From (Quantified) Boolean Formulae to Answer Set Programming
Igor Stéphan, Benoit da Mota, Pascal Nicolas

To cite this version:
Igor Stéphan, Benoit da Mota, Pascal Nicolas. From (Quantified) Boolean Formulae to Answer Set Programming. Journal of Logic and Computation, 2009, 19 (4), pp.565 - 590. 10.1093/logcom/exn041. hal-03255405

HAL Id: hal-03255405
https://univ-angers.hal.science/hal-03255405
Submitted on 9 Jun 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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) \cite{22,26,9}. Some recent works show that the CNF transformation seems to disrupt too much the original structure of the problem \cite{36}. However, there exist only few implementations of solvers for non-CNF SAT formulae \cite{36,19,24,13}.

In the same way, the problem of validity of quantified Boolean formulae (QBF) \cite{24} 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 \cite{30,17,7,21,5,31,4}. For QBF, the impact of the transformation in prenex CNF formulae seems even larger \cite{2}. As for SAT solvers, there exist very few QBF solvers for (non prenex) non-CNF formulae \cite{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 developed 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 enounced 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 t and 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 F. The set of propositional symbols (or variables) is denoted with V. The symbols ⊤ and ⊥ are the propositional constants. The symbol ∧ is used for conjunction, ∨ for disjunction, ¬ for negation, → for implication, ↔ for equivalence and ⊕ for xor (O = {∧, ∨, ¬, →, ↔}). A literal is a propositional variable or the negation of a propositional variable. The set of literals is denoted with 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 ¬F is an element of PROP; if F and G are elements of PROP and * is an element of O then (F * G) is an element of PROP. The symbol ∃ is used for existential quantification and ∀ 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 $V(F)$. 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 of variable $x$ by $F$. A binder $Q$ is a string $q_1x_1 \ldots q_nx_n$ with $x_1, \ldots, x_n$ distinct variables and $q_1, \ldots, q_n$ quantifiers. The function $Q$ from the set of variables of a binder to $\{\exists, \forall\}$ associates to a variable its quantifier in the binder $Q$. A QBF $QF$ is in prenex form if it is constituted of a binder and a Boolean formula called the matrix. 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: $\epsilon$ is the occurrence of $\Sigma$ in itself; if $o$ an occurrence of $\sigma$ in $\Sigma_0$ then $0.o$ is an occurrence of $\sigma$ in $\neg\Sigma_0$; if $o$ is an occurrence of $\sigma$ in $\Sigma_0$ then $0.o$ is an occurrence of $\sigma$ in $(\Sigma_0 * \Sigma_1)$, $* \in O$ and $\sigma$ is a sub-formula of $\Sigma_0$; if $o$ is an occurrence of $\sigma$ in $\Sigma_1$ then $1.o$ is an occurrence of $\sigma$ in $(\Sigma_0 * \Sigma_1)$, $* \in O$ and $\sigma$ is a sub-formula of $\Sigma_1$. 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 $\mathcal{M}$. For every Boolean variable $y$ and QBF $F$, $(\exists y \, F) = (F[y \leftarrow \top] \lor F[y \leftarrow \bot])$ and $(\forall y \, F) = (F[y \leftarrow \bot] \land F[y \leftarrow \top])$. A QBF $F$ is valid if $F \equiv \top$. If $y$ is an existentially quantified variable preceded by the universally quantified variables $x_1, \ldots, x_n$ we denote $\hat{y}_{x_1,\ldots,x_n}$ its Skolem function from $BOOL^n$ to $BOOL$. A model for a QBF $F$ is a sequence $s$ of satisfying Skolem functions for $F$ (denoted with $s \models F$). For example, the QBF $\exists y \exists x \forall z ((x \lor y) \equiv z)$ is valid and its possible sequence of satisfying Skolem functions is $\hat{y}_x(t) = t, \hat{y}_z(f) = f, \hat{x}_z(t) = f$ and $\hat{x}_z(f) = f$. Skolem functions are sometimes represented by policies $\mathbb{S}$ or strategies $\mathbb{G}$ which clariﬁes 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 $\exists y \exists x \exists z ((x \lor y) \equiv z)$, the Skolem functions $\hat{x} = t, \hat{y} = f$ and $\hat{z} = t$ correspond to the Boolean model $\nu(x) = t, \nu(y) = f$ and $\nu(z) = t$ for the propositional formula $((x \lor y) \equiv z)$. A QBF is valid if and only if there exists a

\footnote{since all the variables are different}

\footnote{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 $((x \lor y) \equiv z)$.}
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 program-
mong 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 \cite{16} for normal
logic programs.

A normal logic program\footnote{For sake of simplicity we use program in the sequel.} is a finite set of rules like

\[(c \leftarrow a_1, \ldots, a_n, \neg b_1, \ldots, \neg b_m)\]

\[n \geq 0, m \geq 0\] where \(c, a_1, \ldots, a_n, b_1, \ldots, b_m\) are atoms all gathered in the set
\(A\); \(P\) 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 \(S \subseteq A\) such that \(S = Cn(P^S)\). Let us note that a program may
have no, one or many stable models. For instance, \(\{(a.), (b \leftarrow a, \neg d.), (c \leftarrow a, \neg a, \not b.)\}\) has the unique stable model \(\{a, b\}\), \(\{(a \not b.), (b \not a.)\}\) 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 ap-
proach introduced in \cite{27} in the case of propositional formulae given in CNF.
Let \(\Sigma\) be a clause set. The translation of the formula produces a program \(LP(\Sigma)\)
containing rules \((na \leftarrow \neg a.)\) and \((a \leftarrow \neg na.)\) for every atom \(a\) occurring
in \(\Sigma\). For every clause in \(\Sigma\), a new atom \(c\) is created and the rule \((\neg \not c.)\)
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, \(\Sigma\) 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 \(\Sigma\) and rules whose head is \(c\) per-
mitt to infer \(c\) if the interpretation satisfies the corresponding clause. Lastly, for

\footnote{Such a headless rule is called a constraint and is given for a rule like \((bug \leftarrow \neg c, \not bug.)\)
where \(bug\) is a new symbol.}
every clause, constraints forbid all sets not containing \( c \) (that are not models of \( \Sigma \)) 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 \( \pi \) and \( \pi^{-1} \) functions which associate a set of literals of a formula to a set of atoms of a program and reciprocally.

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

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

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

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

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, \( P_Q : 2^V \rightarrow \mathcal{P} \) be a function such that

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

**Theorem 1.** Let \( \Sigma \) be a propositional formula and \( \nu \) be a set of literals. \( \nu \) 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 \( F = ((c \lor b) \land (b \rightarrow ((c \rightarrow d) \land (c \lor (a \leftrightarrow \neg d)))) \) be a propositional formula. \( \mathcal{V}(F) = \{ a, b, c, d \} \) and

\[
P_Q(\mathcal{V}(F)) = \left\{ (a \leftarrow \text{not } na.), (na \leftarrow \text{not } a.), (b \leftarrow \text{not } nb.), (nb \leftarrow \text{not } b.), (c \leftarrow \text{not } nc.), (nc \leftarrow \text{not } c.), (d \leftarrow \text{not } nd.), (nd \leftarrow \text{not } d.) \right\}
\]
For example, the set of atoms \( m = \{na,b,nc,d\} \) is a stable model of the normal logic program \( P_Q(V(F)) \) and \( \pi^{-1}(m,F) = \{-a,b,\neg c,d\} \) is a valuation of \( V(F) \).

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 \( \Sigma, \Sigma_0 \) and \( \Sigma_1 \) 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:

- if \( \Sigma = x \) then \( P(\Sigma,o) = \{(s_o \leftarrow x)\} \)
- if \( \Sigma = \neg \Sigma_0 \) then \( P(\Sigma,o) = \{ (s_o \leftarrow \neg s_{o.0}) \} \cup P(\Sigma_0, o.0) \)
- if \( \Sigma = \Sigma_0 \cup \Sigma_1 \) then \( P(\Sigma,o) = \{ (s_o \leftarrow s_{o.0}, s_{o.1})\} \cup P(\Sigma_0, o.0) \cup P(\Sigma_1, o.1) \)
- if \( \Sigma = (\Sigma_0 \rightarrow \Sigma_1) \) then \( P(\Sigma,o) = \{ (s_o \leftarrow \neg s_{o.0}, s_{o.1})\} \cup P(\Sigma_0, o.0) \cup P(\Sigma_1, o.1) \)
- if \( \Sigma = (\Sigma_0 \oplus \Sigma_1) \) then \( P(\Sigma,o) = \{ (s_o \leftarrow s_{o.0}, not s_{o.1}), (s_o \leftarrow not s_{o.0}, s_{o.1})\} \cup P(\Sigma_0, o.0) \cup P(\Sigma_1, o.1) \)

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

**Example 2** (Example [1] continued). For the propositional formula \( F = ((c \lor b) \land (b \rightarrow ((c \rightarrow d) \land (c \rightarrow (a \rightarrow \neg d)))) \)

\[
P(F,e) = \{ (s_e \leftarrow s_0, s_1), (s_0 \leftarrow s_{02}), (s_0 \leftarrow s_{01}), (s_{02} \leftarrow c),
(s_{01} \leftarrow b), (s_1 \leftarrow not s_{10}), (s_1 \leftarrow s_{12}), (s_1 \leftarrow b),
(s_{12} \leftarrow s_{12}, s_{13}), (s_{12} \leftarrow not s_{12}, s_{2}), (s_{12} \leftarrow s_{12}, s_{10}),
(s_{12} \leftarrow d), (s_1 \leftarrow s_{12}, s_{01}), (s_1 \leftarrow s_{12}, s_{13}),
(s_{13} \leftarrow c), (s_{14} \leftarrow s_{14}, s_{15}), (s_{14} \leftarrow not s_{14}, not s_{15}),
(s_{14} \leftarrow a), (s_{15} \leftarrow not s_{15}, s_{15}), (s_{15} \leftarrow d)\}
\]

From the two above definitions, we introduce in the following definition the function \( \Pi \) which generates from a propositional formula a normal logic program.

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

The following lemma shows that for a formula \( \Sigma \), 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 $\Sigma$ be a propositional formula. The set of literals $\nu$ is a valuation for $V(\Sigma)$ 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 $s_0$ 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 $\Sigma_{init}$ be a propositional formula, $\Sigma$ be a sub-formula of $\Sigma_{init}$ and $m$ be a stable model of $P_Q(V(\Sigma_{init})) \cup P(\Sigma, \epsilon)$. For every occurrence $o$ of $\Sigma$ in $\Sigma_{init}$

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

Theorem 2. Let $\Sigma$ 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 with $\Sigma = \Sigma_{init}$.

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

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

\[
\Pi^+(\Sigma) = \Pi(\Sigma) \cup \{(\leftarrow \text{ not } s_\epsilon.)\}
\]

\[
\Pi^-(\Sigma) = \Pi(\Sigma) \cup \{(\leftarrow s_\epsilon.)\}
\]

The following corollary to the previous Theorem 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 $\Sigma$ be a propositional formula, the normal logic program $\Pi^+(\Sigma)$ has a stable model if and only if $\Sigma$ is satisfiable. Moreover, if $m$ is a stable model of $\Pi^+(\Sigma)$ then $\pi^{-1}(m, \Sigma) \models \Sigma$ ; if $\nu$ is a (Boolean) model for $\Sigma$ 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 $s_\epsilon$ : these eight stable models are such that $\pi^{-1}(m_i, F) \models F, 1 \leq i \leq 8$ ; for example the stable model

\[ m_1 = \{na, nb, c, d\} \cup \{s_\epsilon, s_0, s_1, s_2, s_12, s_120, s_1202, s_12^2, s_120^2, s_2, s_13, s_130, s_14\} \]

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

7
The translation $\Pi^+$ 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 $\Pi^-$ 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 $\Sigma$ be a propositional formula, the normal logic program $\Pi^-(\Sigma)$ has no stable model if and only if $\Sigma$ is a tautology. Moreover, if $m$ is a stable model of $\Pi^-(\Sigma)$ then $\pi^{-1}(m, \Sigma)$ falsifies the formula $\Sigma$; if $\nu$ falsifies the formula $\Sigma$ then there exists a (unique) stable model $m$ of $\Pi^-(\Sigma)$ such that $\pi(\nu) \subseteq m$.

Let us remark that if the formula $\Sigma$ 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 $\Pi$ is polynomial with respect the length of the given formula. Moreover, as introduced in [27] we can say that $\Pi$ 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 $(\Sigma \ast \Sigma')$, $\ast \in \mathcal{O}$. 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 $\ast$ to obtain the whole desired translation. The important point is that we do not have to recompute anything else for $\Sigma$.

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 $P(F, c)$ may be optimised in

$$
\left\{ 
\begin{array}{l}
(s, \leftarrow s_0, s_1), (s_0 \leftarrow c.), (s_0 \leftarrow b.), (s_1 \leftarrow \text{not } b.), (s_1 \leftarrow s_2.), \\
(s_1^2 \leftarrow s_1^2, 0, s_1^2.), (s_1^2, 0 \leftarrow \text{not } c.), (s_1^2, 0 \leftarrow d.), (s_1^3 \leftarrow c.), \\
(s_1^3 \leftarrow s_1^3.), (s_1^4 \leftarrow a, \text{nd}.), (s_1^4 \leftarrow \text{not } a, \text{not } \text{nd}.)
\end{array}
\right\}
$$

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

\[5\text{taking into account that we have to use a new symbol } s' \neq s \text{ in } \Pi(\Sigma')\]
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: $(\forall x \Sigma) = (\Sigma[x \leftarrow T] \land \Sigma[x \leftarrow \bot])$ and then to apply
on this formula, which only contains existentially quantified variables, the re-
sult 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 propos-
itonal 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 quanti-
fied 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 $i(0) = f$ and $i(1) = t$. Definitions of $\pi$ and $\pi^{-1}$ are extended to Skolem
functions and to n-ary predicates.

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

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

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

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

In the following definition, translation functions $P_\forall^Q$ and $P_\forall^p$ have an argu-
ment $S$ (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 $P_\forall^Q$
corresponds to the treatment of the quantifiers: the existential quantification is
treated in a similar way to the propositional case since the universal quantifi-
cation introduces a rule expliciting the constant 1 as interpreted to $t$ and the
semantics of the quantifier as a conjunction (the interpretation of the constant
0 as $f$ 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 \texttt{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** ($P_Q^\Psi$, $P^\Psi$ and $\Pi^\Psi$ functions). Let $\Sigma$, $\Sigma_0$ and $\Sigma_1$ be QBF, $V$ a set of variables, $n$ and $N_\Psi$ in $\mathbb{N}$, $o$ an occurrence and $S$ a function from $V$ to $\mathbb{N}$. Let $P^{\Psi}_Q : QBF \times \mathbb{N} \times \mathbb{N} \times (V \rightarrow \mathbb{N}) \rightarrow \mathcal{P}$ be a function such that

\[
\begin{align*}
\text{if } \Sigma = (\exists x \Sigma_0) \text{ then } P^{\Psi}_Q(\Sigma, N_\Psi, n, S) &= P^{\Psi}_Q(\Sigma_0, N_\Psi, n, S \cup \{(x \mapsto n)\}) \\
&\cup\{(x(U_1, \ldots, U_n) \leftarrow \not n x(U_1, \ldots, U_n)),
\langle n x(U_1, \ldots, U_n) \leftarrow \not x(U_1, \ldots, U_n)\}\} \\
\text{if } \Sigma = (\forall x \Sigma_0) \text{ then } P^{\Psi}_Q(\Sigma, N_\Psi, n, S) &= P^{\Psi}_Q(\Sigma_0, N_\Psi, n + 1, S) \\
&\cup\{(x(U_1, \ldots, U_{N_\Psi}) \leftarrow U_{n+1} = 1))\} \\
&\cup\{(s_0(U_1, \ldots, U_n) \leftarrow s_0^{o+1}(U_1, \ldots, U_n, 0), s_0^{o+1}(U_1, \ldots, U_n, 1))\} \\
\text{if } \Sigma \text{ is a propositional formula then } \\
P^{\Psi}_Q(\Sigma, N_\Psi, n, S) &= P^{\Psi}(\Sigma, 0^{N_\Psi}, N_\Psi, S)
\end{align*}
\]

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

Existentially quantified variable considered as a formula
\[\text{if } \Sigma = x \text{ and } (x \mapsto n) \in S \text{ then } P^{\Psi}(\Sigma, o, N_\Psi, S) = \{(s_o(U_1, \ldots, U_{N_\Psi}) \leftarrow x(U_1, \ldots, U_n))\}\]

Universally quantified variable considered as a formula
\[\text{if } \Sigma = x \text{ and } (x \mapsto n) \notin S \text{ then } P^{\Psi}(\Sigma, o, N_\Psi, S) = \{(s_o(U_1, \ldots, U_{N_\Psi}) \leftarrow x(U_1, \ldots, U_n))\}\]

\[\text{if } \Sigma = \neg \Sigma_0 \text{ then } P^{\Psi}(\Sigma, o, N_\Psi, S) = \{(s_o(U_1, \ldots, U_{N_\Psi}) \leftarrow \not s_o,0(U_1, \ldots, U_{N_\Psi}))\}_{\cap} \cup P^{\Psi}(\Sigma_0, o,0, N_\Psi, S)\]

\[\text{if } \Sigma = (\Sigma_0 \land \Sigma_1) \text{ then } P^{\Psi}(\Sigma, o, N_\Psi, S) = \{(s_o(U_1, \ldots, U_{N_\Psi}) \leftarrow s_o,0(U_1, \ldots, U_{N_\Psi}), s_o,1(U_1, \ldots, U_{N_\Psi}))\}_{\cup} \cup P^{\Psi}(\Sigma_0, o,0, N_\Psi, S) \cup P^{\Psi}(\Sigma_1, o,1, N_\Psi, S)\]

\[\text{if } \Sigma = (\Sigma_0 \lor \Sigma_1) \text{ then } P^{\Psi}(\Sigma, o, N_\Psi, S) = \{(s_o(U_1, \ldots, U_{N_\Psi}) \leftarrow s_o,0(U_1, \ldots, U_{N_\Psi})), \}
\{(s_o(U_1, \ldots, U_{N_\Psi}) \leftarrow s_o,1(U_1, \ldots, U_{N_\Psi})), \}
\cup P^{\Psi}(\Sigma_0, o,0, N_\Psi, S) \cup P^{\Psi}(\Sigma_1, o,1, N_\Psi, S)\]
Let \( \Pi \) Example 5 with \( F \) placed by \( s \) with intermediary call in which every variable is replaced by 0 or 1.

Let \( \Pi \) be a QBF and \( N_v \) the number of universally quantified variables of \( \Sigma \). Let \( \Pi_\Sigma : QBF \rightarrow P \) be a function such that \( \Pi_\Sigma = P^\Sigma(\Sigma, N_v, 0, \emptyset) \).

By construction the translation is polynomial.

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

\[
\Pi_\Sigma(F_{a \vee b \vee c \vee d}) = P^\Sigma(F_{a \vee b \vee c \vee d})\cup
\begin{cases}
(a \leftarrow \not\text{not } a),
\begin{cases}
(b(U_1, U_2) \leftarrow U_1 = 1),
(s_0 \leftarrow s_0(0), s_0(1)),
(c(U_1) \leftarrow \not\text{not } c(U_1)),
(d(U_1, U_2) \leftarrow U_2 = 1),
(s_0(U_1) \leftarrow s_0(U_1, 0), s_0(U_1, 1))
\end{cases}
\end{cases}
\]

with \( P_{a \vee b \vee c \vee d} \) equal to \( P(F, 0^2) \) in which all \( s \) (resp. \( b, c \) and \( d \)) are replaced by \( s_0(U_1, U_2) \) (resp. \( b(U_1, U_2), c(U_1) \) and \( d(U_1, U_2) \)) and with intermediary call \( P^\Sigma(F, 2, 2, \{a \leftrightarrow 0, \{c \rightarrow 1\}) \).

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

\[
\Pi_\Sigma(F_{a \exists b \forall c \exists d}) = P^\Sigma(F_{a \exists b \forall c \exists d})\cup
\begin{cases}
(a(U_1, U_2) \leftarrow U_1 = 1),
\begin{cases}
(b(U_1) \leftarrow \not\text{not } b(U_1)),
(s_r \leftarrow s_0(0), s_0(1)),
(c(U_1, U_2) \leftarrow U_2 = 1),
(s_0(U_1) \leftarrow s_0(U_1, 0), s_0(U_1, 1)),
(d(U_1, U_2) \leftarrow \not\text{not } d(U_1, U_2)),
(n)(U_1, U_2) \leftarrow \not\text{not } d(U_1, U_2))
\end{cases}
\end{cases}
\]

with \( P_{a \exists b \forall c \exists d} \) equal to \( P(F, 0^2) \) in which all \( s \) (resp. \( a, b, c \) and \( d \)) are replaced by \( s_0(U_1, U_2) \) (resp. \( a(U_1, U_2), b(U_1), c(U_1, U_2) \) and \( d(U_1, U_2) \)) and with intermediary call \( P^\Sigma(F, 2, 2, \{b \rightarrow 1, \{d \rightarrow 2\}) \).

Let the reader note that the obtained program is a first order one in which only two constants \( (0 \text{ and } 1) \) 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 $\Sigma$ 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 $F_{3a\lor b\lor c\lor d} = \exists a \lor b \lor c \lor d F$ has no satisfying Skolem function and $\Pi_\forall(F_{3a\lor b\lor c\lor d})$ has no stable model.

As in the SAT case, the function $\Pi$ is extended to the function $\Pi^+$ (resp. $\Pi^-$) to restrict the stable models corresponding to the valuations which falsify the formula). We define the function $\Pi^+$ (resp. $\Pi^-$) as an extension of the function $\Pi_\forall$ to restrict the Skolem functions to those which satisfy (resp. not satisfy) the QBF.

Definition 8 ($\Pi^+$ and $\Pi^-$ functions). Let $\Sigma$ be a QBF. Let $\Pi^+$ and $\Pi^- : QBF \to \mathcal{P}$ be functions such that

$$\Pi^+(\Sigma) = \Pi_\forall(\Sigma) \cup \{(\neg \text{ not } s_\epsilon,\}$$
$$\Pi^-(\Sigma) = \Pi_\forall(\Sigma) \cup \{(\neg s_\epsilon,\}$$

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 $\Sigma$ be a QBF, the normal logic program $\Pi_\forall^+(\Sigma)$ has a stable model if and only if $\Sigma$ is valid. Moreover, if $m$ is a stable model of $\Pi_\forall^+(\Sigma)$ then $\pi_\forall^+(m,\Sigma)$ is a set of Skolem functions which satisfy the QBF ; if $sk$ is a set of Skolem functions which satisfy $\Sigma$ 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:

$$sk = \{ \hat{b}_a(t) = t, \hat{b}_a(f) = t, \hat{d_{ac}}(t,f) = f, \hat{d_{ac}}(f,t) = t \}$$

which corresponds to the valid policy or winning strategy

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

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

$$\begin{array}{l}
\{b(1), b(0), d(1,1), d(0,1), d(0,0), nd(1,0)\} \cup \\
\{a(1,0), a(1,1), c(0,1), c(1,1)\} \cup \\
\{s_\epsilon, s_0(0), s_0(1), s_0(0,0), s_0(1,0), s_0(0,1), s_0(1,1)\} \cup \\
\{ s_\epsilon(A,C) \mid A, C \in \{0,1\}, o \in \{0^2,1,0^2.1,0^2.1^2,0^2.1^2.0,0^2.1^3\} \} \cup \\
\{s_0.1^2(0,0), s_0.1^2(1,0), s_0.1^2(0,1)\}
\end{array}$$
So, we have $\pi^{-1}_y(m, F_{\forall a \exists b \exists c}) = sk$ and $\pi_y(sk) \subseteq m$.

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 $s_{0,14}(1,1)$ is not in the stable model since it represents the sub-formula $(a \leftrightarrow \neg d)$ for the variable $a$ interpreted to $t$ (the first 1 of $s_{0,14}(1,1)$) and the variable $d$ interpreted to $t$ (since the atom $d(1,1)$ is in the stable model) and have to be interpreted to $f$.

## 5 Implementation

The theoretical principles exposed in previous sections have been implemented in three different tools: edimacs2nlp, qbf2nlp, qedimacs2nlp, freely available\[1\]. These tools admit two kinds of input formats: EDIMACS format for SAT competition\[2\] and QBF 1.0 format (extended with implication, equivalence and xor) for QBF competition\[3\]. 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).
b(U0) :- not b(U0),bool(U0).
d(U0,U1) :- not nd(U0,U1),bool(U0),bool(U1).
d(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 := sigma_1(0),sigma_1(1).
```

\[2\] http://www.satcompetition.org/2005/edimacs.pdf
\[3\] http://www.qbfilib.org/boole.html
is the output of qbf2nlp on file in QBF 1.0 format

\[ \forall a \exists b \forall c \exists d \ ( (c \lor b) \land (b \rightarrow ((c \rightarrow d) \land (c \lor (a \leftrightarrow \neg d)))) \]

for the QBF \( \forall a \exists b \forall c \exists d ((c \lor b) \land (b \rightarrow ((c \rightarrow d) \land (c \lor (a \leftrightarrow \neg d)))) \).

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 [28] (zChaff 2007.3.12.) and MiniSat. The experiments have been realized on a 2 × 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
Table 1: Experimental results for SAT.

<table>
<thead>
<tr>
<th>QG6 SAT</th>
<th>SAT CNF</th>
<th>SAT non CNF</th>
</tr>
</thead>
<tbody>
<tr>
<td>NbR</td>
<td>zChaff</td>
<td>MiniSat</td>
</tr>
<tr>
<td>% solved</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>ATR</td>
<td>19.39</td>
<td>3.25</td>
</tr>
<tr>
<td>ATRS</td>
<td>19.39</td>
<td>3.25</td>
</tr>
<tr>
<td>QG6 UNSAT</td>
<td>zChaff</td>
<td>MiniSat</td>
</tr>
<tr>
<td>NbR</td>
<td>137</td>
<td>165</td>
</tr>
<tr>
<td>% solved</td>
<td>79.19%</td>
<td>95.38%</td>
</tr>
<tr>
<td>ATR</td>
<td>1260.08</td>
<td>680.15</td>
</tr>
<tr>
<td>ATRS</td>
<td>645.21</td>
<td>538.58</td>
</tr>
</tbody>
</table>

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 QBFE-VAL07 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/](http://www.qbflib.org/)

\(^{10}\) programs with a disjunction in the head of rules
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

<table>
<thead>
<tr>
<th>Instance</th>
<th>NbV</th>
<th>smodels</th>
<th>CLASP</th>
<th>Nb CP</th>
<th>Nb atoms</th>
<th>Nb rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter5,2</td>
<td>15/10</td>
<td>0.180</td>
<td>0.141</td>
<td>0</td>
<td>2 824</td>
<td>2 869</td>
</tr>
<tr>
<td>counter6,2</td>
<td>18/12</td>
<td>0.432</td>
<td>0.423</td>
<td>0</td>
<td>10 108</td>
<td>10 173</td>
</tr>
<tr>
<td>counter7,2</td>
<td>21/14</td>
<td>2.420</td>
<td>2.574</td>
<td>0</td>
<td>37 741</td>
<td>37 830</td>
</tr>
<tr>
<td>counter8,2</td>
<td>24/16</td>
<td>20.465</td>
<td>22.292</td>
<td>0</td>
<td>144 547</td>
<td>144 664</td>
</tr>
<tr>
<td>counter4,4</td>
<td>20/16</td>
<td>3.290</td>
<td>4.911</td>
<td>0</td>
<td>131 925</td>
<td>131 988</td>
</tr>
<tr>
<td>counter5,5</td>
<td>24/20</td>
<td>55.730</td>
<td>513.861</td>
<td>0</td>
<td>2 099 267</td>
<td>2 098 352</td>
</tr>
<tr>
<td>counter5,4</td>
<td>25/20</td>
<td>64.923</td>
<td>531.140</td>
<td>0</td>
<td>2 099 278</td>
<td>2 099 373</td>
</tr>
<tr>
<td>ring4,2</td>
<td>21/14</td>
<td>1.006</td>
<td>1.075</td>
<td>24</td>
<td>34 466</td>
<td>34 665</td>
</tr>
<tr>
<td>ring5,2</td>
<td>24/16</td>
<td>4.024</td>
<td>5.734</td>
<td>25</td>
<td>135 085</td>
<td>135 314</td>
</tr>
<tr>
<td>ring6,2</td>
<td>27/18</td>
<td>22.300</td>
<td>45.870</td>
<td>10</td>
<td>534 572</td>
<td>534 831</td>
</tr>
<tr>
<td>ring4,3</td>
<td>28/21</td>
<td>154.864</td>
<td>M</td>
<td>305</td>
<td>4 197 482</td>
<td>4 199 521</td>
</tr>
<tr>
<td>semaphore,2</td>
<td>21/14</td>
<td>1.090</td>
<td>1.130</td>
<td>5</td>
<td>34 967</td>
<td>35 156</td>
</tr>
<tr>
<td>semaphore,3</td>
<td>28/21</td>
<td>159.452</td>
<td>M</td>
<td>9</td>
<td>4 198 407</td>
<td>4 198 996</td>
</tr>
<tr>
<td>semaphore3,2</td>
<td>27/18</td>
<td>18.327</td>
<td>43.140</td>
<td>5</td>
<td>529 994</td>
<td>530 284</td>
</tr>
</tbody>
</table>

Table 2: Experimental results for QBFEVAL07.
Table 3: Number of rules.

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

Table 4: Results for Lparse + smodels.

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

Table 5: Results for GrinGo + smodels.

<table>
<thead>
<tr>
<th>NBC</th>
<th>CPU GrinGo + smodels</th>
<th>GrinGo</th>
<th>smodels</th>
<th>Total</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td>0.27 (63%)</td>
<td>0.11 (24%)</td>
<td>0.38</td>
<td>0.09</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>0.56 (52%)</td>
<td>0.47 (43%)</td>
<td>1.03</td>
<td>0.59</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>1.15 (44%)</td>
<td>1.42 (54%)</td>
<td>2.63</td>
<td>1.32</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>2.33 (13%)</td>
<td>15.78 (87%)</td>
<td>18.11</td>
<td>21.55</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>4.75 (14%)</td>
<td>29.83 (86%)</td>
<td>34.58</td>
<td>45.75</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>10.01 (11%)</td>
<td>77.85 (89%)</td>
<td>87.86</td>
<td>101.24</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>19.96 (7%)</td>
<td>274.56 (93%)</td>
<td>294.52</td>
<td>291.79</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>42.85 (3%)</td>
<td>1189.43 (97%)</td>
<td>1232.28</td>
<td>1648.32</td>
</tr>
</tbody>
</table>

Table 6: Results for Lparse + CLASP.

<table>
<thead>
<tr>
<th>NBC</th>
<th>CPU Lparse + CLASP</th>
<th>Lparse</th>
<th>CLASP</th>
<th>Total</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td>0.21 (64%)</td>
<td>0.07 (23%)</td>
<td>0.33</td>
<td>0.06</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>0.57 (73%)</td>
<td>0.17 (22%)</td>
<td>0.74</td>
<td>0.27</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>1.59 (80%)</td>
<td>0.37 (19%)</td>
<td>2.00</td>
<td>0.23</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>6.46 (88%)</td>
<td>0.82 (11%)</td>
<td>7.28</td>
<td>3.69</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>17.68 (91%)</td>
<td>1.79 (9%)</td>
<td>19.51</td>
<td>1.75</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>77.43 (95%)</td>
<td>4.29 (5%)</td>
<td>81.72</td>
<td>12.50</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>265.95 (96%)</td>
<td>10.02 (4%)</td>
<td>276.03</td>
<td>43.53</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>1257.33 (98%)</td>
<td>29.71 (2%)</td>
<td>1287.11</td>
<td>255.18</td>
</tr>
</tbody>
</table>
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 $\Sigma = (a \rightarrow (b \lor c))$, the normal form translations presented in [29, 10] lead to the clause set $C = \{L_a, \neg L_a \lor \neg L_{b\lor c}, \neg L_{b\lor c} \lor b \lor c\}$. Then, applying the transformation for CNF formulas recalled in subsection 2.2 leads to a program containing five pairs of rules $(x \leftarrow \text{not } nx), (nx \leftarrow \text{not } x)$ one for each variable $x \in C$, 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 $\Sigma$. 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

<table>
<thead>
<tr>
<th>NBC</th>
<th>CPU GrinGo + CLASP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GrinGo</td>
</tr>
<tr>
<td>10</td>
<td>0.30 (73%)</td>
</tr>
<tr>
<td>11</td>
<td>0.56 (74%)</td>
</tr>
<tr>
<td>12</td>
<td>1.16 (76%)</td>
</tr>
<tr>
<td>13</td>
<td>2.42 (77%)</td>
</tr>
<tr>
<td>14</td>
<td>4.93 (76%)</td>
</tr>
<tr>
<td>15</td>
<td>10.28 (76%)</td>
</tr>
<tr>
<td>16</td>
<td>20.68 (72%)</td>
</tr>
<tr>
<td>17</td>
<td>43.50 (67%)</td>
</tr>
</tbody>
</table>

Table 7: Results for GrinGo + CLASP.

<table>
<thead>
<tr>
<th>NBC</th>
<th>CPU DLV</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Instantiation</td>
</tr>
<tr>
<td>10</td>
<td>0.26 (70%)</td>
</tr>
<tr>
<td>11</td>
<td>0.54 (68%)</td>
</tr>
<tr>
<td>12</td>
<td>1.19 (66%)</td>
</tr>
<tr>
<td>13</td>
<td>2.70 (50%)</td>
</tr>
<tr>
<td>14</td>
<td>5.44 (55%)</td>
</tr>
<tr>
<td>15</td>
<td>11.96 (43%)</td>
</tr>
<tr>
<td>16</td>
<td>25.04 (32%)</td>
</tr>
<tr>
<td>17</td>
<td>35.73 (14%)</td>
</tr>
</tbody>
</table>

Table 8: Results for DLV.
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


7 Proofs

Proof of Theorem 7. Let \( \Sigma \) be a propositional formula such that \( \mathcal{V}(\Sigma) = \{x\} \). \( \Sigma \) has two valuations \( \{x\} \) and \( \{\neg x\} \). \( \pi(\{x\}) = \{x\} \) and \( \pi(\{\neg x\}) = \{nx\} \).

Let \( \nu \) be a set of literals, \( S \) a set of atoms and the normal logic program such that \( P_Q(\nu) = \{ (x \leftarrow \text{not } nx) , (nx \leftarrow \text{not } x) \} \)

If \( S = \emptyset \)

\[
\begin{align*}
P_Q(\nu)^S &= \{ (x) , (nx) \} \\
C_n(P_Q(\nu)^S) &= \{x, nx\} \neq S ;
\end{align*}
\]

If \( S = \{x\} \)

\[
\begin{align*}
P_Q(\nu)^S &= \{ (x) \} \\
C_n(P_Q(\nu)^S) &= \{x\} = S ;
\end{align*}
\]

If \( S = \{nx\} \)

\[
\begin{align*}
P_Q(\nu)^S &= \{ (nx) \} \\
C_n(P_Q(\nu)^S) &= \{nx\} = S ;
\end{align*}
\]

If \( S = \{x, nx\} \)

\[
\begin{align*}
P_Q(\nu)^S &= \emptyset \\
C_n(P_Q(\nu)^S) &= \emptyset \neq S .
\end{align*}
\]
So, $P_Q(\nu)$ has two stable models $\{x\}$ and $\{nx\}$.

\[ \pi^{-1}(\{x\}), \Sigma = \{x\} \text{ and } \pi^{-1}(\{nx\}), \Sigma = \{\neg x\}. \]

The theorem holds for $\nu = \mathcal{V}(\Sigma)$ and $|\mathcal{V}(\Sigma)| = 1$.

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

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

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 $\{ \{x \Leftarrow not nx\}, \{nx \Leftarrow not x\} \}$ is a stable model of $P_Q(\nu')$.

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

If $S = \pi(\nu) \cup \emptyset$ \hspace{1cm} $P_Q(\nu')^S = P_Q(\nu)^S \cup \{ \{x\}, \{nx\} \}$ \hspace{1cm} $C_n(P_Q(\nu')^S) = \pi(\nu) \cup \{x, nx\} \neq S$

If $S = \pi(\nu) \cup \{x\}$ \hspace{1cm} $P_Q(\nu')^S = P_Q(\nu)^S \cup \{ \{x\} \}$ \hspace{1cm} $C_n(P_Q(\nu')^S) = \pi(\nu) \cup \{x\} = S$

If $S = \pi(\nu) \cup \{nx\}$ \hspace{1cm} $P_Q(\nu')^S = P_Q(\nu)^S \cup \{ \{nx\} \}$ \hspace{1cm} $C_n(P_Q(\nu')^S) = \pi(\nu) \cup \{nx\} = S$

If $S = \pi(\nu) \cup \{x, nx\}$ \hspace{1cm} $P_Q(\nu')^S = P_Q(\nu)^S \cup \emptyset$ \hspace{1cm} $C_n(P_Q(\nu')^S) = \pi(\nu) \cup \emptyset \neq S$

$P_Q(\nu')$ has two stable models for each stable model $\pi(\nu)$ of $P_Q(\nu)$: $\pi(\nu) \cup \{x\}$ and $\pi(\nu) \cup \{nx\}$.

\[ \pi^{-1}(\pi(\nu) \cup \{x\}), \Sigma = \nu \cup \{x\} \text{ and } \pi^{-1}(\pi(\nu) \cup \{nx\}), \Sigma = \nu \cup \{nx\}. \]

By induction the theorem holds.

\[ \square \]

Proof of Lemma 2. Let $\Sigma_{init}$ and $\Sigma$ be propositional formulae such that $\Sigma$ is a subformula of $\Sigma_{init}$. 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 $\Sigma$ and $m$ be a stable model of $\Pi(\Sigma)$. We prove the following lemma: “The set of literals $\nu$ 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.

- $\Sigma = x$.

\[ \Pi(\Sigma) = P_Q(\mathcal{V}(\Sigma_{init}) \setminus \{x\}) \cup P_Q(\{x\}) \cup P(\Sigma, o) \]

\[ = P_Q(\mathcal{V}(\Sigma_{init}) \setminus \{x\}) \cup \{ \{x \Leftarrow not nx\}, \{nx \Leftarrow not x\} \} \]

Let $\nu$ be a valuation of $\mathcal{V}(\Sigma_{init})$ and if $x \in \nu$ then $\nu' = \nu \setminus \{x\}$ else $\nu' = \nu \setminus \{\neg x\}$ ($\nu'$ 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 $\nu$ be a set of literals and $m$ be a stable model of $\Pi(\Sigma)$ such that
Lemma 2 with $\Sigma = \Sigma$.

Proof of Lemma 2 and Theorem 2.

Let $\Sigma$ be a propositional formula such that $\Pi$ is a subformula of $\Sigma$, and let $\Sigma'$ be the valuation of $V(\Sigma_{init})$. Then in both cases $\nu$ is a valuation of $V(\Sigma_{init})$.

Assuming Lemma 2 holds for $\Sigma_1$ and $\Sigma_2$ subformulae of $\Sigma$ and $m$ a stable model of $\Pi'(\Sigma)$:

- $\Sigma = -\Sigma_1$
  
  $\Pi'(\Sigma) = P_Q(V(\Sigma_{init})) \cup \bigcup \{(s_o \leftarrow not s_{o,0.1})\}$

- $\Sigma = (\Sigma_1 \land \Sigma_2)$
  
  Cases for $\lor$, $\land$, $\lor$ and $\land$ are similar.

Therefore by induction the lemma holds. Since $\Sigma$ is a subformula of $\Sigma_{init}$, it also holds for $\Sigma_{init}$. Hence Lemma 2 holds. ✷

Proof of Lemma 2 and Theorem 2. Let $\Sigma$ be a propositional formula such that $\Sigma$ is a subformula of $\Sigma_{init}$. Let $\Pi'(\Sigma)$ be a normal logic program such that $\Pi'(\Sigma) = P_Q(V(\Sigma_{init})) \cup \bigcup \{(s_o \leftarrow not s_{o,0.1})\}$, $\nu$ be the occurrence of $\Sigma$ and $m$ be a stable model of $\Pi'(\Sigma)$. Proof of Lemma 2 is by induction. Theorem 2 follows from Lemma 2 with $\Sigma = \Sigma_{init}$.

- $\Sigma = x$
  
  $\Pi'(\Sigma) = P_Q(V(\Sigma_{init}) \setminus \{x\}) \cup \bigcup \{(s_o \leftarrow x_{o,0.1})\}$

$s_o \in m$ 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 $\pi^{-1}$, $x \in m$. 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 $\Sigma_1$ and $\Sigma_2$ subformulae of $\Sigma$ and $m$ a stable model of $\Pi'(\Sigma)$:
• $\Sigma = \neg \Sigma_1$.

\[
\Pi'(\Sigma) = P_Q(V(\Sigma_{init})) \cup P(\Sigma_1, o.0) \cup \{ (s_o \leftarrow not s_{o.0}) \}
\]

$\Pi'(\Sigma_1) \cup \{ (s_o \leftarrow not s_{o.0}) \}$

$s_o \in m$ 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 $\pi^{-1}(m, \Sigma_1) \neq \Sigma_1$, therefore $\pi^{-1}(m, \Sigma) \neq \Sigma_1$ (because $V(\Sigma_1) = V(\Sigma)$) and $\pi^{-1}(m, \Sigma) = \Sigma$.

Conversely, if $\pi^{-1}(m, \Sigma) = \Sigma$, then $\pi^{-1}(m, \Sigma) \neq \Sigma_1$, 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 1 holds for this case.

• $\Sigma = (\Sigma_1 \land \Sigma_2)$. Cases for $\lor$, $\rightarrow$, $\oplus$ and $\leftrightarrow$ are similar.

\[
\Pi'(\Sigma) = P_Q(V(\Sigma_{init})) \cup P(\Sigma_1, o.0) \cup P(\Sigma_2, o.1) \cup \{ (s_o \leftarrow s_{o.0}, s_{o.1.}) \}
\]

$\Pi'(\Sigma_1) \cup \Pi'(\Sigma_2) \cup \{ (s_o \leftarrow s_{o.0}, s_{o.1.}) \}$

$s_o \in m$ 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 $\pi^{-1}(m, \Sigma_1) = \Sigma_1$ and $\pi^{-1}(m, \Sigma_2) = \Sigma_2$, therefore $\pi^{-1}(m, \Sigma) = \Sigma_1$ and $\pi^{-1}(m, \Sigma) = \Sigma_2$ (because $V(\Sigma_1) \subseteq V(\Sigma)$ and $V(\Sigma_2) \subseteq V(\Sigma)$) and $\pi^{-1}(m, \Sigma) = \Sigma$.

Conversely, if $\pi^{-1}(m, \Sigma) = \Sigma$, then $\pi^{-1}(m, \Sigma) = \Sigma_1$ and $\pi^{-1}(m, \Sigma) = \Sigma_2$ 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 1 holds for this case.

Therefore by induction Lemma 1 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 $(\leftarrow not s_e)$, $s_e \in m$. $m$ is also a stable model of $\Pi(\Sigma)$ (since $m = Cn(\Pi(\Sigma)^m) = Cn(\Pi^+(\Sigma)^m)$).

Then by Theorem 2, $\pi^{-1}(m, \Sigma) = \Sigma$ and $\Sigma$ is satisfiable.

• Conversely, let $\Sigma$ be satisfiable and $\nu$ a (Boolean) model of $\Sigma$. 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 $s_e \in m$. $m = Cn(\Pi(\Sigma)^m)$ and since $s_e \in m$, $m = Cn(\Pi^+(\Sigma)^m)$. 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 $\Sigma$ be a QBF. If $m$ is a stable model of $\Pi_{\nu}(\Sigma)$ such that $s_e \in m$ then $\pi_{\nu}^{-1}(m, \Sigma) is a set of Skolem functions which satisfy the QBF : if $sk$ is a set of Skolem functions which satisfy $\Sigma$ then there exists a stable model $m$ of $\Pi_{\nu}(\Sigma)$ such that $\pi_{\nu}(sk) \subseteq m$ and $s_e \in m$.”
The proof is in two steps. First we prove that there exists a normal logic program \( nlp_F \) (and an atom \( s \)) for a QBF \( F \) such that: If \( m \) is a stable model of \( nlp_F \) such that \( s \in m \) 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 \( \Sigma \) then there exists a stable model \( m \) of \( nlp_F \) such that \( \pi v(sk) \subseteq m \) and \( s \in m \). Then, we prove that \( nlp_F \) has the same stable models (w.r.t. the atoms containing Skolem function symbols) as \( \Pi v(F) \).

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 \( L \) be a first-order language with \( PS = \{ p / 1 \} \) the set of predicate symbols and \( CS = \{ 1, 0 \} \) the set of constant symbols (the set of function symbols is empty). Let \( M \) be a structure such that the interpretation of predicate \( p \) is: \( p(1) \) is true and \( p(0) \) is false. Let \( fol : QB F \to F O L \) 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 \( F_{\forall a \exists b \forall c : d} = \forall a \exists b \forall c : d((c \vee b) \land (b \rightarrow ((c \rightarrow d) \land (c \leftrightarrow (a \rightarrow d)))) \) and \( fol(F_{\forall a \exists b \forall c : d}) = \forall a \exists b \forall c : d((p(c) \lor (p(b) \land (p(b) \rightarrow ((p(c) \rightarrow p(d)) \land (p(c) \lor (p(a) \rightarrow p(d))))))) \) be a normal first-order logic program.

Proof continued. Let \( sk : F O L \to F O L \) be the function that computes the Skolem form of a (prefix) 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.

\[
\begin{align*}
\text{sk}(\text{fol}(F_{\forall a \exists b \forall c : d})) &= \forall a \exists b ((p(c) \lor (p(b) \land (p(b) \rightarrow ((p(c) \rightarrow p(d(a, c))) \land (p(c) \lor (p(a) \rightarrow p(d(a, c)))))))
\end{align*}
\]

Proof continued. Using the Skolem theorem we obtain that \( F \) is satisfiable if and only if \( sk(f ol(F)) \) 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 detailed since we do not use it). Finally, \([4]\) states that \( prop(sk(f ol(F))) \) 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:

\[
\text{univ}(sk(f ol(F))) = \bigwedge_{(b_1 \ldots b_{\forall \forall}) \in \{0,1\}^{\forall \forall}} \sigma_{b_1 \ldots b_{\forall \forall}},
\]

with for each \( b_1 \ldots b_{\forall \forall} \in \{0,1\}^{\forall \forall}, \sigma_{b_1 \ldots b_{\forall \forall}} \) is \( sk(f ol(F)) \) in which for each universal variable \( u_i \), \( p(u_i) \) is replaced by \( p(u_i(b_1, \ldots, b_{\forall \forall})) \), for each universal variable \( u_i \), \( \neg p(u_i(b_1, \ldots, b_{\forall \forall})) \) if \( b_i = 0 \) and \( p(u_i(b_1, \ldots, b_{\forall \forall})) \) 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 $\forall a \forall c M = sk(fol(F_{a \forall b \forall c \exists d}))$.

\[
univ(sk(fol(F_{a \forall b \forall c \exists d}))) = \\
[M[a \leftarrow 1][e \leftarrow 1] \land p(a(1,1)) \land p(c(1,1))] \\
\land [M[a \leftarrow 1][e \leftarrow 0] \land p(a(1,0)) \land \neg p(c(1,0))] \\
\land [M[a \leftarrow 0][e \leftarrow 1] \land \neg p(a(0,1)) \land p(c(0,1))] \\
\land [M[a \leftarrow 0][e \leftarrow 0] \land \neg p(a(0,0)) \land \neg p(c(0,0))]
\]

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 $\pi'_\nu : 2^F \rightarrow 2^\xi$ be a function such that, for every $sk \in 2^F$,

\[
\pi'_\nu(sk) = \\
\{x(i^{-1}(u_1),\ldots,i^{-1}(u_n)) \mid \hat{x} \in sk, u_1,\ldots,u_n \in BOOL, \hat{x}(u_1,\ldots,u_n) = t\} \cup \\
\{\neg x(i^{-1}(u_1),\ldots,i^{-1}(u_n)) \mid \hat{x} \in sk, u_1,\ldots,u_n \in BOOL, \hat{x}(u_1,\ldots,u_n) = f\}
\]

Example continued.

\[
sk = \{ \hat{b}_a(t) = t, \hat{b}_a(f) = t, \hat{d}_{ac}(t,t) = t, \hat{d}_{ac}(t,f) = f, \hat{d}_{ac}(f,t) = t, \hat{d}_{ac}(f,f) = t \}
\]

\[
\pi'_\nu(sk) = \{b(1),b(0),d(1,1),d(0,1),d(0,0),\neg d(1,0)\}
\]

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

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

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

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

We have proven the first part.

Example continued.

\[
m = \\
\{b(1),b(0),d(1,1),d(0,1),d(0,0),nd(1,0)\} \cup \\
\{a(1,0),a(1,1),c(0,1),c(1,1)\} \cup \\
\{s_0(s_0(0),s_0(1),s_0(0),s_0(1),s_0(0),s_0(1))\} \cup \\
\{s_A(A,C) \mid A,C \in \{0,1\}, o \in \{0^3, 0^2.1, 0^2.1^2, 0^2.1^2, 0^2.1^3\}\} \cup \\
\{s_0(0), s_0(0), s_0(1), s_0(1), s_0(0), s_0(1), s_0(1), s_0(1)\} \cup \\
\{s_0^2.1(0,0), s_0^2.1(1,0), s_0^2.1(0,1)\}
\]
\[
\pi'_\nu(sk) = \{b(1), b(0), d(1,1), d(0,1), d(0,0), \neg d(1,0)\}
\]
\[
\pi(m, \text{univ}(sk(fol(F_{\forall x \exists y z d(x,y,z)}))) = \{b(1), b(0), d(1,1), d(0,1), d(0,0), \neg d(1,0)\} = \pi'_\nu(sk)
\]
\[
\pi(\pi'_\nu(sk)) = \{b(1), b(0), d(1,1), d(0,1), d(0,0), nd(1,0)\} = \pi'_\nu(sk)
\]

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 \(nlp_1 = \Pi(\text{univ}(sk(fol(F)))\) and \(m_1\) one of its stable models. By construction and Lemma \(4\) for all universal variable \(u_i\) and for all \(b_1, \ldots, b_{N^v} \in \{0,1\}^{N^v}\), if \(b_i = 1\) then \(u_i(b_1, \ldots, b_{N^v}) \in m_1\) and \(nu_i(b_1, \ldots, b_{N^v}) \notin m_1\) otherwise \(b_i = 0\), \(u_i(b_1, \ldots, b_{N^v}) \notin m_1\) and \(nu_i(b_1, \ldots, b_{N^v}) \in m_1\).

Example continued.

\[
nlp_1 = \{(s_0 \leftarrow s_{s,0}, s_{s,1})\} \cup
\]
\[
\{ \text{With root } s_0, \text{ the translation of } a(1,1) \land c(1,1) \land a(1,0) \land \neg c(1,0) \land a(0,1) \land c(0,1) \land \neg a(0,0) \land \neg c(0,0) \} \cup
\]
\[
\{(s_{s,1} \leftarrow s_{s,0}, s_{s,1}), (s_{s,0} \leftarrow s_{s,0}, s_{s,0}, s_{s,1}), (s_{s,1} \leftarrow s_{s,1}, s_{s,1})\} \cup
\]
\[
\{(s_1,1 \leftarrow \text{not } nc(1,1)), (nc(1,1) \leftarrow \text{not } c(1,1)), (c(1,0) \leftarrow \text{not } nc(1,0)), (nc(1,0) \leftarrow \text{not } c(1,0)),
\]
\[
(c(0,0) \leftarrow \text{not } nc(0,0)), (nc(0,0) \leftarrow \text{not } c(0,0)), (a(1,1) \leftarrow \text{not } na(1,1)),
\]
\[
(na(1,1) \leftarrow \text{not } a(1,1)), (a(0,1) \leftarrow \text{not } na(0,1)), (na(0,1) \leftarrow \text{not } a(0,1)),
\]
\[
(a(0,0) \leftarrow \text{not } na(0,0)), (na(0,0) \leftarrow \text{not } a(0,0)), (d(1,1) \leftarrow \text{not } nd(1,1)),
\]
\[
(nd(1,1) \leftarrow \text{not } d(1,1)), (d(1,0) \leftarrow \text{not } nd(1,0)), (nd(1,0) \leftarrow \text{not } d(1,0)),
\]
\[
(d(0,1) \leftarrow \text{not } nd(0,1)), (nd(0,1) \leftarrow \text{not } d(0,1)), (d(0,0) \leftarrow \text{not } nd(0,0)), (nd(0,0) \leftarrow \text{not } d(0,0)),
\]
\[
(b(1) \leftarrow \text{not } nb(1)), (nb(1) \leftarrow \text{not } b(1)), (b(0) \leftarrow \text{not } nb(0)), (nb(0) \leftarrow \text{not } b(0))\}
\]
\[
\cup \sigma_{11} \cup \sigma_{01} \cup \sigma_{10} \cup \sigma_{00}
\]
\[
\sigma_{11} = \{(s_1,1,3 \leftarrow s_{s,1,3,0}, s_{s,1,3,1}), (s_{s,1,3,0} \leftarrow s_{s,1,3,0}),
\]
\[
(s_{s,1,3,0} \leftarrow s_{s,1,3,0}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}),
\]
\[
(s_{s,1,3,0} \leftarrow s_{s,1,3,0}), (s_{s,1,3,0} \leftarrow s_{s,1,3,0}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}),
\]
\[
(s_{s,1,3,1} \leftarrow s_{s,1,3,1}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}),
\]
\[
(s_{s,1,3,1} \leftarrow s_{s,1,3,1}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}),
\]
\[
(s_{s,1,3,1} \leftarrow s_{s,1,3,1}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}),
\]
\[
(s_{s,1,3,1} \leftarrow s_{s,1,3,1}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}),
\]
\[
(s_{s,1,3,1} \leftarrow s_{s,1,3,1}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1}), (s_{s,1,3,1} \leftarrow s_{s,1,3,1})\}
\]

Proof continued. Let \(nlp_2\) be the normal logic program obtained from \(nlp_1\) by deleting all rules with \(s_{s,0,0}\) as head (rules obtained by translation of the added atoms), deleting \(s_{s,0}\) in \((s_\epsilon \leftarrow s_{s,0}, s_{s,1})\), deleting all rules containing \(u_i(b_1, \ldots, b_{N^V})\) if \(b_i = 1\) \((u_i(b_1, \ldots, b_{N^V})\) always in the model), deleting all rules containing \(nu_i(b_1, \ldots, b_{N^V})\) if \(b_i = 0\) \((nu_i(b_1, \ldots, b_{N^V})\) always in the model), deleting the atoms \(nu_i(b_1, \ldots, b_{N^V})\) if \(b_i = 1\) \((nu_i(b_1, \ldots, b_{N^V})\) never in the model), deleting the atoms \(u_i(b_1, \ldots, b_{N^V})\) if \(b_i = 0\) \((u_i(b_1, \ldots, b_{N^V})\) never in the model), deleting the facts \((nu_i(b_1, \ldots, b_{N^V})\) if \(b_i = 0\) (since there is no more instance of the atoms \(nu_i(b_1, \ldots, b_{N^V})\)). 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.
\[ nl_{p_2} = \{(s_c \leftarrow s_{1.1})\} \cup \]
\[ \{(s_c \leftarrow s_{1.0}, s_{1.1}), (s_{1.0} \leftarrow s_{1.0.0}, s_{1.1}), (s_{1.1} \leftarrow s_{1.1.0}, s_{1.1.1})\} \cup \]
\[ \{(c(1.1)), (c(0.1)), (a(1.1)), (a(0.1)), (d(1.1) \leftarrow not nd(1.1)), \]
\[ (nd(1.1) \leftarrow not d(1.1)), (d(1.0) \leftarrow not nd(1.0)), (nd(1.0) \leftarrow not d(1.0)), \]
\[ (d(0.1) \leftarrow not nd(0.1)), (nd(0.1) \leftarrow not d(0.1)), (d(0.0) \leftarrow not nd(0.0)), \]
\[ (nd(0.0) \leftarrow not d(0.0)), (b(1) \leftarrow not nb(1)), (nb(1) \leftarrow not b(1)), \]
\[ (b(0) \leftarrow not nb(0)), (nb(0) \leftarrow not b(0))\} \cup \sigma_{11} \cup \sigma_{01} \cup \sigma_{10} \cup \sigma_{00} \]

with \[ \sigma_{11} = \{(s_{1.1} \leftarrow s_{e_{1.1.0}, s_{e_{1.1.1}}}), (s_{e_{1.1.0}} \leftarrow s_{e_{1.1.0.0}, s_{e_{1.1.1}}}), (s_{e_{1.1.1}} \leftarrow s_{e_{1.1.1.0}, s_{e_{1.1.1.1}}})\} \]

Proof continued. Now we rename the predicate symbols \( s_{e_{1.0}} \) in order to have the same predicate symbols as in Definition 2. Let \( nl_{p_3} \) be the normal logic program obtained from \( nl_{p_2} \) by deleting the rule \( (s_c \leftarrow s_{1.1}) \), replacing the atom \( s_{1.1} \) by the atom \( s_c \), replacing all the atoms \( s_{e_{1.0.0}} \) in \( \sigma_o \) with \( o = b_1 \ldots b_{N^y} \) by the atoms \( s_{e_{0.0}}(b_1, \ldots, b_{N^y}) \), replacing all the atoms \( s_{e_{1.0}} \) in \( \sigma_o \) with \( o = b_1 \ldots b_n, n \leq N^y \) by the atoms \( s_{e_{0.0}}(b_1, \ldots, b_n) \).

Example continued.

\[ nl_{p_3} = \{(s_c \leftarrow s_{0.0}, s_{1.0}), (s_{0.0} \leftarrow s_{0.0}, s_{0.0}), (s_{1.0} \leftarrow s_{1.0}, s_{1.0}), (s_{1.1} \leftarrow s_{1.0}, s_{1.1})\} \cup \]
\[ \{(c(1.1)), (c(0.1)), (a(1.1)), (a(0.1)), (d(1.1) \leftarrow not nd(1.1)), \]
\[ (nd(1.1) \leftarrow not d(1.1)), (d(1.0) \leftarrow not nd(1.0)), (nd(1.0) \leftarrow not d(1.0)), \]
\[ (d(0.1) \leftarrow not nd(0.1)), (nd(0.1) \leftarrow not d(0.1)), (d(0.0) \leftarrow not nd(0.0)), \]
\[ (nd(0.0) \leftarrow not d(0.0)), (b(1) \leftarrow not nb(1)), (nb(1) \leftarrow not b(1)), \]
\[ (b(0) \leftarrow not nb(0)), (nb(0) \leftarrow not b(0))\} \cup \sigma_{11} \cup \sigma_{01} \cup \sigma_{10} \cup \sigma_{00} \]

with \[ \sigma_{11} = \{(s_{1.1} \leftarrow s_{0.0}, s_{1.1}), (s_{0.0} \leftarrow s_{0.0}, s_{0.0}), (s_{1.1} \leftarrow s_{0.0}, s_{1.1})\} \]

The normal logic program \( nl_{p_2} \) has clearly the same stable models (w.r.t. the atoms containing Skolem function symbols) as the normal logic program \( nl_{p_3} \). Since the normal logic program \( nl_{p_3} \) is the ground instance of \( \Pi_{\gamma}(F) \), by the semantics of first-order program in normal logic program, the normal logic program \( \Pi_{\gamma}(F) \) has the same stable models (w.r.t. the atoms containing Skolem function symbols) as the normal logic program \( nl_{p_3} \).

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

1) If \( m \) is a stable model of \( \Pi_{\gamma}(F) \) such that \( s_c \in m \) then there exists a stable model \( m' \) of \( \Pi(univ(sk(fol(F)))) \) such that \( s_c \in m' \).

2) If there exists a stable model \( m \) of \( \Pi_{\gamma}(F) \) such that \( \pi_{\gamma}(sk) \subseteq m \) and
$s_c \in m$ then there exists a stable model $m'$ of $\Pi(\text{univ}(sk(fol(F))))$ such that $\pi_v(sk) \subseteq m'$ and $s_c \in m'$.

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.