of variables X_k. This differ from the DLL satisfiability version, since variables are instantiated according to their prefix ordering. Depending on the quantifier Q_k of the chosen variable, left and/or right branchs are generated. If $Q_k = \forall$ (resp. $Q_k = \exists$) the right branch is generated only if the value returned in left branch is true (resp. false). The search tree developped by the QBF DLL procedure is usually called an and/or tree search.

Algorithm 1: DLL for QBF

Data : Ψ : set of clauses;
$< X_k, \ldots, X_1 >$ the prefix set of the QBF
Result : true if the QBF is valid, false otherwise
begin
Simplify(Ψ);
if $\emptyset \in \Psi$ **then return** false;
if $\Psi = \emptyset$ **then return** true;
if $X_k = \emptyset$ **then return**
QDLL($\Psi, < X_{k-1}, \ldots, X_1 >$);
choose (by heuristic) a literal $l \in X_k$;
if *(($Q_k = \forall$) and*
QDLL($\Psi \cup \{l\}, < X_k - \{l\}, \ldots, X_1 >$)=false)
then
return false;
if *(($Q_k = \exists$) and*
QDLL($\Psi \cup \{l\}, < X_k - \{l\}, \ldots, X_1 >$)=true)
then
return true;
return
QDLL($\Psi \cup \{\neg l\}, < X_k - \{l\}, \ldots, X_1 >$);
end

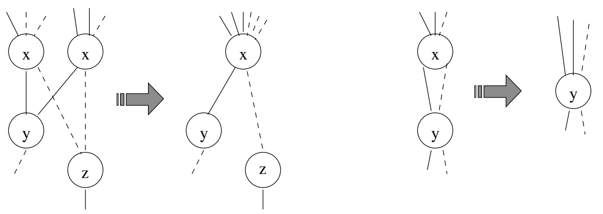
One of the major drawback of the extension of the DLL procedures to QBF concerns the imposed prefix ordering. Such restrictive ordering might lead to performance degradation of the QBF solver. Since, good ordering may be lost. Furthermore, such limitation makes difficult the extension of certain interesting results obtained on the satisfiability problem (see for example [8] for random problem or [15] for structured problems).

2.2. Binary decision diagram

A Binary Decision Diagram (BDD) [1, 3] is a rooted directed acyclic graph with two terminal nodes that are referred to as the 0-terminal and the 1-terminal. Every non-terminal node is associated with a primary input variable such that it has two outgoing edges called the 0-edge corresponding to assigning the variable a false truth value, and the 1-edge corresponding to assigning the variable a true truth value. An Ordered Binary Decision Diagram (OBDD) is a BDD such that the input variables appear in a fixed order on all the paths of the graph, and no variable appears more than once in the path. A Reduced Ordered BDD (ROBDD) is an OBDD that results from the repeated application of the following two rules:

1. Share all equivalent sub-graphs (Figure 2.a).
2. Eliminate all redundant nodes whose outgoing edges point to the same node (Figure 2.b).

A (RO)BDD representing a boolean formula Ψ is noted $(RO)BDD(\Psi)$.



a. Sharing sub-formulas b. Redundant node

Figure 2. Reduction Rules

Figure 3 illustrates the ROBDD representation of the policy π ($ROBDD(\pi)$) shown in example 1 where the solid edges denote the 1-edges and the dashed edges denote the 0-edges. The ROBDD order is the same as the prefix ordering of variables ($\{x_5, x_6\} < \{x_2, x_4\} < \{x_1, x_3\}$) of the QBF Φ.

ROBDDs have some interesting properties. They provide compact and unique representations of boolean functions, and there are efficient algorithms performing all kinds of logical operations on ROBBDs. For example, it is possible to check in constant time whether an ROBDD is true or

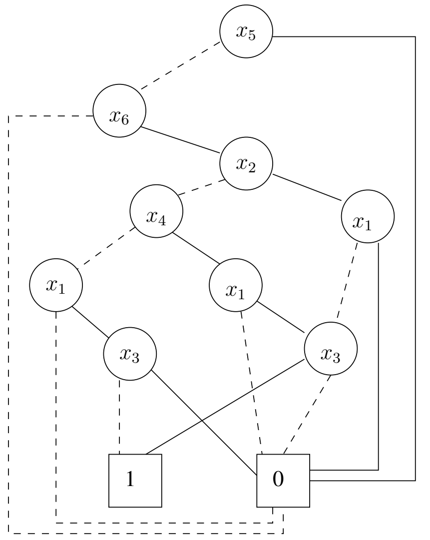


Figure 3. BDD representation of a policy (example 1)

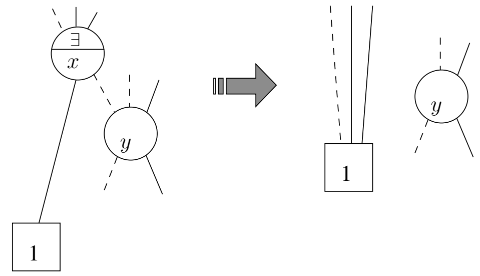


Figure 4. Existential Reduction

false. Let us recall that for boolean formula, such problem is NP-complete. Despite the exponential size of the ROBDD in the worst case, ROBDD is one of the most used data structure in practice.

In the rest of this paper, only reduced ordered BDDs are considered and for short we denote them as BDDs. When Quantified boolean formulas are considered, the order used by the ROBDD follows the prefix ordering of the QBF.

3. Solving QBF : a DLL + BDD approach

In this section, to make the QBFs solver freed from the preset ordering of the variables (i.e. fixed by the QBF prefix), we propose to combine classical SAT solver with binary decision diagram. More precisely, to check the validity of a QBF $\Phi = QX\Psi$, our approach makes use of a DLL-like technique to search for all models of the boolean formula Ψ. Such models are encoded by a BDD according to the prefix variables ordering. To reduce the BDD size and to answer the validity of the QBF, additional new reduction operator is given in Figure 4. When, a node x is existentially quantified and one of its child nodes is the 1-terminal node, any reference to the node x is simply replaced by a reference to its 1-terminal node. We call such reduction operation *existential reduction*. Interestingly enough, when x is universally quantified and its two child nodes are 1-terminal, such node is eliminated using the classical BDD node reduction (Figure 2.b).

In short, BDD is used to represent a set of models (following the prefix variables ordering of the QBF formula). During the BDD construction process, classical reduction operation and existential reduction are applied. If set of models represents a total policy of the QBF formula then the BDD built from such models is reduced to a 1-terminal node as stated by the following property :

Property 1 *Let $\Phi = Q_kX_k, \ldots, Q_1X_1\Psi$ be a QBF formula and $\pi = \{\overrightarrow{x}_1, \overrightarrow{x}_2, \ldots, \overrightarrow{x}_n\}$ a set of models of Ψ. If π is a total policy of Φ then the BDD(π) is reduced to the 1-terminal node.*

Proof: The proof is obtained by induction on k. For $k = 0$, the BDD representing the constant $\top$ is a 1-terminal node (by definition of a total policy). Suppose that the property holds for $k-1$, let us prove that it holds for k. By definition of a total policy two case are considered :

1. if $Q_k = \forall$, then $\pi \uparrow X_k = 2^{X_k}$, and $\forall \overrightarrow{x}_k \in 2^{X_k}$, $\pi \downarrow \overrightarrow{x}_k$ is a total policy of the QBF formula $Q_{k-1}X_{k-1}, \ldots, Q_1X_1\Psi(\overrightarrow{x}_k)$. By induction hypothesis, we can deduce that BDD($\pi \downarrow \overrightarrow{x}_k$) is reduced to 1-terminal node. Consequently all the leaf of the BDD(2^{X_k}) are 1-terminal nodes. Then by repeatedly applying the Redundant node rule on such a BDD, we obtain a BDD reduced to a 1-terminal node.
2. if $Q_k = \exists$, then $\pi \uparrow \overrightarrow{X}_k = \{\overrightarrow{x}_k\}$ and $\pi \downarrow \overrightarrow{x}_k$ is a total policy of the QBF formula $Q_{k-1}X_{k-1}, \ldots, Q_1X_1\Psi(\overrightarrow{x}_k)$. By induction hypothesis BDD($\pi \downarrow \overrightarrow{x}_k$) is a 1-terminal node. Consequently, the BDD($\{\overrightarrow{x}_k\}$) can be seen as a branch ended on a 1-terminal node. Repeatedly applying the existential reduction rule, the BDD(π) is reduced to a 1-terminal node.

The following example shows the dynamic reduction of the BDD associated with the set of models representing the total policy of example 1.

Example 2 *Let Φ be the QBF formula of the example 1 and π its associated policy.*

- *The figure 5 represents the reduction phase of the BDD(π) representation.*

- *The figure 5.a is a BDD representation of the policy π (for clarity reason, only paths with final 1-terminal node are represented).*
- *The existential reduction rule allows the x_3 elimination (figure 5.b) and the x_1 elimination (figure 5.c).*
- *The redundant node reduction rule allows the x_4 elimination (figure 5.d) and the x_2 elimination (figure 5.e).*
- *Finally the existential reduction rule suppresses x_5 and x_6 and the bdd of the policy is restricted to the 1-terminal node representing the true formula (figure 5.f) and proving that π is a total policy.*

The algorithm 2 represents the QBF solver obtained by a combination of DLL procedure with BDD. As we can see, the algorithm search for all models until the BDD representation (characterized by the global variable bdd initialized to the 0-terminal node) is reduced to a 1-terminal node, in that case the algorithm terminates and answers the validity of the QBF formula. If the algorithm backtrack to level 0, then the QBF formula is not valid. In other case search continues (back is returned). The function $Simplify()$ enforces the well known unit propagation process. While the function $conflictAnalysis()$ implements learning scheme used by the most efficient satisfiability solvers. Finally, $modelAnalysis()$ is a new function that is called when a model $\overrightarrow{x}$ is found. It allows us to avoid repeated search. More precisely, if $\overrightarrow{x}$ is model of Ψ, a nogood $\neg(\overrightarrow{x} \downarrow X_1)$ is added to the formula, since the variables X_1 of the inner quantifier group are existential. Example 3 gives a possible trace of QBFBDD algorithm.

Example 3 *Let us consider the formula Φ of example 1. Suppose the algorithm* QBFBDD *starts the search by assigning x_1 and x_5. At this step, one has to satisfy the clause $(\neg x_4 \vee x_3)$. Suppose, now, the choice is done on the variable $\neg x_4$. A first model $(x_5, \neg x_4, x_1)$) is found and is added to the bdd. This variable is reduced to a single path to the 1-terminal node : $< x_5, \neg x_4 >$ (variable x_1 is deleted by existential reduction). The clause $(\neg x_5 \vee x_4)$ is added to the formula Ψ. A backtrack is done to search for other models and the assignment of x_4 implies x_3 using unit propagation. A second model (x_5, x_4, x_1, x_3) is added to the variable* `bdd` *which becomes reduced to the 1-terminal node by using existential and redundant reduction. So, search ends and returns the validity of the formula Φ.*

Algorithm 2: Combining BDD and DLL : QBFBDD

Data : Ψ : set of clauses;
X=$\{X_k, \ldots, X_1\}$ the prefix set of the QBF;
I a partial interpretation;
d level in the search tree, initially set to 0

Result : $valid$ if the QBF is valid, $unvalid$ otherwise;
$back$ is returned to continue the search for other models.

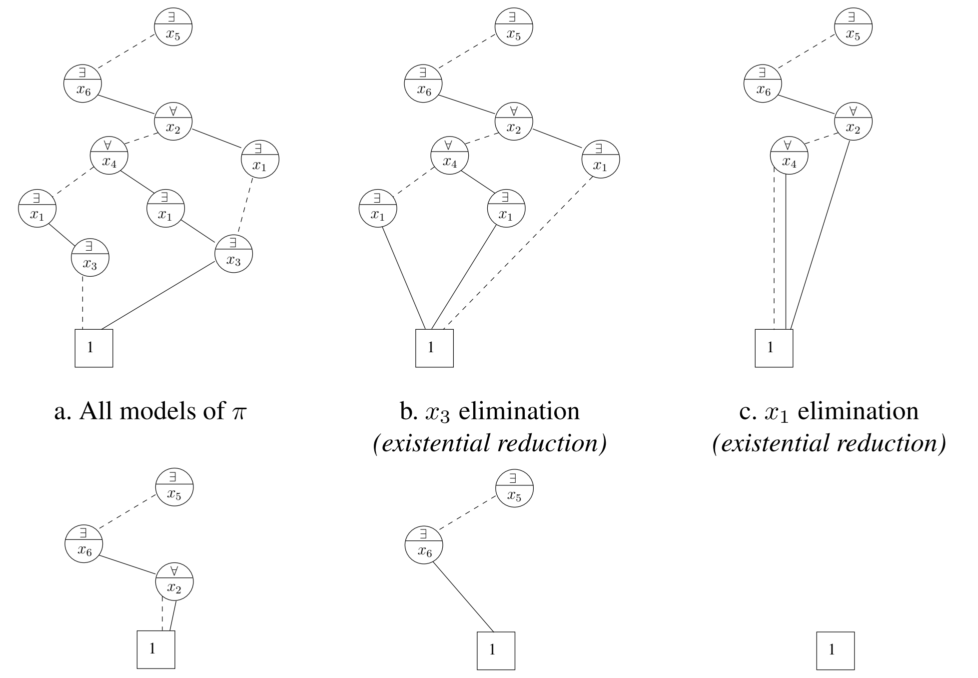
begin
  Simplify(Ψ);
  if $\emptyset \in \Psi$ **then**
    conflictAnalysis();
    return $back$;
  if $\Psi = \emptyset$ **then**
    bdd := or(bdd, I);
    if *equal(bdd,1-terminal)* **then return** $valid$;
    modelAnalysis();
    return $back$;
  Let $l \in X_i$ $(i \in \{1 \ldots k\})$ be the chosen branching variable;
  if
  (QDLL($\Psi \cup \{l\}, X - \{l\}, I \cup \{l\}, d+1$*)=*$valid$*)*
  or QDLL($\Psi \cup \{\neg l\}, X - \{l\}, I \cup \{\neg l\}, d+1$*)=*$valid$*)* **then**
    return $valid$;
  if *(*$d = 0$*)* **then**
    return $unvalid$;
  return $back$;
end

---

4. Empirical Evaluation

The experimental results reported in this section where obtained on a Pentium IV 3 GHz with 1GB RAM, and performed on a large panel of the QBF'03 evaluation instances [2]. These instances are divided into three categories according to their hardness. The first one contains *easy* instances, the second, instances solved by only a few number of solvers (*medium*) and the last one, contains the open *hard* instances. We compare our solver QBFBDD to state of the art QBF solver QUBE [10] and OPENQBF. Since QBFBDD can choose variables using any ordering, we have implemented a variant of *Boehm* heuristic [4] which selects in priority existential variables. Let us note that QBFBDD is a basic preliminary version implemented in Java. For each instance, the cpu time (in seconds) is limited to 600 seconds.

Table 1 summarizes the results obtained on the QBF'03 evaluation instances, restricted to the real world instances. The time column represents the sum of cpu time needed to solve all problems (600 seconds is added when the problem is not solved during the allowed time limit). First of all, QUBE one of the state of the art solver is able to solve all easy instances, a majority of medium ones, but none of the open and hard instances. The solver OPENQBF solves all easy instances in the allowed time limits, only 36% of the medium ones and zero instances of the hard category. Our proposed QBF solver QBFBDD is less competitive than QUBE and OPENQBF on easy and medium instances. The most interesting result is that QBFBDD is the most efficient on the hard category, where the instances are solved



a. All models of π

b. x_3 elimination *(existential reduction)*

c. x_1 elimination *(existential reduction)*

d. x_4 elimination *(redundant node reduction)*

e. x_2 elimination *(redundant node reduction)*

f. x_5, x_6 elimination *(existential reduction)*

Figure 5. BDD reduction of a QBF total policy

for the first time. Such results are confirmed on hard robots instances (see. table 2).

		QUBE		OPENQBF		QBFBDD	
Category	nb	solved	time	solved	time	solved	time
Easy	178	178	46	178	1 062	60	73 232
Medium	331	328	5 830	276	37 215	121	129 221
Hard	135	0	81 000	0	81 000	**12**	**74 739**

Table 1. From easy to hard QBFs

Table 2 highlights cpu time and number of nodes of the search tree for different instances of the QBF'03 evaluation. Each line contains problem name, number of different quantifier groups (#Q), number of variables (#V), number of clauses (#C), the validity of the formula (V ?), the cpu time (in seconds) and the number of nodes of the search tree for the three different solvers (nodes are not available with QUBE). In general, QUBE is the best and the fastest solver. Our method QBFBDD is very competitive with respect to some QBFs. Indeed, the solver QBFBDD solves all robot instances encoding the robot navigation problem [6] whereas the other solvers fail on some of them. Note that 12 instances are solved for the first time in a reasonable amount of time using QBFBDD. Interestingly enough, QBFBDD is not restricted to particular kind of QBF formulas. For example, it solves quite easily the *impl08* instance with 17 different quantifier groups and *lognBWLARGEA1* instance with a great number of clauses. We have remarked that the OPENQBF solver make use of a monotone literal propagation rule that seems to be very effective with respect to some instances, since without such a rule, OPENQBF fails to solve them in the allowed amount of time.

As a summary, our approach outperform the other solvers on hard QBF instances, and it is competitive on easy instances.

5. Future Work

Use of binary decision diagram to solve quantified boolean formula is very promising and opens interesting perspectives. We particularly plan to investigate the following :

- Binary decision diagram is recognized as one of the most efficient data structures for representing boolean formula. In this paper, new BDD reduction operators have been proposed, showing that one can answer efficiently if a set of models is total policy of a quantified boolean formula. Consequently, the design of incomplete QBF solver is a very promising path of re-

					QUBE		OPENQBF		QBFBDD	
problem	#Q	#V	#C	V ?	time	nodes	time	nodes	time	nodes
robots_1_5_2_1.2	2	2964	8640	Y	0.13	?	2.6	233	0.8	288
robots_1_5_5_3.3	2	3 952	12277	Y	0.75	?	25	1 293	1.5	433
robots_1_5_2_1.3	2	3952	12274	Y	0.9	-	49	3 866	1.5	435
robots_1_5_2_2.7	2	7 904	26 810	Y	-	-	-	-	**9**	**1 129**
robots_1_5_3_1.10	2	10 868	37 713	Y	-	-	-	-	**177**	**2 455**
robots_1_5_2_1.9	2	9880	34078	Y	-	-	-	-	**45**	**1800**
term1.blif_0.10_0.20_0_1_inp_exact	23	1116	3763	Y	0.53	?	4.6	12544	0.7	1198
comp.blif_0.10_0.20_0_1_inp_exact	7	311	833	Y	0.3	?	146	304243	2.5	12815
flip-flop-5-c	3	3279	3630	N	0.1	?	0.2	4	-	-
adder-2-unsat	3	334	114	N	0.1	?	5	21 552	423	192 763
impl08	17	34	66	Y	0.01	?	0.1	103	18	7 719
lognBWLARGEA1	3	1099	62820	N	5	?	-	-	10	1 181
toilet_g_10_01.2	3	54	190	Y	0.01	?	0.01	19	-	-
z4ml.blif_0.10_1.00_0_1_out_exact	3	64	196	Y	0.01	?	0.1	63	0.1	10

Table 2. Results on real-world QBFs

search. Indeed, one can use Local search techniques that have been shown very efficient in finding models(e.g. [17, 12]) and binary decision diagram to represent them and to answer the validity or the non validity of quantified boolean formulas.

- Our implementation of QBFBDD is a straightforward one that do not use efficient coding tricks and extra features leading to greater efficiency. Clearly, integrating the most efficient satisfiability solver (e.g. Zchaff[15], Berkmin51 [11]) could lead to significant improvements of the QBFBDD solver. As our approach is freed from the prefix ordering of the QBF, such extension could be done in an obvious way.
- Another interesting path for future research concerns the design of QBF solver that is only BDD-based. In other word, one can construct the BDD associated to the boolean matrix of the QBF instead of representing the models found by a DLL-like techniques. Then, an experimental comparison of the two approaches on a large set of QBFs is necessary.

6. Conclusion

In this paper, a new QBF solver made of a combination of the well known DLL procedure with binary decision diagram is presented. Reduction operations that prevent to some extent the blowup of the BDD size and to answer the validity of the QBF. The main advantage of our approach is that it is freed from any ordering of variables. This facilitates the extension of satisfiability solver to solve quantified boolean formulas. Experimental results on instances from the QBFs competition show the effectiveness of our approach. More interestingly, some open hard QBF instances are solved for the first time.

References

[1] S. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 27(6):509–516, 1978.

[2] D. L. Berre, L. Simon, and A. Tachella. Challenges in the qbf arena: the sat'03 evaluation of qbf solvers. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT2003)*, volume 2919 of *LNAI*, pages 452–467, 2003.

[3] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 8:677–692, C-35.

[4] M. Buro and H. K. Büning. Report on a SAT competition. *Bulletin of the European Association for Theoretical Computer Science*, 49:143–151, 1993.

[5] M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified boolean formulae. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, pages 262–267, Madison (Wisconsin - USA), 1998.

[6] C. Castellini, E. Giunchiglia, and A. Tachella. Sat based planning in complex domains : Concurency, constraints and non determinism. *Artificial Intelligence*, 147(1):85–117, 2003.

[7] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, July 1962.

[8] O. Dubois and G. Dequen. A backbone-search heuristic for efficient solving of hard 3–sat formulae. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI–01)*, Aug. 4–10 2001.

[9] I. P. Gent, H. H. Hoos, A. G. D. Rowley, and K. Smyth. Using stochastic local search to solve quantified boolean formulae. In *Proceedings of the 9th international conference of principles and practice of constraint programming*, volume 2833 of *LNCS*, pages 348–362, 2003.

[10] E. Giunchiglia, M. Narizzano, and A. Tacchella. QuBE : A system for deciding Quantified Boolean Formulas Satisfiability. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR'01)*, Siena, Italy, June 2001.

[11] E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT-solver. In *Proceedings of International Conference on Design, Automation, and Test in Europe (DATE '02)*, pages 142–149, 2002.

[12] H. H. Hoos and T. Stützle. Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481, 2000.

[13] R. Letz. Lemma and model caching in decision procedures for quantified boolean formulas. In *Proceedings of Tableaux 2002*, pages 160–175, Copenhagen, Denmark, 2002.

[14] S. C. Marquis, H. Fargier, J. Lang, D. L. Berre, and P. Marquis. Function problems for quantified boolean formulas. Technical report, CRIL - France, 2003.

[15] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering an efficient sat solver. In *Proceedings of 38th Design Automation Conference (DAC01)*, 2001.

[16] J. Rintanen. Partial implicit unfolding in the Davis-Putnam procedure for Quantified Boolean Formulae. In *Proceedings of the First International Conference on Quantified Boolean Formulae (QBF'01)*, pages 84–93, 2001.

[17] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 459–465, 1992.

[18] L. Zhang and S. Malik. Towards a symetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP'02)*, pages 200–215, Ithaca, NY, USA, Sept. 2002.