From (Quantified) Boolean Formulae to Answer Set Programming

Benoit Da Mota and Igor Stéphan and Pascal Nicolas

LERIA, University of Angers, 2 boulevard Lavoisier, 49045, Angers, France

Abstract. We propose in this article a translation from Quantified Boolean Formulae to Answer Set Programming. The computation of a solution of a Quantified Boolean Formula is then equivalent to the computation of a stable model for a normal logic program. The case of unquantified Boolean formulae is also considered since it is equivalent to the case of Quantified Boolean Formulae with only existential quantifiers.

1 Introduction

The problem of satisfiability of a Boolean or propositional formula (SAT) is a combinatorial problem which held the attention like a canonical problem of the NP-complete complexity class. Many decision procedures have been proposed, mainly for the conjunctive normal form (CNF) [9, 18, 23]. Some recent works show that the CNF transformation seems to disrupt too much the original structure of the problem [31]. However, there exists only few implementations of solvers for non-CNF SAT formulae [11, 15, 21, 31].

In the same way, the problem of validity of quantified Boolean formulae (QBF) [30] is a combinatorial problem but with PSPACE-complete complexity. SAT problem is equivalent to QBF problem with only existentially quantified variables. Most of the decision procedures for QBF treat only the restriction to prenex CNF formulae since they are extensions of SAT decision procedures [4, 5, 7, 14, 17, 26, 27]. For QBF, the impact of the transformation in prenex CNF formulae seems even larger [2]. As for SAT solvers, there exists very few QBF solvers for (non prenex) non-CNF formulae [2, 3, 10, 32, 33].

In both cases (SAT or QBF), the representation of combinatorial problems, or more generally of knowledge representation problems of artificial intelligence,are usually natural, direct and compact when it is allowed to use all the expressivity of the language. That is why, in this article we propose to deal with the development of a solver for (quantified) Boolean formulae not CNF.

Answer Set Programming (ASP) is a formalism of non monotonic logic programming appropriated to represent and solve different combinatorial problems. Many decision procedures for ASP have been proposed and have been developed in different softwares which have proved their efficiency and their robustness on large problems.

A polynomial translation form SAT to ASP has already been proposed but only for CNF formulae [24]. As well-fitted and efficient tool to encode combinatorial problems, we propose in this article to explore the development of SAT or

QBF solver by the translation of non CNF formulae in normal logic programs and to solve them with any ASP software.

We want to demonstrate by this work that the paradigm of answer set programming (ASP, see subsection 2.2) is sufficiently expressive to represent quantified Boolean formulae (QBF, see subsection 2.1). To reach our goal, we first demonstrate in section 3 that the representation of the satisfiability problem for non-CNF formulae is possible in ASP. Then in the section 4, we extend the principle to every quantified Boolean formulae. We complete this study in section 5 by some experimental results illustrating that the best ASP solvers may be used to compute the models of (quantified) Boolean formulae without necessarily formulae in conjunctive normal form which is the case for most of the available solvers. But first we begin by a section that introduces the necessary materials to understand this work.

2 Preliminaries

2.1 (Quantified) Boolean Formulae

The Boolean values are denoted $\mathbf{t}$ and $\mathbf{f}$, the set of Boolean values is denoted $BOOL$ and the set of Boolean functions is denoted $\mathcal{F}$. The set of propositional symbols (or variables) is denoted $\mathcal{V}$. The symbols $\top$ and $\bot$ are the propositional constants. The symbol $\wedge$ is used for conjunction, $\vee$ for disjunction, $\neg$ for negation, $\rightarrow$ for implication, $\leftrightarrow$ for equivalence and $\oplus$ for xor ($\mathcal{O} = \{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$). A literal is a propositional variable or the negation of a propositional variable. The set of literals is denoted $\mathcal{L}$. The set $PROP$ of propositional formulae (called also Boolean formulae) is inductively defined as follows : every propositional symbol (constant or variable) is an element of $PROP$; if F is an element of $PROP$ then $\neg F$ is an element of $PROP$; if F and G are elements of $PROP$ and $\circ$ is an element of $\mathcal{O}$ then $(F \circ G)$ is an element of $PROP$. The symbol $\exists$ is used for existential quantification and $\forall$ for universal quantification (q is used as a quantification variable). The set of quantified Boolean formula (QBF) is also defined by induction as follows: Every Boolean formula is also a quantified Boolean formula ; if F is a QBF and x is a propositional variable then $(\exists x\ F)$ and $(\forall x\ F)$ are QBF. It is assumed that distinct quantifiers bind occurrences of distinct variables. The set of variables or symbols of a formula F is denoted $\mathcal{V}(F)$. A substitution is a function from the set of variables to the set of (quantified) Boolean formulae. We define a substitution of x by F in G, denoted $G[x \leftarrow F]$, as the formula G obtained by replacing all the occurrences[1] of the variable x by the formula F. A binder Q is a string $q_1 x_1 \ldots q_n x_n$ with $x_1, \ldots, x_n$ distinct variables and $q_1 \ldots q_n$ quantifiers. The function q from the set of variables of a binder to $\{\exists, \forall\}$ associates to a variable its quantifier in the binder Q. A QBF QF is in prenex form if it is constituted of a binder and a Boolean formula called the matrix and is in conjunctive normal form (CNF) if F is itself in conjuntive normal form (i.e. a conjunction of disjunctions of literals). In the following we

[1] since all the variables are different

only deal with prenex QBF. Let Σ and σ be two formulae such that σ is a sub-formula of Σ, the function $o : PROP \times PROP \rightarrow \{0,1\}^*$ is such that $o(\sigma, \Sigma)$ computes the occurrence of σ in Σ as follows: $o(\Sigma, \Sigma) = \epsilon$; $o(\sigma, \Sigma) = 0.o(\sigma, \Sigma_0)$ if $\Sigma = \neg\Sigma_0$; $o(\sigma, \Sigma) = 0.o(\sigma, \Sigma_0)$ if $\Sigma = (\Sigma_0 \circ \Sigma_1), \circ \in \mathcal{O}$ and σ is a sub-formula of Σ_0 ; $o(\sigma, \Sigma) = 1.o(\sigma, \Sigma_1)$ if $\Sigma = (\Sigma_0 \circ \Sigma_1), \circ \in \mathcal{O}$ and σ is a sub-formula of Σ_1.

Semantics of all the Boolean symbols is defined in standard way. A valuation (or Boolean interpretation) is a function from the set of variables to $BOOL$. Prepositional satisfaction is denoted $\models$ and logical equivalence is denoted $\equiv$. A model (i.e. a valuation satisfying the formula) is denoted by a set of literals ; for example, for the valuation ν defined by $\nu(x) = \mathbf{t}$, $\nu(y) = \mathbf{f}$ and $\nu(z) = \mathbf{t}$ and which satisfies the formula $((x \vee y) \leftrightarrow z)$ is denoted $\{x, \neg y, z\}$. The semantics of QBF is defined as follows: for every Boolean variable y and QBF F, a formula $(\exists y\ F) = (F[y \leftarrow \top] \vee F[y \leftarrow \bot])$ and $(\forall y\ F) = (F[y \leftarrow \top] \wedge F[y \leftarrow \bot])$. A QBF F is valid if $F \equiv \top$. If y is an existentially quantified variable preceded by the universally quantified variables $x_1, \ldots, x_n$ we denote $\hat{y}_{x_1,\ldots,x_n}$ its Skolem function from $BOOL^n$ to $BOOL$. A model for a QBF F is a sequence s of satisfying Skolem functions for F (denoted $s \models F$). For example, the QBF $\exists y \exists x \forall z((x \vee y) \leftrightarrow z)$ is not valid but the QBF $\forall z \exists y \exists x((x \vee y) \leftrightarrow z)$ is valid and its possible sequence of satisfying Skolem functions is $\hat{y}_z(\mathbf{t}) = \mathbf{t}$, $\hat{y}_z(\mathbf{f}) = \mathbf{f}$, $\hat{x}_z(\mathbf{t}) = \mathbf{f}$ and $\hat{x}_z(\mathbf{f}) = \mathbf{f}$[2]. Skolem functions are sometimes represented by policies [8] or strategies [6] which clarify them by trees ; for example the term $\{z \mapsto y; \neg x, \neg z \mapsto \neg y; \neg x\}$ is a valid policy or winning strategy corresponding to the Skolem functions $\hat{x}$ and $\hat{y}$. A (Boolean) model of an unquantified Boolean formula corresponds exactly to a (QBF) model of its existential closure ; for example for the QBF $\exists y \exists x \exists z((x \vee y) \leftrightarrow z)$, the Skolem functions $\hat{x} = \mathbf{t}$, $\hat{y} = \mathbf{f}$ and $\hat{z} = \mathbf{t}$ correspond to the Boolean model $\nu(x) = \mathbf{t}$, $\nu(y) = \mathbf{f}$ and $\nu(z) = \mathbf{t}$ for the propositional formula $((x \vee y) \leftrightarrow z)$. A QBF is valid if and only if there exists a sequence of satisfying Skolem functions. We recall that the SAT problem which decides if a Boolean formula is satisfiable or not is the canonical problem of the NP-complete class and the QBF problem which decides if a quantified Boolean formula is valid or not is the canonical problem for the PSPACE-complete class.

2.2 Answer Set Programming

Since few years, Answer Set Programming (ASP) is a very active research field involved in knowledge representation, non monotonic reasoning, logic programming and combinatorial problem resolution. In a fully declarative manner, ASP can represent a problem with a logic program of which the semantics defines a set of answers (the models of the program) encoding the solutions of the initial problem. Under this generic term of ASP, many syntactic and semantic variants have been defined. In this work we use the original stable model semantics [13] for normal logic programs.

[2] For every $b \in BOOL$, the valuation $\nu(z) = b$, $\nu(x) = \hat{x}_z(b)$, $\nu(y) = \hat{y}_z(b)$ is a (Boolean) model of $((x \vee y) \leftrightarrow z)$.

A normal logic program[3] is a finite set of rules like

$$(c \leftarrow a_1,\ \ldots,\ a_n, not\ b_1,\ \ldots, not\ b_m.)$$

$n \geq 0, m \geq 0$ where $c, a_1, \ldots, a_n, b_1, \ldots, b_m$ are atoms all gathered in the set $\mathcal{A}$ and $\mathcal{P}$ represents the set of all programs. For a rule r, we note $head(r) = c$ its head, $body^+(r) = \{a_1, \ldots, a_n\}$ its positive body and $body^-(r) = \{b_1, \ldots, b_m\}$ its negative body. The Gelfond-Lifschitz reduct of a program P by an atom set X is the program $P^X = \{(head(r) \leftarrow body^+(r).) \mid body^-(r) \cap X = \emptyset\}$. Since it has no default negation, such a program is definite and then it has a unique minimal Herbrand model denoted $Cn(P)$. By definition, a stable model of P is an atom set $S \subseteq \mathcal{A}$ such that $S = Cn(P^S)$. Let us note that a program may have no, one or many stable models. For instance, $\{(a.), (b \leftarrow a, not\ d)., (c \leftarrow a, not\ b.)\}$ has the unique stable model $\{a, b\}$, $\{(a \leftarrow not\ b.), (b \leftarrow not\ a.)\}$ has two stable models $\{a\}$ and $\{b\}$ and $\{(a.), (b \leftarrow a, not\ d.), (d \leftarrow b.)\}$ has no stable model at all and is said to be inconsistent.

To determine if a program has, or not, a stable model is an NP-complete problem. Thus, the link with the canonical NP-complete problem, SAT, has to be find easily. That is what has been introduced in [24] in the case of propositional formulae given in CNF and we recall here the proposed approach. Let Σ be a clause set. The translation of the formula produces a program $LP(\Sigma)$ containing rules $(na \leftarrow not\ a.)$ and $(a \leftarrow not\ na.)$ for every atom a occurring in Σ. For every clause in Σ, a new atom c is created and the rule $(\leftarrow not\ c.)$[4] is added to $LP(\Sigma)$. For every literal l in this clause, the rule $(c \leftarrow a.)$ if l is an atom a or the rule $(c \leftarrow na.)$ if l is the negation of an atom a, is added to $LP(\Sigma)$. By this way, Σ has a propositional model if and only if $LP(\Sigma)$ has a stable model. The reader can observe that the first rule pairs in $LP(\Sigma)$ allows to generate all possible interpretations for Σ and rules of which head is c permits to infer c if the interpretation satisfies the corresponding clause. Lastly, for every clause, constraints forbid to all sets not containing c (that are not models of Σ) to be a stable model. The first part of our contribution is an extension of this approach to all propositional formula without syntactic restriction (like CNF) and it is described in the next section.

3 From a Boolean Formula to a Normal Logic Program

We propose a (polynomial) translation of every Boolean formula in a normal logic program and prove that a formula is satisfiable if and only if the program, obtained by this translation, has a stable model.

Our result is not only an existence result but it gives the correspondence between the (Boolean) models of the formula and the stable models of the obtained program. It allows ASP solvers to be used as a tool to solve SAT problems

[3] For sake of simplicity we use program in the sequel.

[4] Such a headless rule is called a constraint and is given for a rule like $(bug \leftarrow not\ c, not\ bug.)$ where bug is a new symbol.

without any restriction on formulae (as CNF). The following definition describes the π and π^{-1} functions which associate a set of literals of a formula to a set of atoms of a program and reciprocally.

Definition 1 (π and π^{-1} functions). *Let $\pi : 2^{\mathcal{L}} \rightarrow 2^{\mathcal{A}}$ be a function such that, for every $L \in 2^{\mathcal{L}}$,*

$$\pi(L) = \{x \mid x \in L\} \cup \{nx \mid \neg x \in L\}.$$

Let $\pi^{-1} : 2^{\mathcal{A}} \times PROP \rightarrow 2^{\mathcal{L}}$ be a function such that for every $A \in 2^{\mathcal{A}}$ and every $\Sigma \in PROP$,

$$\pi^{-1}(A, \Sigma) = \{x \mid x \in A, x \in \mathcal{V}(\Sigma)\} \cup \{\neg x \mid nx \in A, x \in V(\Sigma)\}$$

The following definition describes the P_Q function which generates from a set of propositional variables a program whose stable models are in bijection with all the possible valuations of the variables.

Definition 2 (P_Q function). *Let V be a set of variables, $P_Q : 2^{\mathcal{V}} \rightarrow \mathcal{P}$ be a function such that*

$$P_Q(V) = \bigcup_{x \in V} \{\, (x \leftarrow not\ nx.), (nx \leftarrow not\ x.) \,\}$$

Theorem 1. *Let Σ be a propositional formula and ν be a set of literals. ν is a valuation of Σ if and only if $P_Q(V(\Sigma))$ has a stable model $\pi(\nu)$.*

Example 1. Let $F = ((c \vee b) \wedge (b \rightarrow ((c \rightarrow d) \wedge (c \vee (a \leftrightarrow \neg d)))))$ be a propositional formula with $\mathcal{V}(F) = \{a, b, c, d\}$ and

$$P_Q(\mathcal{V}(F)) = \left\{ \begin{array}{l} (a \leftarrow not\ na.), (na \leftarrow not\ a.), \\ (b \leftarrow not\ nb.),\ (nb \leftarrow not\ b.), \\ (c \leftarrow not\ nc.),\ (nc \leftarrow not\ c.), \\ (d \leftarrow not\ nd.), (nd \leftarrow not\ d.) \end{array} \right\}$$

For example, the set of atoms $m = \{na, b, nc, d\}$ is a stable model of the normal logic program $P_Q(\mathcal{V}(F))$ and $\pi^{-1}(m, F) = \{\neg a, b, \neg c, d\}$ is a valuation of $\mathcal{V}(F)$.

The (Boolean) models of a propositional formula are the valuations that satisfy the constraints linked to the operators constituting the formula. To obtain a program corresponding to the formula, we add to the P_Q function another function which translates those constraints (or sub-formulae). The chosen method introduces for every operator instances of the formula a new atom which represents the results of the operator on its arguments.

Definition 3 (P translation function). *Let Σ, Σ_0 and Σ_1 be three propositional formulae, x propositional variable and o an occurrence, Let $P : PROP \times \{0,1\}^* \rightarrow \mathcal{P}$ be a function defined by induction as follows:*

$$\begin{array}{lll}
\text{if } \Sigma = x & \text{then } P(\Sigma, o) = \{(s_o \leftarrow x.)\} \\
\text{if } \Sigma = \neg\Sigma_0 & \text{then } P(\Sigma, o) = \{(s_o \leftarrow not\ s_{o.0}.)\} \cup P(\Sigma_0, o.0) \\
\text{if } \Sigma = (\Sigma_0 \wedge \Sigma_1) & \text{then } P(\Sigma, o) = \{(s_o \leftarrow s_{o.0}, s_{o.1}.)\} \cup P(\Sigma_0, o.0) \cup P(\Sigma_1, o.1) \\
\text{if } \Sigma = (\Sigma_0 \vee \Sigma_1) & \text{then } P(\Sigma, o) = \{(s_o \leftarrow s_{o.0}.), (s_o \leftarrow s_{o.1}.)\} \\
 & \qquad \cup P(\Sigma_0, o.0) \cup P(\Sigma_1, o.1) \\
\text{if } \Sigma = (\Sigma_0 \rightarrow \Sigma_1) & \text{then } P(\Sigma, o) = \{(s_o \leftarrow not\ s_{o.0}.), (s_o \leftarrow s_{o.1}.)\} \\
 & \qquad \cup P(\Sigma_0, o.0) \cup P(\Sigma_1, o.1)
\end{array}$$

$$\begin{array}{ll} \textit{if } \Sigma = (\Sigma_0 \leftrightarrow \Sigma_1) & \textit{then } P(\Sigma, o) = \{(s_o \leftarrow s_{o.0}, s_{o.1}.), (s_o \leftarrow \textit{not } s_{o.0}, \textit{not } s_{o.1}.)\} \\ & \quad \cup P(\Sigma_0, o.0) \cup P(\Sigma_1, o.1) \\ \textit{if } \Sigma = (\Sigma_0 \oplus \Sigma_1) & \textit{then } P(\Sigma, o) = \{(s_o \leftarrow s_{o.0}, \textit{not } s_{o.1}.), (s_o \leftarrow \textit{not } s_{o.0}, s_{o.1}.)\} \\ & \quad \cup P(\Sigma_0, o.0) \cup P(\Sigma_1, o.1) \end{array}$$

The translation of an initial formula Σ is given by $P(\Sigma, \epsilon)$.

Example 2 (Example 1 continued).
For the propositional formula $F = ((c \vee b) \wedge (b \rightarrow ((c \rightarrow d) \wedge (c \vee (a \leftrightarrow \neg d)))))$

$$P(F, \epsilon) = \left\{ \begin{array}{l} (s_\epsilon \leftarrow s_{\epsilon.0}, s_{\epsilon.1}.), (s_{\epsilon.0} \leftarrow s_{\epsilon.0^2}.), (s_{\epsilon.0} \leftarrow s_{\epsilon.0.1}.), (s_{\epsilon.0^2} \leftarrow c.), \\ (s_{\epsilon.0.1} \leftarrow b.), (s_{\epsilon.1} \leftarrow \textit{not } s_{\epsilon.1.0}.), (s_{\epsilon.1} \leftarrow s_{\epsilon.1^2}.), (s_{\epsilon.1.0} \leftarrow b.), \\ (s_{\epsilon.1^2} \leftarrow s_{\epsilon.1^2.0}, s_{\epsilon.1^3}.), (s_{\epsilon.1^2.0} \leftarrow \textit{not } s_{\epsilon.1^2.0^2}.), (s_{\epsilon.1^2.0} \leftarrow s_{\epsilon.1^2.0.1}.), \\ (s_{\epsilon.1^2.0^2} \leftarrow c.), (s_{\epsilon.1^2.0.1} \leftarrow d.), (s_{\epsilon.1^3} \leftarrow s_{\epsilon.1^3.0}.), (s_{\epsilon.1^3} \leftarrow s_{\epsilon.1^4}.), \\ (s_{\epsilon.1^3.0} \leftarrow c.), (s_{\epsilon.1^4} \leftarrow s_{\epsilon.1^4.0}, s_{\epsilon.1^5}.), (s_{\epsilon.1^4} \leftarrow \textit{not } s_{\epsilon.1^4.0}, \textit{not } s_{\epsilon.1^5}.), \\ (s_{\epsilon.1^4.0} \leftarrow a.), (s_{\epsilon.1^5} \leftarrow \textit{not } s_{\epsilon.1^5.0}.), (s_{\epsilon.1^5.0} \leftarrow d.) \end{array} \right\}$$

From the two above definitions, we deduce in the following definition the function Π which generates from a propositional formula a normal logic program.

Definition 4 (Π function). *Let Σ be a propositional formula. Let $\Pi : PROP \rightarrow \mathcal{P}$ be a function such that $\Pi(\Sigma) = P_Q(V(\Sigma)) \cup P(\Sigma, \epsilon)$.*

The following lemma shows that for a formula Σ, the program $\Pi(\Sigma)$ does not eliminate nor adds stable model to the program $P_Q(\Sigma)$: it only adds to the stable models of $P_Q(\Sigma)$ some atoms s_o.

Lemma 1. *Let Σ be a propositional formula. The set of literals ν is a valuation for $\mathcal{V}(\Sigma)$ if and only if there exists a (unique) stable model m of $\Pi(\Sigma)$ such that $\pi(\nu) \subseteq m$.*

The following lemma shows that every intermediary atom s_o introduced to represent the result of a sub-formula is in the stable model if and only if its valuation is **t**.

Lemma 2. *Let Σ_{init} be a propositional formula, Σ be a sub-formula of Σ_{init} and m be a stable model of $P_Q(V(\Sigma_{init})) \cup P(\Sigma, \epsilon)$.*

$$s_{o(\Sigma, \Sigma_{init})} \in m \textit{ if and only if } \pi^{-1}(m, \Sigma) \models \Sigma.$$

Theorem 2. *Let Σ be a propositional formula and m a stable model of $\Pi(\Sigma)$,*

$$s_\epsilon \in m \textit{ if and only if } \pi^{-1}(m, \Sigma) \models \Sigma.$$

The result follows from the previous lemma 2 with $\Sigma = \Sigma_{init}$.

To obtain only stable models containing (or not) the atom s_ϵ, we extend the definition of function Π to Π^+ (Π^-) which includes in the program a new constraint.

Definition 5 (Π^+ and Π^- functions). *Let Σ a propositional formula. Let Π^+ and $\Pi^- : PROP \rightarrow \mathcal{P}$ be two functions such that:*

$$\Pi^+(\Sigma) = \Pi(\Sigma) \cup \{(\leftarrow not\ s_\epsilon.)\}$$
$$\Pi^-(\Sigma) = \Pi(\Sigma) \cup \{(\leftarrow s_\epsilon.)\}$$

The following corollary to the previous Theorem 2 establishes the wished result: the equivalence between the existence of a stable model of the program and the satisfiability of the propositional formula. This result is not only an existence result : to every stable model of a program corresponds a (Boolean) model to the formula and reciprocally.

Corollary 1. *Let Σ be a propositional formula, the normal logic program $\Pi^+(\Sigma)$ has a stable model if and only if Σ is satisfiable. Moreover, if m is a stable model of $\Pi^+(\Sigma)$ then $\pi^{-1}(m, \Sigma) \models \Sigma$; if ν is a (Boolean) model for Σ then there exists a (unique) stable model m of $\Pi^+(\Sigma)$ such that $\pi(\nu) \subseteq m$.*

Example 3 (Example 2 continued). The propositional formula F has eight models, for example : $\nu_1 = \{\neg a, c, d, \neg b\} \models F$. $\Pi(F)$ has sixty four stable models with eight $\{m_i\}_{1 \leq i \leq 8}$ ones which contain s_ϵ ; these eight stable models are such that $\pi^{-1}(m_i, F) \models F, 1 \leq i \leq 8$; for example the stable model

$$m_1 = \{na, nb, c, d\} \cup \{s_\epsilon, s_0, s_1, s_{0^2}, s_{1^2}, s_{1^2.0}, s_{1^2.0^2}, s_{1^2.0.1}, s_{1^3}, s_{1^3.0}, s_{1^4}\}$$

is such that $\pi^{-1}(m_1, F) = \{\neg a, \neg b, c, d\} = \nu_1$.

The translation Π^+ allows to decide not only if a propositional formula is satisfiable or not but also by the corollary 1 if a formula is a tautology or not indirectly by exhaustively (or the number) of the models. The translation Π^- allows to decide directly if a propositional formula is a tautology or not as it is expressed in the following corollary of the theorem 2.

Corollary 2. *Let Σ be a propositional formula, the normal logic program $\Pi^-(\Sigma)$ has no stable model if and only if Σ is a tautology. Moreover, if m is a stable model of $\Pi^-(\Sigma)$ then $\pi^{-1}(m, \Sigma)$ falsifies the formula Σ ; if ν falsifies the formula Σ then there exists a (unique) stable model m of $\Pi^-(\Sigma)$ such that $\pi(\nu) \subseteq m$.*

It is obvious by construction that translation Π is polynomial and modular [24]. It is also obvious that the translation is simple and may be optimised to decrease the number of introduced symbols and rules. We can see first that every occurrence of a variable introduces a new symbol (and a new rule) in the base cases of the induction : the first optimisation modifies those two base cases and mixes them with the induction cases. We can also increase the number of connectors and consider the "nand" (not and), "nor" (not or) and "non implication" connectors (the negation of the equivalence is the xor) : with this second modification combined with the previous one, negation disappears from the translated connectors ; considered as a formal system the translation is very close to the semantics tableaux of [28].

Example 4 (Example 2 continued).

The program $P(F, \epsilon)$ may be optimised in

$$\left\{ \begin{array}{l} (s_\epsilon \leftarrow s_{\epsilon.0}, s_{\epsilon.1}.), (s_{\epsilon.0} \leftarrow c.), (s_{\epsilon.0} \leftarrow b.), (s_{\epsilon.1} \leftarrow not\ b.), (s_{\epsilon.1} \leftarrow s_{\epsilon.1^2}.), \\ (s_{\epsilon.1^2} \leftarrow s_{\epsilon.1^2.0}, s_{\epsilon.1^3}.), (s_{\epsilon.1^2.0} \leftarrow not\ c.), (s_{\epsilon.1^2.0} \leftarrow d.), (s_{\epsilon.1^3} \leftarrow c.), \\ (s_{\epsilon.1^3} \leftarrow s_{\epsilon.1^4}.), (s_{\epsilon.1^4} \leftarrow a, nd.), (s_{\epsilon.1^4} \leftarrow not\ a, not\ nd.) \end{array} \right\}$$

4 From Quantified Boolean Formulae to Normal Logic Programming

We have described in the previous section the translation of a propositional formula to a normal logic program such as to compute the stable models of the program corresponds to compute the models of the (Boolean) formula. We extend these results to the quantified Boolean formulae (QBF) thanks to a translation from a QBF to a normal logic program such as to compute the stable models of the program corresponds to compute the models (satisfying Skolem functions) of the QBF. The simplest way to extend results of the previous section is to apply the semantics of the universal quantifier which explicits this quantification as a conjunction: $(\forall x\ \Sigma) = (\Sigma[x \leftarrow \top] \wedge \Sigma[x \leftarrow \bot])$ and then to apply on this formula which only contains existentially quantified variables the result obtain in propositional case (this technique is very close to [5]). The obvious price of this method is the exponential increase of the obtained propositional formula and then of the normal logic program. The other price is the exponential increase of the intermediary symbols introduced during the translation of the connectors. These two exponential increasings may be avoided by computing a first order normal logic program. Unfortunately, a final exponential increasing of the actual program treated by the ASP solvers can not be avoided since the ASP solvers are propositional ones.

The technique we present here is inspired by the Skolemization and is close to the propositional symbolic skolemization of [4] : the existentially quantified variables are replaced by functions whose arguments are the universally quantified variables which precede the existentially quantified variable in the binder. Those functions are converted in predicate symbols in the normal logic program. Two new symbols 0 and 1 are introduced and an interpretation function i from $\{0, 1\}$ to $BOOL$ such that $i(0) = \mathbf{f}$ and $i(1) = \mathbf{t}$. Definitions of π and π^{-1} are extended to Skolem functions and to n-ary predicates.

Definition 6 ($\pi_\forall$ and $\pi_\forall^{-1}$ functions for QBF). *Let $\pi_\forall : 2^{\mathcal{F}} \rightarrow 2^{\mathcal{A}}$ be a function such that, for every $sk \in 2^{\mathcal{F}}$,*

$$\begin{array}{l} \pi_\forall(sk) = \\ \{x(i^{-1}(u_1), \ldots, i^{-1}(u_n)) \mid \hat{x} \in sk, u_1, \ldots, u_n \in BOOL, \hat{x}(u_1, \ldots, u_n) = \mathbf{t}\} \cup \\ \{nx(i^{-1}(u_1), \ldots, i^{-1}(u_n)) \mid \hat{x} \in sk, u_1, \ldots, u_n \in BOOL, \hat{x}(u_1, \ldots, u_n) = \mathbf{f}\} \end{array}$$

Let $\pi_\forall^{-1} : 2^{\mathcal{A}} \times QBF \to 2^{\mathcal{F}}$ *be a function such that, for every* $A \in 2^{\mathcal{A}}$ *and every* $\Sigma \in QBF$,

$\pi_\forall^{-1}(A, \Sigma) =$
$\{\hat{x}(i(U_1), \ldots, i(U_n)) = \mathbf{t} \mid x(U_1, \ldots, U_n) \in A, q(x) = \exists, U_1, \ldots, U_n \in \{0,1\}\} \cup$
$\{\hat{x}(i(U_1), \ldots, i(U_n)) = \mathbf{f} \mid nx(U_1, \ldots, U_n) \in A, q(x) = \exists, U_1, \ldots, U_n \in \{0,1\}\}$

In the following definition, translation functions $P_Q^\forall$ and $P^\forall$ have an argument S (for "Skolem") which associates to every existentially quantified variable the number of universally quantified variables which precede it (and then the arity of the function and of the predicate symbol). The translation function $P_Q^\forall$ corresponds to the treatment of the quantifiers: the existential quantification is treated in a similar way to the propositional case since the universal quantification introduces a rule expliciting the constant 1 as interpreted to $\mathbf{t}$ and the semantics of the quantifier as a conjunction (the interpretation of the constant 0 as $\mathbf{f}$ is not explicited since it is not useful, this fact illustrates the disymmetry in the normal logic program). The intermediary symbols introduced in the propositional case are considered for the QBF case as existentially quantified intern variables coming from the decomposition by introduction of existentially quantified variables as in [29]. As existentially quantified variables, they are also skolemized : every intermediary symbol has as many arguments as universally quantified variables in the QBF.

Definition 7 ($P_Q^\forall$, $P^\forall$ and $\Pi_\forall$ functions). *Let* Σ, Σ_0 *and* Σ_1 *be QBF,* V *a set of variables,* n *and* $N_\forall$ *in* $\mathbb{N}$, o *an occurrence and* S *a function from* $\mathcal{V}$ *to* $\mathbb{N}$. *Let* $P_Q^\forall : QBF \times \mathbb{N} \times \mathbb{N} \times (\mathcal{V} \to \mathbb{N}) \to \mathcal{P}$ *be a function such that*

if $\Sigma = (\exists x\ \Sigma_0)$ *then* $P_Q^\forall(\Sigma, N_\forall, n, S) =$
$P_Q^\forall(\Sigma_0, N_\forall, n, S \cup \{(x, n)\}) \cup$
$\{(x(U_1, \ldots, U_n) \leftarrow not\ nx(U_1, \ldots, U_n).), (nx(U_1, \ldots, U_n) \leftarrow not\ x(U_1, \ldots, U_n).)\}$
if $\Sigma = (\forall x\ \Sigma_0)$ *then* $P_Q^\forall(\Sigma, N_\forall, n, S) =$
$P_Q^\forall(\Sigma_0, N_\forall, n+1, S) \cup$
$\{(x(U_1, \ldots, U_{N_\forall}) \leftarrow U_{n+1} = 1.)\} \cup$
$\{(s_{0^n}(U_1, \ldots, U_n) \leftarrow s_{0^{n+1}}(U_1, \ldots, U_n, 0), s_{0^{n+1}}(U_1, \ldots, U_n, 1).)\}$
if Σ *is a propositional formula then*
$P_Q^\forall(\Sigma, N_\forall, n, S) = P^\forall(\Sigma, 0^{N_\forall}, N_\forall, S)$

Let $P^\forall : PROP \times \{0,1\}^* \times \mathbb{N} \times (\mathcal{V} \to \mathbb{N}) \to \mathcal{P}$ *be a function such that*

Existentially quantified variable considered as a formula
if $\Sigma = x$ *and* $(x, n) \in S$ *then* $P^\forall(\Sigma, o, N_\forall, S) =$
$\{(s_o(U_1, \ldots, U_{N_\forall}) \leftarrow x(U_1, \ldots, U_n).)\}$
Universally quantified variable considered as a formula
if $\Sigma = x$ *and* $(x, n) \notin S$ *then* $P^\forall(\Sigma, o, N_\forall, S) =$
$\{(s_o(U_1, \ldots, U_{N_\forall}) \leftarrow x(U_1, \ldots, U_{N_\forall}).)\}$

$$
\begin{array}{l}
\textit{if } \Sigma = \neg\Sigma_0 \textit{ then } P^{\forall}(\Sigma, o, N_\forall, S) = \\
\left\{ (s_o(U_1, \ldots, U_{N_\forall}) \leftarrow \textit{not } s_{o.0}(U_1, \ldots, U_{N_\forall}).) \right\} \cup \\
P^{\forall}(\Sigma_0, o.0, N_\forall, S) \\
\textit{if } \Sigma = (\Sigma_0 \wedge \Sigma_1) \textit{ then } P^{\forall}(\Sigma, o, N_\forall, S) = \\
\{(s_o(U_1, \ldots, U_{N_\forall}) \leftarrow s_{o.0}(U_1, \ldots, U_{N_\forall}), s_{o.1}(U_1, \ldots, U_{N_\forall}).)\} \cup \\
P^{\forall}(\Sigma_0, o.0, N_\forall, S) \cup P^{\forall}(\Sigma_1, o.1, N_\forall, S) \\
\textit{if } \Sigma = (\Sigma_0 \vee \Sigma_1) \textit{ then } P^{\forall}(\Sigma, o, N_\forall, S) = \\
\left\{ \begin{array}{l} (s_o(U_1, \ldots, U_{N_\forall}) \leftarrow s_{o.0}(U_1, \ldots, U_{N_\forall}).), \\ (s_o(U_1, \ldots, U_{N_\forall}) \leftarrow s_{o.1}(U_1, \ldots, U_{N_\forall}).) \end{array} \right\} \cup \\
P^{\forall}(\Sigma_0, o.0, N_\forall, S) \cup P^{\forall}(\Sigma_1, o.1, N_\forall, S) \\
\textit{if } \Sigma = (\Sigma_0 \rightarrow \Sigma_1) \textit{ then } P^{\forall}(\Sigma, o, N_\forall, S) = \\
\left\{ \begin{array}{l} (s_o(U_1, \ldots, U_{N_\forall}) \leftarrow \textit{not } s_{o.0}(U_1, \ldots, U_{N_\forall}).), \\ (s_o(U_1, \ldots, U_{N_\forall}) \leftarrow s_{o.1}(U_1, \ldots, U_{N_\forall}).) \end{array} \right\} \cup \\
P^{\forall}(\Sigma_0, o.0, N_\forall, S) \cup P^{\forall}(\Sigma_1, o.1, N_\forall, S) \\
\textit{if } \Sigma = (\Sigma_0 \leftrightarrow \Sigma_1) \textit{ then } P^{\forall}(\Sigma, o, N_\forall, S) = \\
\left\{ \begin{array}{l} (s_o(U_1, \ldots, U_{N_\forall}) \leftarrow s_{o.0}(U_1, \ldots, U_{N_\forall}), s_{o.1}(U_1, \ldots, U_{N_\forall}).), \\ (s_o(U_1, \ldots, U_{N_\forall}) \leftarrow \textit{not } s_{o.0}(U_1, \ldots, U_{N_\forall}), \textit{not } s_{o.1}(U_1, \ldots, U_{N_\forall}).) \end{array} \right\} \cup \\
P^{\forall}(\Sigma_0, o.0, N_\forall, S) \cup P^{\forall}(\Sigma_1, o.1, N_\forall, S) \\
\textit{if } \Sigma = (\Sigma_0 \oplus \Sigma_1) \textit{ then } P^{\forall}(\Sigma, o, N_\forall, S) = \\
\left\{ \begin{array}{l} (s_o(U_1, \ldots, U_{N_\forall}) \leftarrow s_{o.0}(U_1, \ldots, U_{N_\forall}), \textit{not } s_{o.1}(U_1, \ldots, U_{N_\forall}).), \\ (s_o(U_1, \ldots, U_{N_\forall}) \leftarrow \textit{not } s_{o.0}(U_1, \ldots, U_{N_\forall}), s_{o.1}(U_1, \ldots, U_{N_\forall}).) \end{array} \right\} \cup \\
P^{\forall}(\Sigma_0, o.0, N_\forall, S) \cup P^{\forall}(\Sigma_1, o.1, N_\forall, S)
\end{array}
$$

Let Σ be a QBF and $N_\forall$ the number of universally quantified variables of Σ. Let $\Pi_\forall(\Sigma) : QBF \rightarrow \mathcal{P}$ be a function such that $\Pi_\forall(\Sigma) = P^{\forall}_{\sigma}(\Sigma, N_\forall, 0, \emptyset)$.

By construction the translation is polynomial and modular.

Example 5 (Example 3 continued). Let $F_{\exists a \forall b \exists c \forall d} = \exists a \forall b \exists c \forall d F$ be a QBF with $F = ((c \vee b) \wedge (b \rightarrow ((c \rightarrow d) \wedge (c \vee (a \leftrightarrow \neg d)))))$. We obtain

$$
\begin{array}{l}
\Pi^{\forall}(F_{\exists a \forall b \exists c \forall d}) = P_{\exists a \forall b \exists c \forall d} \cup \\
\left\{ \begin{array}{ll}
(a \leftarrow \textit{not } na.), & (na \leftarrow \textit{not } a.), \\
(b(U_1, U_2) \leftarrow U_1 = 1.), & (s_\epsilon \leftarrow s_0(0), s_0(1).), \\
(c(U_1) \leftarrow \textit{not } nc(U_1).), & (nc(U_1) \leftarrow \textit{not } c(U_1).), \\
(d(U_1, U_2) \leftarrow U_2 = 1.), & (s_0(U_1) \leftarrow s_{0^2}(U_1, 0), s_{0^2}(U_1, 1).)
\end{array} \right\}
\end{array}
$$

with $P_{\exists a \forall b \exists c \forall d}$ equal to $P(F, 0^2)$ in which all s_o (resp. b, c and d) are replaced by $s_o(U_1, U_2)$ (resp. $b(U_1, U_2)$, $c(U_1)$ and $d(U_1, U_2)$) and with intermediary call $P^{\forall}(F, 2, 2, \{(a \mapsto 0), (c \mapsto 1)\})$.

Let $F_{\forall a \exists b \forall c \exists d} = \forall a \exists b \forall c \exists d F$ be a QBF. We obtain

$$
\begin{array}{l}
\Pi^{\forall}(F_{\forall a \exists b \forall c \exists d}) = P_{\forall a \exists b \forall c \exists d} \cup \\
\left\{ \begin{array}{ll}
(a(U_1, U_2) \leftarrow U_1 = 1.), & (s_\epsilon \leftarrow s_0(0), s_0(1).), \\
(b(U_1) \leftarrow \textit{not } nb(U_1).), & (nb(U_1) \leftarrow \textit{not } b(U_1).), \\
(c(U_1, U_2) \leftarrow U_2 = 1.), & (s_0(U_1) \leftarrow s_{0^2}(U_1, 0), s_{0^2}(U_1, 1).), \\
(d(U_1, U_2) \leftarrow \textit{not } nd(U_1, U_2).), & (nd(U_1, U_2) \leftarrow \textit{not } d(U_1, U_2).)
\end{array} \right\}
\end{array}
$$

with $P_{\forall a \forall b \forall c \exists d}$ equal to $P(F, 0^2)$ in which all s_o (resp. a, b, c and d) are replaced by $s_o(U_1, U_2)$ (resp. $a(U_1, U_2)$, $b(U_1)$, $c(U_1, U_2)$ and $d(U_1, U_2)$) and with intermediary call $P^{\forall}(F, 2, 2, \{(b \mapsto 1), (d \mapsto 2)\})$.

Theorem 3. *Let Σ be a QBF and m be a stable model of $\Pi_{\forall}(\Sigma)$.*

$$s_\epsilon \in m \text{ if and only if } \Sigma \text{ is valid.}$$

Example 6 (Example 5 continued). The QBF $F_{\exists a \forall b \exists c \forall d} = \exists a \forall b \exists c \forall d F$ has no satisfying Skolem function and $\Pi_{\forall}(F_{\exists a \forall b \exists c \forall d})$ has no stable model.

In the same way the function Π is extended to the function Π^+ (resp. Π^-) to restrict the stable models corresponding to the (Boolean) models (resp. corresponding to the valuations which falsify the formula), we define the function ${\Pi_{\forall}}^+$ (resp. ${\Pi_{\forall}}^-$) as an extension of the function $\Pi_{\forall}$ to restrict the Skolem functions to those which satisfy (resp. not satisfy) the QBF.

Definition 8 ($\Pi^+_{\forall}$ and $\Pi^-_{\forall}$ functions). *Let Σ be a QBF. Let $\Pi^+_{\forall}$ and $\Pi^-_{\forall}$: $QBF \rightarrow \mathcal{P}$ be functions such that*

$$\Pi^+_{\forall}(\Sigma) = \Pi_{\forall}(\Sigma) \cup \{(\leftarrow not\ s_\epsilon.)\}$$
$$\Pi^-_{\forall}(\Sigma) = \Pi_{\forall}(\Sigma) \cup \{(\leftarrow s_\epsilon.)\}$$

As for the corollary 1 of the theorem 2 which computes the (Boolean) model of a propositional formula thanks to the stable models of an associated normal logic program, the following corollary to the theorem 3 allows to extract from the stable models of an associated normal logic program the Skolem functions which satisfy a QBF.

Corollary 3. *Let Σ be a QBF, the normal logic program $\Pi^+_{\forall}(\Sigma)$ has a stable model if and only if Σ is valid. Moreover, if m is a stable model of $\Pi^+_{\forall}(\Sigma)$ then $\pi^{-1}_{\forall}(m, \Sigma)$ is a set of Skolem functions which satisfy the QBF ; if sk is a set of Skolem functions which satisfy Σ then there exists a (unique) stable model m of $\Pi^+_{\forall}(\Sigma)$ such that $\pi_{\forall}(sk) \subseteq m$.*

Example 7 (Example 6 continued). The QBF $F_{\forall a \exists b \forall c \exists d} = \forall a \exists b \forall c \exists d F$ has a (unique) set of satisfying Skolem functions:

$$sk = \left\{ \begin{array}{l} \hat{b}_a(\mathbf{t}) = \mathbf{t}, \hat{b}_a(\mathbf{f}) = \mathbf{t}, \\ \hat{d}_{ac}(\mathbf{t}, \mathbf{t}) = \mathbf{t}, \hat{d}_{ac}(\mathbf{t}, \mathbf{f}) = \mathbf{f}, \hat{d}_{ac}(\mathbf{f}, \mathbf{t}) = \mathbf{t}, \hat{d}_{ac}(\mathbf{f}, \mathbf{f}) = \mathbf{t} \end{array} \right\}$$

which corresponds to the valid policy or winning strategy

$$\left\{ \begin{array}{l} a \mapsto b; \left\{ \begin{array}{l} c \mapsto d, \\ \neg c \mapsto \neg d \end{array} \right\}, \\ \neg a \mapsto b; \left\{ \begin{array}{l} c \mapsto d, \\ \neg c \mapsto d \end{array} \right\} \end{array} \right\}$$

and $\Pi^+_\forall(F_{\forall a \exists b \forall c \exists d})$ has a (unique) stable model

$$
\begin{array}{l}
m = \\
\{b(1), b(0), d(1,1), d(0,1), d(0,0), nd(1,0)\} \cup \\
\{a(1,0), a(1,1), c(0,1), c(1,1)\} \cup \\
\{s(0), s(1), s_0(0,0), s_0(1,0), s_0(0,1), s_0(1,1)\} \cup \\
\{\, s_o(A,C) \mid A, C \in \{0,1\}, o \in \{0^3, 0^2.1, 0^2.1^2, 0^2.1^2.0, 0^2.1^3\}\,\} \cup \\
\{s_{0^2.1^4}(0,0), s_{0^2.1^4}(1,0), s_{0^2.1^4}(0,1)\}
\end{array}
$$

So, we have $\pi^{-1}_\forall(m, F_{\forall a \exists b \forall c \exists d}) = sk$ and $\pi_\forall(sk) \subseteq m$.

Since the symbols s and s_0 represent the semantics of the universal quantification for the variables a and c, it is necessary to all their instances to be in the stable model. Only the atom $s_{0^2.1^4}(1,1)$ is not in the stable model since it represents the sub-formula $(a \leftrightarrow \neg d)$ for the variable a interpreted to $\mathbf{t}$ (the first 1 of $s_{0^2.1^4}(1,1)$) and the variable d interpreted to $\mathbf{t}$ (since the atom $d(1,1)$ is in the stable model) and have to be interpreted to $\mathbf{f}$.

5 Experimental Results

We want to evaluate in that section if our approach represent an efficient alternative to compute the (Boolean) model of a propositional formula or the satisfying Skolem functions of a QBF. We have selected four ASP solvers: dlv [16], clasp [12], noMoRe++ [1] and smodels [25]. We first study the result of our approach on propositional formulae and then on QBF.

5.1 From SAT to ASP

We have chosen to evaluate and compare our approach in propositional case on the set of QG6 problems [22] which has the advantage to be in CNF format (from 1301 to 2109 variables and from 6089 to 9964 clauses) and non-CNF format (either 252 or 324 variables and from 3532 to 7850 disjunctions or conjunctions) ; there are 83 satisfiable instances and 173 unsatisfiable instances. We compare the execution time and the percentage of successful runs of ASP solvers on the result of our translations applied to the SAT non-CNF instances and those of the SAT non-CNF solver Satmate [15] (the only one which admits the file format of the set of problems QG6). We include also in the comparison the execution time and percentage of successful runs for the SAT CNF solvers zchaff [23] and minisat [9].

The experiments have been realized on a $2 \times Intel\ Xeon\ CPU\ 2.80Ghz$ with $2GB$ of memory. The following table shows if the problems are satisfiable (SAT) or not (UNSAT), the number of solved problems (NbR), the average execution time in seconds (ATR) and the average execution time in seconds when the run succeeds (ATRS).

	SAT CNF		SAT non CNF	ASP			
QG6 SAT	zchaff	minisat	satmate	smodels	nomore	dlv	clasp
NbR	83	83	83	67	83	78	83
% solved	100,00%	100,00%	100,00%	80,72%	100,00%	93,98%	100,00%
ATR	19,39	3,25	4,31	857,98	118,98	416,48	11,98
ATRS	19,39	3,25	4,31	203,17	118,98	212,40	11,98
QG6 UNSAT	zchaff	minisat	satmate	smodels	nomore	dlv	clasp
NbR	137	165	152	88	77	83	154
% solved	79,19%	95,38%	87,86%	50,87%	44,51%	47,98%	89,02%
ATR	1260,08	680,15	501,18	1912,02	2067,62	1934,17	711,97
ATRS	645,21	538,58	73,05	281,58	157,12	127,86	355,66

Our approach associated with the ASP solver Clasp is really efficient on this set of problems since it does better than the SAT non-CNF solver Satmate (dedicated to this set of problems) and Zchaff (on the CNF benchmarks) ; only Minisat is better in number of solved problems and execution time.

5.2 From QBF to ASP

In the case of QBF, we only evaluate our approach on the set of benchmarks of the QBFEVAL07 evaluation[5] ; all the instances have an $\exists\forall$ alternation of quantifiers and only negation, conjunction and disjunction connectors. The result have been obtained for a 'Core Duo T2400" with $3GB$ of memory. The following table shows the instance, the number of existentially/universally quantified variables (NbV), the execution time for Smodels and for Clasp, the number of choice points given by Smodels, the number of atoms and the number of rules of the normal logic program.

Instance	NbV	Smodels	Clasp	Choice Points	Nb atoms	Nb rules
counter5_2	15/10	0,140	0,152	0	4 584	5 746
counter6_2	18/12	0,548	0,508	0	15 584	19 876
counter7_2	21/14	2,968	3,000	0	56 734	73 382
counter8_2	24/16	23,093	24,434	0	215 082	280 960
counter5_4	25/20	105,751	592.725	0	3 149 672	4 198 500
ring4_2	21/14	1,932	1,520	10	52 180	68 590
ring5_2	24/16	12,985	8,585	9	203 698	269 314
ring6_2	27/18	94,086	64,084	9	803 989	1 066 267
semaphore_2	21/14	2,364	1,604	5	52 862	69 458
semaphore3_2	27/18	130.164	58,476	5	798 972	1 061 495

It is worth to note that when the memory is insufficient or the execution time limit (1 hour) is exceeded it is always due to the grounding step. Indeed all the current ASP solvers admit for input first order programs but they use a "front-end" (external like Lparse or internal for DLV) ; thus, the replacement of the variables by constants (in our case 0 and 1) leads to program of exponential size.

[5] `http://www.qbflib.org/`

6 Conclusion

In this work, we have generalized a translation from SAT to ASP allowing to use any existing ASP solver as a SAT solver for every kind of propositional formula. Experiments show that our approach is realistic for non trivial problems. An interesting and complementary point would be to study how an ASP solver (like ASSAT [20] or Cmodels [19]) that computes stable models by calling an underlying SAT solver (inverse approach to ours) deals with our programs. For QBF, the lack of various benchmarks does not allow us to realize a deep evaluation of the performance of our proposal. Indeed, we know that space complexity is a central problem. But, the theoretical concepts that we have introduced here are new and provide QBF researchers with a practical tool that can be useful to compute solutions. One of the challenge for ASP community is to be able to deal with very large programs, maybe by escaping the instanciation phase. If such a goal is reached it could be of great interest for our methodology.

References

1. C. Anger, M. Gebser, T. Linke, A. Neumann, and T. Schaub. The nomore++ system. In *LPNMR*, pages 422–426, 2005.
2. C. Ansotegui, C. Gomes, and B. Selman. Achilles' heel of QBF. In *Nat. Conf. on Artificial Intelligence (AAAI)*, 2005.
3. A. Ayari and D. Basin. Qubos: Deciding quantified boolean logic using propositional satisfiability solvers. In *Formal Methods in Computer-Aided Design, Fourth International Conference, FMCAD 2002*. Springer-Verlag, 2002.
4. M. Benedetti. Evaluating QBFs via Symbolic Skolemization. In *Proc. of the 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR04)*, number 3452 in LNCS. Springer, 2005.
5. A. Biere. Resolve and expand. In *7th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'04)*, 2004.
6. L. Bordeaux. Boolean and interval propagation for quantified constraints. In *First International Workshop on Quantification in Constraint Programming*, 2005.
7. M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An algorithm to evaluate quantified boolean formulae and its experimental evaluation. *Journal of Automated Reasoning*, 28(2):101–142, 2002.
8. S. Coste-Marquis, H. Fargier, J. Lang, D. Le Berre, and P. Marquis. Representing policies for quantified boolean formulae. In *KR06*, 2006.
9. N. Een and N. Srensson. An extensible sat-solver. In *6th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'03)*, 2003.
10. Uwe Egly, Martina Seidl, and Stefan Woltran. A solver for QBFs in nonprenex form. In *ECAI*, pages 477–481, 2006.
11. G. Parthasarathy F. Lu, M. K. Iyer and K.-T. Cheng. An efficient sequential sat solver with improved search strategies. In *Design, Automation and Test in Europe (DATE)*, 2005.
12. M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In *IJCAI*, pages 386–, 2007.

13. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.
14. E. Giunchiglia, M. Narizzano, and A. Tacchella. Backjumping for quantified boolean logic satisfiability. *Artificial Intelligence*, 145:99–120, 2003.
15. H. Jain, C. Bartzis, and E. Clarke. Satisfiability checking of non-clausal formulas using general matings. In *9th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'06)*, 2006.
16. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The dlv system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.
17. R. Letz. Lemma and model caching in decision procedures for quantified boolean formulas. In *TABLEAUX*, pages 160–175, 2002.
18. C.-M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *IJCAI*, pages 366–371, 1997.
19. Yuliya Lierler. Cmodels for tight disjunctive logic programs. In Armin Wolf, Thom W. Frühwirth, and Marc Meister, editors, *W(C)LP*, volume 2005-01 of *Ulmer Informatik-Berichte*, pages 163–166. Universität Ulm, Germany, 2005.
20. F. Lin and Y. Zhao. Assat: computing answer sets of a logic program by sat solvers. *Artif. Intell.*, 157(1-2):115–137, 2004.
21. F. Lu, L.-C. Wang, J. Moondanos, and Z. Hanna. A signal correlation guided circuit-sat solver. *Journal of Universal Computer Science*, 10(12):1629–1654, 2004.
22. A. Meier and V. Sorge. Applying sat solving in classification of finite algebras. *J. Autom. Reason.*, 35(1-3):201–235, 2005.
23. M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.
24. I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999.
25. I. Niemela and P. Simons. Evaluating an algorithm for default reasoning. In *Workshop on Applications and Implementations of Nonmonotomic Reasonigs Systems*, 1995.
26. J. Rintanen. Improvements to the evaluation of quantified boolean formulae. In *IJCAI*, pages 1192–1197, 1999.
27. H. Samulowitz and F. Bacchus. Binary clause reasoning in QBF. In *9th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'06)*, pages 353–367, 2006.
28. R.M. Smullyan. *First Order Logic*. Springer Verlag, 1969.
29. I. Stéphan. Boolean propagation based on literals for quantified boolean formulae. In *17th European Conference on Artificial Intelligence*, 2006.
30. L.J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.
31. C. Thiffault, F. Bacchus, and T. Walsh. Solving non-clausal formulas with dpll search. In *7th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'04)*, 2004.
32. L. Zhang. Solving QBF with combined conjunctive and disjunctive normal form. In *National Conference on Artificial Intelligence (AAAI 2006)*, 2006.
33. L. Zhang and S. Malik. Conflict driven learning in a quantified boolean satisfiability solver. In *International Conference on Computer Aided Design (ICCAD2002)*, 2002.