From (Quantified) Boolean Formulae to Answer Set Programming

Igor Stéphan Benoit Da Mota
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 known as the canonical problem of the NP-complete complexity class. Many decision procedures have been proposed, mainly for the conjunctive normal form (CNF) [22, 26, 9]. Some recent works show that the CNF transformation seems to disrupt too much the original structure of the problem [36]. However, there exist only few implementations of solvers for non-CNF SAT formulae [36, 19, 24, 13].

In the same way, the problem of validity of quantified Boolean formulae (QBF) [34] 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 [30, 17, 7, 21, 5, 31, 4]. For QBF, the impact of the transformation in prenex CNF formulae seems even larger [2]. As for SAT solvers, there exist very few QBF solvers for (non prenex) non-CNF formulae [3, 2, 38, 12, 37].

In both cases (SAT or QBF), the encoding of combinatorial problems, or more generally knowledge representation in artificial intelligence, are usually natural, direct and compact when an unrestricted syntax is allowed. That is why in this article we propose to deal with satisfiability for non CNF (quantified) Boolean formulae.

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 tools which have proved their efficiency and their robustness on large problems.

A polynomial translation from SAT to ASP has already been proposed but only for CNF formulae [27] or a particular case [18]. In this article, we formalize a translation that can be applied to every kind of propositional formula and then, we extend it to quantified Boolean formulae. 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 quantified Boolean formulae. We complete this study in section 5 by presenting the tools that we have developped to practically realize our translations from SAT or QBF to ASP. Furthermore, we give some experimental results illustrating that the best ASP solvers may be used to compute the models of (quantified) Boolean formulae in an arbitrary syntax and not just in conjunctive normal form which is the case for most of the available QBF solvers. The section 6 concludes our work by relating it to some others and discussing some points about efficiency. The proofs of all theoretical results enonced in our work are grouped in the special section 7. But first, we start with a section that introduces the necessary materials to understand this work.

2 Preliminaries

The aim of the two following subsections is to recall the formal definitions necessary for the understanding of our work.

2.1 (Quantified) Boolean Formulae

The Boolean values are denoted with $\mathbf{t}$ and $\mathbf{f}$, the set of Boolean values is denoted with $BOOL$ and the set of Boolean functions (i.e. functions from $BOOL^n$ to $BOOL$) is denoted with $\mathcal{F}$. The set of propositional symbols (or variables) is denoted with $\mathcal{V}$. 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 with $\mathcal{L}$. 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 $*$ is an element of $\mathcal{O}$ then $(F * G)$ is an element of $PROP$. The symbol $\exists$ is used for existential quantification and $\forall$ for universal quantification (q is

used in place of $\exists$ and $\forall$). The set of quantified Boolean formula (QBFs) 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 with $\mathcal{V}(F)$. 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 with $G[x \leftarrow F]$, as the formula obtained by replacing in G all the occurrences[1] of variable x by the formula F. A binder Q is a string $q_1x_1 \dots q_nx_n$ with $x_1, \dots, x_n$ distinct variables and $q_1, \dots, q_n$ quantifiers. The function $\mathcal{Q}$ 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. A QBF QF is in conjunctive normal form (CNF) if F is itself in conjunctive normal form (i.e. a conjunction of disjunctions of literals). In the following, we only deal with prenex QBFs. We define an occurrence $o \in \{0,1\}^*$ of a formula in another one as follows: ϵ is the occurrence of Σ in itself; if o is an occurrence of σ in Σ_0 then $0.o$ is an occurrence of σ in $\neg\Sigma_0$; if o is an occurrence of σ in Σ_0 then $0.o$ is an occurrence of σ in $(\Sigma_0 * \Sigma_1)$, $* \in \mathcal{O}$ and σ is a sub-formula of Σ_0; if o is an occurrence of σ in Σ_1 then $1.o$ is an occurrence of σ in $(\Sigma_0 * \Sigma_1)$, $* \in \mathcal{O}$ and σ is a sub-formula of Σ_1. As usual $\epsilon.o$ is simplified in o.

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 $BOOL$. Propositional satisfaction is denoted with $\models$ and logical equivalence is denoted with $\equiv$. A model (i.e. a valuation satisfying a formula) is denoted with a set of literals; for example, 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 with $\{x, \neg y, z\}$. The semantics of QBF is defined as follows: for every Boolean variable y and QBF F, $(\exists y\ F) = (F[y \leftarrow \top] \vee F[y \leftarrow \bot])$ 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 preceeded by the universally quantified variables $x_1, \dots, x_n$ we denote $\hat{y}_{x_1,\dots,x_n}$ 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 with $s \models F$). 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

[1] since all the variables are different

[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 $((x \vee y) \leftrightarrow z)$.

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 whose semantics defines a set of answers (the models of the program) encoding the solutions of a given problem. Under the generic term of ASP, many syntactic and semantic variants have been defined. In this work we use the original stable model semantics [16] for normal logic programs.

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 $\mathcal{A}$; $\mathcal{P}$ 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 with $Cn(P)$. 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, and then the relation between ASP and the canonical NP-complete problem, SAT, has already been studied. For instance, we recall here the approach introduced in [27] in the case of propositional formulae given in CNF. 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 $LP(\Sigma)$. In 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 $LP(\Sigma)$ allow one to generate all possible interpretations for Σ and rules whose head is c permit to infer c if the interpretation satisfies the corresponding clause. Lastly, for

[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.

every clause, constraints forbid 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 formulae 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 into 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 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 $\Sigma \in PROP$,

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

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*

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

Theorem 1. *Let Σ be a propositional formula and ν be a set of literals. ν is a valuation of $\mathcal{V}(\Sigma)$ if and only if $P_Q(\mathcal{V}(\Sigma))$ 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. $\mathcal{V}(F) = \{a, b, c, d\}$ and*

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

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 valuations satisfying the constraints linked to the operators constituting the formula. To obtain a program corresponding to the formula, we add to 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 *a propositional variable and* o *an occurrence. Let* $P : PROP \times \{0,1\}^* \rightarrow \mathcal{P}$ *be a function defined by induction as follows:*

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

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

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

$$
P(F, \epsilon) = \left\{
\begin{array}{l}
(s_\epsilon \leftarrow s_0, s_1.), (s_0 \leftarrow s_{0^2}.), (s_0 \leftarrow s_{0.1}.), (s_{0^2} \leftarrow c.), \\
(s_{0.1} \leftarrow b.), (s_1 \leftarrow \textit{not } s_{1.0}.), (s_1 \leftarrow s_{1^2}.), (s_{1.0} \leftarrow b.), \\
(s_{1^2} \leftarrow s_{1^2.0}, s_{1^3}.), (s_{1^2.0} \leftarrow \textit{not } s_{1^2.0^2}.), (s_{1^2.0} \leftarrow s_{1^2.0.1}.), \\
(s_{1^2.0^2} \leftarrow c.), (s_{1^2.0.1} \leftarrow d.), (s_{1^3} \leftarrow s_{1^3.0}.), (s_{1^3} \leftarrow s_{1^4}.), \\
(s_{1^3.0} \leftarrow c.), (s_{1^4} \leftarrow s_{1^4.0}, s_{1^5}.), (s_{1^4} \leftarrow \textit{not } s_{1^4.0}, \textit{not } s_{1^5}.), \\
(s_{1^4.0} \leftarrow a.), (s_{1^5} \leftarrow \textit{not } s_{1^5.0}.), (s_{1^5.0} \leftarrow d.)
\end{array}
\right\}
$$

From the two above definitions, we introduce 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* $\Pi(\Sigma) = P_Q(\mathcal{V}(\Sigma)) \cup P(\Sigma, \epsilon)$.

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 intermediate 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 $P_Q(\mathcal{V}(\Sigma_{init})) \cup P(\Sigma, \epsilon)$. For every occurrence o of Σ in Σ_{init}*

$$s_o \in m \text{ if and only if } \pi^{-1}(m, \Sigma) \models \Sigma.$$

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

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

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 $\Pi^- : PROP \to \mathcal{P}$ be two functions such that:*

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

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 of 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 one 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 computing the models (or their number). 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 $\pi(\nu) \subseteq m$.*

Let us remark that if the formula Σ contains n different propositional variables, m occurrences of these variables and p operators then $\Pi(\Sigma)$ contains at most $2n + m + 2p$ rules. So, our translation Π is polynomial with respect the length of the given formula. Moreover, as introduced in [27] we can say that Π is modular in the sense that the translation of a formula can be locally computed. Indeed, let us suppose that we have already computed the program $\Pi(\Sigma)$ and decide to deal with a new formula as $(\Sigma * \Sigma'), * \in \mathcal{O}$. Then, we just have to compute $\Pi(\Sigma')$, join it with $\Pi(\Sigma)$ and add the one or two rules necessary to encode the operator $*$ to obtain the whole desired translation[5]. The important point is that we do not have to recompute anything else for Σ.

It is also obvious that the translation is simple and may be optimized 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 induction : a first optimization 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 closed to the semantic tableaux of [32].

Example 4 (Example 2 continued)**.** *The program $P(F, \epsilon)$ may be optimised in*

$$\left\{ \begin{array}{l} (s_\epsilon \leftarrow s_0, s_1.), (s_0 \leftarrow c.), (s_0 \leftarrow b.), (s_1 \leftarrow not\ b.), (s_1 \leftarrow s_{1^2}.), \\ (s_{1^2} \leftarrow s_{1^2.0}, s_{1^3}.), (s_{1^2.0} \leftarrow not\ c.), (s_{1^2.0} \leftarrow d.), (s_{1^3} \leftarrow c.), \\ (s_{1^3} \leftarrow s_{1^4}.), (s_{1^4} \leftarrow a, nd.), (s_{1^4} \leftarrow not\ a, not\ nd.) \end{array} \right\}$$

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

[5] taking into account that we have to use a new symbol $s' \neq s$ in $\Pi(\Sigma')$

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 makes this quantification explicit 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 obtained in the 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 number of intermediate 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 cannot be avoided since the actual 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 them in the binder. Those functions are converted into 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 $sk \in 2^{\mathcal{F}}$,*

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

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

$$\begin{array}{l}\pi_\forall^{-1}(A, \Sigma) = \\ \quad \{\hat{x}(i(U_1), \ldots, i(U_n)) = \mathbf{t} \mid x(U_1, \ldots, U_n) \in A, \mathcal{Q}(x) = \exists, U_1, \ldots, U_n \in \{0,1\}\} \\ \cup \{\hat{x}(i(U_1), \ldots, i(U_n)) = \mathbf{f} \mid nx(U_1, \ldots, U_n) \in A, \mathcal{Q}(x) = \exists, U_1, \ldots, U_n \in \{0,1\}\}\end{array}$$

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 asymmetry in the non monotonic logic program, that deals with negation by the absence

of something instead of its explicit representation). The intermediate symbols introduced in the propositional case are considered for the QBF case as existentially quantified variables coming from the decomposition by introduction of existentially quantified variables as in [33]. As existentially quantified variables, they are also skolemized : every intermediate symbol has as many arguments as universally quantified variables in the QBF.

Definition 7 ($P^{\forall}_Q$, $P^{\forall}$ 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 $P^{\forall}_Q : QBF \times \mathbb{N} \times \mathbb{N} \times (\mathcal{V} \to \mathbb{N}) \to \mathcal{P}$ be a function such that*

$$
\begin{array}{l}
\textit{if } \Sigma = (\exists x\ \Sigma_0) \textit{ then } P^{\forall}_Q(\Sigma, N_{\forall}, n, S) = P^{\forall}_Q(\Sigma_0, N_{\forall}, n, S \cup \{(x \mapsto n)\}) \\
\quad \cup \{(x(U_1, \ldots, U_n) \leftarrow \textit{not } nx(U_1, \ldots, U_n).), \\
\qquad (nx(U_1, \ldots, U_n) \leftarrow \textit{not } x(U_1, \ldots, U_n).)\} \\
\textit{if } \Sigma = (\forall x\ \Sigma_0) \textit{ then } P^{\forall}_Q(\Sigma, N_{\forall}, n, S) = P^{\forall}_Q(\Sigma_0, N_{\forall}, n+1, S) \\
\quad \cup \{(x(U_1, \ldots, U_{N_{\forall}}) \leftarrow U_{n+1} = 1.)\} \\
\quad \cup \{(s_{0^n}(U_1, \ldots, U_n) \leftarrow s_{0^{n+1}}(U_1, \ldots, U_n, 0), s_{0^{n+1}}(U_1, \ldots, U_n, 1).)\} \\
\textit{if } \Sigma \textit{ is a propositional formula then} \\
\quad P^{\forall}_Q(\Sigma, N_{\forall}, n, S) = P^{\forall}(\Sigma, 0^{N_{\forall}}, N_{\forall}, S)
\end{array}
$$

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

$$
\begin{array}{l}
\textit{Existentially quantified variable considered as a formula} \\
\textit{if } \Sigma = x \textit{ and } (x \mapsto n) \in S \textit{ then } P^{\forall}(\Sigma, o, N_{\forall}, S) = \\
\quad \{(s_o(U_1, \ldots, U_{N_{\forall}}) \leftarrow x(U_1, \ldots, U_n).)\} \\
\textit{Universally quantified variable considered as a formula} \\
\textit{if } \Sigma = x \textit{ and } (x \mapsto n) \notin S \textit{ then } P^{\forall}(\Sigma, o, N_{\forall}, S) = \\
\quad \{(s_o(U_1, \ldots, U_{N_{\forall}}) \leftarrow x(U_1, \ldots, U_{N_{\forall}}).)\} \\
\textit{if } \Sigma = \neg\Sigma_0 \textit{ then } P^{\forall}(\Sigma, o, N_{\forall}, S) = \\
\quad \left\{\ (s_o(U_1, \ldots, U_{N_{\forall}}) \leftarrow \textit{not } s_{o.0}(U_1, \ldots, U_{N_{\forall}}).)\ \right\} \\
\quad \cup P^{\forall}(\Sigma_0, o.0, N_{\forall}, S) \\
\textit{if } \Sigma = (\Sigma_0 \wedge \Sigma_1) \textit{ then } P^{\forall}(\Sigma, o, N_{\forall}, S) = \\
\quad \{(s_o(U_1, \ldots, U_{N_{\forall}}) \leftarrow s_{o.0}(U_1, \ldots, U_{N_{\forall}}), s_{o.1}(U_1, \ldots, U_{N_{\forall}}).)\} \\
\quad \cup P^{\forall}(\Sigma_0, o.0, N_{\forall}, S) \cup P^{\forall}(\Sigma_1, o.1, N_{\forall}, S) \\
\textit{if } \Sigma = (\Sigma_0 \vee \Sigma_1) \textit{ then } P^{\forall}(\Sigma, o, N_{\forall}, S) = \\
\quad \left\{ \begin{array}{l} (s_o(U_1, \ldots, U_{N_{\forall}}) \leftarrow s_{o.0}(U_1, \ldots, U_{N_{\forall}}).), \\ (s_o(U_1, \ldots, U_{N_{\forall}}) \leftarrow s_{o.1}(U_1, \ldots, U_{N_{\forall}}).) \end{array} \right\} \\
\quad \cup P^{\forall}(\Sigma_0, o.0, N_{\forall}, S) \cup P^{\forall}(\Sigma_1, o.1, N_{\forall}, S)
\end{array}
$$

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

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 $\Pi_{\forall}(\Sigma) = P^{\forall}(\Sigma, N_{\forall}, 0, \emptyset)$.

By construction the translation is polynomial.

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*

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

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

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

with $P_{\forall a \exists b \forall c \exists d}$ 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 $P^{\forall}(F, 2, 2, \{(b \mapsto 1), (d \mapsto 2)\})$.

Let the reader note that the obtained program is a first order one in which only two constants (0 and 1) occur. So, as usual in ASP, this program has to be considered as a (exponential) compressed version of the propositional program in which every variable is replaced by 0 or 1.

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.*

As in the SAT case, 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*

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

As for the corollary 1 of 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 one 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

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

So, we have $\pi_\forall^{-1}(m, F_{\forall a \exists b \forall c \exists d}) = sk$ *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 Implementation

The theoretical principles exposed in previous sections have been implemented in three different tools : `edimacs2nlp`, `qbf2nlp`, `qedimacs2nlp`, freely available[6]. These tools admit two kinds of input formats: EDIMACS format for SAT competition [7] and QBF 1.0 format (extended with implication, equivalence and xor) for QBF competition [8]. As mentioned previously, some improvements of our formal method have been introduced in the operational translators that we have developed. For instance, all the theory is based on binary operators, but in EDIMACS format we can represent n-ary operators and our tools are able to deal with them. In example 8 we show the real ASP program for example 7 with no option (two options are possible : `-lparse` to hide the intermediate symbols and `-noreduce` to generate a program with no optimization).

Example 8 (Example 7 continued)**.** *The generated file*

```
bool(0).     bool(1).
b(U0) :- not nb(U0),bool(U0).
nb(U0) :- not b(U0),bool(U0).
d(U0,U1) :- not nd(U0,U1),bool(U0),bool(U1).
nd(U0,U1) :- not d(U0,U1),bool(U0),bool(U1).
sigma_2(U0,U1) :- U1=1,bool(U0),bool(U1).
sigma_2(U0,U1) :- b(U0),bool(U0),bool(U1).
sigma_3(U0,U1) :- U1=0,bool(U0),bool(U1).
sigma_3(U0,U1) :- d(U0,U1),bool(U0),bool(U1).
sigma_4(U0,U1) :- U0=1,nd(U0,U1),bool(U0),bool(U1).
sigma_4(U0,U1) :- U0=0,not nd(U0,U1),bool(U0),bool(U1).
sigma_5(U0,U1) :- U1=1,bool(U0),bool(U1).
sigma_5(U0,U1) :- sigma_4(U0,U1),bool(U0),bool(U1).
sigma_6(U0,U1) :- sigma_3(U0,U1),sigma_5(U0,U1),bool(U0),bool(U1).
sigma_7(U0,U1) :- not b(U0),bool(U0),bool(U1).
sigma_7(U0,U1) :- sigma_6(U0,U1),bool(U0),bool(U1).
sigma_8(U0,U1) :- sigma_2(U0,U1),sigma_7(U0,U1),bool(U0),bool(U1).
sigma :- sigma_1(0),sigma_1(1).
sigma_1(U0) :- sigma_8(U0,0),sigma_8(U0,1),bool(U0).
```

[6] `http://forge.info.univ-angers.fr/~damota/asp/en/index.php`
[7] `http://www.satcompetition.org/2005/edimacs.pdf`
[8] `http://www.qbflib.org/boole.html`

```
:- not sigma .
```

is the output of `qbf2nlp` *on file in QBF 1.0 format*

```
/forall [a] /exists [b] /forall [c] /exists [d]
((c | b) & (b -> ((c -> d) & (c | (a <-> !d)))))
```

for the QBF $\forall a \exists b \forall c \exists d((c \vee b) \wedge (b \rightarrow ((c \rightarrow d) \wedge (c \vee (a \leftrightarrow \neg d)))))$.

We want to evaluate if our approach represents an efficient alternative to computing the (Boolean) model of a propositional formula or the satisfying Skolem functions of a QBF. We have selected five ASP solvers: DLV [20] (July 14th, 2006), CLASP [14] (CLASP 1.0.1), noMoRe++ [1] (noMoRe++ v1.5), smodels [28] (smodels-2.32) and ASSAT [23] (ASSAT 2.0); ASSAT has an approach inverse to ours, it computes stable models by calling an underlying SAT solver, in this study: MiniSat [9] (MiniSat 2.0 beta). We first study the results 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 [25] which has the advantage to be available 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. The CNF version was obtained by directly expressing the problem of classifying quasigroups into CNF as opposed to the translation of non-clausal formulas into CNF. We compare the run 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 [19] (SatMate.20.05.2006) the only one which admits the file format of the set of problems QG6. We also include in the comparison the run time and percentage of successful runs for the SAT CNF solvers zChaff [26] (zChaff 2007.3.12.) and MiniSat. The experiments have been realized on a $2 \times Intel\ Xeon\ CPU\ 2.80Ghz$ with $2GB$ of memory. Table 1 shows if the problems are satisfiable (SAT) or not (UNSAT), the number of solved problems (NbR), the average run time in seconds (ATR) and the average run time in seconds when the run succeeds (ATRS).

Our approach associated with the ASP solvers CLASP and ASSAT is really efficient on this set of problems since it performs 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 run time. It is worth noting that if the time limit is set to 500 hours (instead of 1 hour), SatMate does not solve more problems but the other solvers do.

5.2 From QBF to ASP

All current ASP solvers deal with first order programs, but they use a "front-end", external like Lparse [35] (Lparse 1.0.17) or GrinGo [15] (GrinGo 1.0.0) or

	SAT CNF		SAT non CNF
QG6 SAT	zChaff	MiniSat	SatMate
NbR	83	83	83
% solved	100,00%	100,00%	100,00%
ATR	19,39	3,25	4,31
ATRS	19,39	3,25	4,31
QG6 UNSAT	zChaff	MiniSat	SatMate
NbR	137	165	152
% solved	79,19%	95,38%	87,86%
ATR	1260,08	680,15	501,18
ATRS	645,21	538,58	73,05

	ASP				
QG6 SAT	smodels	noMoRe++	DLV	CLASP	ASSAT
NbR	67	83	78	83	83
% solved	80,72%	100,00%	93,98%	100,00%	100,00%
ATR	857,98	118,98	416,48	11,98	10,78
ATRS	203,17	118,98	212,40	11,98	10,78
QG6 UNSAT	smodels	noMoRe++	DLV	CLASP	ASSAT
NbR	88	77	83	154	157
% solved	50,87%	44,51%	47,98%	89,02%	90,75 %
ATR	1912,02	2067,62	1934,17	711,97	662,52
ATRS	281,58	157,12	127,86	355,66	363,15

Table 1: Experimental results for SAT.

internal for DLV. Thus, the replacement of the variables by constants (in our case 0 and 1) called the grounding step leads to a program with an exponential size.

We first evaluate our approach on the set of benchmarks of the QBFEVAL07 evaluation[9] ; 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. Table 2 shows the instance, the number of existentially/universally quantified variables (NbV), the run time for smodels and for CLASP, the number of choice points given by smodels (Nb CP), the number of atoms and the number of rules of the normal logic program grounded by Lparse.

On problems of class "counter", we can remark that smodels has no need of choice point to find the solution. It clearly justifies that ASP is a good framework to deal with the computation of solutions of QBF. It is worth noting that when the memory is insufficient or the run time limit (1 hour) is exceeded it is always due to the grounding step.

In [11] the authors introduce a methodology to compute answer sets of disjunctive logic programs [10] by means of QBF. So, by chaining their polynomial

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

[10] programs with a disjunction in the head of rules

Instance	NbV	smodels	CLASP	Nb CP	Nb atoms	Nb rules
counter5_2	15/10	0,180	0,141	0	2 824	2 869
counter6_2	18/12	0,432	0,423	0	10 108	10 173
counter7_2	21/14	2,420	2,574	0	37 741	37 830
counter8_2	24/16	20,465	22,292	0	144 547	144 664
counter4_4	20/16	3,290	4,911	0	131 925	131 988
counter4_5	24/20	55,730	513,861	0	2 098 267	2 098 352
counter5_4	25/20	64,923	531,140	0	2 099 278	2 099 373
ring4_2	21/14	1,006	1,075	24	34 466	34 665
ring5_2	24/16	4,024	5,734	25	135 085	135 314
ring6_2	27/18	22,300	45,870	10	534 572	534 831
ring4_3	28/21	154,864	M	305	4 197 482	4 199 521
semaphore_2	21/14	1,090	1,130	5	34 967	35 156
semaphore_3	28/21	159,452	M	9	4 198 407	4 198 996
semaphore3_2	27/18	18,327	43,140	5	529 994	530 284

Table 2: Experimental results for QBFEVAL07.

translation from a disjunctive logic program to a QBF and our polynomial translation from QBF to first order normal logic program, we obtain a complete procedure to compute answer sets of disjunctive programs by means of any answer set programming solver, even if it is not dedicated to disjunctive programs. We have experimented this point on the strategic companie example as it is described in [20]. We have generated 10 disjunctive programs for each number of companies between 10 and 17. Every disjunctive program contains 2 rules with variables. The experiments have been realized on an $Intel\ Pentium\ 4\ CPU\ 1.40Ghz$ with $2GB$ of memory. For all these programs the CPU time for DLV (grounding step plus computation of one model) is lower than 0.01 second. On these disjunctive programs we have applied our translation : Table 3 shows the number of companies (NBC), the number of rules of the disjunctive program (NBDR = 4NBC+2), the average number of first order rules in the normal program after the 2 translations (NBNR) and the average number of normal grounded rules after instantiation (NBGR) processed either by Lparse or GrinGo; Table 4 reports the results of the application of Lparse plus smodels (time spent in Lparse with its pourcentage w.r.t. total time, time spent in CLASP with its pourcentage w.r.t. total time, total time and standard deviation σ); Table 5 reports the results of the application of GrinGo plus smodels; Table 6 reports the results of the application of Lparse plus CLASP; Table 7 reports the results of the application of GrinGo plus CLASP; Table 8 reports the results of the application of DLV (with its internal grounder).

6 Concluding discussion

In the first part of this work, we have generalized a translation from SAT to ASP allowing one to use any existing ASP solver as a SAT solver for every kind of propositional formula without syntactic restrictions like CNF as required for

NBC	NBDR	NBNR	NBNGR Lparse	NBNGR GrinGo
10	42	296	12637	11082
11	46	300	24214	21123
12	50	355	42922	41902
13	54	395	97781	85471
14	58	402	187719	163121
15	62	488	387631	338459
16	66	480	745502	647180
17	70	547	1510118	1313501

Table 3: Number of rules.

NBC	CPU Lparse + smodels			
	Lparse	smodels	Total	σ
10	0.27 (55%)	0.17 (35%)	0.49	0.11
11	0.68 (59%)	0.42 (37%)	1.15	0.30
12	1.76 (57%)	1.28 (41%)	3.10	0.87
13	6.74 (50%)	6.67 (50%)	13.46	8.76
14	18.17 (56%)	14.21 (44%)	32.43	19.57
15	74.10 (75%)	24.79 (25%)	98.95	32.46
16	248.25 (71%)	102.57 (29%)	350.87	143.24
17	1047.67 (75%)	347.04 (25%)	1394.28	373.71

Table 4: Results for Lparse + smodels.

NBC	CPU GrinGo + smodels			
	GrinGo	smodels	Total	σ
10	0.27 (63%)	0.11 (24%)	0.43	0.09
11	0.56 (52%)	0.47 (43%)	1.09	0.59
12	1.15 (44%)	1.42 (54%)	2.63	1.32
13	2.33 (13%)	15.78 (87%)	18.17	21.55
14	4.75 (14%)	29.83 (86%)	34.64	45.75
15	10.01 (11%)	77.85 (89%)	87.93	101.24
16	19.96 (7%)	274.56 (93%)	294.60	291.79
17	42.85 (3%)	1189.43 (97%)	1232.37	1648.32

Table 5: Results for GrinGo + smodels.

NBC	CPU Lparse + CLASP			
	Lparse	CLASP	Total	σ
10	0,21 (64%)	0,07 (23%)	0,33	0.06
11	0,57 (73%)	0,17 (22%)	0,72	0.27
12	1,59 (80%)	0,37 (19%)	2,00	0.23
13	6,46 (88%)	0,82 (11%)	7,35	3.69
14	17,68 (91%)	1,79 (9%)	19,52	1.75
15	77,43 (95%)	4,29 (5%)	81,78	12.50
16	265,95 (96%)	10,02 (4%)	276,03	43.53
17	1257,33 (98%)	29,71 (2%)	1287,11	255.18

Table 6: Results for Lparse + CLASP.

NBC	CPU GrinGo + CLASP			
	GrinGo	CLASP	Total	σ
10	0,30 (73%)	0,06 (15%)	0,41	0.04
11	0,56 (74%)	0,14 (18%)	0,76	0.05
12	1,16 (76%)	0,30 (20%)	1,52	0.06
13	2,42 (77%)	0,65 (21%)	3,14	0.24
14	4,93 (76%)	1,46 (23%)	6,45	0.24
15	10,28 (76%)	3,24 (24%)	13,59	0.51
16	20,68 (72%)	8,07 (28%)	28,83	0.88
17	43,50 (67%)	20,96 (33%)	64,53	2.56

Table 7: Results for GrinGo + CLASP.

NBC	CPU DLV			
	Instantiation	Model generator	Total	σ
10	0,26 (70%)	0,03 (8%)	0,37	0.04
11	0,54 (68%)	0,16 (20%)	0,80	0.20
12	1,19 (66%)	0,42 (23%)	1,81	0.41
13	2,70 (50%)	2,34 (43%)	5,43	4.13
14	5,44 (55%)	3,69 (37%)	9,86	4.81
15	11,96 (43%)	14,00 (51%)	27,54	25.27
16	25,04 (32%)	49,36 (64%)	77,37	43.57
17	35,73 (14%)	212,34 (85%)	250,88	188.15

Table 8: Results for DLV.

most of SAT solvers.

In a certain way, our translation deals with the input logical formula as it has been already done by [29, 10] since we introduce a label for every subformula. But our methodology can not be reduce to the application of such a normal form translation followed by a conversion to a normal logic program as in [27]. For instance, if the given formula is $\Sigma = (a \rightarrow (b \vee c))$, the normal form translations presented in [29, 10] lead to the clause set $\mathcal{C} = \{L_a, \neg L_a \vee \neg a \vee L_{b \vee c}, \neg L_{b \vee c} \vee b \vee c\}$. Then, applying the transformation for CNF formulas recalled in subsection 2.2 leads to a program containing five pairs of rules $(x \leftarrow not\ nx.), (nx \leftarrow not\ x.)$ one for each variable $x \in \mathcal{C}$, included the new variables encoding the labels of formulae. With our translation, we obtain a program $\Pi(\Sigma)$ containing three pairs of rules only those for variables occurring in the input formula Σ. In fact, it is easy to check that chaining the two already known techniques leads to $n+m$ pair of rules of this type if n is the number of propositional variables and m the number of operators occurring in the formula, when our methodology leads only to n pairs (note that in whole generality $n \ll m$). This is an important feature for efficiency of the resolution since for up-to-date ASP solvers each pair of such rules induces a choice point to determine if x or nx belongs to a solution. In our translation, the search space is not enlarged to useless variables but is limited to the original ones.

Experiments show that our approach is realistic for non trivial problems even

if it does not speed up the global computation time. In fact, we knew that some very deep studies are required to improve the performances of the better SAT solvers available today, and the improvement of calculus performance for SAT was not our main goal. But, establishing a clear mapping between propositional models and stable models was the first necessary step to reach our initial goal : solving the problem of QBF validity by using ASP paradigm.

That is why in this article we have defined (theoretically and practically) a translation process that provides the user with a normal logic program whose stable models encode (if they exist) the Skolem functions validating the given QBF. The lack of various benchmarks does not allow us to realize a deep evaluation of the performance of our proposal. Therefore, we verify that space complexity resulting from the grounding preprocessing in ASP is the central problem for us and it reflects the particular nature of QBF. But, this point is not surprising with respect to the theory of complexity. Indeed, our translation takes a QBF and generates a normal logic program whose size is still polynomial with respect to the size of the input QBF. Here we take the advantage of the usage of variables in ASP that allows one to write programs representing in intension and in a very compact way, a knowledge base that can have an exponential size if we represent it in extension. But, when the program is processed by an ASP solver, the problem of exponential size, inherent to QBF, arises again since all the available ASP solvers begin their computation by a grounding phase in order to deal with a propositional program.

In conclusion, we can say that the theoretical concepts and the tools that we have introduced here are new and provide QBF researchers with a practical tool that can be useful to compute solutions, to verify solutions, to build benchmarks,... For the ASP community, our work exhibits a kind of benchmarks particularly difficult because of their size. It points out that one of the challenges for the ASP community is to be able to deal with very large programs, maybe by escaping the grounding phase and dealing with first order programs. If one day such a goal is reached then it would be of great interest for our approach of QBF solving.

References

[1] C. Anger, M. Gebser, T. Linke, A. Neumann, and T. Schaub. The nomore++ system. In *Proceedings of the 5th Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, pages 422–426, 2005.

[2] C. Ansotegui, C. Gomes, and B. Selman. Achilles' heel of QBF. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI'05)*, pages 275–281, 2005.

[3] 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)*, pages 187–201, 2002.

[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] A. Biere. Resolve and expand. In *Proceedings of the 7th International Confrerence on Theory and Applications of Satisfiability Testing (SAT'04)*, pages 59–70, 2004.

[6] L. Bordeaux. Boolean and interval propagation for quantified constraints. In *Proceedings of the 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 *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 286–296, 2006.

[9] N. Een and N. Sörensson. An extensible SAT-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 502–518, 2003.

[10] U. Egly. On different structure-preserving translations to normal form. *Journal of Symbolic Computation*, 22(2):121–142, 1996.

[11] U. Egly, T. Eiter, V. Klotz, H. Tompits, and S. Woltran. Computing stable models with quantified boolean formulas: Some experimental results. In *Proceedings of the AAAI Spring Symposium*, pages 53–59, 2001.

[12] U. Egly, M. Seidl, and S. Woltran. A solver for QBFs in nonprenex form. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06)*, pages 477–481, 2006.

[13] G. Parthasarathy F. Lu, M. K. Iyer and K.-T. Cheng. An efficient sequential SAT solver with improved search strategies. In *Proceedings of the 8th Conference on Design, Automation and Test in Europe (DATE'05)*, pages 1102–1107, 2005.

[14] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 386–392, 2007.

[15] M. Gebser, T. Schaub, and S. Thiele. Gringo: A new grounder for answer set programming. In *Proceedings of the 7th Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, pages 266–271, 2007.

[16] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming (ICLP'88)*, pages 1070–1080, 1988.

[17] E. Giunchiglia, M. Narizzano, and A. Tacchella. Backjumping for quantified boolean logic satisfiability. *Artificial Intelligence*, 145:99–120, 2003.

[18] M. Hietalahti, F. Massacci, and I. Niemelä. DES: a challenge problem for nonmonotonic reasoning systems. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning (NMR'00)*, 2000.

[19] H. Jain, C. Bartzis, and E. Clarke. Satisfiability checking of non-clausal formulas using general matings. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT'06)*, pages 75–89, 2006.

[20] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[21] R. Letz. Lemma and model caching in decision procedures for quantified boolean formulas. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'02)*, pages 160–175, 2002.

[22] C.-M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 366–371, 1997.

[23] F. Lin and Y. Zhao. Assat: computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, 157(1-2):115–137, 2004.

[24] 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.

[25] A. Meier and V. Sorge. Applying SAT solving in classification of finite algebras. *Journal of Automated Reasoning*, 35(1-3):201–235, 2005.

[26] 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)*, pages 530–535, 2001.

[27] 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.

[28] I. Niemelä and P. Simons. Evaluating an algorithm for default reasoning. In *Proceedings of the Workshop on Applications and Implementations of Nonmonotomic Reasonigs Systems*, pages 66–72, 1995.

[29] D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986.

[30] J. Rintanen. Improvements to the evaluation of quantified boolean formulae. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 1192–1197, 1999.

[31] H. Samulowitz and F. Bacchus. Binary clause reasoning in QBF. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT'06)*, pages 353–367, 2006.

[32] R.M. Smullyan. *First Order Logic*. Springer Verlag, 1969.

[33] I. Stéphan. Boolean propagation based on literals for quantified boolean formulae. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06)*, pages 452–456, 2006.

[34] L.J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.

[35] T. Syrjänen. Implementation of local grounding for logic programs for stable model semantics. Technical report, Helsinki University of Technology, 1998.

[36] C. Thiffault, F. Bacchus, and T. Walsh. Solving non-clausal formulas with DPLL search. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, pages 663–678, 2004.

[37] L. Zhang. Solving QBF with combined conjunctive and disjunctive normal form. In *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI'06)*, pages 143–150, 2006.

[38] L. Zhang and S. Malik. Conflict driven learning in a quantified boolean satisfiability solver. In *Proceedings of the International Conference on Computer Aided Design (ICCAD'02)*, pages 442–449, 2002.

7 Proofs

Proof of Theorem 1. Let Σ be a propositional formula such that $\mathcal{V}(\Sigma) = \{x\}$. Σ has two valuations $\{x\}$ and $\{\neg x\}$. $\pi(\{x\}) = \{x\}$ and $\pi(\{\neg x\}) = \{nx\}$.
Let ν be a set of literals, S a set of atoms and the normal logic program such that $P_Q(\nu) = \{\ (x \leftarrow not\ nx.),\quad (nx \leftarrow not\ x.)\ \}$

If $S = \emptyset$	$P_Q(\nu)^S = \{\ (x.),\quad (nx.)\ \}$
	$C_n(P_Q(\nu)^S) = \{x, nx\} \neq S$;
If $S = \{x\}$	$P_Q(\nu)^S = \{\ (x.)\ \}$
	$C_n(P_Q(\nu)^S) = \{x\} = S$;
If $S = \{nx\}$	$P_Q(\nu)^S = \{\ (nx.)\ \}$
	$C_n(P_Q(\nu)^S) = \{nx\} = S$;
If $S = \{x, nx\}$	$P_Q(\nu)^S = \emptyset$
	$C_n(P_Q(\nu)^S) = \emptyset \neq S$.

So, $P_Q(\nu)$ has two stable models $\{x\}$ and $\{nx\}$.
$\pi^{-1}(\{x\}, \Sigma) = \{x\}$ and $\pi^{-1}(\{nx\}, \Sigma) = \{\neg x\}$.
The theorem holds for $\nu = \mathcal{V}(\Sigma)$ and $|\mathcal{V}(\Sigma)| = 1$.

Let Σ' be a propositional formula such that $\mathcal{V}(\Sigma') = \mathcal{V}(\Sigma) \cup \{x\}$ with x a new propositional symbol such that $x \notin \mathcal{V}(\Sigma)$. Σ' has two valuations for each valuation ν of Σ: $\nu \cup \{x\}$ and $\nu \cup \{\neg x\}$. Assuming the theorem holds for a set of literals ν and a propositional formula Σ, $P_Q(\mathcal{V}(\Sigma))$ has a stable model $\pi(\nu)$ if and only if ν is a valuation of Σ. Then, $\pi(\nu \cup \{x\}) = \pi(\nu) \cup \{x\}$ and $\pi(\nu \cup \{\neg x\}) = \pi(\nu) \cup \{nx\}$.

Let ν' be a set of literals, S a set of atoms and $P_Q(\nu')$ a normal logic program such that $P_Q(\nu') = P_Q(\nu) \cup \{\ (x \leftarrow not\ nx.),\quad (nx \leftarrow not\ x.)\ \}$

The atoms x and nx are not present in any body or head of a rule in $P_Q(\nu)$, because $x \notin \mathcal{V}(\Sigma)$. Then the union of a stable model of $P_Q(\nu)$ and a stable model of $\{\ (x \leftarrow not\ nx.),\quad (nx \leftarrow not\ x.)\ \}$ is a stable model of $P_Q(\nu')$.

Let $\pi(\nu)$ be a stable model of $P_Q(\nu)$:

If $S = \pi(\nu) \cup \emptyset$	$P_Q(\nu')^S = P_Q(\nu)^S \cup \{\ (x.),\quad (nx.)\ \}$
	$C_n(P_Q(\nu')^S) = \pi(\nu) \cup \{x, nx\} \neq S$
If $S = \pi(\nu) \cup \{x\}$	$P_Q(\nu')^S = P_Q(\nu)^S \cup \{\ (x.)\ \}$
	$C_n(P_Q(\nu')^S) = \pi(\nu) \cup \{x\} = S$
If $S = \pi(\nu) \cup \{nx\}$	$P_Q(\nu')^S = P_Q(\nu)^S \cup \{\ (nx.)\ \}$
	$C_n(P_Q(\nu')^S) = \pi(\nu) \cup \{nx\} = S$
If $S = \pi(\nu) \cup \{x, nx\}$	$P_Q(\nu')^S = P_Q(\nu)^S \cup \emptyset$
	$C_n(P_Q(\nu')^S) = \pi(\nu) \cup \emptyset \neq S$

$P_Q(\nu')$ has two stable models for each stable model $\pi(\nu)$ of $P_Q(\nu)$: $\pi(\nu) \cup \{x\}$ and $\pi(\nu) \cup \{nx\}$. $\pi^{-1}(\pi(\nu) \cup \{x\}, \Sigma) = \nu \cup \{x\}$ and $\pi^{-1}(\pi(\nu) \cup \{nx\}, \Sigma) = \nu \cup \{\neg x\}$. By induction the theorem holds. □

Proof of Lemma 1. Let Σ_{init} and Σ be propositional formulae such that Σ is a subformula of Σ_{init}. Let $\Pi'(\Sigma)$ be a normal logic program such that $\Pi'(\Sigma) = P_Q(\mathcal{V}(\Sigma_{init})) \cup P(\Sigma, o)$, o be the occurrence of Σ and m be a stable model of $\Pi'(\Sigma)$. We prove the following lemma: "The set of literals ν is a valuation of $\mathcal{V}(\Sigma_{init})$ if and only if there exists a stable model m of $\Pi'(\Sigma)$ such that $\pi(\nu) \subseteq m$."

The proof of this lemma is by induction.

- $\Sigma = x$.

$$\begin{aligned} \Pi'(\Sigma) &= P_Q(\mathcal{V}(\Sigma_{init}) \setminus \{x\}) \cup P_Q(\{x\}) \cup P(\Sigma, o) \\ &= P_Q(\mathcal{V}(\Sigma_{init}) \setminus \{x\}) \cup \left\{ \begin{array}{l} (x \leftarrow not\ nx.), (nx \leftarrow not\ x.), \\ (s_o \leftarrow x.) \end{array} \right\} \end{aligned}$$

  Let ν be a valuation of $\mathcal{V}(\Sigma_{init})$ and if $x \in \nu$ then $\nu' = \nu \setminus \{x\}$ else $\nu' = \nu \setminus \{\neg x\}$ (ν' is a valuation of $\mathcal{V}(\Sigma_{init} \setminus \{x\})$). By Theorem 1, $P_Q(\mathcal{V}(\Sigma_{init}) \setminus \{x\})$ has necessarily a stable model $\pi(\nu')$. If $x \in \nu$ then $m = \pi(\nu') \cup \{x, s_o\}$ is a stable model of $\Pi'(\Sigma)$ such that $\pi(\nu) \subseteq m$ else $\neg x \in \nu$ and $m = \pi(\nu') \cup \{nx\}$ is a stable model of $\Pi'(\Sigma)$ such that $\pi(\nu) \subseteq m$. Conversely, let ν be a set of literals and m be a stable model of $\Pi'(\Sigma)$ such that

$\pi(\nu) \subseteq m$. Either $x, s_o \in m$ and $nx \notin m$, or $x, s_o \notin m$ and $nx \in m$. By Theorem 1, the set of literals ν', such that $\nu' = \nu \setminus \{x\}$ in first case and $\nu' = \nu \setminus \{\neg x\}$ in the second case, is a valuation of $\mathcal{V}(\Sigma_{init} \setminus \{x\})$. Then in both cases ν is a valuation of $\mathcal{V}(\Sigma_{init})$.

Assuming Lemma 1 holds for Σ_1 and Σ_2 subformulae of Σ and m a stable model of $\Pi'(\Sigma)$:

- $\Sigma = \neg\Sigma_1$.

$$\begin{array}{rcl} \Pi'(\Sigma) & = & P_Q(\mathcal{V}(\Sigma_{init})) \cup P(\Sigma_1, o.0) \cup \{(s_o \leftarrow not\ s_{o.0}.)\} \\ & = & \Pi'(\Sigma_1) \cup \{(s_o \leftarrow not\ s_{o.0}).)\} \end{array}$$

Let ν be a valuation of Σ_{init}. By induction hypothesis, there exists a stable model m_1 of $\Pi'(\Sigma_1)$ such that $\pi(\nu) \subseteq m_1$. If $s_{o.0} \notin m_1$ then $m_1 \cup \{s_o\}$ is a stable model for $\Pi'(\Sigma)$ and $\pi(\nu) \subseteq m_1 \cup \{s_o\}$, if $s_{o.0} \in m_1$ then m_1 is a stable model for $\Pi'(\Sigma)$ and $\pi(\nu) \subseteq m$. The converse is trivial.

- $\Sigma = (\Sigma_1 \wedge \Sigma_2)$. Cases for $\vee$, $\rightarrow$, $\oplus$ and $\leftrightarrow$ are similar.

$$\begin{array}{rcl} \Pi'(\Sigma) & = & P_Q(\mathcal{V}(\Sigma_{init})) \cup P(\Sigma_1, o.0) \cup P(\Sigma_2, o.1) \cup \{(s_o \leftarrow s_{o.0}, s_{o.1}.)\} \\ & = & \Pi'(\Sigma_1) \cup \Pi'(\Sigma_2) \cup \{(s_o \leftarrow s_{o.0}, s_{o.1}.)\} \end{array}$$

Let ν be a valuation of Σ_{init}. By induction hypothesis, there exists a stable model m_1 of $\Pi'(\Sigma_1)$ such that $\pi(\nu) \subseteq m_1$ and there exists a stable model m_2 of $\Pi'(\Sigma_2)$ such that $\pi(\nu) \subseteq m_2$. Since $\mathcal{V}(\Sigma_{init})$ is in common between $\Pi'(\Sigma_1)$ and $\Pi'(\Sigma_2)$, and $Head(P(\Sigma_1, o.0)) \cap Head(P(\Sigma_2, o.1)) =:$ if $s_{o.0} \in m_1$ and $s_{o.1} \in m_2$ then $m = m_1 \cup m_2 \cup \{s_o\}$ is a stable model of $\Pi'(\Sigma)$ and $\pi(\nu) \subseteq m$, otherwise, $m = m_1 \cup m_2$ is a stable model of $\Pi'(\Sigma)$ and $\pi(\nu) \subseteq m$. The converse is trivial.

Therefore by induction the lemma holds. Since Σ is a subformula of Σ_{init}, it also holds for Σ_{init}. Hence Lemma 1 holds. □

Proof of Lemma 2 and Theorem 2. Let Σ be a propositional formula such that Σ is a subformula of Σ_{init}. Let $\Pi'(\Sigma)$ be a normal logic program such that $\Pi'(\Sigma) = P_Q(\mathcal{V}(\Sigma_{init})) \cup P(\Sigma, o)$, o be the occurence of Σ and m be a stable model of $\Pi'(\Sigma)$. Proof of Lemma 2 is by induction. Theorem 2 follows from Lemma 2 with $\Sigma = \Sigma_{init}$.

- $\Sigma = x$.

$$\begin{array}{rcl} \Pi'(\Sigma) & = & P_Q(\mathcal{V}(\Sigma_{init}) \setminus \{x\}) \cup P_Q(\{x\}) \cup P(\Sigma, o) \\ & = & P_Q(\mathcal{V}(\Sigma_{init}) \setminus \{x\}) \cup \left\{ \begin{array}{l} (x \leftarrow not\ nx.), (nx \leftarrow not\ x.), \\ (s_o \leftarrow x.) \end{array} \right\} \end{array}$$

$s_o \in m$ if and only if $x \in m$ because $(s_o \leftarrow x.)$ is the only rule whose head is s_o. Then, $x \in \pi^{-1}(m, \Sigma)$ and $\pi^{-1}(m, \Sigma) \models \Sigma$.

If $\pi^{-1}(m, \Sigma) \models \Sigma$ then $x \in \pi^{-1}(m, \Sigma)$ and by definition of π^{-1}, $x \in m$. Due to the rule $(s_o \leftarrow x.)$, m is a stable model of $\Pi'(\Sigma)$ if and only if $s_o \in m$. Lemma 2 holds for this case.

Assuming Lemma 2 holds for Σ_1 and Σ_2 subformulas of Σ and m a stable model of $\Pi'(\Sigma)$:

- $\Sigma = \neg\Sigma_1$.

  $$\begin{array}{rcl}\Pi'(\Sigma) & = & P_Q(\mathcal{V}(\Sigma_{init})) \cup P(\Sigma_1, o.0) \cup \{\ (s_o \leftarrow not\ s_{o.0}.)\ \} \\ & = & \Pi'(\Sigma_1) \cup \{\ (s_o \leftarrow not\ s_{o.0}).)\ \}\end{array}$$

  $s_o \in m$ if and only if $s_{o.0} \notin m$ because $(s_o \leftarrow not\ s_{o.0}.)$ is the only rule whose head is s_o. By hypothesis, $s_{o.0} \notin m$ is equivalent to $\pi^{-1}(m, \Sigma_1) \not\models \Sigma_1$, therefore $\pi^{-1}(m, \Sigma) \not\models \Sigma_1$ (because $\mathcal{V}(\Sigma_1) = \mathcal{V}(\Sigma)$) and $\pi^{-1}(m, \Sigma) \models \Sigma$.
  Conversely, if $\pi^{-1}(m, \Sigma) \models \Sigma$, then $\pi^{-1}(m, \Sigma) \not\models \Sigma_1$, equivalent to $s_{o.0} \notin m$ by hypothesis. Due to the rule $(s_o \leftarrow not\ s_{o.0}.)$, $s_0 \in m$. Then Lemma 2 holds for this case.

- $\Sigma = (\Sigma_1 \wedge \Sigma_2)$. Cases for $\vee$, $\rightarrow$, $\oplus$ and $\leftrightarrow$ are similar.

  $$\begin{array}{rcl}\Pi'(\Sigma) & = & P_Q(\mathcal{V}(\Sigma_{init})) \cup P(\Sigma_1, o.0) \cup P(\Sigma_2, o.1) \cup \{\ (s_o \leftarrow s_{o.0}, s_{o.1}.)\ \} \\ & = & \Pi'(\Sigma_1) \cup \Pi'(\Sigma_2) \cup \{\ (s_o \leftarrow s_{o.0}, s_{o.1}.)\ \}\end{array}$$

  $s_o \in m$ if and only if $s_{o.0} \in m$ and $s_{o.1} \in m$ because $(s_o \leftarrow s_{o.0}, s_{o.1}.)$ is the only rule whose head is s_o. By hypothesis (since m is a stable model of $\Pi'(\Sigma_1)$ and $\Pi'(\Sigma_2)$), $s_{o.0} \in m$ and $s_{o.1} \in m$ is equivalent to $\pi^{-1}(m, \Sigma_1) \models \Sigma_1$ and $\pi^{-1}(m, \Sigma_2) \models \Sigma_2$, therefore $\pi^{-1}(m, \Sigma) \models \Sigma_1$ and $\pi^{-1}(m, \Sigma) \models \Sigma_2$ (because $\mathcal{V}(\Sigma_1) \subseteq \mathcal{V}(\Sigma)$ and $\mathcal{V}(\Sigma_2) \subseteq \mathcal{V}(\Sigma)$) and $\pi^{-1}(m, \Sigma) \models \Sigma$. Conversely, if $\pi^{-1}(m, \Sigma) \models \Sigma$, then $\pi^{-1}(m, \Sigma) \models \Sigma_1$ and $\pi^{-1}(m, \Sigma) \models \Sigma_2$ equivalent to $s_{o.0} \in m$ and $s_{o.1} \in m$ by hypothesis. Due to the rule $(s_o \leftarrow s_{o.0}, s_{o.1}.)$, $s_0 \in m$. Then Lemma 2 holds for this case.

Therefore by induction Lemma 2 holds. □

Proof of Corollary 1 and Corollary 2. We prove Corollary 1. Proof is similar for Corollary 2.

- Let m be a stable model of $\Pi^+(\Sigma)$. Due to the rule $(\leftarrow not\ s_\epsilon.)$, $s_\epsilon \in m$. m is also a stable model of $\Pi(\Sigma)$ (since $m = Cn(\Pi(\Sigma)^m) = Cn(\Pi^+(\Sigma)^m)$). Then by Theorem 2, $\pi^{-1}(m, \Sigma) \models \Sigma$ and Σ is satisfiable.

- Conversely, let Σ be satisfiable and ν a (Boolean) model of Σ. According to Lemma 1, there exists a stable model $m \supseteq \pi(\nu)$ of $\Pi(\Sigma)$. According to Theorem 2, if $\nu \models \Sigma$ then $s_\epsilon \in m$. $m = Cn(\Pi(\Sigma)^m)$ and since $s_\epsilon \in m$, $m = Cn(\Pi^+(\Sigma)^m)$. So m is stable model of $\Pi^+(\Sigma)$ such that $m \supseteq \pi(\nu)$. Then there exists a stable model $m \supseteq \pi(\nu)$ of $\Pi^+(\Sigma)$.

Hence Corollary 1 holds.

□

Proof of Theorem 3. We prove more than the theorem : "Let Σ be a QBF. If m is a stable model of $\Pi_\forall(\Sigma)$ such that $s_\epsilon \in m$ then $\pi_\forall^{-1}(m, \Sigma)$ 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 stable model m of $\Pi_\forall(\Sigma)$ such that $\pi_\forall(sk) \subseteq m$ and $s_\epsilon \in m$."

The proof is in two steps. First we prove that there exists a normal logic program nlp_F (and an atom s_ϵ) for a QBF F such that : If m is a stable model of nlp_F such that $s_\epsilon \in m$ then $\pi_\forall^{-1}(m, \Sigma)$ 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 stable model m of nlp_F such that $\pi_\forall(sk) \subseteq m$ and $s_\epsilon \in m$. Then, we prove that nlp_F has the same stable models (w.r.t. the atoms containing Skolem function symbols) as $\Pi_\forall(F)$.

The starting point of the first part of this proof is the *Propositional Skolemization* of [4] which extracts from a QBF an equivalent SAT instance by Skolemization of an equivalent First-Order Logic (FOL) formula. Let $\mathbf{L}$ be a first-order language with $\mathbf{PS} = \{p/1\}$ the set of predicate symbols and $\mathbf{CS} = \{1, 0\}$ the set of constant symbols (the set of function symbols is empty). Let $\mathbf{M}$ be a structure such that the interpretation of predicate p is: $p(1)$ is true and $p(0)$ is false. Let $fol : QBF \rightarrow FOL$ be the function that replaces in a QBF the propositional symbol x by an atom $p(x)$. If the domain of the structure is the Boolean domain, [4] states that a QBF F is satisfiability equivalent to $fol(F)$.
Example 7 continued. *Let* $F_{\forall a \exists b \forall c \exists d} = \forall a \exists b \forall c \exists d((c \vee b) \wedge (b \rightarrow ((c \rightarrow d) \wedge (c \vee (a \leftrightarrow \neg d)))))$.
$fol(F_{\forall a \exists b \forall c \exists d}) = \forall a \exists b \forall c \exists d((p(c) \vee p(b)) \wedge (p(b) \rightarrow ((p(c) \rightarrow p(d)) \wedge (p(c) \vee (p(a) \leftrightarrow \neg p(d))))))$
Proof continued. Let $sk : FOL \rightarrow FOL$ be the function that computes the Skolem form of a (prenex) FOL formula (replacing the existential variables by new Skolem function symbols which depend on the universal variables that have those existential variables in their scope).
Example continued.

$$\begin{array}{ll} & sk(fol(F_{\forall a \exists b \forall c \exists d})) \\ = & \forall a \forall c((p(c) \vee p(b(a))) \wedge (p(b(a)) \rightarrow ((p(c) \rightarrow p(d(a,c))) \wedge (p(c) \vee (p(a) \leftrightarrow \neg p(d(a,c)))))))) \end{array}$$

Proof continued. Using the Skolem theorem we obtain that F is satisfiable if and only if $sk(fol(F))$ is also satisfiable (with a bijection between the models of the QBF and the satisfying Skolem functions). In $fol(F)$, we keep the name of the existential variables as Skolem constant or function symbols. We apply the Skolem theorem and obtain a formula with only universal quantifiers. The last step of Propositional Skolemization flattens the FOL formula to a SAT instance (this function called *prop* is not detailled since we do not use it). Finally, [4] states that $prop(sk(fol(F)))$ is a satisfiability equivalent propositional formula to F. We replace the application of *prop* function by a more simple one called *univ* which expands the universal quantifiers as follow :

$$univ(sk(fol(F))) = \bigwedge_{(b_1 \ldots b_{N^\forall}) \in \{0,1\}^{N^\forall}} \sigma_{b_1 \ldots b_{N^\forall}}.$$

with for each $b_1 \ldots b_{N^\forall} \in \{0,1\}^{N^\forall}$, $\sigma_{b_1 \ldots b_{N^\forall}}$ is $sk(fol(F))$ in which for each universal variable u_i, $p(u_i)$ is replaced by $p(u_i(b_1, \ldots, b_{N^\forall}))$, for each universal variable u_i, $\neg p(u_i(b_1, \ldots, b_{N^\forall}))$ if $b_i = 0$ and $p(u_i(b_1, \ldots, b_{N^\forall}))$ if $b_i = 1$ is added (with a conjunction), for each existential variable e (and Skolem function symbol e) the atoms $p(e(u_1, \ldots, u_n))$ is replaced by $p(e(b_1, \ldots, b_n))$.

Example continued. *Let* $\forall a \forall c M = sk(fol(F_{\forall a \exists b \forall c \exists d}))$.

$$\begin{array}{l} univ(sk(fol(F_{\forall a \exists b \forall c \exists d}))) = \\ \quad [M[a \leftarrow 1][c \leftarrow 1] \wedge p(a(1,1)) \wedge p(c(1,1))] \\ \wedge [M[a \leftarrow 1][c \leftarrow 0] \wedge p(a(1,0)) \wedge \neg p(c(1,0))] \\ \wedge [M[a \leftarrow 0][c \leftarrow 1] \wedge \neg p(a(0,1)) \wedge p(c(0,1))] \\ \wedge [M[a \leftarrow 0][c \leftarrow 0] \wedge \neg p(a(0,0)) \wedge \neg p(c(0,0))] \end{array}$$

Proof continued. Since $univ(sk(fol(F)))$ is free of variables it may be consider as a propositional formula. We rename all "propositional symbol" $p(x)$ by the "propositional symbol" x.

Let $\pi'_\forall : 2^{\mathcal{F}} \rightarrow 2^{\mathcal{L}}$ be a function such that, for every $sk \in 2^{\mathcal{F}}$,

$$\begin{array}{l} \pi'_\forall(sk) = \\ \quad \{x(i^{-1}(u_1), \ldots, i^{-1}(u_n)) \mid \hat{x} \in sk, u_1, \ldots, u_n \in BOOL, \hat{x}(u_1, \ldots, u_n) = \mathbf{t}\} \cup \\ \quad \{\neg x(i^{-1}(u_1), \ldots, i^{-1}(u_n)) \mid \hat{x} \in sk, u_1, \ldots, u_n \in BOOL, \hat{x}(u_1, \ldots, u_n) = \mathbf{f}\} \end{array}$$

Example continued.

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

$$\pi'_\forall(sk) = \{b(1), b(0), d(1,1), d(0,1), d(0,0), \neg d(1,0)\}$$

Proof continued. By definition of satisfiability of QBF: if a set of Skolem functions sk satisfy a QBF F then $\pi'_\forall(sk)$ is a model for $univ(sk(fol(F)))$; if ν is a (Boolean) model of $univ(sk(fol(F)))$ then there exists a set of Skolem functions sk such that $\nu = \pi'_\forall(sk)$ and sk satisfy F.

Now we can apply Theorem 2 and Corollary 1 to $univ(sk(fol(F)))$ which is a propositional formula:

(1) If m is a stable model of $\Pi(univ(sk(fol(F))))$ such that $s_\epsilon \in m$ then

$$\pi^{-1}(m, univ(sk(fol(F)))) \models univ(sk(fol(F))).$$

Let sk be a set of Skolem functions such that $\pi^{-1}(m, univ(sk(fol(F)))) = \pi'_\forall(sk)$ then $\pi^{-1}(m, univ(sk(fol(F)))) = \pi_\forall^{-1}(m, F)$ is a set of Skolem functions that satisfy F.

(2) If sk is a set of Skolem functions which satisfy F then $\pi'_\forall(sk)$ is a (Boolean) model of $univ(sk(fol(F)))$ then there exists a stable model m of $\Pi(univ(sk(fol(F))))$ such that $\pi(\pi'_\forall(sk)) = \pi_\forall(sk) \subseteq m$ and $s_\epsilon \in m$.

We have proven the first part.
Example continued.

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

$$\pi'_\forall(sk) = \{b(1), b(0), d(1,1), d(0,1), d(0,0), \neg d(1,0)\}$$
$$\pi(m, univ(sk(fol(F_{\forall a \exists b \forall c \exists d}))) = \{b(1), b(0), d(1,1), d(0,1), d(0,0), \neg d(1,0)\} = \pi'_\forall(sk)$$
$$\pi(\pi'_\forall(sk)) = \{b(1), b(0), d(1,1), d(0,1), d(0,0), nd(1,0)\} = \pi_\forall(sk)$$

Proof continued. Without lose of generality and by commutativity and associativity of conjunction, we isolate under the root s_0 the translation of the conjunction of the added atoms. Let $nlp_1 = \Pi(univ(sk(fol(F))))$ and m_1 one of its stable models. By construction and Lemma 1, for all universal variable u_i and for all $b_1 \ldots b_{N^\forall} \in \{0,1\}^{N^\forall}$, if $b_i = 1$ then $u_i(b_1, \ldots, b_{N^\forall}) \in m_1$ and $nu_i(b_1, \ldots, b_{N^\forall}) \notin m_1$ otherwise $b_i = 0$, $u_i(b_1, \ldots, b_{N^\forall}) \notin m_1$ and $nu_i(b_1, \ldots, b_{N^\forall}) \in m_1$.

Example continued.

$nlp_1 = \{(s_\epsilon \leftarrow s_{\epsilon.0}, s_{\epsilon.1}.)\} \cup$
$\{$*With root* $s_{\epsilon.0}$*, the translation of*
$a(1,1) \wedge c(1,1) \wedge a(1,0) \wedge \neg c(1,0) \wedge \neg a(0,1) \wedge c(0,1) \wedge \neg a(0,0) \wedge \neg c(0,0)\} \cup$
$\{(s_{\epsilon.1} \leftarrow s_{\epsilon.1.0}, s_{\epsilon.1.1}.), (s_{\epsilon.1.0} \leftarrow s_{\epsilon.1.0.0}, s_{\epsilon.1.0.1}.), (s_{\epsilon.1} \leftarrow s_{\epsilon.1.1.0}, s_{\epsilon.1.1.1}.)\} \cup$
$\{(c(1,1) \leftarrow not\ nc(1,1).), (nc(1,1) \leftarrow not\ c(1,1).), (c(1,0) \leftarrow not\ nc(1,0).),$
$(nc(1,0) \leftarrow not\ c(1,0).), (c(0,1) \leftarrow not\ nc(0,1).), (nc(0,1) \leftarrow not\ c(0,1).),$
$(c(0,0) \leftarrow not\ nc(0,0).), (nc(0,0) \leftarrow not\ c(0,0).), (a(1,1) \leftarrow not\ na(1,1).),$
$(na(1,1) \leftarrow not\ a(1,1).), (a(1,0) \leftarrow not\ na(1,0).), (na(1,0) \leftarrow not\ a(1,0).),$
$(a(0,1) \leftarrow not\ na(0,1).), (na(0,1) \leftarrow not\ a(0,1).), (a(0,0) \leftarrow not\ na(0,0).),$
$(na(0,0) \leftarrow not\ a(0,0).), (d(1,1) \leftarrow not\ nd(1,1).), (nd(1,1) \leftarrow not\ d(1,1).),$
$(d(1,0) \leftarrow not\ nd(1,0).), (nd(1,0) \leftarrow not\ d(1,0).), (d(0,1) \leftarrow not\ nd(0,1).),$
$(nd(0,1) \leftarrow not\ d(0,1).), (d(0,0) \leftarrow not\ nd(0,0).), (nd(0,0) \leftarrow not\ d(0,0).),$
$(b(1) \leftarrow not\ nb(1).), (nb(1) \leftarrow not\ b(1).), (b(0) \leftarrow not\ nb(0).), (nb(0) \leftarrow not\ b(0).)\}$
$\cup \sigma_{11} \cup \sigma_{01} \cup \sigma_{10} \cup \sigma_{00}$

with

$\sigma_{11} = \{(s_{\epsilon.1^3} \leftarrow s_{\epsilon.1^3.0}, s_{\epsilon.1^3.1}.), (s_{\epsilon.1^3.0} \leftarrow s_{\epsilon.1^3.0^2}.), (s_{\epsilon.1^3.0} \leftarrow s_{\epsilon.1^3.0.1}.),$
$(s_{\epsilon.1^3.0^2} \leftarrow c(1,1).), (s_{\epsilon.1^3.0.1} \leftarrow b(1).), (s_{\epsilon.1^3.1} \leftarrow not\ s_{\epsilon.1^3.1.0}.), (s_{\epsilon.1^3.1} \leftarrow s_{\epsilon.1^3.1^2}.),$
$(s_{\epsilon.1^3.1.0} \leftarrow b(1).), (s_{\epsilon.1^3.1^2} \leftarrow s_{\epsilon.1^3.1^2.0}, s_{\epsilon.1^3.1^3}.), (s_{\epsilon.1^3.1^2.0} \leftarrow not\ s_{\epsilon.1^3.1^2.0^2}.),$
$(s_{\epsilon.1^3.1^2.0} \leftarrow s_{\epsilon.1^3.1^2.0.1}.), (s_{\epsilon.1^3.1^2.0^2} \leftarrow c(1,1).), (s_{\epsilon.1^3.1^2.0.1} \leftarrow d(1,1).),$
$(s_{\epsilon.1^3.1^3} \leftarrow s_{\epsilon.1^3.1^3.0}.), (s_{\epsilon.1^3.1^3} \leftarrow s_{\epsilon.1^3.1^4}.), (s_{\epsilon.1^3.1^3.0} \leftarrow c(1,1).),$
$(s_{\epsilon.1^3.1^4} \leftarrow s_{\epsilon.1^3.1^4.0}, s_{\epsilon.1^3.1^5}.), (s_{\epsilon.1^3.1^4} \leftarrow not\ s_{\epsilon.1^3.1^4.0}, not\ s_{\epsilon.1^3.1^5}.),$
$(s_{\epsilon.1^3.1^4.0} \leftarrow a(1,1).), (s_{\epsilon.1^3.1^5} \leftarrow not\ s_{\epsilon.1^3.1^5.0}.), (s_{\epsilon.1^3.1^5.0} \leftarrow d(1,1).)\}$

Proof continued. Let nlp_2 be the normal logic program obtained from nlp_1 by deleting all rules with $s_{\epsilon.0.o}$ as head (rules obtained by translation of the added atoms), deleting $s_{\epsilon.0}$ in $(s_\epsilon \leftarrow s_{\epsilon.0}, s_{\epsilon.1}.)$, deleting all rules containing *not* $u_i(b_1, \ldots, b_{N^\forall})$ if $b_i = 1$ ($u_i(b_1, \ldots, b_{N^\forall})$ always in the model), deleting all rules containing *not* $nu_i(b_1, \ldots, b_{N^\forall})$ if $b_i = 0$ ($nu_i(b_1, \ldots, b_{N^\forall})$ always in the model), deleting the atoms *not* $nu_i(b_1, \ldots, b_{N^\forall})$ if $b_i = 1$ ($nu_i(b_1, \ldots, b_{N^\forall})$ never in the model), deleting the atoms *not* $u_i(b_1, \ldots, b_{N^\forall})$ if $b_i = 0$ ($u_i(b_1, \ldots, b_{N^\forall})$ never in the model), deleting the facts $(nu_i(b_1, \ldots, b_{N^\forall}).)$ if $b_i = 0$ (since there is no more instance of the atoms $nu_i(b_1, \ldots, b_{N^\forall})$). By the above remarks, the normal logic program nlp_1 has the same stable models (w.r.t. the atoms containing Skolem function symbols) as the normal logic program nlp_2.

Example continued.

$nlp_2 = \{(s_\epsilon \leftarrow s_{\epsilon.1}.)\} \cup$
$\{(s_{\epsilon.1} \leftarrow s_{\epsilon.1.0}, s_{\epsilon.1.1}.), (s_{\epsilon.1.0} \leftarrow s_{\epsilon.1.0.0}, s_{\epsilon.1.0.1}.), (s_{\epsilon.1.1} \leftarrow s_{\epsilon.1.1.0}, s_{\epsilon.1.1.1}.)\} \cup$
$\{(c(1,1).), (c(0,1).), (a(1,1).), (a(1,0).), (d(1,1) \leftarrow not\ nd(1,1).),$
$(nd(1,1) \leftarrow not\ d(1,1).), (d(1,0) \leftarrow not\ nd(1,0).), (nd(1,0) \leftarrow not\ d(1,0).),$
$(d(0,1) \leftarrow not\ nd(0,1).), (nd(0,1) \leftarrow not\ d(0,1).), (d(0,0) \leftarrow not\ nd(0,0).),$
$(nd(0,0) \leftarrow not\ d(0,0).), (b(1) \leftarrow not\ nb(1).), (nb(1) \leftarrow not\ b(1).),$
$(b(0) \leftarrow not\ nb(0).), (nb(0) \leftarrow not\ b(0).)\} \cup \sigma_{11} \cup \sigma_{01} \cup \sigma_{10} \cup \sigma_{00}$
with
$\sigma_{11} = \{(s_{\epsilon.1^3} \leftarrow s_{\epsilon.1^3.0}, s_{\epsilon.1^3.1}.), (s_{\epsilon.1^3.0} \leftarrow s_{\epsilon.1^3.0^2}.), (s_{\epsilon.1^3.0} \leftarrow s_{\epsilon.1^3.0.1}.),$
$(s_{\epsilon.1^3.0^2} \leftarrow c(1,1).), (s_{\epsilon.1^3.0.1} \leftarrow b(1).), (s_{\epsilon.1^3.1} \leftarrow not\ s_{\epsilon.1^3.1.0}.), (s_{\epsilon.1^3.1} \leftarrow s_{\epsilon.1^3.1^2}.),$
$(s_{\epsilon.1^3.1.0} \leftarrow b(1).), (s_{\epsilon.1^3.1^2} \leftarrow s_{\epsilon.1^3.1^2.0}, s_{\epsilon.1^3.1^3}.), (s_{\epsilon.1^3.1^2.0} \leftarrow not\ s_{\epsilon.1^3.1^2.0^2}.),$
$(s_{\epsilon.1^3.1^2.0} \leftarrow s_{\epsilon.1^3.1^2.0.1}.), (s_{\epsilon.1^3.1^2.0^2} \leftarrow c(1,1).), (s_{\epsilon.1^3.1^2.0.1} \leftarrow d(1,1).),$
$(s_{\epsilon.1^3.1^3} \leftarrow s_{\epsilon.1^3.1^3.0}.), (s_{\epsilon.1^3.1^3} \leftarrow s_{\epsilon.1^3.1^4}.), (s_{\epsilon.1^3.1^3.0} \leftarrow c(1,1).),$
$(s_{\epsilon.1^3.1^4} \leftarrow s_{\epsilon.1^3.1^4.0}, s_{\epsilon.1^3.1^5}.), (s_{\epsilon.1^3.1^4} \leftarrow not\ s_{\epsilon.1^3.1^4.0}, not\ s_{\epsilon.1^3.1^5}.),$
$(s_{\epsilon.1^3.1^4.0} \leftarrow a(1,1).), (s_{\epsilon.1^3.1^5} \leftarrow not\ s_{\epsilon.1^3.1^5.0}.), (s_{\epsilon.1^3.1^5.0} \leftarrow d(1,1).)\}$

Proof continued. Now we rename the predicate symbols $s_{\epsilon.1.o}$ in order to have the same predicate symbols as in Definition 7. Let nlp_3 be the normal logic program obtained from nlp_2 by deleting the rule $(s_\epsilon \leftarrow s_{\epsilon.1}.)$, replacing the atom $s_{\epsilon.1}$ by the atom s_ϵ, replacing all the atoms $s_{\epsilon.1.o.o'}$ in σ_o with $o = b_1.....b_{N^\forall}$ by the atoms $s_{\epsilon.o'}(b_1.....b_{N^\forall})$, replacing all the atoms $s_{\epsilon.1.o}$ in σ_o with $o = b_1.....b_n, n \leq N^\forall$ by the atoms $s_{\epsilon.o'}(b_1.....b_n)$,

Example continued.

$nlp_3 = \{(s_\epsilon \leftarrow s_\epsilon(0), s_\epsilon(1).), (s_\epsilon(0) \leftarrow s_\epsilon(0,0), s_\epsilon(0,1).), (s_\epsilon(1) \leftarrow s_\epsilon(1,0), s_\epsilon(1,1).)\} \cup$
$\{(c(1,1).), (c(0,1).), (a(1,1).), (a(1,0).), (d(1,1) \leftarrow not\ nd(1,1).),$
$(nd(1,1) \leftarrow not\ d(1,1).), (d(1,0) \leftarrow not\ nd(1,0).), (nd(1,0) \leftarrow not\ d(1,0).),$
$(d(0,1) \leftarrow not\ nd(0,1).), (nd(0,1) \leftarrow not\ d(0,1).), (d(0,0) \leftarrow not\ nd(0,0).),$
$(nd(0,0) \leftarrow not\ d(0,0).), (b(1) \leftarrow not\ nb(1).), (nb(1) \leftarrow not\ b(1).),$
$(b(0) \leftarrow not\ nb(0).), (nb(0) \leftarrow not\ b(0).)\} \cup \sigma_{11} \cup \sigma_{01} \cup \sigma_{10} \cup \sigma_{00}$
with
$\sigma_{11} = \{(s_\epsilon(1,1) \leftarrow s_{\epsilon.0}(1,1), s_{\epsilon.1}(1,1).), (s_{\epsilon.0}(1,1) \leftarrow s_{\epsilon.0^2}(1,1).),$
$(s_{\epsilon.0}(1,1) \leftarrow s_{\epsilon.0.1}(1,1).), (s_{\epsilon.0^2}(1,1) \leftarrow c(1,1).), (s_{\epsilon.0.1}(1,1) \leftarrow b(1).),$
$(s_{\epsilon.1}(1,1) \leftarrow not\ s_{\epsilon.1.0}(1,1).), (s_{\epsilon.1}(1,1) \leftarrow s_{\epsilon.1^2}(1,1).), (s_{\epsilon.1.0}(1,1) \leftarrow b(1).),$
$(s_{\epsilon.1^2}(1,1) \leftarrow s_{\epsilon.1^2.0}(1,1), s_{\epsilon.1^3}(1,1).), (s_{\epsilon.1^2.0}(1,1) \leftarrow not\ s_{\epsilon.1^2.0^2}(1,1).),$
$(s_{\epsilon.1^2.0}(1,1) \leftarrow s_{\epsilon.1^2.0.1}(1,1).), (s_{\epsilon.1^2.0^2}(1,1) \leftarrow c(1,1).), (s_{\epsilon.1^2.0.1}(1,1) \leftarrow d(1,1).),$
$(s_\epsilon(1,1) \leftarrow s_{\epsilon.1^3.0}(1,1).), (s_{\epsilon.1^3}(1,1) \leftarrow s_{\epsilon.1^4}(1,1).), (s_{\epsilon.1^3.0}(1,1) \leftarrow c(1,1).),$
$(s_{\epsilon.1^4}(1,1) \leftarrow s_{\epsilon.1^4.0}(1,1), s_{\epsilon.1^5}(1,1).), (s_{\epsilon.1^4}(1,1) \leftarrow not\ s_{\epsilon.1^4.0}(1,1), not\ s_{\epsilon.1^5}(1,1).),$
$(s_{\epsilon.1^4.0}(1,1) \leftarrow a(1,1).), (s_{\epsilon.1^5}(1,1) \leftarrow not\ s_{\epsilon.1^5.0}(1,1).), (s_{\epsilon.1^5.0}(1,1) \leftarrow d(1,1).)\}$

Proof continued. The normal logic program nlp_2 has clearly the same stable models (w.r.t. the atoms containing Skolem function symbols) as the normal logic program nlp_3. Since the normal logic program nlp_3 is the ground instance of $\Pi_\forall(F)$, by the semantics of first-order program in normal logic program, the normal logic program $\Pi_\forall(F)$ has the same stable models (w.r.t. the atoms containing Skolem function symbols) as the normal logic program nlp_3.

From these equivalences (w.r.t. atoms containing Skolem function symbols):

(1') If m is a stable model of $\Pi_\forall(F)$ such that $s_\epsilon \in m$ then there exists a stable model m' of $\Pi(univ(sk(fol(F))))$ such that $s_\epsilon \in m'$.

(2') If there exists a stable model m of $\Pi_\forall(F)$ such that $\pi_\forall(sk) \subseteq m$ and

$s_\epsilon \in m$ then there exists a stable model m' of $\Pi(univ(sk(fol(F))))$ such that $\pi_\forall(sk) \subseteq m'$ and $s_\epsilon \in m'$.

From (1) plus (1') and (2') plus (2) the theorem holds.

□

Proof of Corollary 3. The proof of Corollary 3 is similar to this of Corollary 1

□