De l'utilisation de l'Answer Set Programming pour la formalisation de logiques de description

Laurent Garcia, Aymeric Le Dorze

De l'utilisation de l'Answer Set Programming pour la formalisation de logiques de description *

Answer Set Programming applied to Description Logic

Laurent GARCIA Aymeric LE DORZE

LERIA, Université d'Angers
2, bd Lavoisier, 49045 ANGERS CEDEX 05
{garcia,ledorze}@info.univ-angers.fr

Résumé

Nous voulons utiliser l'ASP comme un cadre unifiant les aspects du web sémantique correspondant aux niveaux Ontologies et Règles de la pile du web sémantique pour permettre d'utiliser la puissance des solveurs ASP pour interroger les informations décrites. Pour cela, nous étudions les différents opérateurs utilisés en logique de description et en donnons une traduction en ASP. Notre travail porte sur la logique $\mathcal{SHOIQ}$.

Mots-Clés

Answer Set Programming, Logiques de description, Web sémantique

Abstract

We want to use ASP as a framework to unify the layers about Ontology and Rules of the semantic web stack to allow the power of ASP solvers to interrogate the given information. To this aim, we study the operators used in description logic and we give their translation into ASP. Our work deals with the logic $\mathcal{SHOIQ}$.

Keywords

Answer Set Programming, Description Logic, Semantic Web

1 Introduction

De plus en plus de bases de connaissances sont représentées grâce à des logiques de description, notamment dans le cadre du Web sémantique [2] et plus spécifiquement pour le niveau *Ontologie* de la pile du Web sémantique. Celles-ci sont une famille des langages de représentation de connaissances. Elles incluent les notions d'individu, de concept, de rôle, ou encore d'héritage et servent à lever certaines ambiguïtés qu'il peut exister dans les relations entre ces différentes notions. Ce sont des logiques décidables qui possèdent différents niveaux d'expressivité en fonction des opérateurs utilisés dans chacune d'entre elles.

L'Answer Set Programming (ASP) [5, 6] est un langage de programmation logique et donc capable d'induire, à partir de faits, de nouveaux faits suivant des règles. Formalisme non monotone, l'ASP peut induire des faits à partir de règles même si on ne connaît pas la valeur de vérité de chacun des faits qui interviennent dans cette règle. Langage de programmation logique, il est parfaitement adapté pour représenter le niveau *Règles* du Web sémantique. Il dispose en outre d'un atout par rapport aux autres langages de programmation logique classiques. En effet, avec les ontologies du Web sémantique, on fait l'hypothèse du monde ouvert, c'est-à-dire que si on ne peut pas prouver qu'un fait est vrai, alors on ne peut rien déduire sur ce fait. Ce choix a été fait en raison de la quantité très importante d'informations contenues sur le Web : il serait impensable d'interroger la totalité des sites Web pour vérifier la véracité d'une information et on préférera ne pas se prononcer. En programmation logique classique, on fait généralement l'hypothèse du monde clos, c'est-à-dire que si on ne peut pas prouver qu'un fait est vrai, alors on déduit que ce fait est faux. Il n'y a donc pas de distinction entre une information prouvée comme étant fausse et une information supposée fausse car non prouvée vraie. En ASP, on peut distinguer les informations prouvées comme étant fausses et celles étant déclarées fausses par défaut.

En décidant de coder le niveau *Règles* par un ensemble de règles ASP, on doit s'assurer que ces règles puissent interagir avec le niveau *Ontologie* de la pile ; par conséquent, il faut que ces deux niveaux puissent s'échanger des informations. Deux approches ont déjà été proposées : la première sépare l'ASP et la logique de description dans deux modules distincts qui peuvent s'échanger des informations [4] ; la seconde fusionne les niveaux *Règles* et *Ontologie* en exprimant ce dernier niveau sous la forme d'un programme ASP [7].

Tout d'abord, notons que notre travail n'a pas pour objectif principal d'être utilisé pour le web sémantique et nous nous

*Ce travail a été réalisé en collaboration avec Pascal NICOLAS, décédé en décembre 2010.

intéressons aux logiques de description quel que soit leur domaine d'application. Il est néanmoins difficile de faire abstraction du web sémantique et c'est pourquoi nous situons aussi brièvement nos travaux dans ce contexte.
Dans cet article, notre idée est de tout intégrer en ASP afin de n'avoir à gérer qu'un seul formalisme de représentation permettant à la fois de manipuler des connaissances représentées par une ontologie exprimée dans une logique de description et des règles décrivant des connaissances non exprimables en logique de description et permettant des raisonnements non-classiques (en particulier non-monotones).
Ainsi, le but du travail est de parvenir à coder en ASP les informations habituellement représentées en logique de description pour les lier avec des règles ASP. Il faut donc étudier chaque particularité de la logique de description choisie et donner son équivalent en ASP. Dans la section 2, nous donnons quelques rappels sur les logiques de description ainsi que les définitions formelles concernant l'ASP. Dans la section 3, nous présentons le travail proprement dit qui consiste à coder en ASP les ontologies exprimées dans la logique de description $\mathcal{SHOIQ}$. Nous présentons les principes sur lesquels on s'appuie, les problèmes des méthodes existantes et les solutions que nous avons apportées. Nous terminons, en section 4, par une conclusion et des perspectives.

2 Rappels

2.1 Logiques de description

Le niveau *Ontologie* du Web sémantique est basé sur les logiques de description, qui permettent de représenter des connaissances. Elles expriment des relations entre des individus grâce à des concepts et des rôles. Il existe différentes logiques de description, qu'on identifie en fonction de leurs extensions. Les logiques de description sont généralement basées sur la logique de description $\mathcal{ALC}$. Les lettres de cette logique de description représentent chacune une extension :

$\mathcal{AL}$ comprend les concepts, les rôles, le concept $\top$ [1], la hiérarchie des concepts, l'intersection de concepts et la restriction universelle ;

$\mathcal{C}$ comprend le complément de concepts [2].

D'autres extensions (identifiées elles aussi par des lettres) existent et s'ajoutent à la logique $\mathcal{ALC}$:

$\mathcal{S}$ comprend la transitivité des rôles ;

$\mathcal{H}$ comprend la hiérarchie des rôles ;

$\mathcal{R}$ comprend l'inclusion de rôles, la réflexivité et les rôles disjoints ($\mathcal{S} \subset \mathcal{R}$ et $\mathcal{H} \subset \mathcal{R}$) ;

$\mathcal{O}$ comprend les concepts nominaux (c'est-à-dire les concepts définis comme étant des ensembles d'individus explicitement donnés) ;

$\mathcal{I}$ comprend l'inversion des rôles (si R est un rôle, alors son rôle inverse noté R^- est tel que pour tous les individus x et y de l'ontologie, si x est lié à y par le rôle R, alors y est lié à x par le rôle R^-) ;

$\mathcal{F}$ comprend les rôles fonctionnels (si R est un rôle fonctionnel, alors pour tout individu x de l'ontologie il existe au plus un individu y tel que x est lié à y par le rôle R) ;

$\mathcal{N}$ comprend les restrictions cardinales sur les rôles ($\mathcal{F} \subset \mathcal{N}$) ;

$\mathcal{Q}$ comprend les restrictions cardinales qualifiées sur les rôles (restrictions cardinales liées à un concept particulier ; $\mathcal{N} \subset \mathcal{Q}$) ;

$(\mathbf{D})$ correspond à l'utilisation de rôles, de valeurs et de types primitifs.

Nous définissons ici formellement la logique de description $\mathcal{SHOIQ}$. On définit un vocabulaire $\Psi = (\mathbf{A}, \mathbf{R}_A, \mathbf{I})$ où $\mathbf{A}$, $\mathbf{R}_A$ et $\mathbf{I}$ sont deux à deux disjoints et :
– $\mathbf{A}$ est un ensemble de *concepts atomiques* ;
– $\mathbf{R}_A$ un ensemble de *rôles abstraits* (lie un individu à un autre individu) ;
– $\mathbf{I}$ un ensemble d'*individus*.
On note $\mathbf{R}_A^- = \{R^-, R \in \mathbf{R}_A\}$. Un *rôle* est un élément de $\mathbf{R}_A \cup \mathbf{R}_A^-$. Les *concepts* sont définis tels que :

1. tout concept atomique $C \in \mathbf{A}$ est un concept ;
2. si $o_1, \ldots, o_n$ sont des individus de $\mathbf{I}$, alors $\{o_1, \ldots, o_n\}$ est un concept (appelé concept nominal) ;
3. si C et D sont des concepts, alors $(C \sqcap D)$, $(C \sqcup D)$, $\neg C$ sont des concepts (appelés respectivement *intersection*, *union* et *complément*) ;
4. si C est un concept, R un rôle abstrait de $\mathbf{R}_A \cup \mathbf{R}_A^-$, et n un entier positif, alors $\exists R.C$, $\forall R.C$, $\geq nR.C$, $\leq nR.C$ et $=nR.C$ sont des concepts (appelés respectivement *restriction existentielle*, *universelle*, *minimale*, *maximale* et *exacte*).

Les axiomes sont une expression de l'une des formes suivantes :

1. $C \sqsubseteq D$ (*inclusion de concept*) (C et D des concepts) ;
2. $R \sqsubseteq S$ (*inclusion de rôle*) ($R, S \in \mathbf{R}_A$) ;
3. $\mathrm{Trans}(R)$ (*transitivité*) ($R \in \mathbf{R}_A$) ;
4. $a : C$ (*appartenance à un concept*) (C un concept et $a \in \mathbf{I}$) ;
5. $(a, b) : R$ (*appartenance à un rôle*) ($R \in \mathbf{R}_A$ et $a, b \in \mathbf{I}$) ;
6. $a = b$ (respectivement $a \neq b$) (*égalité* (respectivement *inégalité*)) ($a, b \in \mathbf{I}$)

Les logiques de description séparent ces axiomes en deux ensembles distincts :

T-BOX contient la terminologie, c'est-à-dire les relations entre concepts et entre rôles ; elle contient donc les axiomes :

1. $\top$ contient la totalité des individus d'une ontologie
2. à noter que la combinaison de $\mathcal{AL}$ et de $\mathcal{C}$ permet de contruire l'union (extension $\mathcal{U}$) et la restriction existentielle (extension ε)

– d'inclusion de concepts ;
– d'inclusion de rôles ;
– de transitivité.

A-BOX contient les assertions, c'est-à-dire l'appartenance des individus à des concepts ou à des rôles ; elle contient donc les axiomes :
– d'appartenance à un concept ;
– d'appartenance à un rôle ;
– d'égalité et d'inégalité.

Une *base de connaissances* ou *ontologie* est l'association d'une T-BOX et d'une A-BOX.
Il existe quatre manières d'interroger une ontologie exprimée dans une logique de description :

Satisfiabilité de la base teste si l'ontologie est satisfiable (c'est-à-dire s'il ne peut pas exister un individu appartenant à la fois à un concept et à son complément) ;

Satisfiabilité d'un concept teste s'il est possible qu'un individu quelconque appartienne à un concept donné ;

Instanciation teste si un individu donné appartient à un concept donné ;

Subsomption teste si un concept donné est inclus dans un autre concept donné.

2.2 ASP : sémantique des modèles stables

Un *programme logique défini* est un ensemble de règles de la forme :

$$b \leftarrow a_1, \ldots, a_n. \qquad (n \geq 0)$$

Pour une telle règle r, $tête(r) = b$ est un atome appelé *tête* et $corps(r) = \{a_1, \ldots, a_n\}$ est un ensemble d'atomes appelé *corps*. La règle $b \leftarrow a.$ peut se lire : « si l'on peut prouver a, alors on conclut b ». Étant donnée une règle r et un ensemble d'atomes A, on dit que r est *applicable* (ou *déclenchable*) dans A si $corps(r) \subseteq A$. Un ensemble d'atomes A est dit *clos* par rapport à un programme P si et seulement si pour toute règle r de P, $corps(r) \subseteq A \Rightarrow tête(r) \in A$. On appelle $Cn(P)$, ou *modèle de Herbrand*, le plus petit ensemble d'atomes clos par rapport au programme P. Pour un programme P et un ensemble d'atomes A, l'opérateur T_P défini par :

$$T_P(A) = \{tête(r) \,|r \in P, corps(r) \subseteq A\}$$

calcule l'ensemble des atomes déductibles à partir de P et A. À partir de cet opérateur, on définit la suite : $T_P^0 = T_P(\emptyset)$ et $T_P^{k+1} = T_P(T_P^k), \forall k \geq 0$. $Cn(P)$ est le plus petit point fixe de T_P et $Cn(P) = \bigcup_{k \geq 0} T_P^k$. Ainsi, $Cn(P)$ contient tous les atomes que l'on peut déduire du programme P. On dit qu'un programme logique défini est *enraciné* s'il peut être ordonné selon la suite $\langle r_1, \ldots, r_n \rangle$ telle que $\forall i,\ 1 \leq i \leq n$, r_i est applicable dans $\{tête(r_j) \mid 1 \leq j < i\}$.
Un *programme logique normal* est un ensemble de règles de la forme :

$$c \leftarrow a_1, \ldots, a_n, \mathbf{not}\ b_1, \ldots, \mathbf{not}\ b_m. (n \geq 0,\ m \geq 0)$$

Comme précédemment, les a_i, b_j et c sont des atomes et pour une telle règle r on note $corps^+(r) = \{a_1, \ldots, a_n\}$ (respectivement $corps^-(r) = \{b_1, \ldots, b_m\}$) son corps *positif* (respectivement *négatif*). En outre, $r^+ = c \leftarrow a_1, \ldots, a_n.$ désigne la *projection positive* de la règle r. Intuitivement, la règle $c \leftarrow a,\ not\ b.$ peut se lire : « si l'on peut prouver a et si l'on ne peut pas prouver b, alors on peut conclure c ». Les « **not** » du corps négatif s'interprètent donc comme des *négations par défaut* (appelées aussi *négations faibles*). La plupart des solveurs ASP imposent que chaque règle soit *saine*. Une règle est dite *saine* quand toute variable apparaissant dans la tête ou dans le corps négatif apparaît au moins une fois dans le corps positif. Soit une règle r et un ensemble d'atomes A. On dit que r est *applicable* (ou *déclenchable*) dans A si $corps^+(r) \subseteq A$ et $corps^-(r) \cap A = \emptyset$.

Définition 2.1 *Le* réduit *d'un programme logique normal P par rapport à un ensemble d'atomes A est le programme logique défini : $P^A = \{r^+ \,|r \in P, corps^-(r) \cap A = \emptyset\}$.*

La sémantique originelle [5] des programmes logiques normaux est celle des modèles stables rappelée ci-après.

Définition 2.2 *Soit P un programme logique normal et S un ensemble d'atomes. S est un* modèle stable *de P si et seulement si $S = Cn(P^S)$.*

Pour des besoins de représentation des connaissances on peut être amené à utiliser conjointement la « vraie » négation (appelée aussi *négation forte*) et la négation par défaut au sein d'un même programme. Ceci est rendu possible par les *programmes logiques étendus* [6], où l'on autorise les littéraux à la place des atomes dans l'écriture des règles. Mais si, comme c'est notre cas, on refuse les modèles inconsistants alors la sémantique associée à ces programmes est réductible à celle des modèles stables en prenant en compte les considérations suivantes :
– tout littéral $\neg x$ est codé par l'atome nx,
– on ajoute une règle $\bot \leftarrow x,\ nx.$ pour tout atome x,
– un modèle stable ne doit pas contenir le symbole $\bot$.
Les règles de type $\bot \leftarrow x,\ nx.$, parfois simplement notées $\leftarrow x,\ nx.$ (sans tête), sont appelées des *contraintes*. L'ensemble des contraintes d'un programme P est noté P_K. L'ajout des contraintes induit le comportement attendu, à savoir qu'il est impossible d'obtenir un modèle stable contenant à la fois x et nx. Ainsi, seuls les modèles stables consistants sont conservés. Cette manière de faire est celle implantée dans tous les solveurs disponibles qui sont donc capables de traiter les programmes logiques étendus tout en restant dans le cadre des programmes logiques normaux.

3 Traduction de DL en ASP

Dans cette section, nous nous intéressons à la traduction proprement dite d'une logique de description vers ASP avec pour but futur de fusionner les niveaux *Ontologies* et *Règles* de la pile du web sémantique. Nous traduisons les ontologies exprimées par le biais de la logique $\mathcal{SHOIQ}$.

3.1 Bases de la traduction

La méthode que nous suivons est de traduire chaque axiome utilisé dans l'ontologie représentant le problème par des règles ASP. Pour cela, nous utilisons le principe, proposé dans [7], qui consiste à découper chaque axiome en sous-axiomes jusqu'à obtenir une décomposition en axiomes faisant intervenir uniquement des individus, des concepts et des rôles appartenant au vocabulaire.

Concernant les informations élémentaires, on utilise la même méthode que dans [1] et [7] : les individus seront représentés par des constantes, les concepts par des prédicats d'arité 1 et les rôles par des prédicats d'arité 2. Ainsi, si a et b sont des individus, C un concept et R un rôle, l'appartenance de l'individu a au concept C est traduit par la règle $c(a) \leftarrow$. et l'appartenance des individus a et b au rôle R se traduit par $r(a, b) \leftarrow$.

Il s'agit ensuite d'étudier le codage des opérateurs (comme par exemple l'inclusion) qui permettent d'exprimer des concepts ou des rôles plus complexes. L'idée de l'approche développée par Heymans et Vermeir [7] est de raisonner par contraintes : l'ensemble de toutes les combinaisons possibles est généré. Pour chaque individu et pour chaque concept, on génère deux types d'answer sets : l'un dans lequel l'individu est instance du concept et l'autre dans lequel l'individu est instance du complément du concept. Il en va de même pour deux individus et un rôle. Les définitions de concepts sont exprimées sous la forme de contraintes en explicitant les informations qu'on ne veut pas à l'aide de négations faibles. Les answer sets ne vérifiant pas ces contraintes sont supprimés. Finalement, il ne reste que les answer sets compatibles avec l'ontologie. Les informations déductibles de l'ontologie sont celles communes à l'ensemble des answer sets.

Le problème de l'approche par contraintes est qu'elle fait deux hypothèses trop fortes quand il s'agit de représenter des informations issues des logiques de description, notamment celles utilisées pour le Web sémantique.

D'une part, on doit supposer qu'une constante représente de manière unique un individu. Deux constantes différentes ne peuvent pas représenter le même individu. C'est l'hypothèse des noms uniques (Unique Name Assumption). On ne peut donc pas représenter l'axiome d'égalité ($a = b$), très naturel lorsqu'on traite des informations du Web sémantique, surtout dans le cas de la fusion d'ontologies.

D'autre part, on admet que si on ne peut pas prouver qu'une information est vraie, c'est qu'elle est fausse. C'est l'hypothèse du monde clos (Closed World Assumption), liée à l'utilisation des négations faibles dans les règles du programme logique. Or les logiques de description font l'hypothèse du monde ouvert (elles ne se prononcent pas lorsqu'un fait n'a pas pu être prouvé). Dans l'approche par contrainte, suivant l'hypothèse du monde ouvert, il y a potentiellement une infinité d'individus qui peuvent appartenir à une infinité de concepts. Or, pour que les contraintes fonctionnent, il faut absolument que l'ensemble des solutions possibles soit généré, ce qui est impossible car celui-ci est infini.

De plus, un autre problème de la solution proposée dans [7] est que le concept $\top$ n'est pas décrit dans cette traduction. Or, celui-ci est présent dans toutes les logiques de description.

Cette approche par contraintes présente donc des lacunes pour la représentation des nombreuses ontologies.

Notons au passage que l'approche de Eiter et al. [4] n'a pas ces défauts puisque la logique de description est gérée par un solveur spécialement conçu pour le raisonnement sur ces logiques mais elle impose d'utiliser deux solveurs distincts alors que nous voulons tout intégrer en ASP.

Pour pallier aux problèmes liés à l'approche par contraintes, il est préférable de ne générer que les connaissances dont on est sûr qu'elles sont vraies afin de respecter l'hypothèse du monde ouvert. C'est pourquoi nous proposons une approche dite par règles dont le principe est de ne déduire des informations que si on a prouvé ou réfuté un ensemble d'informations. La seule négation apparaissant ainsi dans les règles sera une négation forte.

Pour traduire les axiomes de la logique de description en règles ASP, on procède en deux étapes : les axiomes de la logique de description sont traduits en formules de la logique classique sous la forme de conjonction d'implications puis ces implications sont traduites en règles ASP.

Utiliser des règles avec des négations fortes plutôt que des négations faibles semble donc être une bonne approche pour la traduction en ASP : de nouveaux faits sont déduits seulement si d'autres faits ont déjà été prouvés ou réfutés, et aucun fait n'est supposé vrai ou faux.

3.2 Approche par règles

Concept $\top$

Tous les individus doivent appartenir au concept $\top$. Par conséquent, pour chaque individu a de l'ontologie, on ajoute le fait : $\top(a) \leftarrow$.

Aucun individu a ne pourra appartenir au concept $\neg\top$, car un answer set contiendra nécessairement $\top(a)$ et ne pourra donc pas contenir à la fois $\top(a)$ et $\neg\top(a)$ (puisqu'on refuse les modèles inconsistants).

Égalité

L'égalité permet d'indiquer que deux individus sont équivalents, c'est-à-dire qu'ils représentent la même notion. Si le prédicat $=$ existe bien en ASP, il n'est vrai que si les deux arguments représentent la même constante, par le biais de la constante elle-même ou bien d'une variable l'ayant pour valeur. En effet, en ASP, on suppose qu'une constante a un et un seul nom : deux constantes sont équivalentes si elles ont le même nom et sont différentes sinon. Ce n'est pas le cas en logique de description : deux noms différents d'individus peuvent être équivalents et donc représenter la même notion. Il faut donc indiquer que deux constantes différentes peuvent être équivalentes. Pour cela, on définit un nouveau prédicat d'équivalence qui représente cette égalité. Ce prédicat d'arité 2 se nomme $sameAs$. Il s'utilise comme n'importe quel rôle pour indi-

quer que deux individus sont identiques ou différents. Une relation d'équivalence est *réflexive*, *symétrique* et *transitive*. La réflexivité du rôle $sameAs$ s'exprime en logique classique : $\forall x,\ sameAs(x,x)$. La traduction simple de cette formule en ASP est donc : $sameAs(X,X) \leftarrow$. On doit encore rendre cette règle saine, et comme le prédicat $\top(X)$ est vrai pour tous les individus X de l'ontologie, on écrira la règle : $sameAs(X,X) \leftarrow \top(X)$.

La symétrie du rôle $sameAs$ s'exprime en logique classique : $\forall x,y,\ sameAs(x,y) \Rightarrow sameAs(y,x)$ et se traduit en ASP par les règles :

$$sameAs(X,Y) \leftarrow sameAs(Y,X).$$
$$\neg sameAs(X,Y) \leftarrow \neg sameAs(Y,X).$$

La transitivité du rôle $sameAs$ s'exprime en logique classique :

$$\forall x,y,z,\ \neg sameAs(x,z) \wedge sameAs(y,z) \Rightarrow \neg sameAs(x,y)$$
$$\forall x,y,z,\ \neg sameAs(x,z) \wedge sameAs(x,y) \Rightarrow \neg sameAs(y,z)$$

Ce qui donne les règles ASP suivantes :

$$sameAs(X,Z) \leftarrow sameAs(X,Y), sameAs(Y,Z).$$
$$\neg sameAs(X,Y) \leftarrow \neg sameAs(X,Z), sameAs(Y,Z).$$
$$\neg sameAs(Y,Z) \leftarrow \neg sameAs(X,Z), sameAs(X,Y).$$

En plus de cela, il faut exprimer les conséquences de l'équivalence : si deux individus sont équivalents, alors si l'un appartient à un concept, l'autre doit aussi lui appartenir. De même, si deux individus sont équivalents, alors si l'un d'eux intervient dans un rôle, l'autre doit y intervenir de la même manière. Pour les concepts, on a donc la formule logique : $\forall x,y,\ c(y) \wedge sameAs(x,y) \Rightarrow c(x)$

Cette formule peut également s'exprimer :

$$\forall x,y,\ \neg c(x) \wedge sameAs(x,y) \Rightarrow \neg c(y)$$
$$\forall x,y,\ \neg c(x) \wedge c(y) \Rightarrow \neg sameAs(x,y)$$

On obtient les règles ASP :

$$c(X) \leftarrow c(Y), sameAs(X,Y).$$
$$\neg c(Y) \leftarrow \neg c(X), sameAs(X,Y).$$
$$\neg sameAs(X,Y) \leftarrow \neg c(X), c(Y).$$

Pour les rôles, on a la formule logique : $\forall u,v,x,y,$
$r(u,v) \wedge sameAs(x,u) \wedge sameAs(y,v) \Rightarrow r(x,y)$
qui nous donne les implications : $\forall u,v,x,y,$

$$\neg r(x,y) \wedge sameAs(x,u) \wedge sameAs(y,v) \Rightarrow \neg r(u,v)$$
$$\neg r(x,y), r(u,v) \wedge sameAs(y,v) \Rightarrow \neg sameAs(x,u)$$
$$\neg r(x,y), r(u,v) \wedge sameAs(x,u) \Rightarrow \neg sameAs(y,v)$$

et les règles ASP :

$$r(X,Y) \leftarrow r(U,V), sameAs(X,U), sameAs(Y,V).$$
$$\neg r(U,V) \leftarrow \neg r(X,Y), sameAs(X,U), sameAs(Y,V).$$
$$\neg sameAs(X,U) \leftarrow \neg r(X,Y), r(U,V), sameAs(Y,V).$$
$$\neg sameAs(Y,V) \leftarrow \neg r(X,Y), r(U,V), sameAs(X,U).$$

Autres traductions

Nous donnons maintenant les autres opérateurs de logique de description pour lesquels nous pouvons utiliser l'approche par règles pour effectuer la traduction.

La subsomption de concepts est typiquement représentée par l'axiome $C \sqsubseteq D$ où C et D sont des concepts. Cet axiome s'exprime en logique classique $\forall x, C(x) \Rightarrow D(x)$, qui peut se réécrire $\forall x, \neg D(x) \Rightarrow \neg C(x)$. On peut donc en déduire les règles suivantes qui traduisent cet axiome :

$$d(X) \leftarrow c(X).$$
$$\neg c(X) \leftarrow \neg d(X).$$

On utilise le même raisonnement pour traduire la subsomption de rôles $R \sqsubseteq S$:

$$s(X,Y) \leftarrow r(X,Y).$$
$$\neg r(X,Y) \leftarrow \neg s(X,Y).$$

La traduction de l'équivalence de concepts et de l'équivalence de rôles consiste simplement à exprimer une double subsomption. L'axiome $C \equiv \neg D$ signifie que D est le complément de C. Cet axiome s'exprime en logique classique $\forall x, C(x) \Leftrightarrow \neg D(x)$, ce qui donne les règles :

$$\neg d(X) \leftarrow c(X).$$
$$\neg c(X) \leftarrow d(X).$$
$$d(X) \leftarrow \neg c(X).$$
$$c(X) \leftarrow \neg d(X).$$

L'inversion de rôle exprimée par l'axiome $R \equiv S^-$ s'exprime en logique classique $\forall x,y, R(x,y) \Leftrightarrow S(y,x)$, ce qui donne les règles :

$$s(Y,X) \leftarrow r(X,Y).$$
$$\neg r(X,Y) \leftarrow \neg s(Y,X).$$
$$\neg s(Y,X) \leftarrow \neg r(X,Y).$$
$$r(X,Y) \leftarrow s(Y,X).$$

Pour la transitivité d'un rôle exprimée par l'axiome $\mathrm{Trans}(R)$, on utilise le même raisonnement que pour la transitivité de $sameAs$, ce qui donne les règles :

$$r(X,Z) \leftarrow r(X,Y), r(Y,Z).$$
$$\neg r(X,Y) \leftarrow \neg r(X,Z), r(Y,Z).$$
$$\neg r(Y,Z) \leftarrow \neg r(X,Z), r(X,Y).$$

3.3 Limites de l'approche par règles

Certains axiomes de la logique de description représentent des disjonctions. Ainsi, l'axiome $C \equiv D \sqcup E$ signifie que tout individu appartenant à C appartient à D **ou** à E. Or, il est impossible d'exprimer des disjonctions en tête de règle dans les programmes logiques normaux. On peut néanmoins déduire des connaissances d'une disjonction, même si on est incapable de l'exprimer. Pour ce faire, on utilise le mécanisme d'answer set d'ASP : pour chaque littéral d'une disjonction, on génère une catégorie d'answer sets dans laquelle ce littéral sera vrai et une autre catégorie dans lequel ce littéral sera faux. On interdit cependant d'avoir un answer set dans lequel tous les littéraux de la disjonction sont faux. Pour cela, on suit le principe utilisé dans l'approche par contraintes, ce qui va introduire en plus des négations faibles. La différence avec l'approche de [7] est que les individus concernés par cette génération d'answer sets ne sont pas l'ensemble des individus de l'ontologie, mais uniquement ceux dont on connaît l'appartenance à un concept particulier. Avec l'axiome $C \equiv D \sqcup E$, on générera des answers sets uniquement pour les individus appartenant au concept C.

3.4 Représentation de l'union de concepts, de l'intersection de concepts et des concepts nominaux

Lors de la traduction des axiomes de la logique $\mathcal{SHOIQ}$, trois types d'opérations introduisent une disjonction en

tête de règle : l'union et l'intersection de concepts et les concepts nominaux.

En reprenant l'axiome $C \equiv D \sqcup E$, qui s'exprime en logique classique $\forall x, C(x) \Leftrightarrow D(x) \vee E(x)$, on traduit chacune des deux implications représentées par ces équivalences. Pour $\forall x, C(x) \Leftarrow D(x) \vee E(x)$, qui peut également s'exprimer comme $\forall x, D(x) \Rightarrow C(x)$ et $\forall x, E(x) \Rightarrow C(x)$, on utilise le même raisonnement que précédemment, ce qui nous permet d'obtenir les règles suivantes :

$$\begin{aligned} c(X) &\leftarrow d(X). \\ \neg d(X) &\leftarrow \neg c(X). \\ c(X) &\leftarrow e(X). \\ \neg e(X) &\leftarrow \neg c(X). \end{aligned}$$

Pour traduire l'implication $\forall x, C(x) \Rightarrow D(x) \vee E(x)$, on génère une catégorie d'answer sets dans laquelle $d(X)$ est vrai et une autre dans laquelle $\neg d(X)$ l'est. Pour cela, on utilise les règles suivantes :

$$\begin{aligned} d(X) &\leftarrow c(X), \mathbf{not}\ \neg d(X). \\ \neg d(X) &\leftarrow c(X), \mathbf{not}\ d(X). \end{aligned}$$

On réalise la même opération pour $E(x)$. Reste à exprimer la contraposée $\forall x, \neg D(x) \wedge \neg E(x) \Rightarrow \neg C(x)$:
$\neg c(X) \leftarrow \neg d(X), \neg e(X)$.

Cette règle permet par ailleurs de supprimer les answer sets dans lesquels on aurait eu $\neg d(X)$ et $\neg e(X)$ bien qu'on ait $c(X)$. En étendant ce principe à une disjonction de n littéraux, on obtient les règles suivantes :

$$\begin{aligned} \neg c(X) &\leftarrow \neg d_1(X), \ldots, \neg d_n(X). \\ \forall\, 1 \leq i \leq n, & \\ d_i(X) &\leftarrow c(X), \mathbf{not}\ \neg d_i(X). \\ \neg d_i(X) &\leftarrow c(X), \mathbf{not}\ d_i(X). \\ c(X) &\leftarrow d_i(X). \\ \neg d_i(X) &\leftarrow \neg c(X). \end{aligned}$$

Pour l'intersection de concepts, on se sert du fait que $C \equiv D \sqcap E \Leftrightarrow \neg C \equiv \neg D \sqcup \neg E$. L'axiome $C \equiv D_1 \sqcap \cdots \sqcap D_n$ se traduit donc :

$$\begin{aligned} c(X) &\leftarrow d_1(X), \ldots, d_n(X). \\ \forall\, 1 \leq i \leq n, & \\ d_i(X) &\leftarrow \neg c(X), \mathbf{not}\ \neg d_i(X). \\ \neg d_i(X) &\leftarrow \neg c(X), \mathbf{not}\ d_i(X). \\ \neg c(X) &\leftarrow \neg d_i(X). \\ d_i(X) &\leftarrow c(X). \end{aligned}$$

Pour les concepts nominaux, l'axiome $C \equiv \{a_1, \ldots, a_n\}$ s'exprime en logique classique $\forall x, C(x) \Leftrightarrow x = a_1 \vee \cdots \vee x = a_n$. En utilisant le même raisonnement que pour l'union de concepts, on obtient les règles suivantes :

$$\begin{aligned} \neg c(X) &\leftarrow \neg sameAs(X, a_1), \ldots, \neg sameAs(X, a_n). \\ \forall\, 1 \leq i \leq n, & \\ sameAs(X, a_i) &\leftarrow c(X), \mathbf{not}\ \neg sameAs(X, a_i). \\ \neg sameAs(X, a_i) &\leftarrow c(X), \mathbf{not}\ \neg sameAs(X, a_i). \\ c(X) &\leftarrow sameAs(X, a_i). \\ \neg sameAs(X, a_i) &\leftarrow \neg c(X). \end{aligned}$$

3.5 Skolémisation

Les opérateurs décrits précédemment sont tous basés sur une quantification universelle. La gestion de la quantification existentielle nécessite une traduction moins simple basée sur la skolémisation.

Principe de la skolémisation

Jusqu'ici, notre démarche de traduction consistait à exprimer en règles ASP des formules de la logique classique sous forme prénexe représentant les différents axiomes et opérateurs de la logique de description. Pour l'instant, tous les quantificateurs de ces formules sont des quantificateurs universels. En ASP, on exprime justement des règles qui sont valables pour l'ensemble des valeurs représentées par les variables. Cependant, l'expression des différentes restrictions de la logique de description en logique classique nécessite l'utilisation de quantificateurs existentiels. Pour exprimer des variables quantifiées existentiellement en ASP, on réalise une opération de skolémisation. La skolémisation consiste à transformer une formule de la logique classique en conservant les quantificateurs universels et en remplaçant chaque variable existentielle par un fonction, dite de Skolem, ayant pour paramètres les variables universelles dont le quantificateur est avant le quantificateur existentiel de la variable. Chaque fonction de Skolem attribue ainsi à un ensemble de variables une constante de Skolem.

Traduction des restrictions existentielles et universelles

Soit l'axiome $C \equiv \exists R.D$ représentant une restriction existentielle. Cet axiome s'exprime en logique classique comme étant : $\forall x, C(x) \Leftrightarrow [\exists y, R(x, y) \wedge D(y)]$. La traduction de l'implication $\forall x, C(x) \Leftarrow [\exists y, R(x, y) \wedge D(y)]$ qui peut également s'exprimer sous la forme $\forall x, y, R(x, y) \wedge D(y) \Rightarrow C(x)$ se traduit comme les formules vues jusqu'à présent :

$$\begin{aligned} c(X) &\leftarrow r(X, Y), d(Y). \\ \neg r(X, Y) &\leftarrow \neg c(X), d(Y). \\ \neg d(Y) &\leftarrow \neg c(X), r(X, Y). \end{aligned}$$

L'autre implication $\forall x, \exists y, C(x) \Rightarrow R(x, y) \wedge D(y)$ possède un quantificateur existentiel. Nous allons donc skolémiser cette formule. Pour cela, nous utilisons une fonction de Skolem ψ_r sur le rôle R et le concept D. La formule précédente se réécrit alors $\forall x, C(x) \Rightarrow R(x, \psi_r(x)) \wedge D(\psi_r(x))$. On peut alors appliquer la méthode classique pour obtenir les règles :

$$\begin{aligned} r(X, \psi_r(X)) &\leftarrow c(X). \\ d(\psi_r(X)) &\leftarrow c(X). \end{aligned}$$

Il reste à exprimer deux propriétés des constantes de Skolem : elles représentent des individus de l'ontologie et si deux variables sont équivalentes, alors chaque fonction de Skolem leur attribue des constantes de Skolem équivalentes :

$$\begin{aligned} \top(\psi_r(X)) &\leftarrow c(X). \\ sameAs(\psi_r(X), \psi_r(Y)) &\leftarrow c(X), c(Y), sameAs(X, Y). \end{aligned}$$

Les prémisses de la contraposée $\forall x, \neg R(x, \psi_r(x)) \vee \neg D(\psi_r(x)) \Rightarrow \neg C(x)$ utilisent des constantes de Skolem. Or, pour que ces constantes aient été définies, il faut forcément qu'on ait déduit $C(x)$, puisque c'est le seul moyen de les définir. Autrement dit, cette règle signifie que

si, pour un individu a, on suppose $C(a)$, alors si cela aboutit à une contradiction, on déduit $\neg C(a)$. On réalise ici un raisonnement par l'absurde. Pour représenter ce raisonnement en ASP, on génère une fois de plus deux catégories d'answer sets, pour chaque individu a de l'ontologie : l'une où on suppose que a appartient au concept C et l'autre où on suppose que a appartient au complément de C, $\neg C$. On ajoute donc les règles suivantes :

$$c(X) \leftarrow \top(X), \mathbf{not}\ \neg c(X).$$
$$\neg c(X) \leftarrow \top(X), \mathbf{not}\ c(X).$$

Ainsi, en prenant les informations communes à chaque answer set, on sera capable de déduire si l'individu a n'appartient pas au concept C. En effet, si dans les answer sets dans lesquels on a $a : C$, on aboutit à une contradiction, ces answer sets seront supprimés et seuls resteront les answer sets dans lesquels on a $a : \neg C$. Cependant, un nouveau problème se pose : on crée ici une nouvelle constante de Skolem pour chaque individu de l'ontologie. Or, les constantes de Skolem ont elles-mêmes été déclarées comme étant des individus de l'ontologie. Par conséquent, on est en mesure de créer une nouvelle constante de Skolem pour chaque constante de Skolem. Les fonctions de Skolem risquent alors de s'imbriquer à l'infini. Pour pallier à ce problème, on déclare explicitement que chaque constante de Skolem en est une grâce à un nouveau prédicat d'arité 1 $skolem$ qui sera vrai pour X si et seulement si X est une constante de Skolem. On ajoute alors la règle suivante : $skolem(X) \leftarrow c(X)$.

De plus, on modifie les deux règles générant les answer sets nécessaires au raisonnement par l'absurde pour qu'elles ne prennent pas en compte les constantes de Skolem :

$$c(X) \leftarrow \top(X), \mathbf{not}\ \neg c(X), \mathbf{not}\ skolem(X).$$
$$\neg c(X) \leftarrow \top(X), \mathbf{not}\ c(X), \mathbf{not}\ skolem(X).$$

Finalement, l'axiome $C \equiv \exists R.D$ se traduit par l'ensemble de règles suivant :

$$c(X) \leftarrow r(X,Y), d(Y).$$
$$\neg r(X,Y) \leftarrow \neg c(X), d(Y).$$
$$\neg d(Y) \leftarrow \neg c(X), r(X,Y).$$
$$c(X) \leftarrow \top(X), \mathbf{not}\ \neg c(X), \mathbf{not}\ skolem(X).$$
$$\neg c(X) \leftarrow \top(X), \mathbf{not}\ c(X), \mathbf{not}\ skolem(X).$$
$$skolem(\psi_r(X)) \leftarrow c(X).$$
$$r(X, \psi_r(X)) \leftarrow c(X).$$
$$d(\psi_r(X)) \leftarrow c(X).$$
$$\top(\psi_r(X)) \leftarrow c(X).$$
$$sameAs(\psi_r(X), \psi_r(Y)) \leftarrow c(X), c(Y), sameAs(X,Y).$$

D'autre part, pour représenter l'axiome $C \equiv \forall R.D$, on se sert du fait que $\forall R.D \equiv \neg(\exists R.\neg D)$:

$$\neg c(X) \leftarrow r(X,Y), \neg d(Y).$$
$$\neg r(X,Y) \leftarrow c(X), \neg d(Y).$$
$$d(Y) \leftarrow c(X), r(X,Y).$$
$$\neg c(X) \leftarrow \top(X), \mathbf{not}\ c(X), \mathbf{not}\ skolem(X).$$
$$c(X) \leftarrow \top(X), \mathbf{not}\ \neg c(X), \mathbf{not}\ skolem(X).$$
$$skolem(\psi_r(X)) \leftarrow \neg c(X).$$
$$r(X, \psi_r(X)) \leftarrow \neg c(X).$$
$$\neg d(\psi_r(X)) \leftarrow \neg c(X).$$
$$\top(\psi_r(X)) \leftarrow \neg c(X).$$
$$sameAs(\psi_r(X), \psi_r(Y)) \leftarrow c(X), c(Y), sameAs(X,Y).$$

Gestion des cardinalités

Les restrictions cardinales sont similaires aux restrictions existentielles. Là où les restrictions existentielles imposent l'existence d'un élément, les restrictions cardinales minimales permettent de spécifier le nombre minimal d'éléments souhaités. En réalité, pour un rôle R et un concept C, le concept $\exists R.C$ est complètement équivalent au concept $\geq 1R.C$. Par conséquent, pour traduire l'axiome de la logique de description $C \equiv \geq nR.D$ pour $n \in \mathbb{N}$, il faut appliquer le même raisonnement que précédemment, mais avec n existentielles. La traduction de cet axiome nécessite donc de créer n constantes de Skolem, toutes différentes. On utilise les mêmes règles ASP que précédemment pour créer chacune de ces constantes :

$$skolem(\psi_{r,i}(X)) \leftarrow c(X).$$
$$r(X, \psi_{r,i}(X)) \leftarrow c(X).$$
$$d(\psi_{r,i}(X)) \leftarrow c(X).$$
$$\top(\psi_{r,i}(X)) \leftarrow c(X).$$
$$sameAs(\psi_{r,i}(X), \psi_{r,i}(Y)) \leftarrow c(X), c(Y), sameAs(X,Y).$$

pour tout i tel que $1 \leq i \leq n$ avec $\psi_{r,i}(X)$ une fonction de Skolem. Il reste à préciser que toutes les constantes de Skolem doivent être différentes deux à deux :

$$\neg sameAs(\psi_{r,i}(X), \psi_{r,j}(X)) \leftarrow c(X).$$

pour tout i, j tels que $1 \leq i < j \leq n$.

Les autres règles sont les mêmes que pour la traduction de la restriction existentielle, mais en raisonnant avec n éléments, en précisant bien qu'ils sont tous différents :

$$c(X) \leftarrow r(X,Y_1), d(Y_1), \ldots, r(X,Y_n), d(Y_n), \neg sameAs(Y_1,Y_2), \ldots, \neg sameAs(Y_{n-1},Y_n).$$
$$\neg r(X,Y_i) \leftarrow \neg c(X), r(X,Y_1), d(Y_1), \ldots, d(Y_i), \ldots, r(X,Y_n), d(Y_n), \neg sameAs(Y_1,Y_2), \ldots, \neg sameAs(Y_{n-1},Y_n).$$
$$\neg d(Y_i) \leftarrow \neg c(X), r(X,Y_1), d(Y_1), \ldots, r(X,Y_i), \ldots, r(X,Y_n), d(Y_n), \neg sameAs(Y_1,Y_2), \ldots, \neg sameAs(Y_{n-1},Y_n).$$
$$c(X) \leftarrow \top(X), \mathbf{not}\ \neg c(X), \mathbf{not}\ skolem(X).$$
$$\neg c(X) \leftarrow \top(X), \mathbf{not}\ c(X), \mathbf{not}\ skolem(X).$$

Pour traduire les restrictions cardinales maximales, on se sert du fait que $\leq nR.C \equiv \neg(\geq(n{+}1)R.C)$ pour $n \in \mathbb{N}$, R un rôle et C un concept. Exprimer $C \equiv \leq nR.D$ revient donc à exprimer $\neg C \equiv \geq (n{+}1)R.D$, ce qui donne les règles suivantes :

$$\neg c(X) \leftarrow r(X,Y_1), d(Y_1), \ldots, r(X,Y_{n+1}), d(Y_{n+1}), \neg sameAs(Y_1,Y_2), \ldots, \neg sameAs(Y_n,Y_{n+1}).$$
$$\neg r(X,Y_i) \leftarrow c(X), r(X,Y_1), d(Y_1), \ldots, d(Y_i), \ldots, r(X,Y_{n+1}), d(Y_{n+1}), \neg sameAs(Y_1,Y_2), \ldots, \neg sameAs(Y_n,Y_{n+1}).$$
$$\neg d(Y_i) \leftarrow c(X), r(X,Y_1), d(Y_1), \ldots, r(X,Y_i), \ldots, r(X,Y_{n+1}), d(Y_{n+1}), \neg sameAs(Y_1,Y_2), \ldots, \neg sameAs(Y_n,Y_{n+1}).$$
$$\neg c(X) \leftarrow \top(X), \mathbf{not}\ c(X), \mathbf{not}\ skolem(X).$$
$$c(X) \leftarrow \top(X), \mathbf{not}\ \neg c(X), \mathbf{not}\ skolem(X).$$

$\forall\, 1 \leq i \leq n{+}1,$

$$skolem(\psi r, i(X)) \leftarrow \neg c(X).$$
$$r(X, \psi_{r,i}(X)) \leftarrow \neg c(X).$$
$$d(\psi_{r,i}(X)) \leftarrow \neg c(X).$$
$$\top(\psi_{r,i}(X)) \leftarrow \neg c(X).$$
$$sameAs(\psi_{r,i}(X), \psi_{r,i}(Y)) \leftarrow c(X), c(Y), sameAs(X, Y).$$

$\forall\, 1 \leq i < j \leq n{+}1,$

$$\neg sameAs(\psi_{r,i}(X), \psi_{r,j}(X)) \leftarrow \neg c(X).$$

Enfin, pour traduire la restriction cardinale exacte, on se sert du fait que $=nR.C \equiv \geq nR.C \sqcap \leq nR.C$. En combinant les règles précédentes, on obtient alors :

$$c(X) \leftarrow r(X, Y_1), d(Y_1), \ldots, r(X, Y_n), d(Y_n), \neg sameAs(Y_1, Y_2), \ldots, \neg sameAs(Y_{n-1}, Y_n).$$
$$\neg c(X) \leftarrow r(X, Y_1), d(Y_1), \ldots, r(X, Y_{n+1}), d(Y_{n+1}), \neg sameAs(Y_1, Y_2), \ldots, \neg sameAs(Y_n, Y_{n+1}).$$
$$\neg r(X, Y_i) \leftarrow \neg c(X), r(X, Y_1), d(Y_1), \ldots, d(Y_i), \ldots, r(X, Y_n), d(Y_n), \neg sameAs(Y_1, Y_2), \ldots, \neg sameAs(Y_{n-1}, Y_n).$$
$$\neg r(X, Y_i) \leftarrow c(X), r(X, Y_1), d(Y_1), \ldots, d(Y_i), \ldots, r(X, Y_{n+1}), d(Y_{n+1}), \neg sameAs(Y_1, Y_2), \ldots, \neg sameAs(Y_n, Y_{n+1}).$$
$$\neg d(Y_i) \leftarrow \neg c(X), r(X, Y_1), d(Y_1), \ldots, r(X, Y_i), \ldots, r(X, Y_n), d(Y_n), \neg sameAs(Y_1, Y_2), \ldots, \neg sameAs(Y_{n-1}, Y_n).$$
$$\neg d(Y_i) \leftarrow c(X), r(X, Y_1), d(Y_1), \ldots, r(X, Y_i), \ldots, r(X, Y_{n+1}), d(Y_{n+1}), \neg sameAs(Y_1, Y_2), \ldots, \neg sameAs(Y_n, Y_{n+1}).$$
$$c(X) \leftarrow \top(X), \mathbf{not}\ \neg c(X), \mathbf{not}\ skolem(X).$$
$$\neg c(X) \leftarrow \top(X), \mathbf{not}\ c(X), \mathbf{not}\ skolem(X).$$

$\forall\, 1 \leq i \leq n,$

$$skolem(\psi_{r,i}(X)) \leftarrow c(X).$$
$$r(X, \psi_{r,i}(X)) \leftarrow c(X).$$
$$d(\psi_{r,i}(X)) \leftarrow c(X).$$
$$\top(\psi_{r,i}(X)) \leftarrow c(X).$$
$$sameAs(\psi_{r,i}(X), \psi_{r,i}(Y)) \leftarrow c(X), c(Y), sameAs(X, Y).$$

$\forall\, 1 \leq i < j \leq n,$

$$\neg sameAs(\psi_{r,i}(X), \psi_{r,j}(X)) \leftarrow c(X).$$

$\forall\, 1 \leq i \leq n{+}1,$

$$skolem(\psi_{r,i}(X)) \leftarrow \neg c(X).$$
$$r(X, \psi_{r,i}(X)) \leftarrow \neg c(X).$$
$$d(\psi_{r,i}(X)) \leftarrow \neg c(X).$$
$$\top(\psi_{r,i}(X)) \leftarrow \neg c(X).$$
$$sameAs(\psi_{r,i}(X), \psi_{r,i}(Y)) \leftarrow \neg c(X), \neg c(Y), sameAs(X, Y).$$

$\forall\, 1 \leq i < j \leq n{+}1,$

$$\neg sameAs(\psi_{r,i}(X), \psi_{r,j}(X)) \leftarrow \neg c(X).$$

4 Conclusion et perspectives

Nous avons proposé une méthode de traduction générale en ASP des ontologies exprimées dans la logique de description $\mathcal{SHOIQ}$. Pour cela, nous avons représenté les notions d'individus, de concepts et de rôles en un équivalent ASP. Pour traduire les descriptions de rôles ou de concepts complexes, nous les avons exprimées en logique classique, puis nous avons traduit ces formules de la logique classique en règles ASP. Lorsqu'aucune règle ASP ne pouvait traduire simplement la formule, nous avons utilisé d'autres moyens pour y parvenir, en représentant la disjonction dans un programme ASP et en utilisant des fonctions de Skolem. Faute de place, nous ne donnons pas ici le détail du tableau récapitulatif de traduction mais il peut être trouvé dans [3]. Traduire les logiques de description en règles ASP permet de fusionner facilement les niveaux *Ontologie* et *Règles* de la pile du Web sémantique. De plus, les règles ASP pouvant suivre selon leur définition l'hypothèse du monde clos ou bien celle du monde ouvert, les règles du Web sémantique exprimées en ASP seront d'autant plus souples. Pour représenter ces ontologies en ASP dans leur intégralité, une traduction des langages OWL [3] et RIF [4], généralement utilisés pour représenter respectivement le niveau *Ontologie* et le niveau *Règles*, doit être réalisée. La traduction présentée dans cet article en serait une étape intermédiaire. Il faudra cependant ajouter à celle-ci la gestion des types de données, de manière à représenter la logique $\mathcal{SHOIQ}(\mathbf{D})$, utilisée dans les ontologies du Web sémantique.

Il serait également possible de représenter le niveau *Interrogation* en règles ASP, en générant différents modèles stables selon les réponses possibles à l'interrogation. Pour représenter l'interrogation d'instanciation et de satisfiabilité d'un concept, il suffirait d'ajouter un nouveau concept à l'ontologie représentant le concept ajouté par l'interrogation. Pour l'interrogation de subsomption, il suffirait, comme pour les tests précédents, d'ajouter les deux nouveaux concepts à l'ontologie, puis d'ajouter un individu appartenant au concept subsumé.

Références

[1] Güray Alsaç and Chitta Baral. Reasoning in description logics using declarative logic programms. 2001.

[2] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. pages 34–43, 2001.

[3] Aymeric Le Dorze. *Answer Set Programming et logiques de description*. Rapport de M2 Recherche, LERIA, Université d'Angers, 2010.

[4] Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. *Artif. Intell.*, 172(12-13) :1495–1539, 2008.

[5] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080, 1988.

[6] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4) :365–386, 1991.

[7] Stijn Heymans and Dirk Vermeir. Integrating semantic web reasoning and answer set programming. In Marina De Vos and Alessandro Provetti, editors, *Answer Set Programming*, volume 78 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

3. http://www.w3.org/2007/OWL/wiki/OWL_Working_Group
4. http://www.w3.org/2005/rules/wiki/RIF_Working_Group