Une nouvelle sémantique pour la programmation logique capturant la sémantique des modèles stables : la sémantique des extensions

Belaïd Benhamou, Pierre Siegel

Une nouvelle sémantique pour la programmation logique capturant la sémantique des modèles stables : la sémantique des extensions

Belaïd Benhamou et Pierre Siegel

Université d'Aix-Marseille
Centre de Mathématiques et d'Informatique
39, rue Joliot Curie - 13453 Marseille cedex 13, France
{Belaid.Benhamou;siegel}@cmi.univ-mrs.fr

Abstract

Answer set programming is a well studied framework in logic programming. Many research works had been done in order to define a semantics for logic programs. Most of these semantics are iterated fixed point semantics. The main idea is the canonical model approach which is a declarative semantics for logic programs that can be defined by selecting for each program one of its canonical models. The notion of canonical models of a logic program is what it is called the stable models. The stable models of a logic program are in a certain sense the minimal Herbrand models of its "reduct" programs. Here we introduce a new semantics for logic programs that is different from the known fixed point semantics. In our approach, logic programs are expressed as CNF formulas (sets of clauses) of a propositional logic for which we define a notion of extension. We prove in this semantics, that each consistent CNF formula admits at least an extension and for each given stable model of a logic program there exists an extension of its corresponding CNF formula which logically entails it. On the other hand, we show that some of the extensions do not entail any stable model, in this case, we define a simple descrimination condition which allows to recognize such extensions. These extensions could be very important, but are not captured by the stable models semantics. Our approach, extends the stable model semantics in this sense. Following the new semantics, we give a full characterization of the stable models of a logic program by means of the extensions of its CNF encoding verifying a simple condition, and provide a procedure which can be used to compute such extensions from which we deduce the stable models of the given logic program.

Résumé

La programmation par ensembles réponses (Answer Set Programming) est un cadre bien étudié en programmation logique. Plusieurs travaux ont été faits pour définir une sémantique pour les programmes logiques. La plupart de ces sémantiques sont en fait des sémantiques de point fixe. L'idée principale est le calcul de modèles canoniques du programme logique considéré, appelés modèles stables. Les modèles stables sont dans un certain sens des modèles minimaux des programmes réduits. Nous introduisons une nouvelle sémantique pour les programmes logiques, à partir d'une notion d'extension d'une formule propositionnelle classique. Ces extensions peuvent être calculés de manière itérative. Un programme logique est alors codé par un ensemble de clauses de la logique propositionnelle. On prouve que chaque formule consistante admet au moins une extension et que, pour chaque modèle stable d'un programme logique, il existe une extension de son codage qui l'implique logiquement. Certaines des extensions ne correspondent pas à un modèle stable mais sont intéressantes. Nous donnons une condition discriminante simple qui permet de reconnaitre de telles extensions. Enfin, nous décrivons un algorithme qui calcule les extensions de la formule CNF codant le programme logique. De cet ensemble d'extension on peut extraire les modèles stables du programme logique initial.

1 Introduction

L'étude proposée ici se situe dans le cadre de la programmation par ensemble réponse (ASP). Ce cadre peut être vu comme un cas particulier de la logique des défauts [12]. Une des questions importantes de l'ASP est de définir une sémantique pour les programme logiques.

Un programme logique π est un ensemble de rè-

gles de la forme $r : concl(r) \leftarrow prem(r)$, où $prem(r)$ est l'ensemble des prémisses de la règle. Cet ensemble de prémisses est une conjonction de littéraux qui peut contenir des négations classiques et des négations par échec. La partie gauche $concl(r)$ est la conclusion de la règle, exprimée en général par un seul littéral positif, ou dans certains cas par une disjonction de littéraux positifs (programmes logiques disjonctifs). L'ensemble $prem(r)$ peut être aussi appelé corps de la règle r et $concl(r)$ appelé tête de la règle ($r : head(r) \leftarrow body(r)$). Chaque programme logique π est traduit en un programme logique terminal équivalent $Ground(\pi)$ par le remplacement de chaque règle contenant des variables par toutes ses instances terminales, de sorte que chaque littéral de $Ground(\pi)$ est terminal. Cette technique est utilisée pour éliminer les variables même si le programme contient des symboles fonctionnels et si son univers de Herbrand est infini.

Parmi les sémantiques les plus connues pour les programmes logiques, on a la complétion de Clark [1], et la sémantique des modèles stables (ensembles de réponses) [7]. On sait que chaque modèle stable d'un programme logique est un modèle de sa complétion, mais que l'inverse n'est pas toujours vrai. Fages [6] a montré que les deux sémantiques sont équivalentes pour les programmes logiques sans boucle (tight programs). Une généralisation du résultat de Fage aux programmes logiques avec d'éventuelles expressions imbriquées de négations par échec dans les corps de leurs règles a été donné dans [5]. D'autre part, Fangzhen Lin et Zhao Yutin ont proposé dans [3] d'ajouter à la complétion d'un programme logique, des formules appelées *loop formulas*. Ils ont montré que l'ensemble des modèles de la complétion étendue par ces dernières formules est identique à l'ensemble des modèles stables du programme logique considéré même si ce dernier contient des boucles (not tight).

La quasi-totalité des sémantiques connues pour les programmes logiques sont déclaratives et/ou sont des sémantiques de point fixe (comme pour la logique des défauts un modèle stable est un ensemble de littéraux qui vérifie une équation qui peut ne pas avoir de solution). Ici nous introduisons une nouvelle sémantique qui n'est ni déclarative ni définie par un point fixe classique (on a toujours une solution). Cette sémantique peut être considérée comme un cas particulier de la logique des hypothèses [14, 13] qui, en particulier, étend la logique des défauts. La sémantique est basée sur une notion d'extension d'une formule propositionnelle classique. Une extension est obtenue en ajoutant à la formule un ensemble maximal consistant de littéraux. Ces extensions peuvent être calculés de manière itérative. Un programme logique est alors codé par un ensemble de clauses de logique propositionnelle. On prouve que chaque formule consistante admet au moins une extension et que, pour chaque modèle stable d'un programme logique, il existe une extension de son codage qui l'implique logiquement. En d'autres termes tout modèle stable est représenté par une extension. Certaines des extensions (les extra-extensions) ne correspondent pas à un modèle stable, mais sont intéressantes. Par exemple, en terme de représentation des connaissances, nous pouvons avoir un programme logique qui n'a pas de modèle stable à cause de la présence d'une seule règle. Mais l'ensemble des autres règles a des modèles stables pertinents. Dans ce cas les extra-extension permettent en fait d'isoler la règle gênante (on est proche de la para-consistance). D'autre part l'utilisation des extra-extensions va permettre d'avoir un algorithme simple de calcul d'extensions et de modèles stables.

Pour reconnaitre les extra-extensions, nous donnons une *condition discriminante*. L'ajout de cette condition permet, pour les programmes généraux d'avoir une bijection entre les extensions et les modèles stables. Enfin, nous décrivons un algorithme qui calcule les extensions de la formule CNF codant le programme logique. De cet ensemble d'extension, on peut extraire les modèles stables du programme logique initial. L'algorithme peut être mis en oeuvre avec un Solveur SAT légèrement modifié, qui effectue une énumération sur le codage CNF du programme logique et qui utilise un sous-ensemble de variables comme strong backdoor [16] (notée par STB). Ce strong backdoor donne la complexité de la procédure.

Le reste de l'article est structuré comme suit : dans la partie 2 nous donnons quelque préliminaires sur la programmation ASP. Nous introduisons la nouvelle sémantique des programmes logiques généraux et montrons sa relation avec la sémantique des modèles stables dans la partie 3. La partie 4 donne un algorithme basé sur une procédure de type DLL d'un solveurs SAT, qui permet de calculer les extensions à partir desquelles les modèles stables du programme logique peuvent être déduits. Nous donnons une conclusion et des perspectives dans la partie 5.

2 Notions de base

Cette partie rappelle quelques notions de base sur la programmation ASP. Il existe plusieurs classes de programmes logiques, caractérisées par la présence ou l'absence de la négation classique et de la négation par échec. Nous supposons dans la suite de ce travail que tous les programmes sont terminaux [1] (sans variables).

1. Chaque programme logique π est traduit en un programme logic terminal équivalent $ground(\pi)$ par la substitution de chaque règle contenant des variables par toutes ses instances

Les classes les plus connues sont les suivantes :

- *Les programmes logiques positifs :* un programme logique positif π est un ensemble de règles de la forme $r : L_0 \leftarrow L_1, \ldots, L_m$, $(m \geq 0)$ où L_i $(0 \leq i \leq m)$ est un atome. Un programme positif ne contient pas de négation classique ni de négation par échec. On peut le considérer comme un programme Prolog sans négation par échec ou aussi comme un ensemble de clauses de Horn du calcul propositionnel. Ici on a $head(r) = L_0$ et $body(r) = L_1, \ldots, L_m$. Le sens de la règle est "si on peut prouver le corps de la règle $L_1, L_2, \ldots, L_m$, alors on prouve la tête L_0". Etant donné un ensemble d'atomes A, on dit que la règle r est applicable (active) dans A si $body(r) \subseteq A$. Un ensemble d'atomes A est fermé par rapport à un programme logique π si et seulement si pour toute règle $r \in \pi$, si $body(r) \subseteq A$, alors $head(r) \in A$. Un programme logique positif π contient un seul modèle de Herbrand canonique équivalent à l'unique modèle minimal de Herbrand que nous dénotons par $CM(\pi)$. Le modèle minimal de Herbrand de π est le plus petit ensemble d'atomes fermé par rapport à π. Formellement, pour un programme logic π et un ensemble d'atomes A, l'opérateur $T_\pi(A) = \{head(r)/r \in \pi, body(r) \subseteq A\}$ calcule tous les atomes qui peuvent être déduits de A par utilisation des règles de π. Maintenant, nous définissons la suite : $T_\pi^0 = T_\pi(\emptyset)$, $T_\pi^{k+1} = T_\pi(T_\pi^k), \forall k \geq 0$. Le modèle minimal de Herbrand est le plus petit point fixe de T_π, c'est-à-dire $CM(\pi) = \bigcup_{k \geq 0} T_\pi^k$. Le modèle minimal de Herbrand $CM(\pi)$ contient tous les atomes qui peuvent être déduits de π. Ce modèle est égal au modèle minimal exprimé par un programme Prolog formé par les règles de π ou à la partie positive du modèle minimal (au sens des modèles préférentiels) de l'ensemble de clauses de Horn correspondant.
- *Les programmes logiques généraux (normaux) :* un programme logique général π est un ensemble de règles de la forme $r : L_0 \leftarrow L_1, L_2, \ldots, L_m, notL_{m+1}, \ldots, notL_n$, $(0 \leq m < n)$ où L_i $(0 \leq i \leq n)$ sont des atomes, et *not* le symbole exprimant la négation par échec. Le corps positif de r est dénoté par $body^+(r) = \{L_1, L_2, \ldots, L_m\}$, et le corps négatif par $body^-(r) = \{L_{m+1}, \ldots, L_n\}$. Le mot *général* est du au fait que les règles sont plus générales que les clauses de Horn, car elles contiennent des négations par échec. La sous-règle $r^+ : L_0 \leftarrow L_1, L_2, \ldots, L_m$ exprime la projection positive de la règle r. Intuitivement, la règle r est interprétée par "*Si on infère tous les atomes* $\{L_1, L_2, \ldots, L_m\}$ *et qu'aucun des atomes* $\{L_{m+1}, \ldots, L_n\}$ *n'est inféré , alors on infère* L_0". Etant donné un ensemble d'atomes A, on dit que la règle r est applicable (active) dans A si $body^+(r) \subseteq A$ et $body^-(r) \cap A = \emptyset$. Le réduit du programme π par rapport à l'ensemble d'atomes A est le programme positif π^A où on supprime chaque règle contenant une expression $notL_i$ dans son corps négatif body telle que $L_i \in A$ et où on supprime les autres expressions $notL_i$ des corps des autres règles. Plus précisément, $\pi^A = \{r^+/r \in \pi, body^-(r) \cap A = \emptyset\}$. La sémantique la plus connue pour les programmes généraux est celle des modèles stables définie dans [7]. Cette dernière peut être vue comme une amélioration de la négation par échec de Prolog. Un ensemble d'atomes A est un modèle stable (un ensemble réponse) de π si et seulement si A est identique au modèle minimal de Herbrand de π^A, c'est à dire ssi $A = CM(\pi^A)$. La sémantique des modèles stables est basée sur l'hypothèse du monde clos, un atome qui n'est pas dans le modèle stable A est considéré comme faux.
- *Les programmes logiques étendus :* un programme logique étendu π est un ensemble de règles de la forme $r : L_0 \leftarrow L_1, L_2, \ldots, L_m, notL_{m+1}, \ldots, notL_n$ $(0 \leq m < n)$ où L_i, $(0 \leq i \leq n)$ sont des littéraux (des atomes L_i ou leurs négations $\neg L_i$). La sémantique des programmes logiques étendus [8] est une extension de la sémantique des modèles stables définie pour les programme logiques généraux [7]. Premièrement, considérons un programme logique étendu π qui ne contient pas de négations par échec (m=n, dans chaque règle de π). L'ensemble réponse de π est le plus petit ensemble de littéraux A tel que pour chaque règle $L_0 \leftarrow L_1, \ldots, L_m$ de π, si $L_1, \ldots, L_m$ sont dans A, alors L_0 est dans A, et si deux littéraux complémentaires figurent dans A, alors A devrait contenir tous les littéraux du langage de π. Nous dénotons cet ensemble réponse par $A = \alpha(\pi)$. Il est facile de voir qu'il existe un seul modèle $\alpha(\pi)$, et si π ne contient pas de négation, alors le modèle $\alpha(\pi)$ est identique au modèle minimal de Herbrand de π. Soient π un programme logique étendu et Lit l'ensemble de tous les littéraux de son langage. Pour tout sous-ensemble A de littéraux, le réduit du programme π par rapport à A est le programme π^A où on supprime chaque règle contenant une expression $notL_i$ dans son corps négatif telle que $L_i \in A$ et où on supprime les autres expressions $notL_i$ dans les corps des autres règles. Le programme résultant π^A ne contient pas de *not*, et le sous-ensemble A

terminales de telle sorte que chaque littéral de $ground(\pi)$ est terminal.

est un modèle stable pour π si et seulement si il est identique à l'ensemble réponse de π^A. C'est-à-dire, $A = \alpha(\pi^A)$. Ici, la sémantique est différente de celle des programmes généraux, puisque l'hypothèse du monde clos n'est pas appliquée dans le cadre des programmes étendus.

3 La négation par échec exprimée en logique propositionnelle et la notion d'extension

Notre sémantique, inspirée par celle introduite dans la logique des hypothèses [14, 13], part d'une approche logique classique propositionnelle non modale. Elle est basée sur un langage propositionnel L. Dans L on distingue deux types de variables propositionnelles : un sous ensemble de variables classiques $V = \{L_i : L_i \in L\}$ et un autre sous-ensemble $nV = \{notL_i : notL_i \in L\}$. Pour chaque variable dans $L_i \in V$ il existe une variable correspondante $notL_i \in nV$. L'ensemble total des variables de L est $V \cup nV$. Nous utilisons la variable propositionelle $notL_i$ pour exprimer la négation par échec utilisée dans les programmes logiques. A ce niveau, les variables L_i et $notL_i$ sont logiquement indépendants ; il n y a pas de relation sémantique entre ces variables.

Depuis l'introduction de Prolog en 1972 et sa négation par échec, le but de la programmation logique est de définir un lien entre les deux types de variables. Particulièrement, la logique des défauts [12], où la logique des hypothèses [14, 13] représentent deux façons de formaliser un tel lien. Généralement, toutes les logiques non-monotones oeuvrent dans ce sens.

Ici, dans le même esprit que la logique des hypothèses, nous introduisons une sémantique qui donne un lien entre les deux types de variables dans le cadre des ASP. Ce lien est exprimé par l'ajout au langage propositionnel L de l'ensembles de clauses négatives ME=$\{(\neg L_i \vee \neg notL_i) : L_i \in V\}$ équivalent à chacun des ensembles de formules $\{\neg(L_i \wedge notL_i) : L_i \in V\}$, $\{(L_i \rightarrow \neg notL_i) : L_i \in V\}$ et $\{(notL_i \rightarrow \neg L_i) : L_i \in V\}$. C'est en fait un pseudo axiome logique qui s'applique seulement aux couples de variables $\{L_i, notL_i\}$. Il est important de voir que l'ensemble de clauses ME ainsi généré exprime seulement les exclusions mutuelles entre chaque littéral $L_i \in V$ et son littéral négatif correspondent $notL_i \in nV$, mais n'établit pas l'équivalence entre $\neg L_i$ et $notL_i$. Ici, on peut avoir $\neg notL_i$ sans avoir L_i. Intuitivement, dans une première approche, on peut donc considérer qu'une variable $notL_i$ est une négation par échec classique affaiblie. Pour obtenir une vraie négation par échec, il suffit d'ajouter au système logique une propriété (en fait une règle d'inférence) qui oblige d'inférer L_i quand on infère $\neg notL_i$. Cette propriété sera appelée propriété discriminante dans la suite.

Etant donné un ensemble de formules F exprimées dans le langage propositionnel L et un sous-ensemble S de nV, on définit une extension de $(F \cup ME, S)$ comme la théorie obtenue de $F \cup ME$ en ajoutant un nombre maximal de littéraux $notL_i$ de S à $F \cup ME$ de telle sorte que la théorie résultante reste consistante. Une extension de $(F \cup ME, S)$ est donc une théorie maximalement consistante par rapport à l'inclusion de littéraux $notL_i$ de S. Formellement :

Définition 1 *Etant donné un ensemble de formules F du langage L, un sous-ensemble S de nV et un sous-ensemble S' de S, une extension de $(F \cup ME, S)$ est un ensemble $E = (F \cup ME) \cup S'$ tel que les conditions suivantes sont vérifiées :*

1. *E est consistant.*
2. *$\forall notL_i \in S - S', E \cup \{notL_i\}$ is inconsistant*

Exemple 1 *Soit $F = \{(notb \wedge c) \rightarrow a, a \rightarrow b, notd \rightarrow c, a\}$ un ensemble de formules du langage L, alors $ME = \{\neg a \vee \neg nota, \neg b \vee \neg notb, \neg c \vee \neg notc, \neg d \vee \neg notd\}$ et $S = \{notb, notd\}$ un sous-ensemble variables de nV. La paire $(F \cup ME, S)$ admet une seule extension $E = (F \cup ME) \cup \{notd\}$.*

Exemple 2 *Soit $F = \{(notb \wedge c) \rightarrow a, a \rightarrow b, notd \rightarrow c\}$ un ensemble de formules du langage L, on obtient $ME = \{\neg a \vee \neg nota, \neg b \vee \neg notb, \neg c \vee \neg notc, \neg d \vee \neg notd\}$, et $S = \{notb, notd\}$ un sous ensemble de variables de nV. La paire $(F \cup ME, S)$ admets deux extensions $E = (F \cup ME) \cup \{notd\}$ et $E = (F \cup ME) \cup \{notb\}$.*

On montre qu'un ensemble de formules de L contenant ME, qui est consistant a au moins une extension pour tout sous-ensemble $S \subset nV$.

Proposition 1 *Soient F un ensemble de formules du langage L et S un sous-ensemble de nV ($S \subset nV$). Si $(F \cup ME)$ est consistant alors il existe alors au moins une extension de $(F \cup ME, S)$.*

Preuve 1 *Soit S un sous-ensemble nV. Deux cas sont possibles : Si S est fini, alors la preuve est triviale. En effet, puisque $(E \cup ME)$ est consistant par hypothèse, alors en ajoutant successivement des littéraux $notL_i$ à $(E \cup ME)$, nous allons atteindre l'ensemble maximallement consistant $(E \cup ME) \cup S'$ où $S' \subset S$. Si S est infini, la preuve peut être faite par l'utilisation d'une part du théorème de compacité et d'autre part du lemme de Zorn (la preuve n'est pas donnée par manque de place).*

Proposition 2 *Si F est un ensemble de clauses dont chacune contient au moins un littéral positif de V et qui ne contiennent aucun littéral positif $notL_i$ de nV, alors l'ensemble de clauses $F \cup ME$ est consistant.*

Preuve 2 *L'interprétation composée de tous littéraux positifs $L_i \in V$ et de tous les littéraux $\neg notL_i \in nV$ est un modèle trivial.*

3.1 Une nouvelle sémantique pour la programmation logique

Dans la suite nous allons nous intéresser à la classe des programmes logiques généraux où un programme π est un ensemble de règles de la forme $r : L_0 \leftarrow L_1, L_2, \ldots, L_m, notL_{m+1}, \ldots, notL_n$, $(0 \leq m < n)$ et où tout L_i $(0 \leq i \leq n)$ est un atome (une variable propositionnelle). On considère que l'ensemble STB des littéraux positifs du type $notL_i$ qui apparaissent dans π, $STB = \{notL_i : notL_i \in \pi\} \subset nV$, est un strong backdoor [16] de π. Dans cette approche chaque règle r de π est traduite par une formule propositionnelle (une clause) $C = L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m \vee \neg notL_{m+1} \ldots \neg notL_n$. On ajoute à cet ensemble de clauses traduites, l'ensemble des clauses d'exclusion mutuelle ME=$\{(\neg L_i \vee \neg notL_i) : L_i \in V\}$.

Un programme logique général π=$\{r : L_0 \leftarrow L_1, L_2, \ldots, L_m, notL_{m+1}, \ldots, notL_n\}$, $(0 \leq m < n)$, est donc traduit par un ensemble $L(\pi)$ de clauses de Horn propositionnelles qui représente son codage CNF dans un langage propositionnel L :

$$L(\pi) = \{\bigcup_{r \in \pi}(L_0 \vee \neg L_1 \vee, \ldots, \neg \vee L_m \vee \neg notL_{m+1} \ldots \neg notL_n)$$
$$\bigcup_{L_i \in V}(\neg L_i \vee \neg notL_i)\}$$

La taille du codage $L(\pi)$ est donc égale à $taille(\pi) + 2n$, si n est le nombre de variables propositionnelles de π.

Soit donc $L(\pi)$ le codage en calcul propositionnel du programme logique π et $STB \subset nV$ son strong backdoor. Nous allons nous intéresser dans la suite aux extensions du couple $(L(\pi), STB)$. Une extension de $(L(\pi), STB)$ est la théorie obtenue à partir de $L(\pi)$ en ajoutant un ensemble maximal consistant de littéraux $notL_i \in STB$ to $L(\pi)$ tel que la théorie obtenue soit consistante. A partir de la définition **??** on définit donc un cas particulier d'extensions pour les programmes logiques :

Définition 2 *Soit $L(\pi)$ le codage en logique d'un programme logique π, STB son strong backdoor, et un sous ensemble $S' \subset STB$. Alors $E = L(\pi) \cup S'$ est une extension de $(L(\pi), STB)$ si les conditions suivantes sont vérifiées :*

1. *E est consistant*
2. *$\forall notL_i \in STB - S', E \cup \{notL_i\}$ est inconsistant.*

Exemple 3 *On considère l'ensemble F de formules propositionnelles données dans l'exemple 1 comme la traduction logique d'un ensemble de règles π. On donne ci dessous π et son codage logique $L(\pi) = L(\pi) - Rules \cup L(\pi) - ME$:*

π :	$L(\pi) - Rules$:	$L(\pi) - ME$:
$a \leftarrow c, notb$	$a \vee \neg c \vee \neg notb$	$\neg a \vee \neg nota$
$b \leftarrow a$	$b \vee \neg a$	$\neg b \vee \neg notb$
$c \leftarrow notd$	$c \vee \neg notd$	$\neg c \vee \neg notc$
$a \leftarrow$	a	$\neg d \vee \neg notd$

Le strong backdoor est alors l'ensemble $STB = \{notb, notd\}$ et la paire $(L(\pi), STB)$ a une seule extension $E = L(\pi) \cup \{notd\}$. En effet E est consistant (il existe un modèle) et est maximal consistant sur le STB (si on ajoute $notb$ à E, l'ensemble obtenu est inconsistant).

Exemple 4 *On prend π le programme logique de l'exemple 3 vu ci dessus et on supprime la règle $a \leftarrow$. On obtient le programme logique $\pi' = \pi - \{a \leftarrow\}$ dont le codage logique CNF est $L(\pi') = L(\pi) - \{a\}$ et dont le STB est le même que celui de π. Dans ce cas on obtient deux extensions de $(L(\pi'), STB)$: $E_1 = L(\pi') \cup \{notd\}$ et $E_2 = L(\pi) \cup \{notb\}$. Mais $(L(\pi')$ n'a pas de modèle stable. Les deux extensions sont des extra-extensions dont on rediscutera dans la suite.*

Nous allons maintenant montrer que tout modèle stable correspond à une extension.

Théoreme 1 *Si X est un modèle stable d'un programme logique π, alors il existe une extension E de $(L(\pi), STB)$ telle que $X = \{L_i \in V : E \models L_i\}$. D'autre part, E vérifie la "condition discriminante" : $(\forall L_i \in V, E \models \neg notL_i \Rightarrow E \models L_i)$.*

Preuve 3 *Soit l'ensemble $E = L(\pi) \cup \{notL_i / L_i \notin X\}$. Pour démontrer le théorème on montre que E est une extension (1 and 4 ci dessous), que $X = \{L_i : E \models L_i\}$ (2 and 3) et que E vérifie la condition discriminante $(\forall L_i \in V, E \models \neg notL_i \Rightarrow E \models L_i)$ (5).*

1. *E est consistant : on montre que l'interprétation $I = \{\neg L_i, notL_i / L_i \notin X\} \cup \{L_i, \neg notL_i / L_i \in X\}$ est un modèle de E. Comme $\{notL_i / L_i \notin X\} \subset I$, il suffit de montrer que I est un modèle de $L(\pi)$. On doit montrer que chaque clause $(\neg L_i \vee \neg notL_i)$ de ME et chaque clause de $(L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m \vee \neg notL_{m+1} \cdots \vee \neg notL_i \cdots \vee \neg notL_n)$ de $L(\pi)$ qui correspond à une règle de π est satisfaite par I. Il est évident que chaque clause $(\neg L_i \vee \neg notL_i)$ de ME est satisfaite par I. En effet, si $L_i \in X$, on a $\neg notL_i \in I$ par définition de I et la clause est satisfaite par I. Maintenant, si $L_i \notin X$, alors par définition de I on a $\neg L_i \in I$, donc I satisfait la clause $(\neg L_i \vee \neg notL_i)$. Donc $(\neg L_i \vee \neg notL_i)$ est satisfaite par I dans les deux cas. Maintenant pour prouver que chaque clause $(L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m \vee \neg notL_{m+1} \cdots \vee \neg notL_i \cdots \vee \neg notL_n)$ est satisfaite par I, on étudie aussi deux cas. Si la sous clause $(L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m)$ correspond à une regle du réduit π^X, alors X satisfait $L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m$ puisque X est un modèle stable π,*

alors I satisfait la clause $(L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m)$ car $X \subset I$. Donc la clause $(L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m \vee \neg notL_{m+1} \cdots \vee \neg notL_i \cdots \vee \neg notL_n)$ est également satisfaite par I. Dans l'autre cas, la sous clause $(L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m)$ ne correspond pas à une regle de π^X. Il y a alors un $notL_i$ dans la partie négative de la regle de π qui correspond à la clause $(L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m \vee \neg notL_{m+1} \cdots \vee \neg notL_i \cdots \vee \neg notL_n)$ of $L(\pi)$ tel que $L_i \in X$. Alors, $\neg notL_i \in I$ par définition de I et notre clause est satisfaite par I. I est donc un modele de $L(\pi)$ et de E, donc E est consistant.

2. *$L_i \in X \Rightarrow E \models L_i$: à partir de l'ensemble X on obtient le réduit π^X en enlevant en premier lieu dans π toute règle $r : L_0 \leftarrow L_1, L_2, \ldots, L_m, notL_{m+1}, \ldots, notL_n$ $(0 \leq m < n)$ qui contient un littéral $notL_i$ dans sa partie négative, tel que $L_i \in X$. Dans ce cas la clause correspondante $c : L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m \vee \neg notL_{m+1} \ldots \neg notL_n$ de l'ensemble E n'est pas supprimée. Ensuite on supprime dans les autres règles de π toutes les occurrences de $notL_i$ tel que $L_i \notin X$, et dans ce cas, on en déduit à partir de la définition de E que $notL_i \in E$, puis on supprime les littéraux $\neg notL_i$ dans les clauses correspondantes de E. On en déduit que le réduit π^X est inclus dans l'ensemble E^X obtenu $\tilde{A}$ partir de E en appliquant la résolution unitaire sur les littéraux de la forme $notL_i$ $(E \models E^X)$. Comme X est un modèle stable de π, par définition, L_i appartient au modèle de Herbrand minimal de π^X. On en déduit que $\forall L_i \in X, \pi^X \models L_i$. Comme $\pi^X \subset E^X$, alors $E^X \models \pi^X$ et donc $E \models \pi^X$. Comme l'inférence est monotone on déduit que $\forall L_i \in X, E \models L_i$.*
3. *$E \models L_i \Rightarrow L_i \in X$: c'est équivalent à montrer que $L_i \notin X \Rightarrow E \not\models L_i$. De $L_i \notin X$ on a par définition de E que $notL_i \in E$, donc par application de la unit résolution sur la clause $(\neg L_i \vee \neg notL_i)$ générée par le nouveau pseudo axiome, on déduit que $E \models \neg L_i$. On en conclu que $E \not\models L_i$ car E est consistant (montré en 1).*
4. *E est maximal consistant sur STB : Soit l'ensemble $E' = E \cup \{notL_i\}$, tel que $notL_i \in STB$, but $notL_i \notin E$. Comme $notL_i \notin E$, par définition de E, on déduit que $L_i \in X$. En appliquant la propriété démontrée en 2) on obtient $E \models L_i$. Donc, $E' \models L_i$. En appliquant la résolution unitaire sur la clause $(\neg L_i \vee \neg notL_i)$ générée par le pseudo axiome, on déduit que $E' \models \neg notL_i$ et que E' n'est pas consistant car il infère $notL_i$ et son complément $\neg notL_i$. Donc E est maximal consistant sur STB.*
5. *$(\forall L_i \in V, E \models \neg notL_i \Rightarrow E \models L_i)$: il est équivalent de montrer que $(\forall L_i \in V, E \not\models L_i \Rightarrow E \not\models \neg notL_i)$. Si $E \not\models L_i$, comme $X = \{L_i : E \models L_i\}$, on déduit que $L_i \notin X$, donc $notL_i \in E$ par définition de E. Comme E est consistant on conclut que $E \not\models notL_i$.*

Exemple 5 *Soit le programme logique π de l'exemple 3 et son codage CFN $L(\pi)$. π a un seul modèle stable $X = \{a, b, c\}$ qui correspond à l'unique extension $E = L(\pi) \cup \{notd\}$ de $(L(\pi), STB)$. En effet, après l'affectation des variables du STB, on utilise la résolution unitaire pour montrer que $E \models \{a, b, c, \neg d, \neg nota, \neg notb, \neg notc\}$. On voit ici que E vérifie bien la condition discriminante $(\forall L_i \in V, E \models \neg notL_i \Rightarrow E \models L_i)$ et que $X = \{L_i \in V : E \models L_i\} = \{a, b, c\}$.*

Remarque 1 *Comme (π) est un programme général, une extension E de $(L(\pi), STB)$ est un ensemble consistant de clauses de Horn . Donc tous les littéraux positifs (les clauses unitaires positives) qui peuvent être inférées par E sont produits par résolution unitaire. Par exemple les littéraux du modèle stable $X = \{a, b, c\}$ de l'exemple précédent sont inférés par résolution unitaire. Ceci est important pour la complexité de l'algorithme de calcul des extensions et des modèles stable qui va être donné dans la partie 4.*

Remarque 2 *La proposition 1, dit que $(L(\pi), STB)$ a toujours une extension si $L(\pi)$ est consistant. Mais un programme logique π peut ne pas avoir de modèle stable. Il peut donc exister des extensions (extra-extensions) de $(L(\pi), STB)$ qui ne correspondent à aucun modèle stable de π. On va montrer que ces extra-extensions sont exactement celles qui ne vérifient pas la condition discriminante. Ces extra-extensions, non prises en compte par la sémantique des modèle stables, sont très intéressantes pour simplifier le calcul d'extension (donc de modèles stables). Elles sont également importantes pour la représentation des connaissances, mais on ne discutera pas de cet aspect dans cet article.*

Par exemple, dans l'exemple 4, on a deux extensions de $(L(\pi'), STB) : E_1 = L(\pi') \cup \{notd\}$ et $E_2 = L(\pi) \cup \{notb\}$ qui ne correspondent pas à un modèle stable. D'un autre coté, on peut montrer que $E_1 \models \{c, \neg d, \neg notb, \neg notc\}$ et que $E_2 \models \{\neg b, \neg a, \neg c, \neg notd\}$. On remarque donc que ces deux extensions ne vérifient pas la condition discriminante du Théorème 1, car E_1 implique $\neg notb$ mais n'implique pas b et également E_2 implique $\neg notd$ mais n'implique pas d. On parlera dans la suite d'extra-extensions.

Ci dessous on donne deux exemples d'école très utilisés par la communauté qui étudiait la logique des défauts à une époque. On montre que notre sémantique est valide pour les deux exemples .

Exemple 6 *Soit le programme $\pi = \{a \leftarrow nota\}$ composé dune seule règle. Ce programme n'a pas de modèle stable. Son codage CNF est $L(\pi) = \{a \vee \neg nota, \neg a \vee \neg nota\}$ et son strong backdoor est $STB = \{nota\}$. La paire $(L(\pi), STB)$ a une extra-extension $E = \{\neg nota\}$. En effet par résolution sur les deux clauses de $L(\pi)$ on infère la clause unitaire $\neg nota$ qui subsume les clauses de $L(\pi)$. Donc $E \models \neg nota$ et $E \not\models a$. L'extension E ne vérifie donc pas la condition discriminante. On prouve donc pour cet exemple que $(L(\pi), STB)$ a une seule extension qui est une extra-extension. Notre sémantique capture bien la sémantique classique des modèles stables pour cet exemple.*

Exemple 7 *On prend le programme logique donné dans l'exemple 6 auquel on ajoute une règle $a \leftarrow$. On obtient*

le programme $\pi = \{a \leftarrow nota, a \leftarrow\}$ *qui a un unique modèle stable* $X = \{a\}$. *Son codage CNF est* $L(\pi) = \{a \vee \neg nota, \neg a \vee \neg nota, a\}$ *et son strong backdoor est* $STB = \{nota\}$. *La paire* $(L(\pi), STB)$ *a une extension* $E = L(\pi)$, *telle que* $E \models \neg nota$ *et* $E \models a$. *Cette extension vérifie bien la condition discriminante et implique le modèle stable* $X = \{a\}$ *de* π. *La sémantique capture bien ici aussi les modèles stables.*

Nous allons maintenant généraliser les exemples ci dessus en prouvant que toute extension de $(L(\pi), STB)$ qui vérifie la condition discriminante implique un modèle stable de π. On aura donc une bijection entre les modèles stables et les extensions qui ne sont pas des extra-extensions

Théoreme 2 *Si* E *est une extension de* $(L(\pi), STB)$, *qui vérifie la condition* $(\forall L_i \in V, E \models \neg notL_i \Rightarrow E \models L_i)$ *alors then* $X = \{L_i : E \models L_i\}$ *est un modèle stable de* π.

Preuve 4 *Soit* E *une extension de* $(L(\pi), STB)$. *Il existe donc un ensemble* $S' \subset STB$ *tel que* $E = L(\pi) \cup S'$. *Comme* E *est maximal consistant sur* STB, *on a* $E \models \neg notL_i$ *pour tout* $notL_i \in STB - S'$. *Donc,* $E \models L(\pi) \cup S' \cup \{\neg notL_i : notL_i \in STB - S\}$. *D'un autre coté l'ensemble* $X = \{L_i : E \models L_i\}$ *est un modèle minimal de* E. *En utilisant le pseudo axiome et la condition discriminante on montre que* $X = \{L_i : E \models L_i\} = \{L_i : E \models \neg notL_i\}$. *Pour prouver que* X *est un modèle stable* π, *on doit montrer que* X *est un modèle minimal de* π^X. *Pour ce faire on étudie deux cas. Pour le premier cas, si* $notL_i \in E$, *on supprime toute les occurrences du littéral* $\neg notL_i$ *dans toutes les clauses* c : $(L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m \vee \neg notL_{m+1} \ldots, \neg notL_i, \ldots, \neg notL_n)$ *de* $L(\pi)$. *En utilisant la clause* $(\neg L_i \vee \neg notL_i)$ *de* ME *on déduit que* $E \models \neg L_i$, *et donc que* $E \not\models L_i$ *car* E *est consistant. On en déduit que* $L_i \notin X$, *et dans ce cas, pour obtenir le réduit* π^X, *on supprime toutes les occurrences de* $notL_i$ *dans le corps de la règle* r : $L_0 \leftarrow L_1, L_2, \ldots, L_m, notL_{m+1}, \ldots, notL_i, \ldots, notL_n$ *de* π *qui correspond à la clause* c. *Pour le second cas* $notL_i \notin E$, *et donc* $E \models \neg notL_i$ *et toute clause* c *de* $L(\pi)$ *qui contient un littéral* $\neg notL_i$ *est subsumée dans* E. *En utilisant la condition discriminante* E, *on déduit que* $E \models L_i$. *Donc* $L_i \in X$, *et dans ce cas, pour obtenir la réduction* π^X, *la règle correspondante* r *dans* π *de la clause* c *est supprimée. On en conclut que la simplification de* E *par les littéraux du STB donne une réduction* π^X *qui est une conséquence logique de* E. *Donc* $E \models \pi^X$. *Comme* X *est un modèle minimal de* E, *on a que* X *est un modèle minimal de* π^X. *Par définition des modèles stables, on conclut que* X *est un modèle stable de* π.

On donne ci dessous deux autres exemples classiques qui viennent de la logique des défauts.

Exemple 8 *Soit le programme logique* π *composé de deux règles :*

$$\pi = \{a \leftarrow notb, b \leftarrow nota\}$$

Son codage CNF est l'ensemble de clauses $L(\pi) = Rules \cup ME$ *:*

$$Rules = \{a \vee \neg notb, b \vee \neg nota\}$$

$$ME = \{\neg a \vee \neg nota, \neg b \vee \neg notb\}$$

Son backdoor est $STB = \{nota, notb\}$. *La paire* $(L(\pi), STB)$ *a deux extensions* $E_1 = L(\pi) \cup \{nota\}$ *and* $E_2 = L(\pi) \cup \{notb\}$. *Par résolution unitaire on déduit que* $E_1 \models \{b, \neg a, \neg notb\}$ *and* $E_2 \models \{a, \neg b, \neg nota\}$. *Donc ces extensions vérifient la condition discriminante. A partir de* E_1 *resp.* E_2 *on obtient les ensembles de littéraux positifs impliqués* $X_1 = \{b\}$ *resp.* $X_1 = \{a\}$ *qui sont bien les deux modèles stables du programme* π.

Exemple 9 *Soit le programme logique* π *:*

$$\pi = \{a \leftarrow notb, b \leftarrow notc, c \leftarrow nota\}$$

Son codage CNF est l'ensemble de clauses $L(\pi) = Rules \cup ME$ *:*

$$Rules = \{a \vee \neg notb, b \vee \neg notc, c \vee \neg nota\}$$

$$ME = \{\neg a \vee \neg nota, \neg b \vee \neg notb, \neg c \vee \neg notc\}$$

,

Le backdoor est $STB = \{nota, notb, notc\}$. *La paire* $(L(\pi), STB)$ *a trois extensions* $E_1 = L(\pi) \cup \{nota\}$, $E_2 = L(\pi) \cup \{notb\}$ *et* $E_3 = L(\pi) \cup \{notc\}$. *Par résolution unitaire, on déduit que* $E_1 \models \{\neg a, c, \neg notc, \neg notb\}$, $E_2 \models \{\neg b, a, \neg nota, \neg notc\}$ *et* $E_3 \models \{\neg c, b, \neg notb, \neg nota\}$. *On voit que les trois extensions sont des extra-extensions qui ne vérifient pas la condition* $(\forall L_i \in V, E \models \neg notL_i \Rightarrow E \models L_i)$. *Donc le théorème 2 dit que* π *n'a pas de modèle stable, ce qui est bien le cas.*

On a donc montré avec les théorème 1 et théorème 2 que les modèles stables d'un programme logique π sont en bijection avec un sous ensemble des extensions de $(L(\pi), STB)$. Nous avons aussi caractérisé les extra-extensions par une condition simple à vérifier. D'autre part on a vu qu'un ensemble de formules consistant a toujours une extension. Cette dernière propriété entraine que la définition d'extension n'est pas une définition par point fixe et qu'il est possible de calculer les extensions de manière incrémentale : on ajoute des $notL_i$ du STB en vérifiant à chaque ajout la consistance de l'ensemble obtenu. On peut donc obtenir des algorithmes très simples de calcul d'extension. Si on ne s'intéresse pas aux extra-extension, il suffit de les filtrer à la fin du calcul : dès qu'une extension a été trouvée on teste si elle correspond un modèle stable en vérifiant la propriété discriminante. On va donner dans la partie suivante une méthode de calcul d'extensions et de modèles stables basée sur ces considérations.

4 Un algorithme de calcul d'extensions ou de modèles stables basé sur la nouvelle sémantique

On sait que tout modèle stable d'un programme π est un modèle de sa complétion $comp(\pi)$, mais que la réciproque est fausse dans le cas général. Fages [6] montre que si un programme π est " tight" (sans boucle) alors l'ensemble des modèles stables est égal à l'ensemble des modèles de sa

complétion $comp(\pi)$ [1]. Si la complétion d'un programme sans boucle est traduite par un ensemble de clauses Γ, alors on peut utiliser un solveur SAT Γ comme une boite noire pour générer les modèles stables de π. Lin et Zhao [3] ont montré que pour les programmes avec boucle, les modèles de la complétion qui ne sont pas des modèles stables peuvent être éliminés en ajoutant à la complétion des *formules boucles*. Leur solveur ASSAT est basé sur cette technique et est plus performant que les solveurs ASP classiques tel que Smodels [11, 15] et DLV [4] pour plusieurs instances. Néanmoins le solveur ASSAT a quelques défauts : il ne peut calculer qu'un seul modèle stable et la formule calculée peut exploser de manière combinatoore en espace. En prenant en compte le fait que chaque modèle stable d'un programme π est un modèle de sa complétion $comp(\pi)$, Guinchiglia et al. in [9] n'utilisent pas un solveur SAT comme boite noire, mais implêmentent une méthode basée sur la procédure DLL [2] dans laquelle ils introduisent une fonction qui vérifie si un modèle généré est un modèle stable. Cette méthode a été implémentée dans le système Cmodels-2 [10] et a l'avantage de calculer $comp(\pi)$ sans introduire de variable supplémentaire, autre que celles utilisées par la transformation en clauses de $comp(\pi)$. Elle fonctionne également pour les programmes avec boucles et sans boucles.

4.1 La méthode

La méthode décrite ici utilise également un soldeur SAT mais elle est différente car elle basée sur la nouvelle sémantique décrite plus haut. Pour un programme π la méthode travaille sur $L(\pi)$ et non pas sur la complétion $comp(\pi)$. Si π est un programme général, $L(\pi)$ est un ensemble de clauses de Horn construit sur deux ensembles de variables propositionnelles V and nV. L'ensemble $STB \subset nV$ des variables qui apparaissent dans le corps des règles de π est un strong backdoor [16] que l'on énumère pour obtenir les extensions de $(L(\pi), STB)$ à partir desquelles on peut trouver les modèles stables de π.

Proposition 3 *Soit π un programme général et $L(\pi)$ son codage CNF. Soit E une extension de $L(\pi)$sur laquelle on a appliqué la simplification par résolution unitaire et subsomption (suppression de clauses par subsomption et suppression de variables propositionnelles par résolution unitaire). On a alors les propriétés suivantes :*

1. *L'ensemble de clauses E est l'union d'un ensemble C_1 de clauses unitaires C_1 et d'un ensemble C_2 de clauses de Horn non unitaires qui ne contient pas de variables propositionnelles du STB. De plus les ensembles C_1 and C_2 n'ont pas de variable en commun (variables indépendants).*
2. *Si on affecte à faux tous les $\{notL_i\}$ et tous les $\{L_i\}$ qui ne sont pas déjà affectés dans C_1 on obtient le modèle minimal M de E (il est unique car E est un ensemble de clauses de Horn). Ce modèle minimal sera d'une part la partie positive de l'extension E (l'ensemble des littéraux positifs impliqué par E) et d'autre part un candidat comme modèle stable.*
3. *Pour savoir si M est un modèle stable, il suffit de vérifier si la condition caractéristique est vérifiée. Comme toutes les variables ont été affectées, cette vérification est immédiate.*

Preuve 5
- *Assertion 1 : $L(\pi)$ est un ensemble de clauses de Horn. Donc après l'affectation de toutes les variables du backdoor STB et la simplification par résolution unitaire et subsomption, le codage $L(\pi)$ est l'union de l'ensemble des clauses unitaires obtenues par la résolution unitaire et d'un ensemble de clauses de Horn non unitaires. Le fait que ces ensembles n'aient pas de variable propositionnelle en commun vient du fait que $L(\pi)$ a été simplifié par résolution unitaire et subsomption.*
- *Assertion 2 : Après affectation des $\{notL_i\}$ du STB, simplification par résolution unitaire et subsompsion il ne reste plus de $\{notL_i\}$ du STB dans les clauses non unitaires de E. On peut donc compléter l'interprétation en affectant à faux les variables propositionnelles non affectées. On obtient alors un modèle minimal (au sens des modèles préférentiels) qui est un candidat à la vérification pour être modèle stable.*
- *Assertion 3 : Ceci a été montré dans la démonstration du théoreme 2.*

A partir de ce qui précède on va donner ci dessous les étapes principales d'une méthode de calcul de toutes les d'extension et de tous les modèles stables.

1. Soit $E = L(\pi)$.
2. On affecte les variables du STB en utilisant une méthode d'énumération DLL. De plus on vérifie à chaque affectation la consistance. Comme on a un ensemble de clauses de Horn, une propagation par résolution unitaire assure la consistance. On maintient également à chaque point de choix l'ensemble des littéraux $\{\neg notL_i\}$ impliquées par l'état courant. Pour les maintenir les $\{\neg notL_i\}$ impliqués, une méthode très brutale est de faire k tests de consistance (k propagations unitaire) si k est le cardinal du STB (on peut être bien plus efficaces). Pour la stratégie d'affectation des variables du STB, une bonne approche est d'affecter en premier à Vrai les variables libres du STB. Puis on ne reviendra en arrière que sur les les noeuds qui ont inféré au moins un nouveau littéral négatif du STB. Cette stratégie assure qu'à chaque point de choix consistant, en poursuivant l'énumération du STB, on obtiendra au moins une extension (la maximalité de l'extension est donc obtenue automatiquement).
   Une fois l'affectation du STB effectuée (on est sur une feuille de l'arbre de recherche restreint au STB), on obtient donc une interprétation partielle I_{STB} du STB qui étend E ($E = E \cup I_{STB}$)).
3. On affecte à Faux toutes les variables de E non affectées pour obtenir le modèle minimal M de E.
4. Si la condition ($\forall L_i \in V : E \models \neg notL_i \Rightarrow E \models L_i$) est vérifié, la restriction aux littéraux positifs de E est un modèle stable de π. Cette vérification est

immédiate à partir du modèle minimal M obtenu à l'étape 3.

L'algorithme est décrit ici de manière très générale. Dans la pratique on peut le rendre bien plus efficace, mais ce n'était pas le premier but de cet article.

4.2 Complexité

Si n est le nombre de variables du codage $L(\pi)$ d'un programme π, k est le cardinal du STB , m est le nombre de clauses de $L(\pi)$, l'étape 2 de l'algorithme peut être effectuée en $O(knm2^k)$. Donc la complexité asymptotique dans le pire des cas est $O(knm2^k)$ avec $k \leq n$. Cette estimation est très grossière et peut être très améliorée. En particulier avec des considérations simples on peut diviser k par 2. Dans le cas moyen la complexité est bien moindre. De plus, il est intéressant de remarquer que la première extension est trouvée à la première descente dans l'arbre sans retour en arrière, donc en $O(knm)$. Enfin la méthode a l'avantage de travailler sur un codage CNF en espace constant $L(\pi)$ dont la taille est $taille(\pi) + 2n$. Cette méthode peut être utilisée pour des programmes avec boucles et sans boucles. Elle est capable de calculer tous les modèles stables d'un programme logique général π. La méthode peut être implémentée, avec une modification mineure de la procédure DDL. Ce travail est théorique mais tous les outils pour l'implémenter existent. On pourra alors faire les comparaisons de performance avec les méthodes classiques

5 Conclusion

Nous avons décrit, de manière théorique, une nouvelle sémantique pour la programmation logique. Cette sémantique n'est pas une sémantique de point fixe classique. Les règles d'un programme logique sont codées par un ensemble de clauses propositionnelles dans un cadre de logique classique. On définit, de manière générale, une notion d'extension qui appliquée aux programmes logiques permet de capturer les modèles stables. On montre que chaque programme logique consistant a au moins une extension et que tout modèle stable d'un programme logique est impliqué par une extension. La notion d'extension est plus générale que les modèles stables et on a des extra-extensions qui ne représentent pas un modèle stable. On donne un critère simple pour reconnaitre ces extra-extensions. Enfin on définit un algorithme qui permet de calculer les extensions et les modèles stables. Cet algorithme est basé sur une modification mineure de la procédure DDL de SAT et énumère un ensemble restreint STB de variables propositionnelles, dont dépend la complexité.

Comme travail futur nous nous intéresserons à l'implémentation de l'algorithme et à sa comparaison avec les méthodes existantes. D'autre part le formalisme des extensions peut facilement prendre en compte les programmes étendus avec disjonctions en partie gauche des règles : au lieu de fournir un unique modèle minimal, une extension en fournira plusieurs. Ceci coûte bien entendu une augmentation de la la complexité algorithmique (on quitte les clauses de Horn). Pour étudier les programmes étendus la notion d'extension et les techniques décrites ici fournissent une bonne base. On peut aussi se rapprocher de la logique des défauts et des logiques non monotones plus subtiles, mais pour ceci il faut revenir un peu vers la logique des hypothèses.

Références

[1] K. Clark. Negation as failure. In *Logic and data bases*, pages 293–322. In Herve Gallaire and Jack Minker, editors, 1978.

[2] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *JACM, 5(7).*

[3] E.Erdem and Y.Zhao. Assat : Computing answer sets of a logic program by sat solver. In *AAAI-02*, 2002.

[4] T. Eiter. Th kr sysem dlv :progress report, comparisons and benchmarks. In *KR*, 1978.

[5] E. Erdem and V. Lifschitz. Tight logicprograms. *Thoery and Practice of Logic Programming*, 3 :499–518, 2003.

[6] F. Fages. Consistency ofclark's completion and existence of stable models. *Thoery and Practice of Logic Programming*, 1 :51–60, 1994.

[7] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Logic programming : Fifth Int'l Conf. and Symp.*, pages 1070–1080. In Robert Kawalski and Kenneth Bowen editors, 1988.

[8] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9 :365–385, 1991.

[9] E. Giunchiglia, Y. Lierler, and M. Mratea. Sat-based answer set programming. In *19th National Conference on Artificial Intelligence,July 25-29, San Jose, California. AAAI*, 2004.

[10] Yuliya Lierler and Marco Maratea. Cmodels-2 : Sat-based answer set solver enhanced to non-tight programs. In *Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 346–350, 2004.

[11] I. Niemela. Logic programs with stable models semantics as a constraint programming paradigm. *Anals of mathematics and Artificial Intelligence*, 25 :241–273, 1999.

[12] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13 :81–132, 1980.

[13] C. Schwind and P. Siegl. A modal logic for hypothesis theory. *Fondamunta Informatika*, 21 (1/2) :89–101, 1994.

[14] P. Siegl and C. Schwind. Hypothesis theory for nonmonotonic reasoning. In *Workshop on Nonstandard Queries and Answers*, Toulouse, July 1991.

[15] P. Simons. Extending and implementing the stable model semantics. *In doctoral dessertation*, pages 305–316, 2000.

[16] Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In *IJCAI*, pages 1173–1178, 2003.