A New Prenexing Strategy for Quantified Boolean Formulae with Bi-Implications

Benoit Da Mota, Igor Stéphan, and Pascal Nicolas

LERIA University of Angers, France
email: {damota, stephan, pn}@info.univ-angers.fr

Abstract. Most of the recent and efficient decision procedures for quantified Boolean formulae accept formulae in negation normal form as input or in an even more restrictive format such conjunctive normal form. But real problems are rarely expressed in such forms. For instance, in specification, intermediate propositional symbols are used to capture local results with always the same pattern. So, in order to use most of the state-of-the-art solvers the original formula has firstly to be converted in prenex form. A drawback of this preliminary step is to destroy completely the original structures of the formula. Furthermore, during the prenexing process, bi-implications are translated in such a way that there is a duplication of their sub-formulae including the quantifiers. In general, this process leads to an exponential growth of the formula. In this work, we focus on this very common pattern of intermediate result. We introduce new logical equivalences allowing us to extract these sub-formulae in a way that can improve the performance of the state-of-the-art quantified Boolean solvers.

1 Introduction

The quantified Boolean formula (QBF) validity problem is a generalization of the Boolean formulae satisfiability problem. While the complexity of Boolean satisfiability problem is NP-complete, it is PSPACE-complete for the QBF validity problem. This is the price for a more concise representation of many classes of formulae. Many important problems in several research fields have polynomial-time translations to the QBF validity problem: AI planning [17, 2], Bounded Model Construction [2], Formal Verification (see [6] for a survey).

Most of the recent and efficient decision procedures for QBF have formulae in negation normal form (NNF) as input or even more restrictive format such as formulae in conjunctive normal form (CNF). But problems are rarely expressed in such a form which destroyed completely their original structures. It is much more natural to use the full expressivity of the QBF language: all the usual connectors (including implication, bi-implication and xor) and quantifiers nested in the formula. Then translation schemas are required in order to use available QBF solvers. The NNF translation needs for QBF, as for first order logic, five steps: (i) replacement of the bi-implication and xor connectors by their definitions with implications, negations, conjunctions and/or disjunctions;

(ii) renaming of propositional symbols such that distinct quantifiers bind occurrences of distinct propositional symbols; (iii) extraction of the quantifiers; (iv) replacement of the implications by their definitions with negations, conjunctions and/or disjunctions; (v) application of the DeMorgan's laws. The CNF translation adds one more step to the NNF translation. This latter translation has been largely studied [16, 10, 11] since this normal form is used as input of most of the decision procedure for the SAT problem. The three first steps of the above translation form the so-called prenexing translation. There is not a unique prenex QBF associated with a QBF and according to the chosen prenexing strategy, the computation time of the decision process is highly influenced [12]. As far as we know, there is no study for QBF about the extraction of quantifiers nested in the bi-implications. It is an important issue since the usual elimination of bi-implications duplicates the nested quantifiers leading to an exponential growth of their numbers and the size of the formula.

The article is organized as follows: In Section 2, we present some preliminaries about propositional logic and quantified Boolean formulae. In Section 3, we analyse the links between prenexing and bi-implications, propose a methodology to extract the very common pattern of intermediate results and present one theoretical example and one practical example in order to evaluate our methodology. In Section 4, we report some experimental results for our methodology on our examples for state-of-the-art quanfitifed Boolean solvers. In Section 5, we conclude and draw some perspectives.

2 Preliminaries

2.1 Propositional logic.

The set of propositional symbols is denoted by $\mathcal{PS}$. Symbols $\bot$ and $\top$ are the propositional constants. Symbol $\wedge$ stands for conjunction, $\vee$ for disjunction, $\neg$ for negation, $\rightarrow$ for implication, $\leftrightarrow$ for bi-implication and $\oplus$ for xor. A literal is a propositional symbol or the negation of a propositional symbol. Definitions of the language of propositional formula **PROP** and semantics of all the Boolean symbols are defined in standard way. A formula is in negation normal form (NNF) if it is only constituted of conjunctions, disjunctions and literals. A formula is in conjunctive normal form (CNF) if it is a conjunction of disjunctions of literals. A substitution is a function from propositional symbols to **PROP**. This definition is extended as usual to a function from **PROP** to **PROP**: $[x \leftarrow F](G)$ is the formula obtained from G by replacing each occurrence of the propositional symbol x by the formula F. This definition is also extended as usual for the substitution of a formula by another formula. A valuation v is a function from $\mathcal{PS}$ to {**true**, **false**} ; the extension to **PROP** is denoted v^*. Propositional satisfaction is denoted $\models$ ($v \models F$ means $v^*(F) =$ **true**, the propositional formula F is satisfied by v and v is a model of F). Logical equivalence is denoted $\equiv$.

2.2 Quantified Boolean Formulae.

The symbol $\exists$ stands for the existential quantifier and $\forall$ stands for the universal quantifier (q stands for any quantifier). The set **QBF** of quantified Boolean formulae is defined inductively as follows: if F is in **PROP** then it is also in **QBF**; if F is in **QBF** and x is a propositional symbol then $(\exists x\ F)$ and $(\forall x\ F)$ are also in **QBF**; if F is in **QBF** then $\neg F$ is also in **QBF**; if F and G are in **QBF** and $\circ$ is in $\{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$ then $(F \circ G)$ is in **QBF**. The set of propositional symbols of a formula F is denoted $\mathcal{PS}(F)$. If a propositional symbol x is not under the scope of a quantifier (as in qx), then it is a free propositional symbol. The set of free propositional symbols of a QBF F is denoted $\mathcal{FPS}(F)$. A QBF is closed if its set of free propositional symbols is empty. A binder is a string $q_1 x_1 \dots q_n x_n$ with $x_1, \dots, x_n$ distinct propositional symbols and $q_1 \dots q_n$ quantifiers. The empty string is denoted by ε. A QBF is in prenex form if it is constituted of a binder and a propositional formula called the matrix. A QBF is in conjunctive normal form if it is a prenex QBF and its matrix is in conjunctive normal form. The definition of substitution is extended to QBF as follows: $[x \leftarrow F](G)$ is the formula obtained from G by replacing *free* occurrences of the propositional symbol x by the formula F. The semantics of QBF is defined as follows: for every propositional symbol y and every QBF F

$$(\exists y\ F) \equiv ([y \leftarrow \top](F) \vee [y \leftarrow \bot](F))$$

and

$$(\forall y\ F) \equiv ([y \leftarrow \top](F) \wedge [y \leftarrow \bot](F)).$$

A QBF F is valid if $F \equiv \top$. The Boolean satisfiability (SAT) of an unquantified Boolean formula corresponds exactly to the decision problem of the validity of its existential closure. The complexity of the QBF validity problem is PSPACE-complete while it is NP-complete for the SAT problem.

3 Prenexing and bi-implications

3.1 Motivations

In [12] are defined strategies for prenexing according to an order over the extraction of quantifiers. Some experiments have shown the high influence of this extraction order on the computation time for QBF solvers. However, there exists no rule for the extraction of quantifiers involved in a bi-implication (or xor). As pointed out in the introduction, the prenexing is decomposed in three steps that we detail more hereafter.

(i) The replacement of the bi-implication and xor connectors by their definitions with implications, negations, conjunctions and/or disjunctions is usually realized by the two following equivalences:

1) $(A \leftrightarrow B) \equiv ((A \rightarrow B) \wedge (B \rightarrow A))$ 2) $(A \oplus B) \equiv ((A \vee B) \wedge \neg(A \wedge B))$

One can remark that the two QBF A and B are duplicated.

(ii) The renaming of propositional symbols such that distinct quantifiers bind occurrences of distinct propositional symbols may be done before the extraction of quantifiers but may also be interleaved with this step.

(iii) The extraction of quantifiers is usually based on the following classical first order equivalences which still hold for QBF (F, G and H are QBF and x is a propositional symbol such that $x \notin \mathcal{FPS}(H)$)

3) $(\exists x\ \neg F) \equiv \neg(\forall x\ F)$ 4) $(\forall x\ \neg F) \equiv \neg(\exists x\ F)$
5) $(\forall x\ F) \equiv F$, if $x \notin \mathcal{FPS}(F)$ 6) $(\exists x\ F) \equiv F$, if $x \notin \mathcal{FPS}(F)$
7) $(\forall x\ (F \wedge H)) \equiv ((\forall x\ F) \wedge H)$ 8) $(\forall x\ (F \vee H)) \equiv ((\forall x\ F) \vee H)$
9) $(\exists x\ (F \wedge H)) \equiv ((\exists x\ F) \wedge H)$ 10) $(\exists x\ (F \vee H)) \equiv ((\exists x\ F) \vee H)$
11) $(\forall x\ (F \wedge G)) \equiv ((\forall x\ F) \wedge (\forall x\ G))$ 12) $(\exists x\ (F \vee G)) \equiv ((\exists x\ F) \vee (\exists x\ G))$
13) $(\forall x\ (F \rightarrow H)) \equiv ((\exists x\ F) \rightarrow H)$ 14) $(\exists x\ (F \rightarrow H)) \equiv ((\forall x\ F) \rightarrow H)$
15) $(\forall x\ (H \rightarrow G)) \equiv (H \rightarrow (\forall x\ G))$ 16) $(\exists x\ (H \rightarrow G)) \equiv (H \rightarrow (\exists x\ G))$

Equivalences 13 to 16 for implication are easily deduced from equivalences 3 to 12. The extraction of quantifiers is a process which applies these equivalences from right to left until a fix-point corresponding to a prenex QBF is obtained. Prenexing preserves validity but does not guarantee the preservation of the size of the QBF nor the associated search space. Let us remark that the expressivity of the bi-implication connector exceeds the simple equivalence between $(A \leftrightarrow B)$ and $((A \rightarrow B) \wedge (B \rightarrow A))$: a propositional symbol associated to a quantifier in A or B is duplicated following the equivalences 1 and 13 to 16 in two propositional symbols: one associated to a universal quantifier and another associated to an existential quantifier, and so whatever is the original quantifier. If this has no impact w.r.t. the validity, it is not the case w.r.t. the computation process. We show in the following examples that the increase of the size of the formula is not the only one problem with prenexing, even with simple QBF.

Let a be a propositional symbol, ϕ, ϕ_1 and ϕ_2 three QBF such that $\phi = (\phi_1 \leftrightarrow (\exists a\ \phi_2))$ and $a \notin \mathcal{FPS}(\phi_1)$. By equivalence 1, the bi-implication is eliminated: $\phi \stackrel{1}{\equiv} ((\phi_1 \rightarrow (\exists a\ \phi_2)) \wedge ((\exists a\ \phi_2) \rightarrow \phi_1))$. Formulae ϕ_1 and ϕ_2 have been duplicated. Then the quantifiers are extracted from the implications: $\phi \stackrel{16,13}{\equiv} ((\exists a\ (\phi_1 \rightarrow \phi_2)) \wedge (\forall a\ (\phi_2 \rightarrow \phi_1)))$. Now, one occurrence of the propositional symbol a associated to a quantifier has to be renamed in order to extract the quantifiers from the conjunction. Let b be this new renaming propositional symbol and $\phi'_2 = [a \leftarrow b](\phi_2)$ then: $\phi \stackrel{9,7}{\equiv} (\exists a\ (\forall b\ ((\phi_1 \rightarrow \phi_2) \wedge (\phi'_2 \rightarrow \phi_1))))$. The order of extraction of quantifiers may be reversed to obtain: $\phi \stackrel{7,9}{\equiv} (\forall a\ (\exists b\ ((\phi_1 \rightarrow \phi_2) \wedge (\phi'_2 \rightarrow \phi_1))))$. In the general case, if F is a QBF, $(\forall y\ (\exists x\ F))$ is not equivalent to $(\exists x\ (\forall y\ F))$. The possibility to permute these two quantifiers in this special case is a useful information which could be used by solvers which is not the case as far as we know. Moreover, propositional symbols a and b come from the same propositional symbol of the non prenex QBF but nothing in the prenex QBF keeps this information.

To conclude this motivation section, we consider how a quantifier has to be extracted from a sequence of bi-implications. We show in the following example

that the choice of the position of the parenthesis can influence the size of the result of the prenexing process. Let a be a propositional symbol ϕ_r, ϕ_l, ϕ_1, ϕ_2 and ϕ_3 some QBF such that $\phi_l = (((\phi_1 \leftrightarrow \phi_2) \leftrightarrow (\exists a\ \phi_3))$, $\phi_r = (\phi_1 \leftrightarrow (\phi_2 \leftrightarrow (\exists a\ \phi_3)))$ with $a \notin \mathcal{FPS}(\phi_1)$ and $a \notin \mathcal{FPS}(\phi_2)$ then $\phi_r \equiv \phi_l$ by associativity of the bi-implication. We compute a prenex form for ϕ_l:

$$\begin{array}{lll} \phi_l & \overset{1}{\equiv} & (((\phi_1 \leftrightarrow \phi_2) \rightarrow (\exists a\ \phi_3)) \wedge ((\exists a\ \phi_3) \rightarrow (\phi_1 \leftrightarrow \phi_2))) \\ \phi_l & \overset{16,13,9,7}{\equiv} & \exists a \forall b(((\phi_1 \leftrightarrow \phi_2) \rightarrow \phi_3) \wedge ([a \leftarrow b](\phi_3) \rightarrow (\phi_1 \leftrightarrow \phi_2))) \end{array}$$

and then for ϕ_r:

$$\begin{array}{lll} \phi_r & \overset{1}{\equiv} & (\phi_1 \leftrightarrow ((\phi_2 \rightarrow (\exists a\ \phi_3)) \wedge ((\exists a\ \phi_3) \rightarrow \phi_2))) \\ \phi_r & \overset{1}{\equiv} & ((\phi_1 \rightarrow ((\phi_2 \rightarrow (\exists a\ \phi_3)) \wedge ((\exists a\ \phi_3) \rightarrow \phi_2))) \wedge \\ & & (((\phi_2 \rightarrow (\exists a\ \phi_3)) \wedge ((\exists a\ \phi_3) \rightarrow \phi_2)) \rightarrow \phi_1)) \\ \phi_r & \overset{13,14,15,16,9,7}{\equiv} & \exists a \exists c \forall b \forall d\, ((\phi_1 \rightarrow ((\phi_2 \rightarrow \phi_3) \wedge ([a \leftarrow b](\phi_3) \rightarrow \phi_2))) \wedge \\ & & \qquad (((\phi_2 \rightarrow [a \leftarrow d](\phi_3)) \wedge ([a \leftarrow c](\phi_3) \rightarrow \phi_2)) \rightarrow \phi_1)) \end{array}$$

Formulae ϕ_l and ϕ_r, although equivalent, have different numbers of propositional symbols and different sizes after prenexing. If we consider the formula $(\phi_1 \leftrightarrow (\phi_2 \leftrightarrow \ldots (\phi_n \leftrightarrow (\exists a\ \phi_{n+1}))))$, then 2^n propositional symbols are generated whose half of them are universally quantified. Formulae ϕ_n and ϕ_{n+1} are recopied 2^n times, formula ϕ_{n-1} is recopied 2^{n-1} times, and so on until ϕ_1 which is recopied two times. The size of the formula and the number of propositional symbols grow exponentially w.r.t. the number of crossed bi-implication. Any way, if every ϕ_k has a quantifier to extract, the worst case cannot be avoided.

3.2 Prenexing and intermediate results

Previous section shows how important is the issue of prenexing when quantifiers occur in the scope of bi-implications. Since $(A \oplus B) \equiv \neg(A \leftrightarrow B) \equiv (\neg A \leftrightarrow B)$, in what follows we only focus on bi-implication. This issue is, as far as we know, usually skipped in the process translating the specified problem in QBF to the equivalent CNF QBF given as input to a QBF solver. In this section, we focus on two very frequent cases which occur in the programming or specifying process: the declaration of intermediate results and the declaration of domain. The first case is based on the introduction of an existentially quantified propositional symbol in association with a conjunction in order to improve the readability of the specification or to capture the results of a calculus duplicated in many places. For QBF, by extension of a classical result of [18], we are interested in the pattern $(\exists x\ ((x \leftrightarrow F) \wedge G))$, x being an intermediate propositional symbol not occurring in F; it is easily proven from the semantics of the existential quantifier that this pattern is equivalent to $[x \leftarrow F](G)$. The latter case is based on the introduction of a universally quantified propositional symbol in association with an implication in order to express the domain of the propositional symbol (ie: a property to be verified by the symbol). In both cases, we call $(x \leftrightarrow F)$ the definition of x. The following result shows that these two techniques are actually the same.

Theorem 1. *Let F and G be two QBF and x a propositional symbol which represents an intermediate result F, with $x \notin \mathcal{FPS}(F)$, then the following equivalence holds:*

$$(\exists x\ ((x \leftrightarrow F) \wedge G)) \equiv (\forall x\ ((x \leftrightarrow F) \rightarrow G)).$$

The previous theorem is based on the following (propositional) equivalence:

$$((\neg A \vee B) \wedge (A \vee C) \wedge (B \vee C)) \equiv ((\neg A \vee B) \wedge (A \vee C)).$$

By the previous theorem, we focus only on the existential pattern.

Since solvers usually have as input CNF QBF, the special status of the intermediate propositional symbols is completely lost and they are managed as the other propositional symbols of the problem. There is a simple way to get rid of these intermediate propositional symbols: we can simply apply the already shown equivalence $(\exists x\ ((x \leftrightarrow F) \wedge G)) \equiv [x \leftarrow F](G)$ but it can lead to an exponential growth of the size of the formula. Instead, we propose to extract these propositional symbols thanks to the following theorem.

Theorem 2. *Let F, G and H be three QBF and x be a propositional symbol which represents the intermediate result F, with $x \notin \mathcal{FPS}(H)$, $x \notin \mathcal{FPS}(F)$ then*

$$(H \leftrightarrow (\exists x\ ((x \leftrightarrow F) \wedge G))) \equiv (\exists x\ ((x \leftrightarrow F) \wedge (H \leftrightarrow G))).$$

The Algorithm 1 realizes the recursive application of this theorem by calling the three functions described below.

Function 1 *def_search*

In: A QBF F
In: A set of propositional symbols S_{ps}
Out: A set of definitions

```
switch F do
 case (∃x G)
 return def_search(G, S_ps ∪ {x})
 case (G ∧ H)
 return def_search(G, S_ps) ∪ def_search(H, S_ps)
 case (x ↔ G)
 if x ∈ S_ps then
  return {(x ↔ G)}
 else
  return ∅
 end if
 default
 return ∅
end switch
```

Let $(\phi_H \leftrightarrow \phi)$ be a QBF, the *def_search* algorithm of Function 1 searches for all definitions matching $(\exists x\ ((x \leftrightarrow F) \wedge G))$ in ϕ with the constraint $x \notin \mathcal{FPS}(F)$

relaxed. In fact the pattern can be embedded in an alternation of similar patterns thanks to equivalences 9 and $(\exists x\ (\exists y\ F)) \equiv (\exists y\ (\exists x\ F))$ and associativity and commutativity of conjunction. For instance

$$def\_search((\phi_1 \wedge (\exists a\ (\exists b\ ((a \leftrightarrow \phi_a) \wedge ((b \leftrightarrow \phi_b) \wedge \phi_2))))), \emptyset) = \{(a \leftrightarrow \phi_a), (b \leftrightarrow \phi_b)\}.$$

In what follows, S_{def} denotes a set of definitions as $\{(a \leftrightarrow \phi_a), (b \leftrightarrow \phi_b)\}$ for instance .

Function 2 *def_extract*

In: A set of definitions S_{def}
In: A set of propositional symbols S_{ps}
Out: A list of definitions
  $L_d := nil$
  while there exists $(e \leftrightarrow d) \in S_{def}$ such that $\mathcal{FPS}(d) \subseteq S_{ps}$ **do**
    $S_{def} := S_{def}$ without all the definitions on e
    $S_{ps} := S_{ps} \cup \{e\}$
    $L_d := \cdot((e \leftrightarrow d), L_d)$
  end while
  return L_d

The *def_extract* algorithm of Function 2 applies the constraint $x \notin \mathcal{FPS}(F)$ on the set of definitions obtained from the *def_search* algorithm thanks to a topological sort which extracts a list of definitions. Not all the definitions of the set are inserted in the list: for instance if $a \in \mathcal{FPS}(\phi_b)$, $b \in \mathcal{FPS}(\phi_a)$ and S_{ps} is a set of propositional symbols such that $a \notin S_{ps}$ nor $b \notin S_{ps}$ then $def\_extract(S_{def}, S_{ps}) = nil$ otherwise if $a \notin \mathcal{FPS}(\phi_b)$ but $b \in \mathcal{FPS}(\phi_a)$ then $def\_extract(S_{def}, S_{ps}) = \cdot((b \leftrightarrow \phi_b), \cdot((a \leftrightarrow \phi_a), nil))$ since

$$\begin{array}{l}(\phi_1 \wedge (\exists a\ (\exists b\ ((a \leftrightarrow \phi_a) \wedge ((b \leftrightarrow \phi_b) \wedge \phi_2)))))\\ \equiv (\exists b\ ((b \leftrightarrow \phi_b) \wedge (\exists a\ ((a \leftrightarrow \phi_a) \wedge (\phi_1 \wedge \phi_2)))))\end{array}$$

The *def_applying* algorithm of Function 3 actually applies Theorem 2 in two steps on the QBF ϕ for the list of definitions extracted by the *def_extract* algorithm. Firstly, the definitions are replaced by $\top$ in the QBF since the definitions are conjunctively connected and are reintroduced at the top of the QBF: for instance $(\phi_1 \wedge (\exists a\ (\exists b\ ((a \leftrightarrow \phi_a) \wedge ((b \leftrightarrow \phi_b) \wedge \phi_2)))))$ is replaced by

$$\begin{array}{l}((b \leftrightarrow \phi_b) \wedge ((a \leftrightarrow \phi_a) \wedge (\phi_1 \wedge (\exists a\ (\exists b\ (\top \wedge (\top \wedge \phi_2)))))))\\ \equiv ((b \leftrightarrow \phi_b) \wedge ((a \leftrightarrow \phi_a) \wedge (\phi_1 \wedge (\exists a\ (\exists b\ \phi_2))))).\end{array}$$

Secondly, the existential quantifiers for the propositional symbols of the list of definitions are eliminated from the formula and reintroduced above it: for instance $((b \leftrightarrow \phi_b) \wedge ((a \leftrightarrow \phi_a) \wedge (\phi_1 \wedge (\exists a\ (\exists b\ \phi_2)))))$ is replaced by

$$\begin{array}{l}(\exists b\ (\exists a\ ((b \leftrightarrow \phi_b) \wedge ((a \leftrightarrow \phi_a) \wedge (\phi_1 \wedge \phi_2)))))\\ \equiv (\exists b\ ((b \leftrightarrow \phi_b) \wedge (\exists a\ ((a \leftrightarrow \phi_a) \wedge (\phi_1 \wedge \phi_2))))).\end{array}$$

Function 3 *def_applying*

In: A QBF F
In: A list of definitions L_d
Out: A QBF
  A list of definitions $L_{save} := L_d$
  while not $empty(L_d)$ **do**
   $(e \leftrightarrow d) := head(L_d)$
   $F := ((e \leftrightarrow d) \wedge [(e \leftrightarrow d) \leftarrow \top](F))$
   $L_d := tail(L_d)$
  end while
  while not $empty(L_{save})$ **do**
   $(e \leftrightarrow d) := head(L_{save})$
   delete existential quantifier on e from F
   $F := (\exists e\ F)$
   $L_{save} := tail(L_{save})$
  end while
  return F

Algorithm 1 *rec_def_extraction*

In: A QBF F
In: A set of propositional symbols S_{ps}
Out: An equivalent QBF to F with its definitions extracted
  switch F **do**
   case $(qx\ G)$
   return $(qx\ rec\_def\_extraction(G, \{x\} \cup S_{ps}))$
   case $\neg G$
   return $\neg rec\_def\_extraction(G, S_{ps})$
   case $(G \circ H)$ and $\circ \in \{\wedge, \vee, \rightarrow\}$
   return $(rec\_def\_extraction(G, S_{ps}) \circ rec\_def\_extraction(H, S_{ps}))$
   case $(G \leftrightarrow H)$
   A QBF $G' := rec\_def\_extraction(G, S_{ps})$
   A QBF $H' := rec\_def\_extraction(H, S_{ps})$
   A list of definitions $L_d := def\_search(G', \emptyset) \cup def\_search(H', \emptyset)$
   return $def\_applying((G' \leftrightarrow H'), def\_extract(L_d, S_{ps}))$
   default
   return F
  end switch

The *rec_def_extraction* algorithm of Algorithm 1 applies recusively the extraction of the pattern on all possible definitions in an inside/out process. By this way some definitions can cross many equivalences.

It is not difficult to prove thanks to Theorem 2 the following correction theorem.

Theorem 3. *Let F be a QBF then $rec\_def\_extraction(F, \emptyset) \equiv F$.*

3.3 Bi-implication chains with intermediate results

We define a theoretical example in order to evaluate the impact of our methodology as a chain of bi-implications $th_n = (\xi_1 \leftrightarrow (\xi_2 \leftrightarrow ...(\xi_{n-1} \leftrightarrow (\xi_n))))$ in which each ξ_k contains an intermediate result as proposed in motivation section 3.1. We have chosen some ξ_k of the form $(\exists x_k((x_k \leftrightarrow (c \vee b)) \wedge (x_k \wedge a)))$, with x_k the intermediate result and a, b and c propositional symbols which are in some other links of the bi-implication chain. Formula th_n, for $n \geq 4$ is of the form:

$th_n = \exists e_0 ... \exists e_{n-2} \forall u_0 \forall u_1 \forall u_2$
$(\ (\exists x_n((x_n \leftrightarrow (e_{n-4} \vee e_{n-3})) \wedge (x_n \wedge e_{n-2}))) \leftrightarrow$
$\quad (\ (\exists x_{n-1}((x_{n-1} \leftrightarrow (e_{n-5} \vee e_{n-4})) \wedge (x_{n-1} \wedge e_{n-3}))) \leftrightarrow$
$\quad\quad ...$
$\quad\quad\quad (\ (\exists x_4((x_4 \leftrightarrow (e_0 \vee e_1)) \wedge (x_4 \wedge e_2))) \leftrightarrow$
$\quad\quad\quad\quad (\ (\exists x_3((x_3 \leftrightarrow (u_2 \vee e_0)) \wedge (x_3 \wedge e_1))) \leftrightarrow$
$\quad\quad\quad\quad\quad (\ (\exists x_2((x_2 \leftrightarrow (u_1 \vee u_2)) \wedge (x_2 \wedge e_0))) \leftrightarrow$
$\quad\quad\quad\quad\quad\quad (\exists x_1((x_1 \leftrightarrow (u_0 \vee u_1)) \wedge (x_1 \wedge u_2))) \ ) \ ) \ )...))$

For all n, th_n is not valid.

The nesting of quantifiers and bi-implications leads to an exponential explosion of the size of th_n in CNF w.r.t. n which is not the case with our methodology. For instance, with $n = 7$, the CNF converted QBF has 1148 propositional symbols (98 are universally quantified) and 3038 clauses whereas the QBF obtained by application of the $rec\_def\_extraction$ algorithm and then converted into CNF has only 33 propositional symbols (3 are universally quantified) and 82 clauses. For $n = 18$, the CNF converted QBF has not been decided by any of the state-of-the-art solvers in one hour whereas for $n = 6000$ the QBF obtained by application of the $rec\_def\_extraction$ algorithm and then converted into CNF has been decided in less than one hour.

3.4 Hardware verification: the n-bit ripple-carry adder

We also evaluate our methodology on a more simple (with only one bi-implication crossed by some definitions) and classical example of hardware verification: the n-bit ripple-carry adder. We follow the approach of bounded model construction [1] which is a method for generating models for a monadic formula by reducing its satisfiability problem to a QBF validity problem [3, 2]. The example is about a parametrized family of ripple-carry n-bits adders. For a given number n, one has to verify the equivalence of the digital circuit with its specification. Hereafter, the QBF add_{impl} represents the implementation of the adder and add_{spec} the specification, (n represents the size of the ripple-carry adder, A and B represent n-bit input vectors, S represents the n-bit output and the Booleans c_i and c_o are the carry-in and carry-out respectively). By this way, the validity of the following QBF add_n ensures the correctness of the physical n-bit ripple-carry adder.

$$add_n = \forall A \forall B \forall S \forall c_i \forall c_o (add_{impl}(n, A, B, S, c_i, c_o) \leftrightarrow add_{spec}(n, A, B, S, c_i, c_o))$$

We give an example for the n-bit adder with $n = 1$: QBF

$$add_1 = \forall A_0 \forall B_0 \forall S_0 \forall c_i \forall c_o \; (add_{impl}(1, A_0, B_0, S_0, c_i, c_o) \\ \leftrightarrow add_{spec}(1, A_0, B_0, S_0, c_i, c_o))$$

with

$$\begin{aligned} add_{impl}(1, A_0, B_0, S_0, c_i, c_o) = [\exists x_1 \exists x_2((D_1(x_1) \wedge D_2(x_2)) \wedge (\exists x_3 \exists x_4 \exists x_5 \\ (D_3(x_3) \wedge C_1(x_3, x_1) \wedge D_4(x_4) \wedge \\ D_5(x_5, x_1, x_3) \wedge C_2(x_2, x_5, x_4))))] \\ add_{spec}(1, A_0, B_0, S_0, c_i, c_o) = [\exists x_6 \exists x_7 (D_1(x_6) \wedge D_2(x_7) \wedge C_3(x_6, x_7) \\ \wedge C_4(x_6))] \end{aligned}$$

encodes the check of the adder for $n = 1$ with the following Boolean functions:

$$\begin{aligned} &D_1(x) = (c_i \leftrightarrow x), \\ &D_2(x) = (c_o \leftrightarrow x), \\ &D_3(x) = (x \leftrightarrow (A_0 \oplus B_0)), \\ &D_4(x) = (x \leftrightarrow (A_0 \wedge B_0)), \\ &D_5(x, y, z) = (x \leftrightarrow (y \wedge z)), \\ &C_1(x, y) = (S_0 \leftrightarrow (x \oplus y)), \\ &C_2(x, y, z) = (x \leftrightarrow (y \vee z)), \\ &C_3(x, y) = ((((A_0 \wedge B_0) \vee (A_0 \wedge x)) \vee (B_0 \wedge x)) \leftrightarrow y), \\ &C_4(x) = (((A_0 \leftrightarrow B_0) \leftrightarrow S_0) \leftrightarrow x). \end{aligned}$$

Functions C_k, $1 \leq k \leq 4$, represent the core of the implementation and specification of the n-bit ripple-carry adder. Formulae add_{impl} and add_{spec} contain instances of functions D_k, $1 \leq k \leq 5$, which are definitions that we will extract.

The hereafter QBF add'_1 is the initial QBF add_1 translated by the classical prenexing algorithm (propositional symbols y_k come from the x_k by recopy of the two subformulae of the bi-implication). Universal quantifiers are first extracted from both parts of the bi-implication in order to minimize the alternation of quantifiers.

$$\begin{aligned} &add'_1 = \forall A_0 \forall B_0 \forall S_0 \forall c_i \forall c_o \forall y_1 \forall y_2 \forall y_3 \forall y_4 \forall y_5 \forall y_6 \forall y_7 \exists x_1 \exists x_2 \exists x_3 \exists x_4 \exists x_5 \exists x_6 \exists x_7 \\ &[[(D_1(y_1) \wedge D_2(y_2) \wedge D_3(y_3) \wedge C_1(y_3, y_1) \wedge D_4(y_4) \wedge D_5(y_5, y_1, y_3) \wedge C_2(y_2, y_5, y_4)) \\ &\rightarrow (D_1(x_6) \wedge D_2(x_7) \wedge C_3(x_6, x_7) \wedge C_4(x_6))] \wedge \\ &[(D_1(y_6) \wedge D_2(y_7) \wedge C_3(y_6, y_7) \wedge C_4(y_6)) \\ &\rightarrow (D_1(x_1) \wedge D_2(x_2) \wedge D_3(x_3) \wedge C_1(x_3, x_1) \wedge D_4(x_4) \wedge D_5(x_5, x_1, x_3) \wedge C_2(x_2, x_5, x_4))]] \end{aligned}$$

The order of the quantifiers highly influences the efficiency of the solvers [12]. During the prenexing process of add_1 into add'_1, the order of the propositional symbols is only constrained by the partial order induced by the fact that the universal quantifiers of the initial propositional symbols have to precede the existential quantifiers.

The hereafter QBF add^t_1 is the initial QBF add_1 translated by our algorithm $rec\_def\_extraction$ and then translated into a prenex form according to the classical algorithm.

$$add_1^t = \forall A_0 \forall B_0 \forall S_0 \forall c_i \forall c_o \exists x_1 \exists x_2 \exists x_3 \exists x_4 \exists x_5 \exists x_6 \exists x_7$$
$$[D_1(x_1) \wedge D_2(x_2) \wedge D_3(x_3) \wedge D_4(x_4) \wedge D_5(x_5, x_1, x_3) \wedge D_1(x_6) \wedge D_2(x_7)] \wedge$$
$$[[C_1(x_3, x_1) \wedge C_2(x_2, x_5, x_4)] \leftrightarrow [C_3(x_6, x_7) \wedge C_4(x_6)]]$$

The obtained matrix of the QBF is composed of two parts: the definition of the intermediate propositional symbols followed by the core of the equivalence between the implementation and the specification of the adder which uses these intermediate propositional symbols.

4 Experimental Results

In order to evaluate the impact of our new methodology, we have developed a Prolog program to generate instances of chains of bi-implications with intermediate results th_n as described in subsection 3.3 and instances of the n-bit ripple-carry adder add_n as described in subsection 3.4. The applied prenexing process does not search for an optimal order of the quantifiers. The order of the definitions in the matrix has not been optimized either. The CNF conversion is computed thanks to the introduction of intermediate existentially quantified propositional symbols. This CNF conversion only polynomially increases the size of the QBF. The experiments have been realized on an *Intel(R) Xeon(R) (2.83 GHz)* with $4GB$ of memory. For our experiments we used the following QBF solvers:

- sKizzo v0.8.2-beta [5] which is based on symbolic skolemization [4];
- Quantor 3.0 [7] which combines Q-resolution [15], to eliminate an innermost existential quantifier, and expansion, to eliminate an innermost universal quantifier;
- QuBE-BJ1.2 [14] which extends to QBF the Davis-Logemann-Loveland [9] procedure with backjumping and two more recent versions, QuBE6.5 (resp. QuBE6.1), which integrates (resp. does not integrate) as preprocessing the Q-resolution [8];
- qpro [13] which is a decision procedure for non prenex QBF in negation normal form (i.e. QBF only with conjunction, disjunction, literals and quantifiers) based on a sequent calculus for QBF, the GQBF calculus.

Experiments have been done firstly with the default settings of the different solvers (in particular, sKizzo is set with the Q-resolution preprocessing).

4.1 Chains of bi-implications with intermediate results

Figures 1 and 2 report computation time in seconds for th_n the chains of bi-implications with or without our methodology and converted in prenex CNF for sKizzo, QuBE 6.5 and Quantor and in non prenex NNF for qpro. Actually, the size of the QBF converted in CNF or NNF grows exponentially with n.

Fig. 1. Computation time in seconds for th_n the chains of bi-implications for QuBE 6.5 with and without intermediate results translated by $rec\_def\_extraction$ algorithm and converted in prenex CNF and for qpro with and without intermediate results translated by $rec\_def\_extraction$ algorithm and converted in non prenex NNF.

Fig. 2. Computation time in seconds for th_n the chains of bi-implications with intermediate results translated by $rec\_def\_extraction$ algorithm and converted in prenex CNF for sKizzo, QuBE 6.5 and Quantor and in non prenex NNF for qpro.

None of the tested QBF solvers has succeeded in deciding if th_n is valid or not for $n > 17$ without translation by our $rec\_def\_extraction$ algorithm (QuBE6.5 succeeds in 2657 seconds for $n = 15$ but computation time exceeded the 3600 seconds for $n = 16$, Quantor succeeds in 0.1 second for $n = 5$ but computation space exceeded the available memory space for $n = 6$, sKizzo succeeds in 31.9 seconds for $n = 7$ but computation space exceeded the available memory space for $n = 8$, qpro succeeds in 1145.8 seconds for $n = 17$ but computation time exceeded the 3600 seconds for $n = 18$). Due to the lack of space and since sKizzo without our methodology failed to decide for $n = 8$ and succeeds to decide with our methodology for $n = 2000$, we do not report more results about sKizzo with or without our methodology. For the same reason, since Quantor failed to decide for $n = 5$ and succeeds to decide with our methodology for $n = 6000$, we do not report more results about Quantor with or without our methodology. Figure 1 reports the results for QuBE 6.5 and qpro with or without our methodology. Computation time axis is in log scale. Figure 2 reports the results for the tested QBF solvers on the QBF translated by our $rec\_def\_extraction$ algorithm (and then converted in prenex CNF for sKizzo, QuBE and Quantor and non prenex NNF for qpro). The axes are both in log scale.

4.2 The n-bit ripple-carry adder

Results of the evaluation for the n-bit ripple-carry adder are summarized in Table 1, computation times are in seconds, T (for "time out") means that the computation time exceeded 3600 seconds, M means that the computation space exceeded the available memory space. These results show that the translated QBF have almost always the best results. For all the solvers but sKizzo the results are clear. For sKizzo, with $n < 12$ the improvement is substantial, but speedup decreases for ripple-carry adder with larger size. The result of the translation penalizes sKizzo on the largest instances: it may be observed thanks to the trace that sKizzo, by the Q-resolution preprocessing, removes some intermediate results (canceling part of our translation). Since QuBE6.5, which integrates a Q-resolution preprocessing, has results worse than those obtained by QuBE-BJ1.2, we think that Q-resolution has a great impact on this problem. In Table 1 are

	sKizzo with Q-resolution		sKizzo without Q-resolution		QuBE 6.5	QuBE 6.1	QuBE BJ1.2	Quantor		qpro	
n	without	with	without	with	with	with	with	without	with	without	with
4	**0,1**	**0,1**	0,2	**0,1**	0,7	1,3	**0,1**	5,3	**0,1**	1,3	1,5
5	0,2	**0,1**	0,2	**0,1**	6,7	79,5	1,1	75,2	0,2	21,2	17,3
6	0,4	**0,1**	1,3	**0,1**	61,4	T	9,8	T	1,8	335,9	186,1
7	1,0	**0,1**	1,3	**0,1**	561,0	T	83,8	M	11,1	T	1967,2
8	2,0	0,2	3,3	**0,1**	T	T	714,5	M	106,0	T	T
9	3,6	0,4	1,5	**0,1**	T	T	T	M	2936,6	T	T
10	7,5	1,6	3,2	**0,1**	T	T	T	M	M	T	T
11	32,0	6,7	5,4	**0,1**	T	T	T	M	M	T	T
12	14,1	11,9	4,8	**0,1**	T	T	T	M	M	T	T
13	13,6	16,1	3,6	**0,2**	T	T	T	M	M	T	T
14	16,3	15,6	5,3	**0,3**	T	T	T	M	M	T	T
15	19,6	21,9	4,7	**0,3**	T	T	T	M	M	T	T
16	33,5	29,3	6,5	**0,3**	T	T	T	M	M	T	T
17	22,6	15,0	17,0	**0,4**	T	T	T	M	M	T	T
18	19,3	14,4	14,0	**0,5**	T	T	T	M	M	T	T
19	32,4	26,8	23,4	**0,5**	T	T	T	M	M	T	T
20	82,5	21,7	28,9	**0,4**	T	T	T	M	M	T	T
21	46,6	20,7	26,1	**1,0**	T	T	T	M	M	T	T
22	35,7	27,9	19,6	**1,0**	T	T	T	M	M	T	T
23	37,7	14,0	294,9	**0,8**	T	T	T	M	M	T	T

Table 1. Computation time in seconds for the n-bit ripple-carry adder with intermediate results translated by *rec_def_extraction* algorithm and converted in prenex CNF for sKizzo, QuBE and Quantor and in non prenex NNF for qpro.

also reported the experiments for sKizzo, but without Q-resolution, for which we obtain the best results of our experiments. The improvement provided by our new translation is very substantial.

5 Conclusion

In this work, we proposed different logical equivalences allowing to deal with intermediate results involved in bi-implications or xor inside a non prenex QBF. One consequence of the use of these equivalences is to keep the size of the original QBF, and its number of propositional symbols, during a prenexing process to extract internal quantifiers. It is also possible to choose the nature of the quantifier of the intermediate propositional symbol and then to reduce the number of alternations of quantifiers.

To evaluate the practical impact of our present work, we realized two kinds of experiments. Firstly, we used an artificial example exhibited in our motivation section to illustrate that in some cases the usual prenexing process leads to a QBF with an exponential size. Our experimental results show that our new prenexing strategy avoids this exponential growing of the prenex formula. So, an immediate consequence of our contribution is that available QBF solvers are now able to deal with such large QBF, when they were unable to solve very small ones. Secondly, we chose a practical and realistic example: the formal verification of a digital circuit that has to conform to a given specification. The results show the positive impact of our new prenexing strategy. It is worth noting that if Q-resolution is used by the solver then the efficiency decreases. So, one interesting

topic of research may be to study if the efficiency of Q-resolution can be restored if it is not applied on intermediate results.

In our future works, we plan to extend our results to the general case of quantifiers involved in bi-implications or xor, and not only for intermediate results. Another improvement of our present work would be to compare different variants of our prenexing strategy since our experimental results show that the nature of the chosen quantifier may importantly influence the resolution time. Moreover, having observed that procedures inside QBF solvers use, more and more often, a structure of quantifiers and a non prenex clausal form of QBF, it would be interesting to work directly on the original (non prenex non CNF) encoding of the problem to solve. So, we plan to study how to translate a non prenex non CNF QBF in a non prenex CNF QBF, to sort the propositional symbols in different categories: problem propositional symbols, intermediate results and propositional symbols only dedicated to the encoding. Then, we would be able to exploit more information and to evaluate the impact of heuristics, like Q-resolution, on these different kinds of propositional symbols.

References

1. A. Ayari and D. Basin. Bounded model construction for monadic second-order logics. In *Proocedings of the 12th International Conference on Computer-Aided Verification (CAV'00)*, pages 99–113, 2000.
2. A. Ayari and D. Basin. Qubos: Deciding Quantified Boolean Logic using Propositional Satisfiability Solvers. In *Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design (FMCAD'02)*, 2002.
3. A. Ayari, D. Basin, and S. Friedrich. Structural and behavioral modeling with monadic logics. In *The Twenty-Ninth IEEE International Symposium on Multiple-Valued Logic*, pages 142–151, 1999.
4. M. Benedetti. Evaluating QBFs via Symbolic Skolemization. In *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'05)*, number 3452 in LNCS, pages 285–300. Springer, 2005.
5. M. Benedetti. skizzo: a suite to evaluate and certify QBFs. In *Proceedings of the 20th International Conference on Automated Deduction (CADE'05)*, pages 369–376, 2005.
6. M. Benedetti and H. Mangassarian. Experience and Perspectives in QBF-Based Formal Verification. *Journal on Satisfiability, Boolean Modeling and Computation*, 2008 (to appear).
7. A. Biere. Resolve and Expand. In *7th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'04)*, 2004.
8. U. Bubeck and H. Kleine Büning. Bounded Universal Expansion for Preprocessing QBF. In *Proceedings of the Tenth International Confrerence on Theory and Applications of Satisfiability Testing (SAT'07)*, pages 244–257, 2007.
9. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communication of the ACM*, 5, 1962.
10. T. Boy de la Tour. An Optimality Result for Clause Form Translation. *Journal of Symbolic Computation*, 14(4):283–302, 1992.
11. U. Egly. On Different Structure-Preserving Translations to Normal Form. *Journal of Symbolic Computation*, 22(2):121–142, 1996.

12. U. Egly, M. Seidl, H. Tompits, S. Woltran, and M. Zolda. Comparing Different Prenexing Strategies for Quantified Boolean Formulas. In *SAT*, pages 214–228, 2003.
13. U. Egly, M. Seidl, and S. Woltran. A Solver for QBFs in Nonprenex Form. *Constraints*, 14(1):38–79, 2009.
14. E. Giunchiglia, M. Narizzano, and A. Tacchella. QuBE++: an efficient QBF solver. In *Formal Methods in Computer-Aided Design*, 2004.
15. H. Kleine Büning, M. Karpinski, and A. Flögel. Resolution for quantified boolean formulas. *Information and Computation*, 117(1):12–18, 1995.
16. D. A. Plaisted and S. Greenbaum. A Structure-Preserving Clause Form Translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986.
17. J. Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.
18. G. S. Tseitin. On the complexity of derivation in propositional calculus. In A. O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, pages 115–125. Consultants Bureau, New York, 1970.