Des formules booléennes (quantifiées) à la programmation par ensembles réponses

From (Quantified) Boolean Formulae to Answer Set Programming

B. Da Mota I. Stéphan P. Nicolas

LERIA, Université d'Angers

2, boulevard Lavoisier, 49045, Angers Cedex 01
{damota,stephan,pn}@info.univ-angers.fr

Résumé

Nous proposons dans cet article une traduction des formules booléennes quantifiées vers le paradigme de la programmation par ensembles réponses. Le calcul d'une solution à une formule booléenne quantifiée devient alors équivalent au calcul d'un modèle stable pour un programme logique normal. Le cas des formules propositionnelles est aussi considéré car étant équivalent au cas des formules booléennes quantifiées uniquement existentiellement.

Mots Clef

Formules booléennes quantifiées, programmation par ensembles réponses (ASP), satisfiabilité.

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 propositional formulae is also considered since it is equivalent to the case of Quantified Boolean Formulae with only existential quantifiers.

Keywords

Quantified Boolean Formula (QBF), Answer Set Programming (ASP), Satisfiability (SAT).

1 Introduction

Le problème de satisfiabilité d'une formule propositionnelle (SAT) est un problème combinatoire qui a retenu l'attention comme problème canonique de la classe de complexité NP-complet. De nombreuses procédures de décision ont été proposées, principalement pour la restriction en forme normale conjonctive (FNC) [20, 24, 10]. De récents résultats montrent que la mise en FNC peut altèrer les informations structurelles de la formule initiale qui auraient pu être exploitées dans la procédure de décision pour réduire l'espace de recherche [35]. Il existe peu d'implantations de prouveurs pour le problème SAT pour des formules non FNC [35, 17, 22, 12]. De même, le problème de validité des formules booléennes quantifiées (QBF) [32] est aussi un problème combinatoire mais de complexité PSPACE-complet. Le problème SAT est équivalent au cas particulier du problème QBF où uniquement des quantificateurs existentiels sont considérés. La plus grande part des procédures de décision pour le problème QBF traitent aussi uniquement la restriction aux formules en FNC car elles sont des extensions des procédures pour SAT [29, 15, 7, 19, 5, 27, 30, 4]. Pour les QBF, l'impact de la traduction en FNC apparaît comme étant encore plus important [2]. Là encore, il existe peu d'implantations de prouveurs pour le problème QBF pour des formules non FNC [3, 2, 38, 11, 37, 34].

Dans les deux cas (SAT et QBF), la représentation d'un problème combinatoire, ou plus généralement, la représentation des connaissances en intelligence artificielle, est souvent très naturelle, directe et compacte lorsque l'on s'autorise à utilier toutes les formules bien formées possibles de la logique propositionnelle. C'est pourquoi, cet article propose de s'intéresser à la mise au point de prouveurs pour formules booléennes (quantifiées ou non) non FNC.

La programmation par ensembles réponses (ASP pour Answer Set Programming) est un formalisme de programmation logique non monotone approprié pour représenter et résoudre différents problèmes combinatoires. De nombreuses procédures de décision ont été proposés pour les ASP et ont donné lieu au développement de différents logiciels qui ont prouvé leur efficacité et leur robustesse sur des problèmes de taille importante.

Une traduction polynomiale de SAT vers ASP a déjà été proposée pour des formules en FNC [25] et pour un problème particulier [16]. avec un objectif le plus général possible, nous nous proposons dans cet article d'explorer la

mise au point de prouveurs SAT et QBF par une traduction des formules logiques quelconques en des programmes logiques non monotones et par l'utilisation de prouveurs ASP.

L'objectif essentiel de ce travail est de montrer que le paradigme de la programmation par ensembles réponses (ASP, voir sous-section 2.2) est suffisamment expressif pour représenter des formules booléennes quantifiées (QBF, voir sous-section 2.1). Pour atteindre cet objectif nous démontrons d'abord dans la section 3 que la représentation du problème de satisfaction d'une formule propositionnelle quelconque est possible en ASP. Puis dans la section 4 nous étendrons ce principe à toute formule booléenne quantifiée. En complément, nous mettrons en avant dans la section 5 quelques résultats expérimentaux illustrant que les meilleurs prouveurs ASP actuels peuvent être utilisés pour résoudre des problèmes de calcul de modèles pour des formules booléennes (quantifiées) sans être contraint par la restriction à la FNC des formules, comme c'est le cas pour la quasi totalité des prouveurs SAT et QBF disponibles.

2 Préliminaires

2.1 Formules propositionnelles et booléennes quantifiées

L'ensemble des valeurs booléennes $\mathbf{v}$ et $\mathbf{f}$ est noté $BOOL$ et l'ensemble des fonctions booléennes est noté $\mathcal{F}$. L'ensemble des variables (propositionnelles ou booléennes) est noté $\mathcal{V}$. Les symboles $\top$ et $\bot$ sont les constantes booléennes. Le symbole $\wedge$ est utilisé pour la conjonction, $\vee$ pour la disjonction, $\neg$ pour la négation, $\rightarrow$ pour l'implication, $\leftrightarrow$ pour l'équivalence et $\oplus$ pour le ou exclusif. L'ensemble des opérateurs $\{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$ est noté $\mathcal{O}$. Un littéral est une variable ou la négation de celle-ci. L'ensemble des littéraux est noté $\mathcal{L}$. L'ensemble $PROP$ des formules propositionnelles est défini inductivement ainsi : tout symbole (constante ou variable) propositionnel est élément de $PROP$; si F est élément de $PROP$ alors $\neg F$ est élément de $PROP$; si F et G sont éléments de $PROP$ et $\circ$ est élément de $\mathcal{O}$ alors $(F \circ G)$ est élément de $PROP$. Le symbole $\exists$ est utilisé pour la quantification existentielle et $\forall$ pour la quantification universelle (q est utilisé pour noter un quantificateur quelconque). L'ensemble QBF des formules booléennes quantifiées est défini inductivement ainsi : si F est un élément de $PROP$ alors c'est un élément de QBF ; si F est un élément de QBF et x est une variable alors $(\exists x \ F)$ et $(\forall x \ F)$ sont des éléments de QBF. Par convention, des quantificateurs différents lient des variables différentes. L'ensemble des variables d'une formule F est noté $\mathcal{V}(F)$. Une substitution est une fonction de l'ensemble des variables dans l'ensemble des formules (quantifiées ou non). Nous définissons la substitution de x par F dans G, notée $G[x \leftarrow F]$, comme étant la formule obtenue de G en remplaçant toutes les occurrences de la variable x par la formule F. Un lieur est une chaîne de caractère $q_1 x_1 \dots q_n x_n$ avec $x_1, \dots, x_n$ des variables distinctes et $q_1 \dots q_n$ des quantificateurs. La fonction q de l'ensemble des variables d'un lieur Q vers $\{\exists, \forall\}$ associe à une variable son quantificateur dans le lieur. Une QBF QF est en forme prénexe si F est une formule booléenne, appelée matrice, et sous forme normale conjonctive (FNC) si F est une formule booléenne en forme normale conjonctive (i.e. une conjonction de disjonctions de littéraux). Soient Σ et σ deux formules telles que σ soit une sous formule de Σ, la fonction $o : PROP \times PROP \rightarrow \{0,1\}^*$ est telle que $o(\sigma, \Sigma)$ calcule l'occurrence de σ dans Σ ainsi : $o(\Sigma, \Sigma) = \epsilon$; $o(\sigma, \Sigma) = 0.o(\sigma, \Sigma_0)$ si $\Sigma = \neg\Sigma_0$; $o(\sigma, \Sigma) = 0.o(\sigma, \Sigma_0)$ si $\Sigma = (\Sigma_0 \circ \Sigma_1), \circ \in \mathcal{O}$ et σ est une sous formule de Σ_0 ; $o(\sigma, \Sigma) = 1.o(\sigma, \Sigma_1)$ si $\Sigma = (\Sigma_0 \circ \Sigma_1), \circ \in \mathcal{O}$ et σ est une sous formule de Σ_1[1].

La sémantique des symboles booléens est définie de manière habituelle. Une valuation est une fonction de l'ensemble des variables dans $BOOL$; elle satisfait une formule si appliquée à celle-ci est vaut $\mathbf{v}$ sinon elle la falsifie. La satisfaction propositionnelle est notée $\models$ et l'équivalence logique $\equiv$. Un modèle (i.e. une valuation qui satisfait la formule) est décrit par un ensemble de littéraux ; par exemple, pour la valuation ν qui est telle que $\nu(x) = \mathbf{v}$, $\nu(y) = \mathbf{f}$ et $\nu(z) = \mathbf{v}$ qui satisfait la formule $((x \vee y) \leftrightarrow z)$, ce modèle est noté $\{x, \neg y, z\}$. La sémantique des quantificateurs est la suivante : pour toute variable y et QBF F,

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

et

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

Une QBF F est valide si $F \equiv \top$. Si y est une variable quantifiée existentiellement précédée par les variables quantifiées universellement $x_1, \dots, x_n$, nous notons $\hat{y}_{x_1,\dots,x_n}$ sa fonction de Skolem de $BOOL^n$ dans $BOOL$. Un modèle pour une QBF F est un ensemble*s* de fonctions de Skolem qui satisfait la formule. Par exemple, la QBF $\exists y \exists x \forall z((x \vee y) \leftrightarrow z)$ n'est pas valide tandis que $\forall z \exists y \exists x((x \vee y) \leftrightarrow z)$ l'est avec pour ensemble possible de fonctions de Skolem $\{\hat{y}_z, \hat{x}_z\}$[2] tel que $\hat{y}_z(\mathbf{v}) = \mathbf{v}$, $\hat{y}_z(\mathbf{f}) = \mathbf{f}$, $\hat{x}_z(\mathbf{v}) = \mathbf{f}$ et $\hat{x}_z(\mathbf{f}) = \mathbf{f}$. Les fonctions de Skolem sont parfois représentées par des politiques [8, 9] ou des stratégies [6] qui les explicitent en terme d'arbre ; par exemple le terme $\{z \mapsto y; \neg x, \neg z \mapsto \neg y; \neg x\}$ est la politique valide ou stratégie gagnante correspondant aux fonctions de Skolem $\hat{x}_z$ et $\hat{y}_z$. Un modèle (booléen) pour une formule booléenne non quantifiée correspond exactement au modèle (QBF) de sa clôture existentielle ; par exemple pour la QBF $\exists y \exists x \exists z((x \vee y) \leftrightarrow z)$, les fonctions de Skolem $\hat{x} = \mathbf{v}$, $\hat{y} = \mathbf{f}$ et $\hat{z} = \mathbf{v}$ correspondent au modèle booléen $\nu(x) = \mathbf{v}$, $\nu(y) = \mathbf{f}$ et $\nu(z) = \mathbf{v}$ pour la formule propositionnelle $((x \vee y) \leftrightarrow z)$. Une QBF est valide si et seulement s'il existe un ensemble de fonctions de Skolem qui la satisfait.

Enfin, rappelons que le problème (SAT) consistant à décider si une formule booléenne est satisfiable ou non est le problème canonique de la classe NP-complet. De son

[1] Comme de coutume, $\epsilon.o$ sera simplement noté o.

[2] Pour tout $b \in BOOL$, la valuation $\nu(z) = b$, $\nu(x) = \hat{x}_z(b)$, $\nu(y) = \hat{y}_z(b)$ est un modèle de $((x \vee y) \leftrightarrow z)$.

côté, le problème (QBF) consistant à décider si une formule booléenne quantifiée est valide ou non est le problème PSPACE-complet canonique.

2.2 Programmation par ensembles réponses

Le terme (peu élégant en français) de programmation par ensembles (de) réponses correspond au paradigme de l'Answer Set Programming (ASP) qui, depuis quelques années, est devenu un thème de recherches très actif dans le domaine de la représentation des connaissances, du raisonnement non monotone, de la programmation logique et de la résolution de problèmes combinatoires. L'ASP permet de manière totalement déclarative de représenter un problème à l'aide d'un programme logique dont la sémantique définit un ensemble de réponses (les modèles du programme) représentant les solutions du problème initial. Derrière le terme générique d'ASP, il existe plusieurs variantes syntaxiques et sémantiques, mais dans ce travail nous utiliserons la sémantique originelle des modèles stables [14] pour les programmes logiques normaux.
Un programme logique normal[3] est un ensemble fini de règles de la forme

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

$n \geq 0, m \geq 0$ où $c, a_1, \ldots, a_n, b_1, \ldots b_m$ sont des atomes propositionnels qui constituent l'ensemble $\mathcal{A}$ et $\mathcal{P}$ désigne l'ensemble des programmes logiques normaux. Une telle règle peut se lire « si tous les a_i appartiennent à un ensemble réponse et si aucun des b_j n'y appartient, alors c doit appartenir à cet ensemble réponse ». Le « not » étant une négation par défaut, un tel programme peut-être vu comme un sous cas d'une théorie de défauts de Reiter [28]. Pour une règle r, on note $tête(r) = c$ sa tête, $corps^+(r) = \{a_1, \ldots,\ a_n\}$ son corps positif et $corps^-(r) = \{b_1, \ldots,\ b_m\}$ son corps négatif. Le réduit (dit de Gelfond et Lifschitz) d'un programme P par un ensemble d'atomes X est le programme $P^X = \{(tête(r) \leftarrow corps^+(r).) \mid corps^-(r) \cap X = \emptyset\}$. Un tel programme est dit défini puisqu'il ne possède pas de négation par défaut et il admet un unique modèle de Herbrand minimal noté $Cn(P)$. Par définition, un modèle stable de P est un ensemble d'atomes $S \subseteq \mathcal{A}$ tel que $S = Cn(P^S)$. Notons qu'un programme peut avoir aucun, un ou plusieurs modèles stables. Par exemple, $\{(a.), (b \leftarrow a, not\ d.), (c \leftarrow a, not\ b.)\}$ possède l'unique modèle stable $\{a, b\}$, $\{(a \leftarrow not\ b)., (b \leftarrow not\ a.)\}$ possède deux modèles stables $\{a\}$ et $\{b\}$ et $\{(a.), (b \leftarrow a, not\ d.), (d \leftarrow b.)\}$ n'en possède aucun et il est dit inconsistant. Il est également possible d'écrire des programmes logique normaux du premier ordre en utilisant des variables et des symboles de prédicat comme, par exemple, dans le programme $\{(p(0).), (p(1).), (r(0).), (q(X) \leftarrow p(X), not\ r(X).)\}$; dans ce cas, le programme au premier ordre doit être considéré comme le condensé du programme propositionnel obtenu par instanciation de chaque variable par toute constante du domaine ; pour notre exemple, nous obtenons le programme $\{(p(0).), (p(1).), (r(0).), (q(0) \leftarrow p(0), not\ r(0).), (q(1) \leftarrow p(1), not\ r(1).)\}$.
Déterminer si un programme possède ou non un modèle stable est un problème NP-complet et donc le lien avec le problème NP-complet canonique, à savoir SAT, doit être aisément trouvé. C'est ce qui a été introduit dans [25] dans le cas où l'on s'intéresse à une formule propositionnelle donnée sous FNC. Nous rappelons ici l'approche proposée. Soit Σ un ensemble de clauses. Le processus consiste à fabriquer un programme $LP(\Sigma)$ contenant les règles $(na \leftarrow not\ a.)$ et $(a \leftarrow not\ na.)$ pour tout atome a apparaissant dans Σ. Pour toute clause dans Σ, on crée un nouvel atome c et pour tout littéral l de cette clause, on ajoute dans $LP(\Sigma)$ les règles $\{(c \leftarrow a.)\}$ si l est un atome a et les règles $\{(c \leftarrow na.)\}$ si l est la négation d'un atome a. Enfin, on ajoute à $LP(\Sigma)$ la règle $(\leftarrow not\ c.)$[4]. De cette manière Σ possède un modèle propositionnel si et seulement si $LP(\Sigma)$ possède un modèle stable. On remarque aisément que les premières paires de règles de $LP(\Sigma)$ permettent de générer toutes les interprétations possibles pour Σ, les règles dont la tête est c permettent d'inférer c si l'interprétation satisfait la clause correspondante et les contraintes finales écartent des ensembles de réponses possibles tous ceux qui ne contiennent pas c. C'est-à-dire tous ceux qui ne correspondent pas à des modèles propositionnels de Σ. La première partie de notre contribution consiste à étendre cette approche à toute formule propositionnelle sans restriction syntaxique (de type FNC) et est décrite dans la section suivante.

3 D'une formule propositionnelle à un programme logique normal

Nous nous proposons de traduire (polynomialement) toute formule propositionnelle en un programme logique normal et de démontrer qu'alors une formule propositionnelle est satisfiable si et seulement si le programme obtenu après traduction admet un modèle stable. Le résultat obtenu n'est pas seulement un résultat d'existence mais il met en correspondance les modèles de la formule propositionnelle avec les modèles stables du programme obtenu. Il ouvre donc la voie à l'utilisation des prouveurs ASP comme outil de résolution du problème SAT sans restriction (de type FNC) sur les formules. La définition ci-dessous décrit les fonctions π et π^{-1} qui permettent d'associer un ensemble de littéraux d'une formule à un ensemble d'atomes d'un programme et réciproquement.

Définition 3.1 (Fonctions π et π^{-1}) *La fonction $\pi : 2^{\mathcal{L}} \rightarrow 2^{\mathcal{A}}$ est telle que, pour tout $L \in 2^{\mathcal{L}}$,*

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

La fonction $\pi^{-1} : 2^{\mathcal{A}} \times PROP \rightarrow 2^{\mathcal{L}}$ est telle que, pour tout $A \in 2^{\mathcal{A}}$ et tout $\Sigma \in PROP$,

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

[3] Abrégé en programme par la suite s'il n'y a pas d'ambiguïté.

[4] Une telle règle sans tête est appelée contrainte et est un raccourci pour la règle $(bug \leftarrow not\ c, not\ bug.)$ où bug est un nouveau symbole.

La définition suivante décrit la fonction P_Q qui construit à partir d'un ensemble de variables un programme dont les modèles stables sont en bijection avec toutes les valuations possibles pour ces variables.

Définition 3.2 (Fonction P_Q) *Soit V un ensemble de variables, la fonction $P_Q : 2^{\mathcal{V}} \rightarrow \mathcal{P}$ est telle que*
$P_Q(V) = \bigcup_{x \in V} \{ \ (x \leftarrow not\ nx.), \quad (nx \leftarrow not\ x.) \ \}$

Théorème 3.1 *Soient une formule Σ et un ensemble de littéraux ν. ν est une valuation de Σ si et seulement si $P_Q(\mathcal{V}(\Sigma))$ admet un modèle stable $\pi(\nu)$.*

Exemple 3.1 *Soit la formule propositionnelle*
$F = ((c \vee b) \wedge (b \rightarrow ((c \rightarrow d) \wedge (c \vee (a \leftrightarrow \neg d)))))$
$\mathcal{V}(F) = \{a, b, c, d\}$ *et*
$$P_Q(\mathcal{V}(F)) = \left\{ \begin{array}{ll} (a \leftarrow not\ na.), & (na \leftarrow not\ a.), \\ (b \leftarrow not\ nb.), & (nb \leftarrow not\ b.), \\ (c \leftarrow not\ nc.), & (nc \leftarrow not\ c.), \\ (d \leftarrow not\ nd.), & (nd \leftarrow not\ d.) \end{array} \right\}$$
Par exemple, l'ensemble d'atomes $m = \{na, b, nc, d\}$ est un modèle stable du programme $P_Q(\mathcal{V}(F))$ et $\pi^{-1}(m, F) = \{\neg a, b, \neg c, d\}$ est une valuation de $\mathcal{V}(F)$.

Les modèles d'une formule propositionnelle sont des valuations qui satisfont les contraintes liées aux opérateurs qui constituent la formule. Pour obtenir un programme qui soit en correspondance avec cette formule, nous devons donc adjoindre à la fonction P_Q une seconde fonction qui va traduire ces contraintes (ou sous-formules) en éliminant les ensembles d'atomes qui ne correspondent pas aux modèles. La méthode choisie ici est d'introduire pour chaque instance d'opérateur de la formule un nouvel atome représentant le résultat de l'opérateur sur ces opérandes.

Définition 3.3 (Fonction de traduction P) *Soient Σ, Σ_0 et Σ_1 trois formules propositionnelles, x une variable propositionnelle et o une occurrence, la fonction $P : PROP \times \{0,1\}^* \rightarrow \mathcal{P}$ est définie inductivement par :*

si $\Sigma = x$ *alors* $P(\Sigma, o) = \{(s_o \leftarrow x.)\}$

si $\Sigma = \neg\Sigma_0$ *alors* $P(\Sigma, o) = \{(s_o \leftarrow not\ s_{o.0}.)\} \cup P(\Sigma_0, o.0)$

si $\Sigma = (\Sigma_0 \wedge \Sigma_1)$ *alors* $P(\Sigma, o) = \{ \ (s_o \leftarrow s_{o.0}, s_{o.1}.) \ \} \cup P(\Sigma_0, o.0) \cup P(\Sigma_1, o.1)$

si $\Sigma = (\Sigma_0 \vee \Sigma_1)$ *alors* $P(\Sigma, o) = \left\{ \begin{array}{l} (s_o \leftarrow s_{o.0}.), \\ (s_o \leftarrow s_{o.1}.) \end{array} \right\} \cup P(\Sigma_0, o.0) \cup P(\Sigma_1, o.1)$

si $\Sigma = (\Sigma_0 \rightarrow \Sigma_1)$ *alors* $P(\Sigma, o) = \left\{ \begin{array}{l} (s_o \leftarrow not\ s_{o.0}.), \\ (s_o \leftarrow s_{o.1}.) \end{array} \right\} \cup P(\Sigma_0, o.0) \cup P(\Sigma_1, o.1)$

si $\Sigma = (\Sigma_0 \leftrightarrow \Sigma_1)$ *alors* $P(\Sigma, o) = \left\{ \begin{array}{l} (s_o \leftarrow s_{o.0}, s_{o.1}.), \\ (s_o \leftarrow not\ s_{o.0}, not\ s_{o.1}.) \end{array} \right\} \cup P(\Sigma_0, o.0) \cup P(\Sigma_1, o.1)$

si $\Sigma = (\Sigma_0 \oplus \Sigma_1)$ *alors* $P(\Sigma, o) = \left\{ \begin{array}{l} (s_o \leftarrow s_{o.0}, not\ s_{o.1}.), \\ (s_o \leftarrow not\ s_{o.0}, s_{o.1}.) \end{array} \right\} \cup P(\Sigma_0, o.0) \cup P(\Sigma_1, o.1)$

La traduction d'une formule initiale Σ est $P(\Sigma, \epsilon)$.

Exemple 3.2 (suite de l'exemple 3.1) *Pour la formule*
$F = ((c \vee b) \wedge (b \rightarrow ((c \rightarrow d) \wedge (c \vee (a \leftrightarrow \neg d)))))$

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

Nous sommes maintenant en mesure de définir la fonction Π qui construit un programme logique normal à partir d'une formule propositionnelle.

Définition 3.4 (Fonction Π) *Soit Σ une formule propositionnelle. La fonction $\Pi : PROP \rightarrow \mathcal{P}$ est telle que $\Pi(\Sigma) = P_Q(\mathcal{V}(\Sigma)) \cup P(\Sigma, \epsilon)$.*

Le lemme suivant exprime que pour une formule Σ, le programme $\Pi(\Sigma)$ n'élimine ni ne rajoute de modèles stables au programme $P_Q(\Sigma)$, il ne fait que les augmenter des atomes intermédiaires s_o.

Lemme 3.1 *Soit Σ une formule propositionnelle. L'ensemble de littéraux ν est une valuation pour $\mathcal{V}(\Sigma)$ si et seulement s'il existe un (unique) modèle stable m de $\Pi(\Sigma)$ tel que $\pi(\nu) \subseteq m$.*

Le lemme suivant exprime que tout atome intermédiaire s_o introduit pour représenter le résultat de l'opérateur sur ces opérandes appartient au modèle stable si et seulement si ce résultat est égal à **v**.

Lemme 3.2 *Soit Σ_{init} une formule propositionnelle, Σ une sous formule de Σ_{init} et m un modèle stable de $P_Q(\mathcal{V}(\Sigma_{init})) \cup P(\Sigma, \epsilon)$.*
$s_{o(\Sigma, \Sigma_{init})} \in m$ *si et seulement si* $\pi^{-1}(m, \Sigma) \models \Sigma$.

Le théorème suivant est une instance du théorème précédent avec $\Sigma = \Sigma_{init}$.

Théorème 3.2 *Quels que soient Σ une formule et m un modèle stable de $\Pi(\Sigma)$,*
$s_\epsilon \in m$ *si et seulement si* $\pi^{-1}(m, \Sigma) \models \Sigma$.

Pour n'obtenir que les modèles stables qui contiennent (resp. ne contiennent pas) l'atome s_ϵ, nous étendons la définition de la fonction Π à la fonction Π^+ (Π^-) qui inclut au programme une contrainte spécifique.

Définition 3.5 (Fonctions Π^+ et Π^-) *Soit Σ une formule propositionnelle. Les fonctions Π^+ et Π^- : $PROP \rightarrow \mathcal{P}$ sont telles que*

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

Le corollaire au théorème 3.2 qui suit établit le résultat attendu : l'équivalence entre l'existence d'un modèle stable pour le programme obtenu et la satisfiabilité pour la formule propositionnelle. Ce résultat n'est pas qu'un résultat d'existence : à tout modèle stable du programme correspond un modèle pour la formule et réciproquement.

Corollaire 3.1 *Soit une formule propositionnelle Σ, le programme logique normal $\Pi^+(\Sigma)$ admet un modèle stable si et seulement si Σ est satisfiable. De plus, si m est un modèle stable de $\Pi^+(\Sigma)$ alors $\pi^{-1}(m, \Sigma) \models \Sigma$. Enfin, si ν est un modèle pour Σ alors il existe un (unique) modèle stable m de $\Pi^+(\Sigma)$ tel que $\pi(\nu) \subseteq m$.*

Exemple 3.3 (suite de l'exemple 3.2) *La formule F admet 8 modèles, par exemple, $\nu_1 = \{\neg a, \neg b, c, d\} \models F$. $\Pi(F)$ admet 64 modèles stables dont 8 $\{m_i\}_{1\leq i\leq 8}$ contiennent s_ϵ ; ces 8 modèles stables sont tels que $\pi^{-1}(m_i, F) \models F$ pour tout i, $1 \leq i \leq 8$; par exemple le modèle stable*
$m_1 = \{na, nb, c, d\} \cup$
$\{s_\epsilon, s_0, s_1, s_{0^2}, s_{1^2}, s_{1^2.0}, s_{1^2.0^2}, s_{1^2.0.1}, s_{1^3}, s_{1^3.0}, s_{1^4}\}$
est tel que $\pi^{-1}(m_1, F) = \{\neg a, \neg b, c, d\} = \nu_1$.

La traduction Π^+ permet de décider non seulement du caractère satisfiable ou non d'une formule propositionnelle mais aussi par le corollaire 3.1 de son caractère tautologique, mais indirectement par l'exhaustivité (ou le nombre) des modèles. La traduction Π^- permet de décider directement du caractère tautologique ou non d'une formule comme l'exprime le corollaire au théorème 3.2 suivant.

Corollaire 3.2 *Soit une formule propositionnelle Σ, le programme logique normal $\Pi^-(\Sigma)$ n'admet aucun modèle stable si et seulement si Σ est une tautologie. De plus, si m est un modèle stable de $\Pi^-(\Sigma)$ alors $\pi^{-1}(m, \Sigma)$ falsifie la formule Σ. Enfin, si ν falsifie la formule Σ alors il existe un (unique) modèle stable m de $\Pi^-(\Sigma)$ tel que $\pi(\nu) \subseteq m$.*

Il est évident que notre traduction Π est polynomiale et modulaire [25] par construction. Cette traduction peut être améliorée pour diminuer le nombres de symboles introduits ainsi que le nombre de règles générées. Nous pouvons constater que chaque occurrence d'une variable introduit un nouveau symbole (et une nouvelle règle) dans les cas de base de l'induction : la première amélioration consiste à modifier ces deux cas de base en les intégrant directement dans les cas d'induction. Nous pouvons aussi augmenter le nombre d'opérateurs en considérant le « non et », le « non ou » et la « non implication » (la « non équivalence » étant le ou exclusif) : cette seconde modification combinée à la précédente fait disparaître la négation comme connecteur traduit, le résultat est alors très proche de la méthode des tableaux sémantiques [31]. Enfin, même si elle s'apparente à l'usuelle transformation de Tseitin [36] de mise sous FNC d'une formule logique quelconque, notre traduction en diffère par le fait que les variables introduites ne sont pas des variables propositionnelles au sens de SAT mais des parties gauches d'une affectation booléenne.

Exemple 3.4 (suite de l'exemple 3.2) *Le programme $P(F, \epsilon)$ peut être optimisé en*

$$\left\{\begin{array}{l}(s_\epsilon \leftarrow s_{\epsilon.0}, s_{\epsilon.1}.), (s_0 \leftarrow c.), (s_0 \leftarrow b.),\\ (s_1 \leftarrow not\ b.),\\ (s_1 \leftarrow s_{1^2}.), (s_{1^2} \leftarrow s_{1^2.0}, s_{1^3}.),\\ (s_{1^2.0} \leftarrow not\ c.), (s_{1^2.0} \leftarrow d.),\\ (s_{1^3} \leftarrow c.), (s_{1^3} \leftarrow s_{1^4}.),\\ (s_{1^4} \leftarrow a, nd.),\\ (s_{1^4} \leftarrow not\ a, not\ nd.)\end{array}\right\}$$

4 D'une formule booléenne quantifiée à un programme logique normal

Nous avons décrit dans la section précédente la tradution d'une formule propositionnelle en un programme logique normal telle que calculer les modèles de la formule revient à calculer les modèles stables du programme. Nous étendons ces résultats aux QBF grâce à une traduction d'une QBF en un programme logique normal telle que calculer les fonctions de Skolem qui satisfont la QBF revient à calculer les modèles stables d'un programme logique normal. La manière la plus simple d'étendre les résultats de la section précédente aux QBF est d'appliquer la sémantique du quantificateur universel qui explicite la quantification comme étant une conjonction :

$$(\forall x\ \Sigma) = (\Sigma[x \leftarrow \top] \wedge \Sigma[x \leftarrow \bot])$$

puis d'appliquer sur cette formule ne contenant plus que des quantificateurs existentiels le résultat obtenu au propositionnel (cette technique est employée dans [5]). Le prix évident de cette traduction est une croissance exponentielle de la formule propositionnelle obtenue et donc du programme logique normal. L'autre prix est la croissance exponentielle des symboles intermédiaires introduits lors de la traduction des opérateurs. Ces deux croissances exponentielles peuvent être évitées en calculant un programme logique normal au premier ordre comme nous le proposons. Malheureusement, la croissance exponentielle du programme traité par les prouveurs ASP ne peut pas être évitée puisque, in fine, les prouveurs ASP actuels sont propositionnels.

La technique employée ici s'inspire de la skolémisation et peut être rapprochée de la skolémisation propositionnelle symbolique de [4] : les variables quantifiées existentiellement sont remplacées par des fonctions dont les arguments sont les variables quantifiées universellement qui précèdent la variable quantifiée existentiellement dans le lieur. Ces fonctions sont converties en symboles de prédicat dans le programme logique normal. Deux nouveaux symboles 0 et 1 sont introduits ainsi qu'une fonction d'interprétation i de $\{0,1\}$ dans $BOOL$ telle que $i(0) = \mathbf{f}$ et $i(1) = \mathbf{v}$. Les définitions de π et π^{-1} sont étendues aux fonctions de Skolem et aux prédicats n-aires.

Définition 4.1 (Fonctions $\pi_\forall$ et $\pi_\forall^{-1}$ pour QBF) *La fonction $\pi_\forall : 2^{\mathcal{F}} \to 2^{\mathcal{A}}$ est telle que, pour tout $sk \in 2^{\mathcal{F}}$,*

$$\pi_\forall(sk) = \left\{ \begin{array}{l} x(i^{-1}(u_1),\ldots,i^{-1}(u_n)) \mid \\ \quad \hat{x} \in sk, \hat{x} \textit{ d'arité } n, \\ \quad u_1,\ldots,u_n \in BOOL, \hat{x}(u_1,\ldots,u_n) = \mathbf{v} \\ nx(i^{-1}(u_1),\ldots,i^{-1}(u_n)) \mid \\ \quad \hat{x} \in sk, \hat{x} \textit{ d'arité } n, \\ \quad u_1,\ldots,u_n \in BOOL, \hat{x}(u_1,\ldots,u_n) = \mathbf{f} \end{array} \right\} \cup$$

La fonction $\pi_\forall^{-1} : 2^{\mathcal{A}} \times QBF \to 2^{\mathcal{F}}$ est telle que, pour tout $A \in 2^{\mathcal{A}}$ et tout $\Sigma \in QBF$,

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

Exemple 4.1 (suite de l'exemple 3.3) *Soit un ensemble d'atomes*

$$A = \{b(1), b(0), d(1,1), d(0,1), d(0,0), nd(1,0)\}$$

alors

$$\pi_\forall^{-1}(A, \forall a \exists b \forall c \exists dF) = \left\{ \begin{array}{l} \hat{b}_a(\mathbf{v}) = \mathbf{v}, \hat{b}_a(\mathbf{f}) = \mathbf{v}, \\ \hat{d}_{ac}(\mathbf{v},\mathbf{v}) = \mathbf{v}, \hat{d}_{ac}(\mathbf{f},\mathbf{f}) = \mathbf{v}, \\ \hat{d}_{ac}(\mathbf{f},\mathbf{v}) = \mathbf{v}, \hat{d}_{ac}(\mathbf{v},\mathbf{f}) = \mathbf{f} \end{array} \right\}$$

La définition suivante étend les fonctions P_Q et P aux QBF. Ces fonctions de traduction $P_Q^\forall$ et $P^\forall$ possèdent un argument S (pour « Skolem ») qui associe à chaque variable quantifiée existentiellement le nombre de variables quantifiée universellement qui la précèdent (et donc l'arité de la fonction de Skolem et du symbole de prédicat correspondant). La fonction de traduction $P_Q^\forall$ correspond au traitement des quantificateurs : la quantification existentielle est traitée de manière similaire au cas propositionnel tandis que la quantification universelle introduit une règle explicitant la constante 1 comme étant interprétée à $\mathbf{v}$ et la sémantique du quantificateur comme étant une conjonction (l'explicitation de la constante 0 comme étant interprétée à $\mathbf{f}$ est inutile, ceci illustrant le caractère disymétrique du programme logique normal).

Les symboles intermédiaires introduits dans le cas propositionnel sont considérés pour les QBF comme des existentielles internes nées de la décomposition par introduction de variables existentielles comme dans [36]. Comme variables quantifiées existentiellement, elle subissent aussi une skolémisation : ainsi les symboles intermédiaires ont-ils autant d'arguments qu'il y a de variables quantifiées universellement dans la QBF.

Définition 4.2 (Fonctions $P_Q^\forall$, $P^\forall$ et $\Pi_\forall$) *Soient Σ, Σ_0 et Σ_1 des QBF, V un ensemble de variables, n et $N_\forall$ des entiers, o une occurrence et S une fonction de $\mathcal{V}$ dans $\mathbb{N}$. Les fonctions*

$$P_Q^\forall : QBF \times \mathbb{N} \times \mathbb{N} \times (\mathcal{V} \to \mathbb{N}) \to \mathcal{P}$$

et

$$P^\forall : PROP \times \{0,1\}^* \times \mathbb{N} \times (\mathcal{V} \to \mathbb{N}) \to \mathcal{P}$$

sont telles que

si $\Sigma = (\exists x\ \Sigma_0)$ *alors* $P_Q^\forall(\Sigma, N_\forall, n, S) =$
$\left\{ \begin{array}{l} x(U_1,\ldots,U_n) \leftarrow not\ nx(U_1,\ldots,U_n). \\ nx(U_1,\ldots,U_n) \leftarrow not\ x(U_1,\ldots,U_n). \end{array} \right\} \cup$
$P_Q^\forall(\Sigma_0, N_\forall, n, S \cup \{(x \mapsto n)\})$

si $\Sigma = (\forall x\ \Sigma_0)$ *alors* $P_Q^\forall(\Sigma, N_\forall, n, S) =$
$\{x(U_1,\ldots,U_{N_\forall}) \leftarrow U_{n+1} = 1.\} \cup$
$\left\{ \begin{array}{l} s_{0^n}(U_1,\ldots,U_n) \leftarrow \\ \quad s_{0^{n+1}}(U_1,\ldots,U_n,0), s_{0^{n+1}}(U_1,\ldots,U_n,1). \end{array} \right\} \cup$
$P_Q^\forall(\Sigma_0, N_\forall, n+1, S)$

si Σ *est une formule propositionnelle alors*
$P_Q^\forall(\Sigma, N_\forall, n, S) = P^\forall(\Sigma, 0^{N_\forall}, N_\forall, S)$

Variable quantifiée existentiellement

si $\Sigma = x$ *et* $(x \mapsto n) \in S$ *alors* $P^\forall(\Sigma, o, N_\forall, S) =$
$\{s_o(U_1,\ldots,U_{N_\forall}) \leftarrow x(U_1,\ldots,U_n).\}$

Variable quantifiée universellement

si $\Sigma = x$ *et* $(x \mapsto n) \notin S$ *alors* $P^\forall(\Sigma, o, N_\forall, S) =$
$\{s_o(U_1,\ldots,U_{N_\forall}) \leftarrow x(U_1,\ldots,U_{N_\forall}).\}$

si $\Sigma = \neg\Sigma_0$ *alors* $P^\forall(\Sigma, o, N_\forall, S) =$
$\{\ s_o(U_1,\ldots,U_{N_\forall}) \leftarrow not\ s_{o.0}(U_1,\ldots,U_{N_\forall}).\ \} \cup$
$P^\forall(\Sigma_0, o.0, N_\forall, S)$

si $\Sigma = (\Sigma_0 \wedge \Sigma_1)$ *alors* $P^\forall(\Sigma, o, N_\forall, S) =$
$\left\{ \begin{array}{l} s_o(U_1,\ldots,U_{N_\forall}) \leftarrow \\ \quad s_{o.0}(U_1,\ldots,U_{N_\forall}), s_{o.1}(U_1,\ldots,U_{N_\forall}). \end{array} \right\} \cup$
$P^\forall(\Sigma_0, o.0, N_\forall, S) \cup P^\forall(\Sigma_1, o.1, N_\forall, S)$

si $\Sigma = (\Sigma_0 \vee \Sigma_1)$ *alors* $P^\forall(\Sigma, o, N_\forall, S) =$
$\left\{ \begin{array}{l} s_o(U_1,\ldots,U_{N_\forall}) \leftarrow s_{o.0}(U_1,\ldots,U_{N_\forall}). \\ s_o(U_1,\ldots,U_{N_\forall}) \leftarrow s_{o.1}(U_1,\ldots,U_{N_\forall}). \end{array} \right\} \cup$
$P^\forall(\Sigma_0, o.0, N_\forall, S) \cup P^\forall(\Sigma_1, o.1, N_\forall, S)$

si $\Sigma = (\Sigma_0 \to \Sigma_1)$ *alors* $P^\forall(\Sigma, o, N_\forall, S) =$
$\left\{ \begin{array}{l} s_o(U_1,\ldots,U_{N_\forall}) \leftarrow not\ s_{o.0}(U_1,\ldots,U_{N_\forall}). \\ s_o(U_1,\ldots,U_{N_\forall}) \leftarrow s_{o.1}(U_1,\ldots,U_{N_\forall}). \end{array} \right\} \cup$
$P^\forall(\Sigma_0, o.0, N_\forall, S) \cup P^\forall(\Sigma_1, o.1, N_\forall, S)$

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

si $\Sigma = (\Sigma_0 \oplus \Sigma_1)$ *alors* $P^\forall(\Sigma, o, N_\forall, S) =$

$$\left\{ \begin{array}{l} s_o(U_1,\ldots,U_{N_\forall}) \leftarrow \\ \quad s_{o.0}(U_1,\ldots,U_{N_\forall}), not\ s_{o.1}(U_1,\ldots,U_{N_\forall}). \\ s_o(U_1,\ldots,U_{N_\forall}) \leftarrow \\ \quad not\ s_{o.0}(U_1,\ldots,U_{N_\forall}), s_{o.1}(U_1,\ldots,U_{N_\forall}). \end{array} \right\} \cup$$

$P^\forall(\Sigma_0, o.0, N_\forall, S) \cup P^\forall(\Sigma_1, o.1, N_\forall, S)$

Soit Σ *une QBF prénexe et* $N_\forall$ *le nombre de quantificateurs universels de* Σ*. La fonction* $\Pi_\forall(\Sigma) : QBF \rightarrow \mathcal{P}$ *est telle que*

$$\Pi_\forall(\Sigma) = P_Q^\forall(\Sigma, N_\forall, 0, \emptyset)$$

Par construction la traduction est polynomiale.

Exemple 4.2 (suite de l'exemple 3.3) *Soit la QBF* $F_{\exists a \forall b \exists c \forall d} = \exists a \forall b \exists c \forall d F$ *avec*

$F = ((c \vee b) \wedge (b \rightarrow ((c \rightarrow d) \wedge (c \vee (a \leftrightarrow \neg d)))))$.

Nous obtenons

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

avec $P_{\exists a \forall b \exists c \forall d}$ *égal à* $P(F, 0^2)$*, de l'exemple 3.2, dans lequel tous les* s_o *(resp.* b*,* c *et* d*) sont remplacés par des* $s_o(U_1,U_2)$ *(resp.* $b(U_1,U_2)$*,* $c(U_1)$ *et* $d(U_1,U_2)$*) et pour appel intermédiaire* $P^\forall(F, 2, 2, \{(a \mapsto 0), (c \mapsto 1)\})$ *(*a *est une variable quantifiée existentiellement qui n'est précédée par aucune variable quantifiée universellement d'où le* 0 *tandis que* c *est une variable quantifiée existentiellement qui est précédée d'une variable quantifiée universellement d'où le* 1*).*

Maintenant, soit la QBF $F_{\forall a \exists b \forall c \exists d} = \forall a \exists b \forall c \exists d F$*. Nous obtenons*

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

avec $P_{\forall a \forall b \forall c \exists d}$ *égal à* $P(F, 0^2)$ *dans lequel tous les* s_o *(resp.* a*,* b*,* c *et* d*) sont remplacés par des* $s_o(U_1,U_2)$ *(resp.* $a(U_1,U_2)$*,* $b(U_1)$*,* $c(U_1,U_2)$ *et* $d(U_1,U_2)$*) et pour appel intermédiaire* $P^\forall(F, 2, 2, \{(b \mapsto 1), (d \mapsto 2)\})$ *(*b *est une variable quantifiée existentiellement qui est précédée d'une variable quantifiée universellement d'où le* 1 *et* d *est une variable quantifiée existentiellement qui est précédée de deux variables quantifiées universellement d'où le* 2*).*

Nous obtenons un théorème similaire pour les QBF au théorème 3.2 pour les formules propositionnelles.

Théorème 4.1 *Soit* Σ *une QBF et* m *un modèle stable de* $\Pi_\forall(\Sigma)$*.*

$s_\epsilon \in m$ *si et seulement si* Σ *est valide.*

Exemple 4.3 (suite de l'exemple 4.2) *Soit la QBF* $F_{\exists a \forall b \exists c \forall d} = \exists a \forall b \exists c \forall d F$*. Elle n'admet pas de fonctions de Skolem qui la satisfassent et* $\Pi_\forall(F_{\exists a \forall b \exists c \forall d})$ *n'admet pas de modèle stable.*

A l'instar de la fonction Π étendue en la fonction Π^+ pour se restreindre uniquement aux modèles stables correspondant aux modèles, nous définissons ci-dessous la fonction ${\Pi_\forall}^+$ comme extension de la fonction $\Pi_\forall$ pour se restreindre aux fonctions de Skolem qui satisfont la QBF.

Définition 4.3 (Fonction $\Pi_\forall^+$) *Soit* Σ *une QBF. La fonction* $\Pi_\forall^+ : QBF \rightarrow \mathcal{P}$ *est telle que*

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

Comme pour le corollaire 3.1 du théorème 3.2 qui explicite les modèles d'une formule propositionnelle grâce aux modèles stables du programme logique normal associé, le corollaire suivant au théorème 4.1 permet d'extraire des modèles stables du programme logique normal les fonctions de Skolem qui satisfont la QBF.

Corollaire 4.1 *Soit une QBF* Σ*, le programme logique normal* $\Pi_\forall^+(\Sigma)$ *admet un modèle stable si et seulement si* Σ *est valide. De plus, si* m *est un modèle stable de* $\Pi_\forall^+(\Sigma)$ *alors* ${\pi_\forall}^{-1}(m, \Sigma)$ *constituent un ensemble de fonctions de Skolem qui satisfont la QBF. Enfin, si* sk *est un ensemble de fonctions de Skolem qui satisfont* Σ *alors il existe un (unique) modèle stable* m *de* $\Pi_\forall^+(\Sigma)$ *tel que* $\pi_\forall(sk) \subseteq m$*.*

Exemple 4.4 (suite de l'exemple 4.3) *La QBF* $F_{\forall a \exists b \forall c \exists d} = \forall a \exists b \forall c \exists d F$ *admet pour (unique) ensemble de fonctions de Skolem qui la satisfont :*

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

soit encore la politique valide ou stratégie gagnante

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

et $\Pi_\forall^+(F_{\forall a \exists b \forall c \exists d})$ *admet un (unique) modèle stable*

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

Nous avons donc bien $\pi_{\forall}^{-1}(m, F_{\forall a \exists b \forall c \exists d}) = sk$ *et* $\pi_{\forall}(sk) \subseteq m$.
Les symboles s_0 *et* s_{0^2} *représentant la sémantique de la quantification universelle portant sur les variables* a *et* c*, il est nécessaire que toutes leurs instances appartiennent au modèle stable. Seul l'atome* $s_{0^2.1^4}(1,1)$ *n'appartient pas au modèle stable car il représente la sous formule* $(a \leftrightarrow \neg d)$ *pour la variable* a *interprétée à* **v** *(le premier* 1 *de* $s_{0^2.1^4}(1,1)$*) alors que la variable* d *est interprétée à* **v** *(puisque l'atome* $d(1,1)$ *est dans le modèle stable) et doit donc bien être interprété à* **f**.

5 Résultats expérimentaux

Nous désirons évaluer dans cette partie dans quelle mesure notre approche représente une alternative efficace aux prouveurs dédiés aux problèmes de calcul des modèles d'une formule propositionnelle ou de calcul des fonctions de Skolem satisfaisant une QBF. Nous avons sélectionné pour cela les prouveurs ASP suivants : DLV [18], Clasp [13], noMoRe++ [1] et Smodels [26]. Nous étudions d'abord l'approche sur les formules propositionnelles puis sur les QBF.

5.1 De SAT vers ASP

Nous avons choisi d'effectuer l'évaluation et la comparaison de notre approche dans le cas propositionnel sur la suite de problèmes QG6 [23] qui a l'avantage d'être fournie aussi bien en FNC (de 1301 à 2109 variables et de 6089 à 9964 clauses) qu'en non-FNC (soit 252 soit 324 variables et de 3532 à 7850 disjonctions et conjonctions). Nous comparons les temps d'exécution et les pourcentages de réussite des prouveurs ASP considérés sur le résultat de nos traductions appliquées à ces instances SAT non-FNC et ceux du prouveur SAT non-FNC Satmate [17] (le seul qui accepte le format de la suite de problèmes QG6). Nous incluons aussi dans la comparaison les temps et pourcentages de réussite pour les prouveurs SAT FNC Zchaff [24] et Minisat [10].
Les résultats sont fournis pour des $2 \times Intel\ Xeon\ CPU\ 2.80Ghz$ avec $2Go$ de mémoire. Le tableau ci-dessous indique si les problèmes sont satisfiables (SAT) ou non (UNSAT), le nombre de problèmes résolus (NbR), le temps moyen en secondes (TM) et le temps moyen lorsque le problème est résolu (TMR) en secondes. Le temps maximum est de 3600 secondes CPU.

	SAT			UNSAT		
	83 problèmes			173 problèmes		
	NbR	TM	TMR	NbR	TM	TMR
Zchaff	83	19	19	137	1260	645
Minisat	83	3	3	165	680	538
Satmate	83	4	4	152	501	73
Smodels	67	858	203	88	1912	281
noMoRe++	83	119	120	77	2067	157
DLV	78	416	212	83	1934	127
Clasp	83	12	12	154	712	355

Notre approche secondée par le prouveur ASP Clasp est performante sur cette suite de problèmes puisqu'elle fait mieux que le prouveur SAT non-FNC Satmate et que Zchaff (sur une version FNC du benchmark) ; seul le prouveur Minisat fait mieux en nombre de problèmes résolus ainsi qu'en temps de résolution.

5.2 De QBF vers ASP

Dans le cas des QBF, nous n'effectuons qu'une évaluation de notre approche sur la suite de benchmarks de l'évaluation QBFEVAL07[5]. L'ensemble des instances ont $\exists\forall$ pour alternance de quantificateurs et sont constituées exclusivement de négations, conjonctions et disjonctions. Les résultats sont fournis pour un « Core Duo T2400' » avec $3Go$ de mémoire. Le tableau ci-dessous indique l'instance, son nombre de variables existentiellement/universellement quantifiées (NbV), ainsi que le nombre d'atomes et de règles du programme logique propositionnel en entrée du prouveur ASP.

Instance	NbV	NbAtomes	NbRègles
counter5_2	15/10	4 584	5 746
counter6_2	18/12	15 584	19 876
counter7_2	21/14	56 734	73 382
counter8_2	24/16	215 082	280 960
counter5_4	25/20	3 149 672	4 198 500
counter6_4	30/24	-	-
ring4_2	21/14	52 180	68 590
ring5_2	24/16	203 698	269 314
ring6_2	27/18	803 989	1 066 267
ring4_3	28/21	-	-
ring5_4	40/32	-	-
semaphore_2	21/14	52 862	69 458
semaphore_3	28/21	-	-
semaphore3_2	27/18	798 972	1 061 495
semaphore4_2	36/25	-	-

Tous les prouveurs ASP actuels acceptent en entrée des programmes au premier ordre mais ils s'appuient sur un « front-end » (externe via Lparse[6] ou interne pour DLV) ; ainsi, le remplacement des variables par des termes (ici les constantes 0 et 1) mène à une taille de programme exponentielle. De fait, il est intéressant de remarquer que dans tous les cas où soit le temps limite est dépassé soit la mémoire est insuffisante, c'est Lparse qui dépasse ces limites. Il nous a été possible de comparer notre approche uniquement avec le prouveur expérimental QBF non-CNF $pQBFcompiled.0$ [33, 34] du fait de l'indisponibilité des autres prouveurs QBF non-CNF. Le prouveur $pQBFcompiled.0$ n'intègre aucune optimisation et ne fait que propager les propriétés locales des connecteurs logiques en tenant compte des quantificateurs. Le tableau ci-dessous indique l'instance, le temps en secondes pour le prouveur Clasp (T signifie que le temps limite de 3600 secondes CPU a été dépassé tandis que M signifie que l'espace mémoire alloué de $3Go$ a été dépassé) et pour le prouveur $pQBFcompiled.0$.

[5] `http://www.qbflib.org`
[6] `http://www.tcs.hut.fi/Software/smodels`

Instance	Clasp	$pQBFcompiled.0$
counter5_2	0,152	0,012
counter6_2	0,508	0,016
counter7_2	3,000	0,028
counter8_2	24,434	0,064
counter5_4	592,725	3,012
counter6_4	M	30,614
ring4_2	1,520	0,304
ring5_2	8,585	0,828
ring6_2	64,084	2,328
ring4_3	M	30,542
ring5_4	T	T
semaphore_2	1,604	1,024
semaphore_3	M	701,416
semaphore3_2	58,476	22,457
semaphore4_2	T	1345,836

L'explosion combinatoire de l'instanciation est telle que notre approche ne peut concurrencer les prouveurs QBF. Cette étude n'invalide pas pour autant l'approche, elle ne fait que montrer la nécessité de recherches plus importantes sur les prouveurs ASP au premier ordre.

6 Conclusion

Nous avons généralisé, dans le cadre de ce travail, la traduction de SAT vers ASP en permettant d'utiliser les prouveurs ASP existant pour n'importe qu'elle formule propositionnelle. Les expériences ont montré qu'une telle approche était réaliste pour des problèmes non triviaux. Un travail intéressant et complémentaire serait de comparer avec des prouveurs ASP qui prennent, pour calculer les modèles stables, l'approche inverse à la notre, ie : la traduction de ASP vers SAT (comme par exemple ASSAT [21]).

En ce qui concerne les QBF, l'absence de benchmarks suffisamment variés et de prouveurs disponibles ne permet pas de réaliser une évaluation approfondie de notre proposition. Il apparaît clairement que la complexité en espace est le point central. Nous pensons néanmoins que l'approche nouvelle que nous avons introduite fournit à la communauté un outil utile puisqu'il permet de calculer l'ensemble des solutions.

D'un point de vue pratique, tous les traducteurs présentés ici ont été développés, sont opérationnels et disponibles[7].

Un des défis de la communauté ASP est de fournir des prouveurs capables de gérer des programmes de taille importante ; cela passe en partie, peut-être, par retarder la phase d'instanciation et inférer autant que faire se peut au premier ordre. Si cet objectif était atteint, il serait d'un grand intérêt pour notre méthodologie.

Références

[1] C. Anger, M. Gebser, T. Linke, A. Neumann, and T. Schaub. The nomore++ system. In *LPNMR*, pages 422–426, 2005.

[2] C. Ansotegui, C. Gomes, and B. Selman. Achilles' heel of QBF. In *Nat. Conf. on Artificial Intelligence (AAAI)*, 2005.

[3] A. Ayari and D. Basin. Qubos : Deciding quantified boolean logic using propositional satisfiability solvers. In *Formal Methods in Computer-Aided Design, Fourth International Conference, FMCAD 2002*. Springer-Verlag, 2002.

[4] M. Benedetti. Evaluating QBFs via Symbolic Skolemization. In *Proc. of the 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR04)*, number 3452 in LNCS. Springer, 2005.

[5] A. Biere. Resolve and expand. In *7th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'04)*, 2004.

[6] L. Bordeaux. Boolean and interval propagation for quantified constraints. In *First International Workshop on Quantification in Constraint Programming*, 2005.

[7] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An algorithm to evaluate quantified boolean formulae and its experimental evaluation. *Journal of Automated Reasoning*, 28(2) :101–142, 2002.

[8] S. Coste-Marquis, H. Fargier, J. Lang, D. Le Berre, and P. Marquis. Résolution de formules booléennes quantifiées : problèmes et algorithmes. In *Actes du 13e Congrès AFRIF-AFIA Reconnaissance des Formes et Intelligence Artificielle (RFIA-02)*, Angers, France, 2002.

[9] S. Coste-Marquis, H. Fargier, J. Lang, D. Le Berre, and P. Marquis. Representing policies for quantified boolean formulae. In *KR06*, 2006.

[10] N. Een and N. Sörensson. An extensible sat-solver. In *6th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'03)*, 2003.

[11] Uwe Egly, Martina Seidl, and Stefan Woltran. A solver for QBFs in nonprenex form. In *ECAI*, pages 477–481, 2006.

[12] G. Parthasarathy F. Lu, M. K. Iyer and K.-T. Cheng. An efficient sequential sat solver with improved search strategies. In *Design, Automation and Test in Europe (DATE)*, 2005.

[13] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In *IJCAI*, pages 386–, 2007.

[14] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.

[7] `http://forge.info.univ-angers.fr/~damota/asp/index.php`

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

[16] 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*, Breckenridge, Colorado, USA, April 2000. cs.AI/0003039.

[17] H. Jain, C. Bartzis, and E. Clarke. Satisfiability checking of non-clausal formulas using general matings. In *9th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'06)*, 2006.

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

[19] R. Letz. Lemma and model caching in decision procedures for quantified boolean formulas. In *TABLEAUX*, pages 160–175, 2002.

[20] C.-M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *IJCAI*, pages 366–371, 1997.

[21] F. Lin and Y. Zhao. Assat : computing answer sets of a logic program by sat solvers. *Artif. Intell.*, 157(1-2) :115–137, 2004.

[22] 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.

[23] A. Meier and V. Sorge. Applying sat solving in classification of finite algebras. *J. Autom. Reason.*, 35(1-3) :201–235, 2005.

[24] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.

[25] 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.

[26] I. Niemela and P. Simons. Evaluating an algorithm for default reasoning. In *Workshop on Applications and Implementations of Nonmonotomic Reasonigs Systems*, 1995.

[27] G. Pan and M.Y. Vardi. Symbolic decision procedures for QBF. In *International Conference on Principles and Practice of Constraint Programming*, 2004.

[28] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2) :81–132, 1980.

[29] J. Rintanen. Improvements to the evaluation of quantified boolean formulae. In *IJCAI*, pages 1192–1197, 1999.

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

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

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

[33] I. Stéphan. Boolean propagation based on literals for quantified boolean formulae. In *17th European Conference on Artificial Intelligence*, 2006.

[34] I. Stéphan. Une (presque) génération automatique d'un compilateur de tables de vérité vers un solveur pour formules booléennes quantifiées prénexes. In *Deuxièmes Journées Francophones de Programmation par Contraintes*, 2007.

[35] C. Thiffault, F. Bacchus, and T. Walsh. Solving non-clausal formulas with dpll search. In *7th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'04)*, 2004.

[36] G. S. Tseitin. On the complexity of derivation in propositional calculus. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning 2 : Classical Papers on Computational Logic 1967-1970*, pages 466–483. Springer, Berlin, Heidelberg, 1983.

[37] L. Zhang. Solving QBF with combined conjunctive and disjunctive normal form. In *National Conference on Artificial Intelligence (AAAI 2006)*, 2006.

[38] L. Zhang and S. Malik. Conflict driven learning in a quantified boolean satisfiability solver. In *International Conference on Computer Aided Design (ICCAD2002)*, 2002.