Traduction d'ontologies exprimées en OWL vers des programmes ASP

Délivrable - Projet ANR ASPIQ
Rapport de recherche - décembre 2014

Résumé

Ce rapport présente la démarche suivie pour effectuer la traduction d'une ontologie exprimée dans le format OWL vers un programme ASP traitable par le solveur ASPeRiX. Cette traduction est effectuée en deux étapes : la première étape consiste à traduite une base de connaissances exprimée en OWL vers une base de règles existentielles utilisant le format dlgp et la deuxième étape permet de traduire une base de règles existentielles exprimée en dlgp en un programme (ensemble de règles) ASP. La prise en compte des existentielles en ASP nécessite un traitement particulier pour obtenir un programme ASP standard. Dans ce rapport nous présentons le traitement de la traduction d'une base de connaissances exprimées en OWL vers un programme ASP en expliquant les aspects théoriques mais aussi en donnant les différents outils logiciels développés pour permettre cette traduction ainsi qu'un exemple complet de traduction.

1 Introduction

L'un des aspects du projet ASPIQ et de proposer un traitementd'informations ontologiques exprimées en OWL en utilisant l'ASP (programmation logique non-monotone) [4]. D'un côté, l'utilisation d'OWL permet d'exprimer des ontologies en respectant les standards de représentation utilisés dans le web et de récolter des benchmarks intéressants d'application réelles. D'un autre côté, l'utilisation d'ASP permet d'enrichir la formalisme de représentation d'ontologie en ajoutant un aspect non-monotone, utile par exemple pour le traitement des exceptions ou de problèmes combinatoires, et de fournir des solveurs efficaces pour traiter ces informations non-monotones.

Pour effectuer cette traduction d'ontologies nous devons traiter certains aspects du formalisme OWL qui ne sont pas disponibles en ASP comme par exemple la quantification existentielles de certaines variables. Néanmoins, il est intéressant de noter que le formalisme des règles existentielles [2] dont les propriétés logiques ont été étudiées en profondeur et qui couvre les profils traitables de OWL2 DL offre un point de passage entre OWL et ASP. Ainsi, la traduction d'ontologies exprimées en OWL vers un programme ASP se fait en deux étapes : la première étape consiste

à traduire les ontologies en une base de règles existentielles et la deuxième étape consiste à traduire la base de règles existentielles en règles ASP.

De plus, dans un souci d'implémentation, le travail n'a pas seulement consisté en une étude théorique mais a aussi concerné la mise en œuvre de la traduction avec plusieurs outils : un outil de traduction d'ontologies OWL en règles existentielles et un outil de traduction de règles existentielles en règles ASP intégré au solveur ASPeRiX [6, 7] permettant le traitement des variables existentielles dans les règles pour obtenir un programme ASP standard.

2 Règles existentielles

Dans cette section nous présentons le format des règles existentielles, nommé dlgp, puis nous présentons la traduction de OWL vers dlgp.

2.1 Format d'échange (dlgp)

Notre format d'échange est un format textuel. Il est appelé *dlgp* pour Datalog+. Ce format adopte la syntaxe usuelle des programmes Prolog ou Datalog. Une erreur à ne pas commettre cependant : les variables n'apparaissant qu'en tête de règle sont quantifiées existentiellement et pas universellement comme elles le seraient en Prolog ; Datalog ne permet pas de telles variables et les systèmes ASP les interdisent en général (condition de « safe rule » où une variable présente en tête de règle doit aussi être présente dans le corps positif de la règle).

Pour illustrer le format de représentation, les exemples ci-dessous donnent une base de connaissances en logique du premier ordre puis son encodage en dlgp.

Exemple 1. [Une base en logique du premier ordre]

Nous considérons les prédicats suivants dont on donne l'arité à la suite du nom : area/1, project/1, researcher/1, hasExpertise/2, isMember/2 et isProject/3. Les prédicats unaires peuvent être vus comme des classes de l'ontologie. Les prédicats binaires *hasExpertise* and *isMember* lient respectivement un chercheur à un domaine de recherche et à un projet de recherche. Le prédicat ternaire *isProject* lie un projet, le domaine du projet et le responsable de ce projet.

Les connaissances sont exprimées avec un fait et quatre règles (dont une contrainte et une règle d'égalité) dont on donne la signification en français puis la traduction en logique du premier ordre.

— Fait : *« Le chercheur* a *est membre d'un projet du domaine de la* représentation des connaissances *»*
$\exists X \exists Y (researcher(a) \wedge isMember(a, X) \wedge isProject(X, kr, Y))$

— Règle 1 : *« Chaque responsable d'un projet est membre de ce projet »*
$\forall X \forall Y \forall Z (isProject(X, Y, Z) \rightarrow isMember(Z, X))$

— Règle 2 : *« Tous les chercheurs experts dans un domaine sont membres d'un projet dans ce domaine »*
$\forall X \forall Y (researcher(X) \wedge hasExpertise(X, Y) \rightarrow \exists Z \exists L(isProject(Z, Y, L) \wedge isMember(X, Z)))$

— Règle 3 (Contrainte) *« Les classes chercheurs et projets sont disjointes »*
$\neg \exists X(researcher(X) \wedge project(X))$ *or*
$\forall X((researcher(X) \wedge project(X)) \rightarrow \bot)$

— Règle 4 (Règle d'égalité) *« Chaque projet a au plus un responsable »*
$\forall X \forall Y \forall Z((isProject(X, Y, Z) \wedge isProject(X, Y, Z')) \rightarrow Z = Z')$

Exemple 2. [La même base au format DLGP]

```
   % Fait
researcher(a), isMember(a, X), isProject(X, kr, Y).

   % Règles
% règle Datalog standard
[R1] ismember(Z,X) :- isProject(X, Y, Z).

   % règle Datalog étendue (L est quantifiée existentiellement)
[R2] isProject(Z, Y, L), isMember(X, Z) :-
                          researcher(X), hasExpertise(X, Y).

   % contrainte (! est le symbole pour « toujours faux »)
[R3] ! :- researcher(X), project(X).

   % règle d'égalité
[R4]Z1=Z2 :- isProject(X,Y,Z1), isProject(X,Y,Z2).
```

Un fichier dlgp peut également encoder des requêtes conjonctives, comme illustré par l'exemple ci-après.

Exemple 3. [Requêtes conjonctives]

Q_1 : « Trouver les membres des projets dans le domaine de la représentation des connaissances *»*
$Q_1 = \exists Y(isMember(X, Y) \wedge isProject(Y, kr, Z))$

Q_2 : « Y a-t-il un projet dans le domaine de la représentation des connaissances *? »*
$Q_2 = \exists X \exists Z isProject(X, kr, Z)$

```
% query Q1
? (X) :- isMember(X,Y), isProject(Y, kr, Z).

% query Q2 (boolean query)
? :- isProject(X,kr,Z).
```

Liens pour en savoir plus

— Introduction informelle basée sur des exemples :
`http://www2.lirmm.fr/~mugnier/graphik/kiabora/downloads/framework_en.pdf`
— Description précise du format :
`http://www2.lirmm.fr/~mugnier/graphik/kiabora/downloads/datalog-plus_en.pdf`

Outils associés

— **Parser :** le format dlgp est muni d'un analyseur (parser) écrit en java, dont le principe est de créeer des événements correspondants aux types de connaissances reconnus ; ces événements peuvent être écoutés de façon sélective et utilisés pour créer les objets associés.
— **Analyse d'une base de règles :** l'outil Kiabora permet entre autres de décomposer un ensemble de règles en règles à tête atomique. Kiabora est en ligne à l'adresse suivante :
`http://www2.lirmm.fr/~mugnier/graphik/kiabora/index.html`

2.2 De OWL à dlgp

Les ontologies disponibles sont généralement au format OWL. Nous avons développé un outil qui permet de traduire un fichier RDFS/OWL en dlgp. Cet outil ne traduit que les assertions OWL qui sont traduisibles en règles existentielles (sans chercher s'il est possible de reformuler des axiomes, qui seraient non traduisibles directement, sous une forme équivalente qui serait traduisible). Il traduit notamment toute base écrite en RDFS, ou dans l'un des profils traitables de OWL2 DL (OWL2 QL notamment).

Précisions techniques Le fichier à traduire est supposé être au format RDF/XML. A ce jour les axiomes pris en compte sont les suivants :

— subClassOf
— equivalentClass
— disjointWith
— subPropertyOf
— equivalentProperty
— propertyChainAxiom
— domain
— range
— TransitiveProperty
— IrreflexiveProperty
— SymmetricProperty
— AsymmetricProperty
— FunctionalProperty
— InverseFunctionalProperty

ainsi que les expressions complexes de classe, contenant les constructeurs suivants (et seulement ceux-ci) :

— intersectionOf
— someValuesFrom (avec une Classe ou avec owl :Thing)

— hasValue
— minCardinality à 1 uniquement
— inverseOf

Remarques
1) l'outil prend en compte OWL mais pas OWL2 (il y a des variations dans les mots-clés)
2) nous prévoyons de développer une nouvelle version de l'outil en se basant sur OWL API :
`http://owlapi.sourceforge.net/`.

Utilisation du traducteur Le traducteur OWL2dlgp est disponible à l'adresse suivante : `https://info-depot.lirmm.fr/repos/GraphIK/owl2dlp`

Le fichier README indique comment utiliser l'outil :
— Pour traduire une ontologie sur la sortie standard :
`java -jar owl2dlp.jar monOnto.owl`
— Pour mettre le résultat dans un fichier : `java -jar owl2dlp.jar monOnto.owl > monOnto.dlp`
— Pour mettre le résultat dans un fichier et les erreurs dans un autre :
`java -jar owl2dlp.jar monOnto.owl >& MonOnto.error.txt > monOnto.dlp`
— Pour lister les différents types d'erreur : `cat error.txt | sort -u`
— Pour augmenter la taille de la mémoire ajouter à java l'option -Xmxn où n est la taille, exemple : `java -Xmx512m -jar owl2dlp.jar monOnto.owl`

L'annexe donne un exemple de traduction à partir de l'ontologie "University" (U) [5].

3 ASP avec existentielles

On étudie maintenant la transformation d'une base de règles existentielles exprimée au format dlgp vers un programme ASP. Pour cela, on donne la syntaxe d'un ASP étendu permettant de prendre en compte les variables existentielles et d'autres particularités intéressantes du langage. Nous expliquons ensuite comment cette représentation étendue d'ASP est traitée au sein d'ASPeRiX.

3.1 Format ASP étendu pour ASPeRiX

Nous nous basons sur le vocabulaire reconnu pour les solveurs ASP (et utilisé dans les compétitions ASP) appelé ASP Core [3] avec $\mathcal{C}$ (resp. $\mathcal{X}$, $\mathcal{SF}$) l'ensemble des constantes (resp. variables, symboles de fonction).

Termes Un terme peut être une *constante*, une *variable*, un *terme fonctionnel* ou une *opération arithmétique*.
— Une *constante* est une *constante symbolique* (une chaîne commençant par une minuscule), une *constante numérique* (un entier) ou une *chaîne constante* (chaîne entre guillemets)
— Une *variable* est une chaîne commençant avec une majuscule

— Un *terme fonctionnel* est de la forme f(t_1,...,t_n) où f est un symbole fonctionnel d'arité n et t_i des termes. Un terme fonctionnel f() d'arité 0 est équivalent à une *constante symbolique* f.
— Une *opération arithmétique* est de la forme -(t_1) ou $t_1 \circ t_2$ avec t_1 et t_2 deux termes et $\circ \in \{+, -, *, /\}$

Atomes prédicatifs Les *atomes prédicatifs* sont de la forme p(t_1,...,t_n) avec t_1,...,t_n des termes et p un nom de prédicat (une chaîne commençant par une minuscule ou une chaîne entre guillemets). Un *atome prédicatif* p() d'arité 0 est représenté par son nom de prédicat p sans parenthèses. Un *littéral* p est un atome ou sa négation ¬ p.

Atomes relationnels Un autre type d'atomes est les *atomes relationnels* (aussi appelés *built-in atoms*) de la forme $t_1 \diamond t_2$ avec t_1 et t_2 deux termes et $\diamond \in \{<,<=,=,!=,>,>=\}$. Dans la suite, les *littéraux* et les *atomes relationnels* seront simplement appelés *atomes*.

Range Atomes Nous avons aussi des *range atomes* notés p(t_1..t_2) avec p un prédicat et t_1 et t_2 des termes représentant des constantes numériques ou des variables. Ces range atomes définissent un ensemble d'atomes qui vont de p(X) où X prend les valeurs possibles de t_1 à t_2.

Règles Une règle est de la forme

$$\forall \vec{x} \forall \vec{y}(\ \exists \vec{z}\ \mathrm{H} \text{ :- } \mathrm{B}^+, \text{ not } \exists \vec{n_1} \mathrm{N}_1, \ ..., \text{ not } \exists \vec{n_k} \mathrm{N}_k).$$

H is un ensemble de range atomes et de littéraux, B^+ un ensemble d'atomes et not N_1, ..., not N_k sont des ensembles des NAF-atomes (NAF pour negation as failure, soit négation par défaut) qui sont des atomes liés par une négation par défaut *not* (pour mémoire, différente de la négation forte ¬). Si un *not* est attaché à un ensemble d'atomes, l'ensemble doit être entre parenthèses. Dans de telles règles, on a plusieurs catégories de variables :
— $\vec{x}$ sont les variables de la frontière qui sont les variables universelles partagées entre la tête et les corps
— $\vec{y}$ sont les variables du corps positif (qui sont des variables universelles) qui apparaissent uniquement dans B^+
— $\vec{z}$ sont des variables existentielles qui n'apparaissent qu'en tête de règles et sont quantifiées existentiellement
— $\vec{n_i}$ sont des variables négatives qui apparaissent uniquement dans le corps négatif et sont quantifiées existentiellement

Une variable est quantifiée existentiellement si elle n'apparaît pas dans le corps positif. Comme il est facile de voir quelles sont les variables existentielles et les variables universelles, on peut écrire les règles sans quantificateurs. La forme simplifiée d'une règle est donc :

$$\mathrm{H} \text{ :- } \mathrm{B}^+, \text{ not } \mathrm{N}_1, \ ..., \text{ not } \mathrm{N}_k.$$

Règles absurdes La seule restriction sur les règles est qu'une variable ne peut pas apparaître à la fois dans la tête et le corps négatif sans apparaître dans le corps positif de la règle. Ces règles de la forme p(Y) :- not q(Y). ne sont pas possibles et sont appelées règles absurdes dans la

mesure où on ne peut pas leur donner de signification. Si une variable existentielle apparaît dans deux corps négatifs différents, elle est vue comme deux variables différentes.

Fait Un fait est une règle sans corps et exprime les données de la base de connaissances. Un fait peut représenter un atome unique (comme en ASP standard) ou un ensemble d'atomes.

3.1.1 Éléments utilisés pour définir les entités lexicales

<nombre>	: :=	0..9
	|	0..9 <nombre>
DOT	: :=	'.'
PAREN_OPEN	: :=	'('
PAREN_CLOSE	: :=	')'
MINUS	: :=	'-'
NOT	: :=	"not"

3.1.2 Entités lexicales

<constante symbolique>	: :=	une chaîne de lettres, chiffres et soulignés commençant par une minuscule
<variable>	: :=	une chaîne de lettres et chiffres commençant par une majuscule
<chaîne>	: :=	une séquence de caractères entre guillemets
<constante arithmétique>	: :=	<nombre> | - <nombre>
<portée>	: :=	<constante arithmétique> .. <constante arithmétique>
<expression arithmétique>	: :=	construit avec des constantes arithmétiques, des variables, les opérateurs binaires +, -, * , / et mod, l'opérateur unaire abs et des parenthèses.
<terme>	: :=	<variable>
	|	<expression arithmétique>
	|	<chaîne>
<termes>	: :=	[<termes> ,] <terme>
<termes étendus>	: :=	[<termes étendus> ,] (<terme> | <atome prédicatif>)
<atome relationnel>	: :=	<terme> (< | > | = | == | != | >= | <=) <terme>
<égalité>	: :=	<terme> (= | ==) <terme>
<atome prédicatif>	: :=	<constante symbolique> [PAREN_OPEN <termes étendus> PAREN_CLOSE]
<littéral>	: :=	([MINUS] <atome prédicatif>) | <atome relationnel>)
<littéraux>	: :=	[<littéraux> ,] <littéral>
<range littéral>	: :=	[MINUS] <constante symbolique> [PAREN_OPEN (<termes étendus> | <portée>) PAREN_CLOSE]
<range littéraux>	: :=	[<range littéraux> ,] (<range littéral> | <égalité>)
<corps positif>	: :=	<littéraux>
<corps négatifs>	: :=	[<corps négatifs> ,] NOT (<littéral> | (PAREN_OPEN <littéraux> PAREN_CLOSE))
<corps>	: :=	[(<corps positif> | <corps négatifs> | <atome relationnel>) ,] (<corps positif> | <corps négatifs> | <atome relationnel>)
<tête>	: :=	<range littéraux>

3.2 De dlgp à ASP

Nous expliquons maintenant la sémantique utilisée pour interpréter les programmes ASP étendus. La sémantique utilisée est celle de l'ASP standard avec quelques particularités concernant les ajouts syntaxiques et le traitement des existentielles.

Les exemples suivants sont de nouveau tirés de l'ontologie university avec, éventuellement, quelques adaptations pour illustrer le propos.

Traitement des variables existentielles par skolémisation Chaque variable qui apparaît dans la tête ou dans le corps négatif d'une règle et qui n'apparaît pas dans le corps positif est une variable existentielle. Un programme sans variables existentielles est dit $safe$ et a un programme équivalent en ASP standard.

Nous choisissons comme sémantique des variables existentielles la sémantique des variables skolémisées en ASP standard. Pour traiter les variables existentielles en tête de règles, la skolémisation permet de transformer une règle pour supprimer les variables existentielles. La skolémisation est réalisée de la manière suivante : pour chaque variable existentielle en tête, un nouveau symbole de fonction est créé et nommé skriX(Fr) avec i le numéro de la règle skolémisée, X le nom de la variable skolémisée et Fr l'ensemble des variables de la frontière de la règle.

Exemple 4. Le programme

r1 : student(titi).
r2 : person(X1), takesCourse(X1,X2), course(X2) :- student(X1).

est skolémisé en :

r1 : student(titi).
r2 : person(X1), takesCourse(X1,skr2X2(X1)), course(skr2X2(X1)) :- student(X1).

On peut noter que l'answer set obtenu sur ce programme est :

student(titi), person(titi), takesCourse(titi,skr2X2(titi)), course(skr1X2(titi)).

Une fois que le programme a été skolémisé, on peut traiter correctement les ensembles d'atomes.

Traitement des atomes multiples en tête L'un des ajouts à ASPeRiX est la possibilité d'avoir une tête contenant plusieurs atomes comme c'est souvent utilisé dans les ontologies. Ceci signifie que l'on peut lier deux atomes avec une variable partagée, en particulier pour une variable existentielle.

Le second intérêt est de simplifier les programmes dans la mesure où plusieurs règles ayant des têtes différentes mais ayant le même corps peuvent être regroupées dans une même règle et permet un gain de temps de calcul. Le traitement des têtes multiples est fait au sein d'ASPeRiX : que les règles ayant le même corps soient regroupées ou non, leurs têtes sont traitées conjointement de la même manière.

Exemple 5. r1 : takesCourse(X1,skr1X2(X1)), Course(skr1X2(X1)) :- Student(X1).

est équivalente à :

r1 : takesCourse(X1,skr1X2(X1)) :- Student(X1).
r2 : Course(skr1X2(X1)) :- Student(X1).

On peut aussi utiliser un ensemble d'atomes (en tête) pour représenter un fait (sans variables). Pour simuler des variables existentielles dans un fait, on peut créer une règle avec un corps factice toujours vrai et les atomes avec variables dans la tête.

Traitement des corps négatifs multiples Un autre ajout est une simplification syntaxique qui permet d'utiliser des atomes multiples dans une même négation par défaut ainsi que des variables existentielles dans les corps négatifs. En fait une telle règle est récrite en autant de règles qu'il y a d'atomes négatifs multiples. Pour chaque règle ri et pour chaque corps négatif Nj contenant une variable existentielle ou un ensemble d'atomes, nous créons un nouvel atome riNj(Fr) où i est le numéro de la règle, j la position du corps négatif et Fr l'ensemble des variables de la frontière. Cet atome est utilisé pour remplacer Nj dans la règle initiale et pour ajouter une nouvelle règle contenant le nouvel atome comme tête et Nj comme corps positif.

Exemple 6. La règle

r1 : phdStudent(X) :- person(X), not (lecturer(X), takesCourse(X,Y)).

peut être réécrite en :

r1 : phdStudent(X) :- person(X), not (r1N1(X)).
r0 : r1N1(X) :- lecturer(X), takesCourse(X,Y).

Traduction de dlgp vers ASP

Exemple Si nous reprenons l'exemple 2 proposé dans la section présentant dlgp, on obtient le programme ASP standard suivant obtenu par ASPeRiX à partir de la base exprimée en dlgp :

Exemple 7. [La base de l'exemple 2 au format ASP d'Asperix]

```
   % Fait

   % Skolémisation : X et Y deviennent skF1X et skF1Y
researcher(a), isMember(a, skF1X), isProject(b, kr,skF1Y).

   % Règles

   % règle Datalog standard : ne change pas
[R1] ismember(Z,X) :- isProject(X, Y, Z).

   % règle Datalog étendue : skolémisation de Z en skr2Z(X,Y) et
de L en skr2L(X,Y)
[R2] isProject(skr2Z(X,Y), Y, skr2L(X,Y)), isMember(X, skr2Z(X,Y)) :-
researcher(X), hasExpertise(X, Y).

   % contrainte
[R3] :- researcher(X), project(X).

   % règle d'égalité : devient une contrainte négative en utilisant
la différence d'ASP
[R4] :- isProject(X,Y,Z1), isProject(X,Y,Z2), Z1 != Z2.
```

4 Conclusion

La traduction d'informations écrites en OWL vers un programme ASP est effectuée en plusieurs étapes :

— la traduction d'ontologies au format OWL vers une base de règles existentielles exprimée dans le format dlgp
— la traduction de la base de règles existentielles au format dlgp dans un format de programme ASP acceptant les existentielles et les atomes multiples
— le traitement des règles ASP avec existentielles et atomes multiples pour obtenir un programme ASP standard

Notons que, avec ASPeRiX, la base de règles existentielles au format dlgp est prise en compte telle quelle et correspond au format ASP étendu accepté en entrée par ASPeRiX.

Concernant le compatibilité complète entre les formats dlgp et ASP d'ASPeRiX, on peut dire que le format dlgp peut quasiment être vu comme un sous-ensemble du format ASP d'ASPeRiX, à quelques nuances près :

— les variables existentielles sont traitées par skolémisation des faits et des règles
— la traduction des règles avec égalité se fait par le biais de contraintes négatives
— il n'existe pas d'encodage des requêtes dans le format d'ASPeRiX (ce point fait l'objet des travaux actuels sur ASPeRiX).

Notons que la skolémisation change la sémantique de la base de connaissances, mais il n'existe pas d'implémentation d'ASP qui conserverait les variables existentielles [1].

Concernant la traduction proprement dite qui est l'objet de ce délivrable, nous avons maintenant un outil complet, intégrant des outils fonctionnels pour chacune des étapes, qui permet de traduire, sous réserve des restrictions évoquées dans le document, une ontologie exprimée en OWL pour obtenir un programme ASP traitable par un solveur ASP comme ASPeRiX.

Annexes

5 Traduction de OWL à dlgp

Voici un exemple de traduction de univ-bench (LUBM) à dlgp. Remarque : l'ontologie appelée UNIVERSITY (U) est une version DL-LITE$_{\mathcal{R}}$ dérivée de LUBM [5].

5.1 Fichier d'origine (univ-bench.owl.xml)

source : http ://swat.cse.lehigh.edu/onto/univ-bench.owl (fait partie du benchmark appelé LUBM)

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns="http://swat.cse.lehigh.edu/onto/univ-bench.owl#"
xml:base="http://swat.cse.lehigh.edu/onto/univ-bench.owl"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#">

<owl:Ontology rdf:about="">
  <rdfs:comment>An university ontology for benchmark tests</rdfs:comment>
  <rdfs:label>Univ-bench Ontology</rdfs:label>
  <owl:versionInfo>univ-bench-ontology-owl, ver April 1, 2004</owl:versionInfo>
</owl:Ontology>

<owl:Class rdf:ID="AdministrativeStaff">
  <rdfs:label>administrative staff worker</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Employee"/>
</owl:Class>

<owl:Class rdf:ID="Article">
  <rdfs:label>article</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Publication"/>
</owl:Class>

<owl:Class rdf:ID="AssistantProfessor">
  <rdfs:label>assistant professor</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Professor"/>
</owl:Class>

<owl:Class rdf:ID="AssociateProfessor">
  <rdfs:label>associate professor</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Professor"/>
</owl:Class>

<owl:Class rdf:ID="Book">
  <rdfs:label>book</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Publication"/>
</owl:Class>

<owl:Class rdf:ID="Chair">
  <rdfs:label>chair</rdfs:label>
  <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Person"/>
  <owl:Restriction>
  <owl:onProperty rdf:resource="#headOf"/>
  <owl:someValuesFrom>
  <owl:Class rdf:about="#Department"/>
  </owl:someValuesFrom>
  </owl:Restriction>
  </owl:intersectionOf>
  <rdfs:subClassOf rdf:resource="#Professor"/>
```

```
</owl:Class>

<owl:Class rdf:ID="ClericalStaff">
  <rdfs:label>clerical staff worker</rdfs:label>
  <rdfs:subClassOf rdf:resource="#AdministrativeStaff"/>
</owl:Class>

<owl:Class rdf:ID="College">
  <rdfs:label>school</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Organization"/>
</owl:Class>

<owl:Class rdf:ID="ConferencePaper">
  <rdfs:label>conference paper</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Article"/>
</owl:Class>

<owl:Class rdf:ID="Course">
  <rdfs:label>teaching course</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Work"/>
</owl:Class>

<owl:Class rdf:ID="Dean">
  <rdfs:label>dean</rdfs:label>
  <owl:intersectionOf rdf:parseType="Collection">
  <owl:Restriction>
  <owl:onProperty rdf:resource="#headOf"/>
  <owl:someValuesFrom>
  <owl:Class rdf:about="#College"/>
  </owl:someValuesFrom>
  </owl:Restriction>
  </owl:intersectionOf>
  <rdfs:subClassOf rdf:resource="#Professor"/>
</owl:Class>

<owl:Class rdf:ID="Department">
  <rdfs:label>university department</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Organization"/>
</owl:Class>

<owl:Class rdf:ID="Director">
  <rdfs:label>director</rdfs:label>
  <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Person"/>
  <owl:Restriction>
  <owl:onProperty rdf:resource="#headOf"/>
  <owl:someValuesFrom>
  <owl:Class rdf:about="#Program"/>
  </owl:someValuesFrom>
  </owl:Restriction>
```

```
  </owl:intersectionOf>
</owl:Class>

<owl:Class rdf:ID="Employee">
  <rdfs:label>Employee</rdfs:label>
  <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Person"/>
  <owl:Restriction>
  <owl:onProperty rdf:resource="#worksFor"/>
  <owl:someValuesFrom>
  <owl:Class rdf:about="#Organization"/>
  </owl:someValuesFrom>
  </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>

<owl:Class rdf:ID="Faculty">
  <rdfs:label>faculty member</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Employee"/>
</owl:Class>

<owl:Class rdf:ID="FullProfessor">
  <rdfs:label>full professor</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Professor"/>
</owl:Class>

<owl:Class rdf:ID="GraduateCourse">
  <rdfs:label>Graduate Level Courses</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Course"/>
</owl:Class>

<owl:Class rdf:ID="GraduateStudent">
  <rdfs:label>graduate student</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Person"/>
  <rdfs:subClassOf>
  <owl:Restriction>
  <owl:onProperty rdf:resource="#takesCourse"/>
  <owl:someValuesFrom>
  <owl:Class rdf:about="#GraduateCourse"/>
  </owl:someValuesFrom>
  </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Institute">
  <rdfs:label>institute</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Organization"/>
</owl:Class>

<owl:Class rdf:ID="JournalArticle">
```

```
  <rdfs:label>journal article</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Article"/>
</owl:Class>

<owl:Class rdf:ID="Lecturer">
  <rdfs:label>lecturer</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Faculty"/>
</owl:Class>

<owl:Class rdf:ID="Manual">
  <rdfs:label>manual</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Publication"/>
</owl:Class>

<owl:Class rdf:ID="Organization">
  <rdfs:label>organization</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="Person">
  <rdfs:label>person</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="PostDoc">
  <rdfs:label>post doctorate</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Faculty"/>
</owl:Class>

<owl:Class rdf:ID="Professor">
  <rdfs:label>professor</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Faculty"/>
</owl:Class>

<owl:Class rdf:ID="Program">
  <rdfs:label>program</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Organization"/>
</owl:Class>

<owl:Class rdf:ID="Publication">
  <rdfs:label>publication</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="Research">
  <rdfs:label>research work</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Work"/>
</owl:Class>

<owl:Class rdf:ID="ResearchAssistant">
  <rdfs:label>university research assistant</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Person"/>
  <rdfs:subClassOf>
```

```
  <owl:Restriction>
  <owl:onProperty rdf:resource="#worksFor"/>
  <owl:someValuesFrom>
  <owl:Class rdf:about="#ResearchGroup"/>
  </owl:someValuesFrom>
  </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="ResearchGroup">
  <rdfs:label>research group</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Organization"/>
</owl:Class>

<owl:Class rdf:ID="Schedule">
  <rdfs:label>schedule</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="Software">
  <rdfs:label>software program</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Publication"/>
</owl:Class>

<owl:Class rdf:ID="Specification">
  <rdfs:label>published specification</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Publication"/>
</owl:Class>

<owl:Class rdf:ID="Student">
  <rdfs:label>student</rdfs:label>
  <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Person"/>
  <owl:Restriction>
  <owl:onProperty rdf:resource="#takesCourse"/>
  <owl:someValuesFrom>
  <owl:Class rdf:about="#Course"/>
  </owl:someValuesFrom>
  </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>

<owl:Class rdf:ID="SystemsStaff">
  <rdfs:label>systems staff worker</rdfs:label>
  <rdfs:subClassOf rdf:resource="#AdministrativeStaff"/>
</owl:Class>

<owl:Class rdf:ID="TeachingAssistant">
  <rdfs:label>university teaching assistant</rdfs:label>
  <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Person"/>
```

```
  <owl:Restriction>
  <owl:onProperty rdf:resource="#teachingAssistantOf"/>
  <owl:someValuesFrom>
  <owl:Class rdf:about="#Course"/>
  </owl:someValuesFrom>
  </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>

<owl:Class rdf:ID="TechnicalReport">
  <rdfs:label>technical report</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Article"/>
</owl:Class>

<owl:Class rdf:ID="UndergraduateStudent">
  <rdfs:label>undergraduate student</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Student"/>
</owl:Class>

<owl:Class rdf:ID="University">
  <rdfs:label>university</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Organization"/>
</owl:Class>

<owl:Class rdf:ID="UnofficialPublication">
  <rdfs:label>unnoficial publication</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Publication"/>
</owl:Class>

<owl:Class rdf:ID="VisitingProfessor">
  <rdfs:label>visiting professor</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Professor"/>
</owl:Class>

<owl:Class rdf:ID="Work">
  <rdfs:label>Work</rdfs:label>
</owl:Class>

<owl:ObjectProperty rdf:ID="advisor">
  <rdfs:label>is being advised by</rdfs:label>
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Professor"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="affiliatedOrganizationOf">
  <rdfs:label>is affiliated with</rdfs:label>
  <rdfs:domain rdf:resource="#Organization"/>
  <rdfs:range rdf:resource="#Organization"/>
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID="affiliateOf">
  <rdfs:label>is affiliated with</rdfs:label>
  <rdfs:domain rdf:resource="#Organization"/>
  <rdfs:range rdf:resource="#Person"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="age">
  <rdfs:label>is age</rdfs:label>
  <rdfs:domain rdf:resource="#Person"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="degreeFrom">
  <rdfs:label>has a degree from</rdfs:label>
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#University"/>
  <owl:inverseOf rdf:resource="#hasAlumnus"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="doctoralDegreeFrom">
  <rdfs:label>has a doctoral degree from</rdfs:label>
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#University"/>
  <rdfs:subPropertyOf rdf:resource="#degreeFrom"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="emailAddress">
  <rdfs:label>can be reached at</rdfs:label>
  <rdfs:domain rdf:resource="#Person"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="hasAlumnus">
  <rdfs:label>has as an alumnus</rdfs:label>
  <rdfs:domain rdf:resource="#University"/>
  <rdfs:range rdf:resource="#Person"/>
  <owl:inverseOf rdf:resource="#degreeFrom"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="headOf">
  <rdfs:label>is the head of</rdfs:label>
  <rdfs:subPropertyOf rdf:resource="#worksFor"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="listedCourse">
  <rdfs:label>lists as a course</rdfs:label>
  <rdfs:domain rdf:resource="#Schedule"/>
  <rdfs:range rdf:resource="#Course"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="mastersDegreeFrom">
  <rdfs:label>has a masters degree from</rdfs:label>
```

```
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#University"/>
  <rdfs:subPropertyOf rdf:resource="#degreeFrom"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="member">
  <rdfs:label>has as a member</rdfs:label>
  <rdfs:domain rdf:resource="#Organization"/>
  <rdfs:range rdf:resource="#Person"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="memberOf">
<rdfs:label>member of</rdfs:label>
<owl:inverseOf rdf:resource="#member"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="name">
<rdfs:label>name</rdfs:label>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="officeNumber">
  <rdfs:label>office room No.</rdfs:label>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="orgPublication">
  <rdfs:label>publishes</rdfs:label>
  <rdfs:domain rdf:resource="#Organization"/>
  <rdfs:range rdf:resource="#Publication"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="publicationAuthor">
  <rdfs:label>was written by</rdfs:label>
  <rdfs:domain rdf:resource="#Publication"/>
  <rdfs:range rdf:resource="#Person"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="publicationDate">
  <rdfs:label>was written on</rdfs:label>
  <rdfs:domain rdf:resource="#Publication"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="publicationResearch">
  <rdfs:label>is about</rdfs:label>
  <rdfs:domain rdf:resource="#Publication"/>
  <rdfs:range rdf:resource="#Research"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="researchInterest">
  <rdfs:label>is researching</rdfs:label>
</owl:DatatypeProperty>
```

```
<owl:ObjectProperty rdf:ID="researchProject">
  <rdfs:label>has as a research project</rdfs:label>
  <rdfs:domain rdf:resource="#ResearchGroup"/>
  <rdfs:range rdf:resource="#Research"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="softwareDocumentation">
  <rdfs:label>is documented in</rdfs:label>
  <rdfs:domain rdf:resource="#Software"/>
  <rdfs:range rdf:resource="#Publication"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="softwareVersion">
  <rdfs:label>is version</rdfs:label>
  <rdfs:domain rdf:resource="#Software"/>
</owl:ObjectProperty>

<owl:TransitiveProperty rdf:ID="subOrganizationOf">
  <rdfs:label>is part of</rdfs:label>
  <rdfs:domain rdf:resource="#Organization"/>
  <rdfs:range rdf:resource="#Organization"/>
</owl:TransitiveProperty>

<owl:ObjectProperty rdf:ID="takesCourse">
  <rdfs:label>is taking</rdfs:label>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="teacherOf">
  <rdfs:label>teaches</rdfs:label>
  <rdfs:domain rdf:resource="#Faculty"/>
  <rdfs:range rdf:resource="#Course"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="teachingAssistantOf">
  <rdfs:label>is a teaching assistant for</rdfs:label>
  <rdfs:domain rdf:resource="#TeachingAssistant"/>
  <rdfs:range rdf:resource="#Course"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="telephone">
  <rdfs:label>telephone number</rdfs:label>
  <rdfs:domain rdf:resource="#Person"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="tenured">
  <rdfs:label>is tenured:</rdfs:label>
  <rdfs:domain rdf:resource="#Professor"/>
</owl:ObjectProperty>
```

```
<owl:DatatypeProperty rdf:ID="title">
  <rdfs:label>title</rdfs:label>
  <rdfs:domain rdf:resource="#Person"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="undergraduateDegreeFrom">
  <rdfs:label>has an undergraduate degree from</rdfs:label>
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#University"/>
  <rdfs:subPropertyOf rdf:resource="#degreeFrom"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="worksFor">
  <rdfs:label>Works For</rdfs:label>
  <rdfs:subPropertyOf rdf:resource="#memberOf"/>
</owl:ObjectProperty>

</rdf:RDF>
```

5.2 Fichier résultat (univ-bench.owl.xml)

En dlgp (comme en Prolog, Datalog, ASP, ..), seuls les noms de variable commencent par des majuscules. Un nom de prédicat commence par une minuscule, ou peut-être entouré de doubles guillemets, ce qui permet n'importe quelle chaîne de caractères.

Voici le fichier obtenu à partir du fichier précédent.

Remarque : La présence des caractères # ne pose pas de problème pour le solveur ASPeRiX mais ils devraient être enlevés pour pouvoir être traités dans les autres solveurs ASP existants. Dans le solveur ASPeRiX, le fichier peut être utilisé tel quel.

```
% Ontology :
"#Employee"(X1) :- "AdministrativeStaff"(X1).
"#Publication"(X1) :- "Article"(X1).
"#Professor"(X1) :- "AssistantProfessor"(X1).
"#Professor"(X1) :- "AssociateProfessor"(X1).
"#Publication"(X1) :- "Book"(X1).
"#Person"(X1),"#headOf"(X1,X2),"#Department"(X2) :- "Chair"(X1).
"Chair"(X1) :- "#Person"(X1),"#headOf"(X1,X2),"#Department"(X2).
"#Professor"(X1) :- "Chair"(X1).
"#AdministrativeStaff"(X1) :- "ClericalStaff"(X1).
"#Organization"(X1) :- "College"(X1).
"#Article"(X1) :- "ConferencePaper"(X1).
"#Work"(X1) :- "Course"(X1).
"#headOf"(X1,X2),"#College"(X2) :- "Dean"(X1).
"Dean"(X1) :- "#headOf"(X1,X2),"#College"(X2).
"#Professor"(X1) :- "Dean"(X1).
"#Organization"(X1) :- "Department"(X1).
"#Person"(X1),"#headOf"(X1,X2),"#Program"(X2) :- "Director"(X1).
"Director"(X1) :- "#Person"(X1),"#headOf"(X1,X2),"#Program"(X2).
```

```
"#Person"(X1),"#worksFor"(X1,X2),"#Organization"(X2) :- "Employee"(X1).
"Employee"(X1) :- "#Person"(X1),"#worksFor"(X1,X2),"#Organization"(X2).
"#Employee"(X1) :- "Faculty"(X1).
"#Professor"(X1) :- "FullProfessor"(X1).
"#Course"(X1) :- "GraduateCourse"(X1).
"#Person"(X1) :- "GraduateStudent"(X1).
"#takesCourse"(X1,X2),"#GraduateCourse"(X2) :- "GraduateStudent"(X1).
"#Organization"(X1) :- "Institute"(X1).
"#Article"(X1) :- "JournalArticle"(X1).
"#Faculty"(X1) :- "Lecturer"(X1).
"#Publication"(X1) :- "Manual"(X1).
"#Faculty"(X1) :- "PostDoc"(X1).
"#Faculty"(X1) :- "Professor"(X1).
"#Organization"(X1) :- "Program"(X1).
"#Work"(X1) :- "Research"(X1).
"#Person"(X1) :- "ResearchAssistant"(X1).
"#worksFor"(X1,X2),"#ResearchGroup"(X2) :- "ResearchAssistant"(X1).
"#Organization"(X1) :- "ResearchGroup"(X1).
"#Publication"(X1) :- "Software"(X1).
"#Publication"(X1) :- "Specification"(X1).
"#Person"(X1),"#takesCourse"(X1,X2),"#Course"(X2) :- "Student"(X1).
"Student"(X1) :- "#Person"(X1),"#takesCourse"(X1,X2),"#Course"(X2).
"#AdministrativeStaff"(X1) :- "SystemsStaff"(X1).
"#Person"(X1),"#teachingAssistantOf"(X1,X2),"#Course"(X2) :- "TeachingAssistant"(X1).
"TeachingAssistant"(X1) :- "#Person"(X1),"#teachingAssistantOf"(X1,X2),"#Course"(X2).
"#Article"(X1) :- "TechnicalReport"(X1).
"#Student"(X1) :- "UndergraduateStudent"(X1).
"#Organization"(X1) :- "University"(X1).
"#Publication"(X1) :- "UnofficialPublication"(X1).
"#Professor"(X1) :- "VisitingProfessor"(X1).
"#Person"(X) :- "advisor"(X,Y).
"#Professor"(Y) :- "advisor"(X,Y).
"#Organization"(X) :- "affiliatedOrganizationOf"(X,Y).
"#Organization"(Y) :- "affiliatedOrganizationOf"(X,Y).
"#Organization"(X) :- "affiliateOf"(X,Y).
"#Person"(Y) :- "affiliateOf"(X,Y).
"#Person"(X) :- "age"(X,Y).
"#Person"(X) :- "degreeFrom"(X,Y).
"#University"(Y) :- "degreeFrom"(X,Y).
"#hasAlumnus"(Y,X) :- "degreeFrom"(X,Y).
"degreeFrom"(X,Y) :- "#hasAlumnus"(Y,X).
"#Person"(X) :- "doctoralDegreeFrom"(X,Y).
"#University"(Y) :- "doctoralDegreeFrom"(X,Y).
"#degreeFrom"(X,Y) :- "doctoralDegreeFrom"(X,Y).
"#Person"(X) :- "emailAddress"(X,Y).
"#University"(X) :- "hasAlumnus"(X,Y).
"#Person"(Y) :- "hasAlumnus"(X,Y).
"#degreeFrom"(Y,X) :- "hasAlumnus"(X,Y).
"hasAlumnus"(X,Y) :- "#degreeFrom"(Y,X).
"#worksFor"(X,Y) :- "headOf"(X,Y).
```

```
"#Schedule"(X) :- "listedCourse"(X,Y).
"#Course"(Y) :- "listedCourse"(X,Y).
"#Person"(X) :- "mastersDegreeFrom"(X,Y).
"#University"(Y) :- "mastersDegreeFrom"(X,Y).
"#degreeFrom"(X,Y) :- "mastersDegreeFrom"(X,Y).
"#Organization"(X) :- "member"(X,Y).
"#Person"(Y) :- "member"(X,Y).
"#member"(Y,X) :- "memberOf"(X,Y).
"memberOf"(X,Y) :- "#member"(Y,X).
"#Organization"(X) :- "orgPublication"(X,Y).
"#Publication"(Y) :- "orgPublication"(X,Y).
"#Publication"(X) :- "publicationAuthor"(X,Y).
"#Person"(Y) :- "publicationAuthor"(X,Y).
"#Publication"(X) :- "publicationDate"(X,Y).
"#Publication"(X) :- "publicationResearch"(X,Y).
"#Research"(Y) :- "publicationResearch"(X,Y).
"#ResearchGroup"(X) :- "researchProject"(X,Y).
"#Research"(Y) :- "researchProject"(X,Y).
"#Software"(X) :- "softwareDocumentation"(X,Y).
"#Publication"(Y) :- "softwareDocumentation"(X,Y).
"#Software"(X) :- "softwareVersion"(X,Y).
"#Organization"(X) :- "subOrganizationOf"(X,Y).
"#Organization"(Y) :- "subOrganizationOf"(X,Y).
"subOrganizationOf"(X,Z) :- "subOrganizationOf"(X,Y),"subOrganizationOf"(Y,Z).
"#Faculty"(X) :- "teacherOf"(X,Y).
"#Course"(Y) :- "teacherOf"(X,Y).
"#TeachingAssistant"(X) :- "teachingAssistantOf"(X,Y).
"#Course"(Y) :- "teachingAssistantOf"(X,Y).
"#Person"(X) :- "telephone"(X,Y).
"#Professor"(X) :- "tenured"(X,Y).
"#Person"(X) :- "title"(X,Y).
"#Person"(X) :- "undergraduateDegreeFrom"(X,Y).
"#University"(Y) :- "undergraduateDegreeFrom"(X,Y).
"#degreeFrom"(X,Y) :- "undergraduateDegreeFrom"(X,Y).
"#memberOf"(X,Y) :- "worksFor"(X,Y).
```

6 Traduction de dlgp à ASP

Une fois obtenue la base de règles existentielles au format dlgp, on s'intéresse à la traduction de cette base de règles au format dlgp en un programme ASP. La première transformation donne un programme ASP skolémisé.

```
% Ontology :
"Employee"(X1) :- "AdministrativeStaff"(X1).
"Publication"(X1) :- "Article"(X1).
"Professor"(X1) :- "AssistantProfessor"(X1).
"Professor"(X1) :- "AssociateProfessor"(X1).
"Publication"(X1) :- "Book"(X1).
"Person"(X1),"headOf"(X1,skr6X2(X1)),"Department"(skr6X2(X1)) :- "Chair"(X1).
```

```
"Chair"(X1) :- "Person"(X1),"headOf"(X1,X2),"Department"(X2).
"Professor"(X1) :- "Chair"(X1).
"AdministrativeStaff"(X1) :- "ClericalStaff"(X1).
"Organization"(X1) :- "College"(X1).
"Article"(X1) :- "ConferencePaper"(X1).
"Work"(X1) :- "Course"(X1).
"headOf"(X1,skr13X2(X1)),"College"(skr13X2(X1)) :- "Dean"(X1).
"Dean"(X1) :- "headOf"(X1,X2),"College"(X2).
"Professor"(X1) :- "Dean"(X1).
"Organization"(X1) :- "Department"(X1).
"Person"(X1),"headOf"(X1,skr17X2(X1)),"Program"(skr17X2(X1)) :- "Director"(X1).
"Director"(X1) :- "Person"(X1),"headOf"(X1,X2),"Program"(X2).
"Person"(X1),"worksFor"(X1,skr19X2(X1)),"Organization"(skr19X2(X1)) :- "Employee"(X1).
"Employee"(X1) :- "Person"(X1),"worksFor"(X1,X2),"Organization"(X2).
"Employee"(X1) :- "Faculty"(X1).
"Professor"(X1) :- "FullProfessor"(X1).
"Course"(X1) :- "GraduateCourse"(X1).
"Person"(X1) :- "GraduateStudent"(X1).
"takesCourse"(X1,skr25X2(X1)),"GraduateCourse"(skr25X2(X1)) :- "GraduateStudent"(X1).
"Organization"(X1) :- "Institute"(X1).
"Article"(X1) :- "JournalArticle"(X1).
"Faculty"(X1) :- "Lecturer"(X1).
"Publication"(X1) :- "Manual"(X1).
"Faculty"(X1) :- "PostDoc"(X1).
"Faculty"(X1) :- "Professor"(X1).
"Organization"(X1) :- "Program"(X1).
"Work"(X1) :- "Research"(X1).
"Person"(X1) :- "ResearchAssistant"(X1).
"worksFor"(X1,skr35X2(X1)),"ResearchGroup"(skr35X2(X1)) :- "ResearchAssistant"(X1).
"Organization"(X1) :- "ResearchGroup"(X1).
"Publication"(X1) :- "Software"(X1).
"Publication"(X1) :- "Specification"(X1).
"Person"(X1),"takesCourse"(X1,skr39X2(X1)),"Course"(skr39X2(X1)) :- "Student"(X1).
"Student"(X1) :- "Person"(X1),"takesCourse"(X1,X2),"Course"(X2).
"AdministrativeStaff"(X1) :- "SystemsStaff"(X1).
"Person"(X1),"teachingAssistantOf"(X1,skr42X2(X1)),"Course"(skr42X2(X1)):-
"TeachingAssistant"(X1).
"TeachingAssistant"(X1) :- "Person"(X1),"teachingAssistantOf"(X1,X2),"Course"(X2).
"Article"(X1) :- "TechnicalReport"(X1).
"Student"(X1) :- "UndergraduateStudent"(X1).
"Organization"(X1) :- "University"(X1).
"Publication"(X1) :- "UnofficialPublication"(X1).
"Professor"(X1) :- "VisitingProfessor"(X1).
"Person"(X) :- "advisor"(X,Y).
"Professor"(Y) :- "advisor"(X,Y).
"Organization"(X) :- "affiliatedOrganizationOf"(X,Y).
"Organization"(Y) :- "affiliatedOrganizationOf"(X,Y).
"Organization"(X) :- "affiliateOf"(X,Y).
"Person"(Y) :- "affiliateOf"(X,Y).
"Person"(X) :- "age"(X,Y).
```

```
"Person"(X) :- "degreeFrom"(X,Y).
"University"(Y) :- "degreeFrom"(X,Y).
"hasAlumnus"(Y,X) :- "degreeFrom"(X,Y).
"degreeFrom"(X,Y) :- "hasAlumnus"(Y,X).
"Person"(X) :- "doctoralDegreeFrom"(X,Y).
"University"(Y) :- "doctoralDegreeFrom"(X,Y).
"degreeFrom"(X,Y) :- "doctoralDegreeFrom"(X,Y).
"Person"(X) :- "emailAddress"(X,Y).
"University"(X) :- "hasAlumnus"(X,Y).
"Person"(Y) :- "hasAlumnus"(X,Y).
"degreeFrom"(Y,X) :- "hasAlumnus"(X,Y).
"hasAlumnus"(X,Y) :- "degreeFrom"(Y,X).
"worksFor"(X,Y) :- "headOf"(X,Y).
"Schedule"(X) :- "listedCourse"(X,Y).
"Course"(Y) :- "listedCourse"(X,Y).
"Person"(X) :- "mastersDegreeFrom"(X,Y).
"University"(Y) :- "mastersDegreeFrom"(X,Y).
"degreeFrom"(X,Y) :- "mastersDegreeFrom"(X,Y).
"Organization"(X) :- "member"(X,Y).
"Person"(Y) :- "member"(X,Y).
"member"(Y,X) :- "memberOf"(X,Y).
"memberOf"(X,Y) :- "member"(Y,X).
"Organization"(X) :- "orgPublication"(X,Y).
"Publication"(Y) :- "orgPublication"(X,Y).
"Publication"(X) :- "publicationAuthor"(X,Y).
"Person"(Y) :- "publicationAuthor"(X,Y).
"Publication"(X) :- "publicationDate"(X,Y).
"Publication"(X) :- "publicationResearch"(X,Y).
"Research"(Y) :- "publicationResearch"(X,Y).
"ResearchGroup"(X) :- "researchProject"(X,Y).
"Research"(Y) :- "researchProject"(X,Y).
"Software"(X) :- "softwareDocumentation"(X,Y).
"Publication"(Y) :- "softwareDocumentation"(X,Y).
"Software"(X) :- "softwareVersion"(X,Y).
"Faculty"(X) :- "teacherOf"(X,Y).
"Course"(Y) :- "teacherOf"(X,Y).
"TeachingAssistant"(X) :- "teachingAssistantOf"(X,Y).
"Course"(Y) :- "teachingAssistantOf"(X,Y).
"Person"(X) :- "telephone"(X,Y).
"Professor"(X) :- "tenured"(X,Y).
"Person"(X) :- "title"(X,Y).
"Person"(X) :- "undergraduateDegreeFrom"(X,Y).
"University"(Y) :- "undergraduateDegreeFrom"(X,Y).
"degreeFrom"(X,Y) :- "undergraduateDegreeFrom"(X,Y).
"memberOf"(X,Y) :- "worksFor"(X,Y).
```

La deuxième transformation donne un programme ASP standard (sans atomes multiples) traitable par un solveur ASP standard. Rappelons que, dans ASPeRiX, le traitement est optimisé pour les têtes multiples.

```
% Ontology :
```

```
"Employee"(X1) :- "AdministrativeStaff"(X1).
"Publication"(X1) :- "Article"(X1).
"Professor"(X1) :- "AssistantProfessor"(X1).
"Professor"(X1) :- "AssociateProfessor"(X1).
"Publication"(X1) :- "Book"(X1).
"Person"(X1):- "Chair"(X1).
"headOf"(X1,skr6X2(X1)):- "Chair"(X1).
"Department"(skr6X2(X1)) :- "Chair"(X1).
"Chair"(X1) :- "Person"(X1),"headOf"(X1,X2),"Department"(X2).
"Professor"(X1) :- "Chair"(X1).
"AdministrativeStaff"(X1) :- "ClericalStaff"(X1).
"Organization"(X1) :- "College"(X1).
"Article"(X1) :- "ConferencePaper"(X1).
"Work"(X1) :- "Course"(X1).
"headOf"(X1,skr13X2(X1)) :- "Dean"(X1).
"College"(skr13X2(X1)) :- "Dean"(X1).
"Dean"(X1) :- "headOf"(X1,X2),"College"(X2).
"Professor"(X1) :- "Dean"(X1).
"Organization"(X1) :- "Department"(X1).
"Person"(X1) :- "Director"(X1).
"headOf"(X1,skr17X2(X1)):- "Director"(X1).
"Program"(skr17X2(X1)) :- "Director"(X1).
"Director"(X1) :- "Person"(X1),"headOf"(X1,X2),"Program"(X2).
"Person"(X1) :- "Employee"(X1).
"worksFor"(X1,skr19X2(X1)) :- "Employee"(X1).
"Organization"(skr19X2(X1)) :- "Employee"(X1).
"Employee"(X1) :- "Person"(X1),"worksFor"(X1,X2),"Organization"(X2).
"Employee"(X1) :- "Faculty"(X1).
"Professor"(X1) :- "FullProfessor"(X1).
"Course"(X1) :- "GraduateCourse"(X1).
"Person"(X1) :- "GraduateStudent"(X1).
"takesCourse"(X1,skr25X2(X1)) :- "GraduateStudent"(X1).
"GraduateCourse"(skr25X2(X1)) :- "GraduateStudent"(X1).
"Organization"(X1) :- "Institute"(X1).
"Article"(X1) :- "JournalArticle"(X1).
"Faculty"(X1) :- "Lecturer"(X1).
"Publication"(X1) :- "Manual"(X1).
"Faculty"(X1) :- "PostDoc"(X1).
"Faculty"(X1) :- "Professor"(X1).
"Organization"(X1) :- "Program"(X1).
"Work"(X1) :- "Research"(X1).
"Person"(X1) :- "ResearchAssistant"(X1).
"worksFor"(X1,skr35X2(X1)) :- "ResearchAssistant"(X1).
"ResearchGroup"(skr35X2(X1)) :- "ResearchAssistant"(X1).
"Organization"(X1) :- "ResearchGroup"(X1).
"Publication"(X1) :- "Software"(X1).
"Publication"(X1) :- "Specification"(X1).
"Person"(X1) :- "Student"(X1).
"takesCourse"(X1,skr39X2(X1)) :- "Student"(X1).
"Course"(skr39X2(X1)) :- "Student"(X1).
```

```
"Student"(X1) :- "Person"(X1),"takesCourse"(X1,X2),"Course"(X2).
"AdministrativeStaff"(X1) :- "SystemsStaff"(X1).
"Person"(X1) :- "TeachingAssistant"(X1).
"teachingAssistantOf"(X1,skr42X2(X1)) :- "TeachingAssistant"(X1).
"Course"(skr42X2(X1)) :- "TeachingAssistant"(X1).
"TeachingAssistant"(X1) :- "Person"(X1),"teachingAssistantOf"(X1,X2),"Course"(X2).
"Article"(X1) :- "TechnicalReport"(X1).
"Student"(X1) :- "UndergraduateStudent"(X1).
"Organization"(X1) :- "University"(X1).
"Publication"(X1) :- "UnofficialPublication"(X1).
"Professor"(X1) :- "VisitingProfessor"(X1).
"Person"(X) :- "advisor"(X,Y).
"Professor"(Y) :- "advisor"(X,Y).
"Organization"(X) :- "affiliatedOrganizationOf"(X,Y).
"Organization"(Y) :- "affiliatedOrganizationOf"(X,Y).
"Organization"(X) :- "affiliateOf"(X,Y).
"Person"(Y) :- "affiliateOf"(X,Y).
"Person"(X) :- "age"(X,Y).
"Person"(X) :- "degreeFrom"(X,Y).
"University"(Y) :- "degreeFrom"(X,Y).
"hasAlumnus"(Y,X) :- "degreeFrom"(X,Y).
"degreeFrom"(X,Y) :- "hasAlumnus"(Y,X).
"Person"(X) :- "doctoralDegreeFrom"(X,Y).
"University"(Y) :- "doctoralDegreeFrom"(X,Y).
"degreeFrom"(X,Y) :- "doctoralDegreeFrom"(X,Y).
"Person"(X) :- "emailAddress"(X,Y).
"University"(X) :- "hasAlumnus"(X,Y).
"Person"(Y) :- "hasAlumnus"(X,Y).
"degreeFrom"(Y,X) :- "hasAlumnus"(X,Y).
"hasAlumnus"(X,Y) :- "degreeFrom"(Y,X).
"worksFor"(X,Y) :- "headOf"(X,Y).
"Schedule"(X) :- "listedCourse"(X,Y).
"Course"(Y) :- "listedCourse"(X,Y).
"Person"(X) :- "mastersDegreeFrom"(X,Y).
"University"(Y) :- "mastersDegreeFrom"(X,Y).
"degreeFrom"(X,Y) :- "mastersDegreeFrom"(X,Y).
"Organization"(X) :- "member"(X,Y).
"Person"(Y) :- "member"(X,Y).
"member"(Y,X) :- "memberOf"(X,Y).
"memberOf"(X,Y) :- "member"(Y,X).
"Organization"(X) :- "orgPublication"(X,Y).
"Publication"(Y) :- "orgPublication"(X,Y).
"Publication"(X) :- "publicationAuthor"(X,Y).
"Person"(Y) :- "publicationAuthor"(X,Y).
"Publication"(X) :- "publicationDate"(X,Y).
"Publication"(X) :- "publicationResearch"(X,Y).
"Research"(Y) :- "publicationResearch"(X,Y).
"ResearchGroup"(X) :- "researchProject"(X,Y).
"Research"(Y) :- "researchProject"(X,Y).
"Software"(X) :- "softwareDocumentation"(X,Y).
```

```
"Publication"(Y) :- "softwareDocumentation"(X,Y).
"Software"(X) :- "softwareVersion"(X,Y).
"Faculty"(X) :- "teacherOf"(X,Y).
"Course"(Y) :- "teacherOf"(X,Y).
"TeachingAssistant"(X) :- "teachingAssistantOf"(X,Y).
"Course"(Y) :- "teachingAssistantOf"(X,Y).
"Person"(X) :- "telephone"(X,Y).
"Professor"(X) :- "tenured"(X,Y).
"Person"(X) :- "title"(X,Y).
"Person"(X) :- "undergraduateDegreeFrom"(X,Y).
"University"(Y) :- "undergraduateDegreeFrom"(X,Y).
"degreeFrom"(X,Y) :- "undergraduateDegreeFrom"(X,Y).
"memberOf"(X,Y) :- "worksFor"(X,Y).
```

Références

[1] Jean-François Baget, Fabien Garreau, Marie-Laure Mugnier, and Swan Rocher. Revisiting chase termination for existential rules and their extension to nonmonotonic negation. *CoRR*, abs/1405.1071, 2014.

[2] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables : Walking the decidability line. *Artif. Intell.*, 175(9-10) :1620–1654, 2011.

[3] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, and T. Schaub. Asp-core-2 : Input language format. `https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.03b.pdf`, December 13, 2012.

[4] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. *International Conference on Logic Programming*, 1988.

[5] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM : A benchmark for OWL knowledge base systems. *J. Web Sem.*, 3(2-3) :158–182, 2005.

[6] C. Lefèvre and P. Nicolas. A first order forward chaining approach for answer set computing. In *Proceedings of the 12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, volume 5753 of *LNCS*, pages 196–208. Springer, 2009.

[7] C. Lefèvre and P. Nicolas. The first version of a new ASP solver : `ASPeRiX`. In *Proceedings of the 12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, volume 5753 of *LNCS*, pages 522–527. Springer, 2009.