8 Famille $\mathcal{SH}$

Les langages de la familles $\mathcal{AL}$ nous permettent de faire des descriptions complexes de concepts, mais sont plutôt limités en ce qui concernes les rôles. Tandis qu'on a à notre disposition plusieurs constructeurs pour construire une description en combinant plusieurs concepts, on ne peut rien dire sur les rôles. On ne peut que les utiliser dans les descriptions de concepts. Or, dans plusieurs situations, on aimerait établir certaines restrictions sur les relations.

Supposons par exemple une relation aDescendant, qui relie une personne à un de ses descendants. On sait que la relation aEnfant est un cas particulier de la relation aDescendant. Autrement dit, aEnfant est une sous-propriété de aDescendant : si une paire d'individus $\langle a, b\rangle$ appartient à $\mathcal{I}$(aEnfant), elle appartient aussi à $\mathcal{I}$(aDescendant). On se retrouve donc ici avec une hiérarchie de rôles. On sait aussi que la relation aDescendant est transitive : si A est un descendant de B qui est lui-même un descendant de C, A est aussi un descendant de C.

Si à la logique $\mathcal{ALC}$ on ajoute la possibilité d'établir qu'une relation est une sous-propriété d'une autre relation et la possibilité de définir une relation transitive, on se retrouve avec une logique nettement plus expressive, dénommée $\mathcal{SH}$. On écrira $Tr(R)$ pour signifier qu'une relation R est transitive, et $R_1 \sqsubseteq R_2$ pour signifier que R_1 est une sous-propriété de R_2.

La figure 3 résume les familles de logiques descriptives présentée dans ce document.

9 Le langage OWL

Dans sa première version, proposée en 2004 par le consortium W3C [2], le langage OWL est composé de trois sous-langages : OWL-Lite, OWL-DL et OWL-Full. OWL-Lite correspond essentiellement à la famille $\mathcal{SHIF}$, alors que OWL-DL correspond essentiellement à la famille $\mathcal{SHOIN}$. En fait, plus rigoureusement, il s'agit des familles $\mathcal{SHIF}(D)$ et $\mathcal{SHOIN}(D)$, parce qu'on y distingue deux types distincts de rôles. Les rôles qui lient deux individus, tel que vu tout au long du présent document, et les rôles qui associent un individu avec un littéral (d'où le D, pour *Data property*).

Les sous-langages OWL-Lite et OWL-DL ne sont pas des extensions de RDF, dans le sens qu'un triplet RDF n'est pas nécessairement valide dans ces deux sous-langages. C'est pour cette raison qu'on a ajouté le sous-langage OWL-Full, qui comprend tout OWL-DL, avec en plus tout RDF.

Depuis 2009, le consortium a officiellement lancé OWL 2, qui se distingue de la première version par un pouvoir expressif augmenté et une élimination de la décomposition en trois sous-langages. On parle plutôt de profils, en éliminant de OWL 2 certains éléments du langage, limitant ainsi son pouvoir expressif. Ainsi, OWL-Full n'existe plus, alors que OWL-Lite et OWL-DL peuvent être considérés comme des profils de OWL 2. Actuellement, trois profils de OWL 2 sont

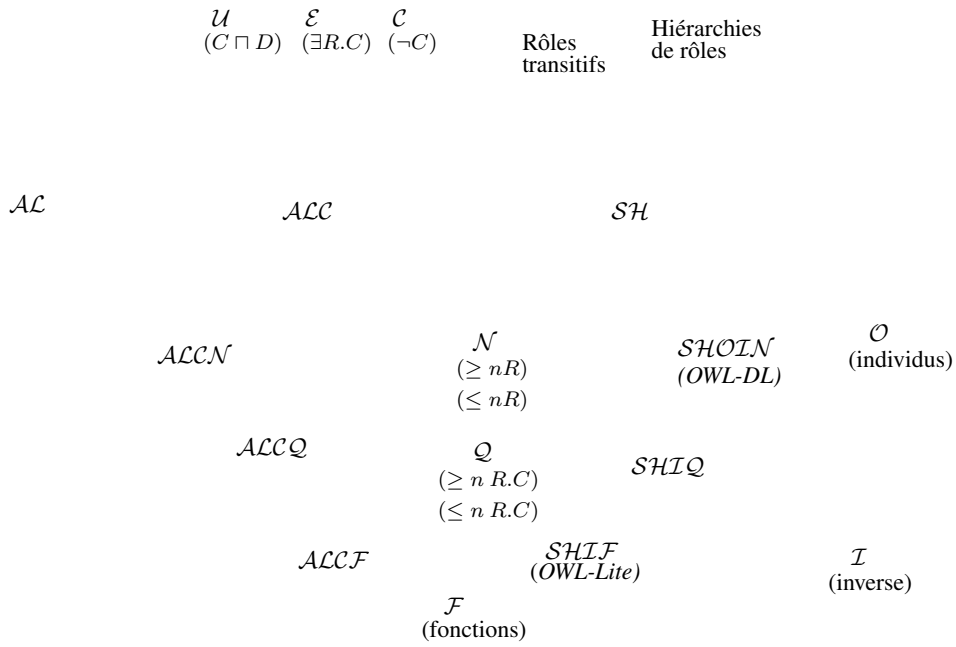

FIGURE 3 – Familles de logiques descriptives

proposés par le W3C. OWL 2 EL est un profil qui sied bien aux ontologies contenant une très grande quantité de classes et de propriétés. L'inférence pour vérifier la consistance, la subsomption et la catégorisation d'instances peut être réalisée en temps polynomial. OWL 2 QL est plus adaptée aux situations où on se retrouve avec beaucoup d'individus. Il s'agit en fait d'un sous-langage plus proche des langages de modélisations UML et Entités/Relations. Il est donc approprié pour interfacer une base de connnaissance avec une base de données. Finalement, OWL 2 RL est un sous-langage qui restreint les ontologies à des formes qui peuvent être traduites en règles de la logiques des prédicats de premier ordre.

Dans la prochaine section, le langage OWL 2 est présenté, suivi d'une descriptions de ses profils.

À noter que dans la présentation qui suit, plusieurs simplifications son été apportées à la définition du langage, pour des raisons de clarté. Par exemple, OWL permet d'ajouter des annotations dans les ontologies. Comme il ne s'agit pas d'éléments essentiels à la compréhension, certains types d'éléments ont tout simplement été omis. Il faut donc se rapporter aux documents du W3C pour une description exhaustive du langage.

Tous les exemples sont présentés dans la syntaxe fonctionnelle qui a été définie pour OWL (assez proche de celle de la logique descriptive) et dans la syntaxe Turtle, afin de bien visualiser la traduction en RDF d'une expression en OWL

9.1 Ontologies en OWL 2

Tout document OWL est une ontologie, qui peut avoir un identificateur unique représenté par une URI, utiliser certains préfixes et, finalement, importer d'autres ontologies. En syntaxe fonctionnelle, toute ontologie est représentée par un terme `Ontology(...)`. À l'intérieur, on y retrouve tous les énoncés qui décrivent l'ontologie. Voici, par exemple, une ontologie qu'on a identifiée par une URI, qui importe une autre ontologie, et qui ne définit rien d'autre (À noter la définition de préfixes) :

Syntaxe fonctionnelle :

```
Prefix(local:=<http://www.polytml.ca/mg/voc#>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Prefix(rdfs:=<http://www.w3.org/2000/01/rdf-schema#>)
Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)


Ontology(local:monOntologie
 Import( <http://www.exemple.org/ontologies/animaux.owl> )
)
```

Syntaxe Turtle :

```
@prefix local: <http://www.polytml.ca/mg/voc#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

local:monOntologie
  a owl:Ontology ;
  owl:imports <http://www.exemple.org/ontologies/animaux.owl> .
```

Évidemment, une ontologie qui en importe une autre et qui n'ajoute rien est sans intérêt. Une ontologie devrait minimalement ajouter au moins un élément nouveau. Par exemple, si l'ontologie importée contient la classe `Chien`, notre ontologie pourrait y ajouter la sous-classe `Caniche` :

Syntaxe fonctionnelle :

```
Prefix(local:=<http://www.polytml.ca/mg/voc#>)
Prefix(autre:=<http://www.exemple.org/ontologies/>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Prefix(rdfs:=<http://www.w3.org/2000/01/rdf-schema#>)
Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)


Ontology(local:monOntologie
 Import(<http://www.exemple.org/ontologies/animaux.owl>)
```

```
 Declaration(Class(local:Caniche))
 SubClassOf(local:Caniche autre:Chien)
)
```

Syntaxe Turtle :

```
@prefix local: <http://www.polytml.ca/mg/voc#> .
@prefix autre: <http://www.exemple.org/ontologies/> .

@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

local:monOntologie
  a owl:Ontology ;
  owl:imports <http://www.exemple.org/ontologies/animaux.owl> .

local:Caniche a owl:Class ;
              rdfs:subClassOf autre:Chien .
```

9.2 Axiomes de base pour décrire des ontologies simples

Nous avons vu à la section précédente comment déclarer une hiérarchie de classes en OWL. Deux types d'axiomes peuvent être utilisés : une déclaration de classe, qui ne fait qu'établir l'existence d'une classe, et l'utilisation directe de la propriété `rdfs:subClassOf`. En fait, le seul nouvel élément de vocabulaire par rapport à RDF est `owl:Class`. Le concept `rdfs:Class` n'est pas utilisé en OWL parce l'interprétation sémantique des classe en OWL est plus restrictive que celle de RDF.

Similiairement, on peut définir des hiérarchies de propriétés en OWL. Mais on distingue deux types de propriété : celles dont la valeur est un individu appartenant à une classe (ObjectProperty) et celles dont la valuer est un littéral (DataProperty). Voici un exmple de description de propriétés en OWL :

Syntaxe fonctionnelle :

```
... (préfixes)

Ontology(local:monOntologie
  Declaration(ObjectProperty(local:epouse))
  Declaration(ObjectProperty(local:conjoint))
  Declaration(DataProperty(local:age))

  SubObjectPropertyOf(local:epouse local:conjoint)

)
```

Syntaxe Turtle :

```
... (préfixes)

local:monOntologie a owl:Ontology .
local:epouse a owl:ObjectProperty ;
             rdfs:subPropertyOf local:conjoint .
local:conjoint a owl:ObjectProperty .
local:age a owl:DatatypeProperty .
```

Comme en RDF, on peut spécifier le domaine et l'image d'une propriété :

Syntaxe fonctionnelle :

```
... (préfixes)

Ontology(local:monOntologie
  Declaration(ObjectProperty(local:epouse))

  ObjectPropertyDomain(local:epouse local:Homme)
  ObjectPropertyRange(local:epouse local:Femme)
)
```

Syntaxe Turtle :

```
... (préfixes)

local:monOntologie  a owl:Ontology .
local:epouse a owl:ObjectProperty ;
             rdfs:domain local:Homme ;
             rdfs:range local:Femme .
```

Pour construire la ABox, on dispose en OWL des termes `ClassAssertion`, `ObjectPropertyAssertion` et `DatPropertyAssertion`. Le premier permet de spécifier la classe d'un individu, alors que les deux autres permettent d'établir une relation entre un individu et un autre indidividu ou un littéral, respectivement. Voici comment on pourrait représenter en OWL que Jean est un homme de 30 ans qui est marié à Marie :

Syntaxe fonctionnelle :

```
... (préfixes)

Ontology(local:monOntologie
  Declaration(ObjectProperty(local:epouse))
  Declaration(DataProperty(local:age))
  Declaration(Class(local:Homme))

  ClassAssertion(local:Homme local:Jean)
  ObjectPropertyAssertion(local:epouse local:Jean local:Marie)
  DataPropertyAssertion(local:age local:Jean "30"^^xsd:integer)

)
```

Syntaxe Turtle :

```
... (préfixes)

local:monOntologie a owl:Ontology .
local:epouse a owl:ObjectProperty .
local:age a owl:DatatypeProperty .
local:Homme a owl:Class .

local:Jean a local:Homme .
local:Jean local:epouse local:MArie .
local:Jean local:age "30"^^xsd:integer .
```

Jusqu'à maintenant, nous avons vu peu de choses qui distinguent OWL de RDF : une redéfinition du concept de classe, qui est en fait un sous-ensemble du concet de classe en RDF, la distinction de deux types de propriétés. Nous allons maintenant aborder des éléments de OWL qui nous mènent nettemetn au delà de RDF.

D'abord, en OWL, on peut déclarer que deux classes sont équivalentes ou disjointes (à noter que le nombre d'items que peut contenir un terme `EquivalentClasses` ou `DisjointClasses` n'est pas limité à 2) :

Syntaxe fonctionnelle :

```
... (préfixes)

Ontology(local:monOntologie
Declaration(Class(local:Homme))
Declaration(Class(local:Femme))
Declaration(Class(local:Feminin))

EquivalentClasses(local:Femme local:Feminin)
DisjointClasses(local:Homme local:Femme)
)
```

Syntaxe Turtle :

```
... (préfixes)

local:monOntologie  a owl:Ontology .
local:Homme a owl:Class.
local:Femme a owl:Class.
local:Feminin a owl:Class.

local:Femme owl:equivalentClass local:Feminin .

[] a owl:allDisjointClasses ;
   owl:members (local:Homme local:Femme) .
```

On remarque que la représentation de classes disjointes en RDF n'est pas aussi directe que celle de classes équivalentes. Il faut supposer une entité anonyme dont les membres sont indiqués par une liste de classes.

Finalement, on ne peut pas en RDF spécifier qu'une relation n'existe pas entre deux entité, alors qu'on peut le faire en OWL. Voci, par exemple, comment on pourrait indiquer que Jean n'a

pas 30 ans et qu'il n'est pas marié avec Anne (à noter que la traduction en RDF se ressemble dans les deux cas, à l'exception de la propriété utilisée dans le dernier triplet, soit `targetValue` au lieu de `targetIndividual` :

Syntaxe fonctionnelle :

```
... (préfixes)

Ontology(local:monOntologie

NegativeObjectPropertyAssertion(local:epouse local:Jean local:Anne)
NegativeDataPropertyAssertion(local:age local:Jean "30"^^xsd:integer )

)
```

Syntaxe Turtle :

```
... (préfixes)

local:monOntologie  a owl:Ontology .

[] a owl:NegativePropertyAssertion ;
  owl:sourceIndividual   local:Jean ;
  owl:assertionProperty  local:epouse ;
  owl:targetIndividual   local:Anne .

[] a owl:NegativePropertyAssertion ;
  owl:sourceIndividual   local:Jean ;
  owl:assertionProperty  local:age ;
  owl:targetValue "30"^^xsd:integer .
```

9.3 Descriptions de classes complexes

À venir

9.4 Descriptions des propriétés

Cette section est à compléter.

La spécification de l'image ou du domaine d'une propriété ne requiert pas une logique plus expressive que celle de la famille $\mathcal{AL}$. En effet, pour exprimer qu'un rôle R ne peut s'appliquer qu'à des individus de la classe C, on écrira l'axiome suivant : $\exists R.\top \sqsubseteq C$. Pour représenter que tous les individus du domaine d'un rôle R appartiennent à la classe C, on écrira l'axiome $\top \sqsubseteq \forall R.C$. Il est très important de ne pas confondre ces axiomes avec des restrictions de rôle. Supposons par exemple un axiome de la forme suivante :

$C_1 \equiv C_2 \sqcap \forall R.C_3$

Cet axiome n'impose pas du tout que tous les éléments du domaine de R appartiennent à la classe C_3. Rien n'empêcherait d'ajouter à l'ontologie un autre axiome de la forme suivante :

$C_4 \equiv C_5 \sqcap \forall R.C_6$

9.5 Déclaration de faits

Les faits en OWL 2 permettent de fournir des informations sur des entités spécifiques (appelées *individus*). Essentiellement, on peut spécifier les classes auxquelles un individu appartient, ainsi que les valeurs des propriétés qui le concerne, qui sont des individus ou des littéraux, selon le type de propriété.

De manière simple, on peut déclarer un indicividu en spécifiant son nom :

Syntaxe fonctionnelle :

```
... (préfixes)

Ontology(local:monOntologie

  Declaration( NamedIndividual(local:Jean) )
)
```

Syntaxe Turtle :

```
... (préfixes)

local:monOntologie  a owl:Ontology .
local:Jean a owl:NamedIndividual .
```

On peut aussi déclarer le type d'un individu en indiquant la classe à laquelle il appartient. Cette classe peut être un concept atomique ou une description plus complexe :

Syntaxe fonctionnelle :

```
... (préfixes)

Ontology(local:monOntologie

  ClassAssertion(foaf:Person local:Jean)
  ClassAssertion(
      ObjectIntersectionOf(foaf:Person local:professeur)
      local:Paul)
)
```

Syntaxe Turtle :

```
... (préfixes)

local:monOntologie  a owl:Ontology .

local:Jean a foaf:Person .
local:Paul a [ a owl:Class ;
               owl:intersectionOf ( foaf:Person local:professeur ) ]
```

On peut spécifier qu'un individu est lié ou n'est pas lié à un autre par un propriété :

Syntaxe fonctionnelle :

```
... (préfixes)

Ontology(local:monOntologie

  ObjectPropertyAssertion(local:Jean local:aime local:Marie)
  NegativeObjectPropertyAssertion(local:Jean local:aime local:Anne)
  DataPropertyAssertion(local:Jean local:age "34"^^xsd:integer)
)
```

Syntaxe Turtle :

```
... (préfixes)

local:monOntologie  a owl:Ontology .

local:Jean local:aime local:Marie .
[] a owl:NegativePropertyAssertion ;
   owl:sourceIndividual   local:Jean ;
   owl:assertionProperty  local:aime ;
   owl:targetIndividual   local:Anne .

local:Jean local:age "34"^^xsd:integer .
```

Un autre type de fait permis en OWL-Lite sert à spécifier que deux ou plusieurs identificateurs d'individus représentent le même individu ou des individus différents :

Syntaxe fonctionnelle :

```
... (préfixes)

Ontology(local:monOntologie
DifferentIndividuals(local:Jean local:Marie)
SameIndividual(local:Marie local:Bouchard)

)
```

Syntaxe Turtle :

```
... (préfixes)

local:monOntologie  a owl:Ontology .
local:Jean owl:differentFrom local:Marie .
local:Marie owl:sameAs local:Bouchard .
```

Dans le cas où on a plus de deux individus qui sont différents, la traduction en RDF est un peu différente :

Syntaxe fonctionnelle :

```
... (préfixes)

Ontology(local:monOntologie
DifferentIndividuals(local:Jean local:Marie local:Paul)
)
```

Syntaxe Turtle :

```
... (préfixes)

local:monOntologie  a owl:Ontology .
[] a owl:AllDifferent ;
   owl:members ( local:Jean local:Marie local:Paul ) .
```

9.6 Les différents profils de OWL 2

9.6.1 OWL Lite

Dans la spécification de OWL 2, le profil Lite n'existe pas. Nous le présentons ici pour des raisons historiques, puisqu'il existait dans la première spécification de OWL.

En OWL-Lite, on peut spécifier que deux ou plusieurs classes sont équivalentes, mais seulement dans le cas où les classes ont un identificateur. L'axiome d'équivalence ne peut donc pas s'appliquer à des descriptions complexes de classes :

axiome ::= *EquivalentClasses(* **classeID classeID** {**classeID**} *)*

Le type de restriction que l'on peut déclarer en OWL-Lite est plutôt limité. On a les quantifications universelle et existentielle, c'est-à-dire des restrictions de la forme $\forall R.C$ et $\exists R.C$, mais la description C ne peut pas être autre chose qu'un concept atomique. À part les quantifications, on n'a que les restrictions de cardinalité, mais où les seules valeurs permises sont 0 et 1. Voici la syntaxe pour les restrictions :

élémentRestrictionIndividu ::= *ObjectAllValuesFrom(* **propID classeID** *)*
| *ObjectSomeValuesFrom(* **propID classeID** *)*
| **cardinalitéPropriété**
élémentRestrictionLittéral ::= *DataAllValuesFrom(* **propId dataRange** *)*
| *DataSomeValuesFrom(* **propID dataRange** *)*
| **cardinalitéLittéral**
cardinalitéPropriété ::= *ObjectMinCardinality(0* ***propID*** *)* | *ObjectMinCardinality(1* ***propID****)*
| *ObjectMaxCardinality(0* ***propID****)* | *ObjectMaxCardinality(1* ***propID****)*
| *ObjectExactCardinality(0* ***propID****)* | *ObjectExactCardinality(1* ***propID****)*
cardinalitéLittéral ::= *DataMinCardinality(0* ***propID****)* | *DataMinCardinality(1* ***propID****)*

| *DataMaxCardinality(0* ***propID****)* | *DataMaxCardinality(1* ***propID****)*
| *DataExactCardinality(0* ***propID****)* | *DataExactCardinality(1* ***propID****)*

dataRange ::= **typeLittéralID** | *rdfs :Literal*

Il y a plusieurs types de données que l'on peut utiliser pour qualifier les restrictions de littéral. En voici une liste non exhaustive : `xsd:string`, `xsd:boolean`, `xsd:decimal`, `xsd:float`, `xsd:double`, `xsd:time`, `xsd:date`, `xsd:int` , `xsd:positiveInteger`.
En guise d'exemple, voici une ontologie et sa traduction en OWL-Lite :

En logique descriptive :

$$\text{Célibataire} \equiv \text{Personne} \sqcap \leq 0\ \text{mariéAvec}$$
$$\text{Végétarien} \equiv \text{Personne} \sqcap \forall \text{mange.Vegetal}$$

En OWL-Lite :

```
Prefix(local:=<http://www.polymtl.ca#>)
Ontology(
  EquivalentClasses(
       local:Celibataire
       ObjectIntersectionOf(
           local:Personne
           ObjectMaxCardinality(0 local:marieAvec)))
  EquivalentClasses(
       local:Vegetarien
       ObjectIntersectionOf(
           local:Personne
           ObjectAllValuesFrom(local:mange local:Vegetal))))
```

Il y a plusieurs limitations en OWL-Lite en ce qui concerne les propriétés. On peut déclarer une propriété comme sous-propriété d'une autre propriété, définir son domaine et son image et, finalement, la déclarer fonctionnelle. Dans le cas des propriétés objet, on peut, en plus de ce qui a été énuméré précédemment, la déclarer symétrique, transitive, ou encore injective (inverse fonctionnelle), c'est-à-dire qu'on ne peut avoir deux individus du domaine qui ont la même valeur dans l'image de la fonction.

axiomePropriété ::=
*SubDataPropertyOf(***propID1 propID2** *)*
| *DataPropertyDomain(* **propID classeID** *)*
| *DataPropertyRange(* **propID dataRange** *)*
| *FunctionalDataProperty(* ***propID1*** *)*]
*SubObjectPropertyOf(***propID1 propID2** *)*
| *FunctionalObjectProperty(* ***propID1*** *)*]
| *ObjectPropertyDomain(* **propID classeID** *)*
| *ObjectPropertyRange(* **propID dataRange** *)*
| *FunctionalObjectProperty(* **propID1** *)*]

| *InverseFunctionalObjectProperty(* **propID1** *)*]
| *TransitiveObjectProperty(* **propID1** *)*]

Finalement OWL-Lite permet d'écrire des axiomes qui spécifient des équivalences de propriétés :

axiomePropriété ::= *EquivalentDataProperties(* **propID propID** {**propID**} *)*
| *EquivalentObjectProperties(* **propID propID** {**propID**} *)*

Il est à noter ici que la classification de OWL-Lite comme un langage de la famille $\mathcal{SHIF}$ est un peu abusive. En effet, OWL-Lite ne contient pas tout le pouvoir expressif de la famille $\mathcal{ALC}$, à partir de laquelle la famille $\mathcal{SH}$ est définie. La négation et l'union de concepts y sont absents. Aussi, dans tous les axiomes exprimant les formes $C_1 \equiv C_2$ ou $C_1 \sqsubseteq C_2$, les descriptions C_1 ne peuvent pas être autre chose qu'un concept atomique, contrairement à la logique $\mathcal{ALC}$, qui accepte aussi des descriptions complexes à cette position. Par exemple, l'axiome suivant ne pourrait pas etre représenté en OWL-Lite : $(A \sqcap B) \sqsubseteq C$. Une conséquence importante est l'impossibilité de spécifier que deux classes A et B sont disjointes, puisque cela exige un axiome de la forme suivante : $(A \sqcap B) \sqsubseteq \bot$. Similairement, les restrictions n'acceptent que des concepts atomiques, ce qui rend impossible la représentation de l'axiome suivant, qui est tout à fait acceptable dans le langage $\mathcal{ALC}$: $A \equiv B \sqcap \exists R.(C \sqcup D)$. La seule exception est l'axiome *ObjectPropertyDomain(P C)* (similairement pour *DataPropertyDomain*), qui correspond à la forme suivante en logique descriptive : $\geq 1\ R \sqsubseteq C$.

Pour terminer, voici un exemple d'ontologie que l'on peut exprimer en OWL-Lite.
En logique descriptive :

Célibataire $\equiv$ Personne $\sqcap \leq 0$ mariéAvec
Personne $\sqsubseteq \exists$nom.`xsd:string` $\sqcap \exists$age.`xsd:int`
Homme $\sqsubseteq$ Personne
Femme $\sqsubseteq$ Personne
Père $\equiv$ Personne $\sqcap \exists$aEnfant.Personne
PèreDeFilles $\equiv$ Père $\sqcap \forall$aEnfant.Femme
$\top \sqsubseteq \forall$aEnfant.Personne (image)
$\exists$aEnfant.$\top \sqsubseteq$ Personne (domaine)
aEnfant $\equiv$ aParent$^-$ (rôle inverse)
$\top \sqsubseteq \leq 1$ estConjointDe (rôle fonctionnel)
$\top \sqsubseteq \leq 1$ estConjointDe$^-$ (rôle injectif)
estConjointDe $\equiv$ estConjointDe$^-$ (rôle symétrique)
$\top \sqsubseteq \forall$estConjointDe.Personne (image)
$\exists$estConjointDe.$\top \sqsubseteq$ Personne (domaine)
mariéAvec $\sqsubseteq$ estConjointDe
$\top \sqsubseteq \forall$age.`xsd:int` (image)
$\exists$age.$\top \sqsubseteq$ Personne (domaine)
$\top \sqsubseteq \forall$nom.`xsd:string` (image)
$\exists$nom.$\top \sqsubseteq$ Personne (domaine)

En OWL 2 Lite :

```
Prefix(local:=<http://www.polymtl.ca#>)
Ontology(
  EquivalentClasses(
           local:Celibataire
           ObjectIntersectionOf(
              local:Personne
              ObjectMaxCardinality(0 local:marieAvec)))

  SubClassOf(
           local:Personne
           ObjectIntersectionOf(
               DataSomeValuesFrom(local:nom xsd:string)
               DataSomeValuesFrom(local:age xsd:int)))
  SubClassOf(local:Homme local:Personne)
  SubClassOf(local:Femme local:Personne)
  EquivalentClasses(
           local:Pere
           ObjectIntersectionOf(
               local:Personne
               ObjectSomeValuesFrom(local:aEnfant local:Personne)))
  EquivalentClasses(
           local:PereDeFilles
           ObjectIntersectionOf(
               local:Pere
               ObjectAllValuesFrom(local:aEnfant local:Femme)))
  ObjectPropertyDomain(local:aEnfant local:Personne)
```

```
ObjectPropertyRange(local:aEnfant local:Personne)
InverseObjectProperties(local:aEnfant local:aParent)
FunctionalObjectProperty(local:estConjointDe)
InverseFunctionalObjectProperty(local:estConjointDe)
SymmetricObjectProperty(local:estConjointDe)
ObjectPropertyDomain(local:estConjointDe local:Personne)
ObjectPropertyRange(local:estConjointDe local:Personne)
SubObjectPropertyOf(local:marieAvec local:estConjointDe)
DataPropertyDomain(local:age local:Personne)
DataPropertyRange(local:age xsd:int)
DataPropertyDomain(local:nom local:Personne)
DataPropertyRange(local:nom xsd:string))
```

9.6.2 OWL EL

Le profil EL est un sous-ensemble de OWL DL, qui correspond essentiellement à la famille $\mathcal{EL}$, obtenu en considérant les contraintes suivantes :

1. Pas de négation de classes.
2. Pas de disjonction.
3. Pas de quantificateur universel.
4. Pas de restriction de cardinalité.
5. Pas de classe définie par énumération.
6. La définition d'une classe par énumération ne peut pas contenir plus d'un élément (ce qui réduit grandement son utilité).
7. Les seuls axiome de propriétés qu'on peut utiliser sont les suivantes : réflexivité, transitivité, équivalence, sous-propriété, domaine et image.
8. Pas d'individus anonymes. Par exemple, l'expression suivante serait interdite : `ObjectPropertyAssertion(:mariéAvec :paul _:x`
9. Les types suivants ne peuvent être utilisés : `xsd:double`, `xsd:float`, `xsd:nonPositiveInteger`, `xsd:positiveInteger`, `xsd:negativeInteger`, `xsd:long`, `xsd:int`, `xsd:short`, `xsd:byte`, `xsd:unsignedLong`, `xsd:unsignedInt`, `xsd:unsignedShort`, `xsd:unsignedByte`, `xsd:language` et `xsd:boolean`.

9.6.3 OWL QL

À venir

9.6.4 OWL RL

À venir

9.7 Traduction en RDF

Pour traduire en RDF une ontologie écrite en syntaxe abstraite OWL, nous utiliserons une fonction de traduction T qui, appliquée récursivement à l'ontologie, transforme ses composantes en triplets RDF.
Traduction d'une ontologie :

Ontology(ID Import(OntURI$_1$) ... Import(OntURI$_n$) axiome$_1$ ... axiome$_n$)	ID `rdf:type owl:Ontology .` ID `owl:imports` OntURI$_1$ ... ID `owl:imports` OntURI$_n$ T(axiome$_1$) ... T(axiome$_n$)
Ontology(Import(OntURI$_1$) ... Import(OntURI$_n$) axiome$_1$ ... axiome$_n$)	ID `owl:imports` OntURI ... ID `owl:imports` OntURI$_n$ T(axiome$_1$) ... T(axiome$_n$)

Pour comprendre certaines règles de traduction qui suivent, nous aurons parfois besoin d'utiliser une fonction de traduction intermédiaire $TLIST$, qui retourne une liste d'items. Soit un ensemble d'items item$_1$, item$_2$ ... item$_n$, la fonction $TLIST$(item$_1$, item$_2$ ... item$_n$) retourne le graphe RDF suivant :

```
[] a rdf:List;
   rdf:first T(item1) ;
   rdf:rest
    [  a rdf:List ;
       rdf:first T(item2)  ;
       ...
           [  a rdf:List ;
              rdf:first T(itemn)  ;
              rdf:rest rdf:nil ] ] .
```

Il s'agit donc d'une liste RDF comprenant des expressions elles-même traduites en RDF.

Traduction des connecteurs :

Declaration(Class(C))	T(C) `a owl:Class \verb.+`
SubClassOf(C_1 $C2_2$)	T(C_1) `rdfs:subClassOf` T(C_1) .
EquivalentClasses($C_1 \dots C_n$)	T(C_1) `owl:equivalentClass` T(C_2) . ... T(C_{n-1}) `owl:equivalentClass` T(C_n) .
DisjointClasses(C_1 C_2)	T(C_1) `owl:disjointWith` T(C_2) .
DisjointClasses($C_1 \dots C_n$), où n > 2	`[] a owl:AlldisjointClasses ;` `owl:members` $TLIST$($C_1 \dots C_n$) .
DisjointUnion(C $C_1 \dots C_n$)	T(C) `owl:disjointUnionOf` $TLIST$($C_1 \dots C_n$) .
ObjectIntersectionOf($C_1 \dots C_n$)	`[] a owl:Class ;` `owl:intersectionOf` $TLIST$($C_1, \dots C_n$) .
ObjectUnionOf($C_1 \dots C_n$)	`[] a owl:Class ;` `owl:unionOf` $TLIST$($C_1, \dots C_n$) .
ObjectComplementOf(C)	`[] owl:Class ;` `owl:complementOf` T(C) .
ObjectOneOf(individu$_1$ ... individu$_n$)	`[] a owl:Class ;` `owl:oneOf` $TLIST$(individu$_1$, ... individu$_n$) .

Traduction des restrictions d'objets :

ObjectAllValuesFrom(P C)	[] a owl:Restriction ; owl:onProperty T(P) ; owl:allValuesFrom T(C) .
ObjectSomeValuesFrom(P C)	[] a owl:Restriction ; owl:onProperty T(P) ; owl:someValuesFrom T(C) .
ObjectHasValue(P valeur))	[] a owl:Restriction ; owl:onProperty T(P) ; owl:hasValue T(valeur) .
ObjectHasSelf(P))	[] a owl:Restriction ; owl:onProperty T(P) ; owl:hasSelf "true"^^xsd:boolean .
ObjectHasMinCardinality(n P)	[] a owl:Restriction ; owl:onProperty T(P) ; owl:minCardinality "n"^^xsd:nonNegativeInteger .
ObjectHasMaxCardinality(n P)	[] a owl:Restriction ; owl:onProperty T(P) ; owl:maxCardinality "n"^^xsd:nonNegativeInteger .
ObjectExactCardinality(n P)	[] a owl:Restriction ; owl:onProperty T(P) ; owl:cardinality "n"^^xsd:nonNegativeInteger .
ObjectHasMinCardinality(n P C)	[] a owl:Restriction ; owl:onProperty T(P) ; owl:minQualifiedCardinality "n"^^xsd:nonNegativeInteger ; owl:onClass T(C) .
ObjectHasMaxCardinality(n P C)	[] a owl:Restriction ; owl:onProperty T(P) ; owl:maxQualifiedCardinality "n"^^xsd:nonNegativeInteger ; owl:onClass T(C) .
ObjectExactCardinality(n P C)	[] a owl:Restriction ; owl:onProperty T(P) ; owl:qualifiedCardinality "n"^^xsd:nonNegativeInteger ; owl:onClass T(C) .

Traduction des restrictions de valeurs :

DataAllValuesFrom(DP Range)	`[] a owl:Restriction ;` `owl:onProperty` T(DP) ; `owl:allValuesFrom` T(Range) .
DataAllValuesFrom(DP$_1$ ... DP$_n$ Range), où n ≥ 2	`[] a owl:Restriction ;` `owl:onProperties` $TLIST$(DP$_1$ ... DP$_n$) ; `owl:allValuesFrom` T(Range) .
DataSomeValuesFrom(DP Range)	`[] a owl:Restriction ;` `owl:onProperty` T(DP) ; `owl:someValuesFrom` T(Range) .
DataSomeValuesFrom(DP$_1$ ... DP$_n$ Range), où n ≥ 2	`[] a owl:Restriction ;` `owl:onProperties` $TLIST$(DP$_1$ ... DP$_n$) ; `owl:someValuesFrom` T(Range) .
DataHasValue(DP valeur))	`[] a owl:Restriction ;` `owl:onProperty` T(DP) ; `owl:hasValue` T(valeur) .

DataHasMinCardinality(n DP)	`[] a owl:Restriction ;`
	`owl:onProperty` T(DP) ;
	`owl:minCardinality`
	`"n"^^xsd:nonNegativeInteger .`

DataHasMaxCardinality(n DP)	`[] a owl:Restriction ;`
	`owl:onProperty` T(DP) ;
	`owl:maxCardinality`
	`"n"^^xsd:nonNegativeInteger .`

DataExactCardinality(n DP)	`[] a owl:Restriction ;`
	`owl:onProperty` T(DP) ;
	`owl:cardinality`
	`"n"^^xsd:nonNegativeInteger .`

DataHasMinCardinality(n DP Range)	`[] a owl:Restriction ;`
	`owl:onProperty` T(DP) ;
	`owl:minQualifiedCardinality`
	`"n"^^xsd:nonNegativeInteger ;`
	`owl:onClass` T(Range) .

DataHasMaxCardinality(n DP Range)	`[] a owl:Restriction ;`
	`owl:onProperty` T(DP) ;
	`owl:maxQualifiedCardinality`
	`"n"^^xsd:nonNegativeInteger ;`
	`owl:onClass` T(Range) .

DataExactCardinality(n DP Range)	`[] a owl:Restriction ;`
	`owl:onProperty` T(DP) ;
	`owl:qualifiedCardinality`
	`"n"^^xsd:nonNegativeInteger ;`
	`owl:onClass` T(Range) .

Traduction des descriptions de types de valeurs :

Declaration(Datatype(DT))	T(DT) `a rdfs:Datatype \verb.+`
DataIntersectionOf($DR_1 \dots DR_n$)	`[] a rdfs:Datatype ;` `owl:intersectionOf` $TLIST(DR_1, \dots DR_n)$.
DataUnionOf($DR_1 \dots DR_n$)	`[] a rdfs:Datatype ;` `owl:unionOf` $TLIST(DR_1, \dots DR_n)$.
DataComplementOf(DR)	`[] rdfs:Datatype ;` `owl:datatypeComplementOf` T(DR) .
DataOneOf(littéral$_1$ ... littéral$_n$)	`[] a rdfs:Datatype ;` `owl:oneOf` $TLIST$(littéral$_1$, ... littéral$_n$) .

DatatypeRestriction(DT F_1 littéral$_1$ ... F_1 littéral$_n$)	`[] a rdfs:Datatype ;`
	`owl:onDatatype T(DT) ;`
	`owl:withRestrictions (_:y1 ... _:yn) .`
	`_:y1` F_1 littéral$_1$.
	...
	`_:yn` F_n littéral$_n$.

Traduction des axiomes de description de propriétés objets :

Declaration(ObjectProperty(P))	T(P) `a owl:ObjectProperty \verb.+`
ObjectInverseOf(P)	`[] owl:inverseOf` T(P) .
SubObjectPropertyOf(P_1 P_2)	T(P_1) `rdfs:subPropertyOf` T(P_2) .
SubObjectPropertyOf(ObjectPropertyChain($P_1 \dots P_n$) P)	T(P) `owl:propertyChainAxiom` $TLIST$($P_1 \dots P_n$) .
ObjectPropertyDomain(P C)	T(P) `rdfs:domain` T(C) .
ObjectPropertyRange(P C)	T(P) `rdfs:range` T(C) .
InverseObjectProperties(P_1 P_2)	T(P_1) `owl:inverseOf` T(P_2) .
FunctionalObjectProperty(P)	T(P) `a owl:FunctionalProperty .`
InverseFunctionalObjectProperty(P)	T(P) `a owl:InverseFunctionalProperty .`
TransitiveObjectProperty(P)	T(P) `a owl:TransitiveProperty .`
SymmetricObjectProperty(P)	T(P) `a owl:SymmetricProperty .`
AymmetricObjectProperty(P)	T(P) `a owl:AymmetricProperty .`
ReflexiveObjectProperty(P)	T(P) `a owl:ReflexiveProperty .`
IrreflexiveObjectProperty(P)	T(P) `a owl:IrreflexiveProperty .`
EquivalentObjectProperties($P_1 \dots P_n$)	T(P_2) `owl:equivalentProperty` T(P_2) . ... T(P_{n-1}) `owl:equivalentProperty` T(P_n) .
DisjointObjectProperties(P_1 P_2)	T(P_1) `owl:propertyDisjointWith` T(P_2) .
DisjointObjectProperties($P_1 \dots P_n$), où n $>$ 2	`[] a owl:AlldisjointProperties ;` `owl:members` $TLIST$($P_1 \dots P_n$) .

Description de propriétés valeurs :

À venir

Description d'individus :

ClassAssertion(C individu)	T(individu) `a` T(C) .
ObjectPropertyAssertion(P individu$_1$ individu$_2$)	T(individu$_1$) T(P) T(individu$_2$) .
DataPropertyAssertion(DP individu littéral)	T(individu) T(DP) T(littéral) .
NegativeObjectPropertyAssertion(P individu$_1$ individu$_2$)	`[] a owl:NegativePropertyAssertion ;` `owl:sourceIndividual` T(individu$_1$) ; `owl:assertionProperty` T(P) ; `owl:targetIndividual` T(individu$_2$) ;
NegativeDataPropertyAssertion(DP individu littéral)	`[] a owl:NegativePropertyAssertion ;` `owl:sourceIndividual` T(individu) ; `owl:assertionProperty` T(DP) ; `owl:targetIndividual` T(littéral) ;
SameIndividual(individu$_1$ ... individu$_n$)	individu$_1$ `owl:sameAs` individu$_2$ . ... individu$_{n-1}$ `owl:sameAs` individu$_n$.
DifferentIndividuals(individu$_1$ individu$_2$)	individu$_1$ `owl:differentFrom` individu$_2$.
DifferentIndividuals(individu$_1$ ... individu$_n$)	`[] a owl:AllDifferent ;` `owl:members` $TLIST$(individu$_1$,...,individu$_n$) .

En terminant, nous allons voir un exemple de traduction en RDF d'une base de connaissances. Voici d'abord cette base de connaissances exprimée en logique descriptive :

Personne $\equiv \exists$nom.`xsd:string` $\sqcap$ (Femme $\sqcup$ Homme)
Livre $\sqsubseteq$ ProduitCulturel $\sqcap \exists$auteur.Personne $\sqcap \exists$titre.`xsd:string`
Livre(LIV203)
auteur(LIV203, JLB)
titre(LIV203, `"Ficciones"`)
Homme(JLB)
nom(JLB, `"Jorge Luis Borges"`)

Voici une version OWL de cette base de connaissances :

```
Prefix(local:=<http://www.polymtl.ca#>)
Ontology(
  Declaration( Class(local:Femme) )
  Declaration( Class(local:Homme) )
  Declaration( Class(local:ProduitCulturel) )
  Declaration( ObjectProperty(local:auteur) )
  Declaration( DataProperty(local:nom) )
  Declaration( DataProperty(local:titre) )

  EquivalentClasses(
     local:Personne
     ObjectIntersectionOf(
          DataSomeValuesFrom(local:nom xsd:string)
          ObjectUnionOf(local:Femme local:Homme)))

  SubClassOf(
     local:Livre
     ObjectIntersectionOf(
```

```
            local:ProduitCulturel
            ObjectSomeValuesFrom(local:auteur local:Personne)
            DataSomeValuesFrom(local:titre xsd:string)))

    ClassAssertion(local:Livre local:LIV203)
    ObjectPropertyAssertion(local:auteur local:Livre local:JLB)
    DataPropertyAssertion(local:titre local:Livre "Ficciones"^^xsd:string)

    ClassAssertion(local:Homme local:JLB)
    DataPropertyAssertion(local:nom local:JLB"Jorge Luis Borges"^^xsd:string)
)
```

Traduite en RDF, cette base de connaissances aura alors la forme suivante :

```
@prefix local: <http://www.polymtl.ca#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

local:Femme a owl:Class .
local:Homme a owl:Class .
local:ProduitCulturel a owl:Class .

local:auteur a owl:ObjectProperty .
local:nom a owl:DatatypeProperty .
local:titre a owl:DatatypeProperty .

local:Personne
   owl:equivalentClass
       [ a owl:Class;
         owl:intersectionOf
           ( [a owl:Restriction ;
               owl:onProperty local:nom ;
               owl:someValuesFrom xsd:string ]
              [a owl:Class ;
               owl:unionOf (local:Femme local:Homme ) ] ) ] .
local:Livre
   owl:subclassOf
       [ a owl:Class;
         owl:intersectionOf
           ( local:ProduitCulturel

             [a owl:Restriction ;
              owl:onProperty local:auteur ;
              owl:someValuesFrom local:Personne ]

             [a owl:Restriction ;
              owl:onProperty local:titre ;
              owl:someValuesFrom xsd:string ] ) ] .
```

```
local:LIV203
   a local:Livre ;
   local:auteur local:JLB ;
   local:titre "Ficciones"^^xsd:string .
local:JLB
   a local:Homme ;
   local:nom "Jorge Luis Borges"^^xsd:string .
```

Voici maintenant la même base de connaissances en RDF/XML :

```
<rdf:RDF xmlns="http://www.polymtl.ca#"
    xmlns:local="http://www.polymtl.ca#"
    xmlns:log="http://www.w3.org/2000/10/swap/log#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

    <owl:Class rdf:about="http://www.polymtl.ca#Femme">
    </owl:Class>

    <owl:Class rdf:about="http://www.polymtl.ca#Homme">
    </owl:Class>

    <Homme rdf:about="http://www.polymtl.ca#JLB">
        <nom rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Jorge Luis Borges</nom>
    </Homme>

    <Livre rdf:about="http://www.polymtl.ca#LIV203">
        <auteur rdf:resource="http://www.polymtl.ca#JLB"/>
        <titre rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Ficciones</titre>
    </Livre>

    <rdf:Description rdf:about="http://www.polymtl.ca#Livre">
        <owl:subclassOf rdf:parseType="Resource">
            <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
            <owl:intersectionOf rdf:parseType="Resource">
                <rdf:first rdf:resource="http://www.polymtl.ca#ProduitCulturel"/>
                <rdf:rest rdf:parseType="Resource">
                    <rdf:first rdf:parseType="Resource">
                        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Restriction"/>
                        <owl:onProperty rdf:resource="http://www.polymtl.ca#auteur"/>
                        <owl:someValuesFrom rdf:resource="http://www.polymtl.ca#Personne"/>
                    </rdf:first>
                    <rdf:rest rdf:parseType="Resource">
                        <rdf:first rdf:parseType="Resource">
                            <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Restriction"/>
                            <owl:onProperty rdf:resource="http://www.polymtl.ca#titre"/>
                            <owl:someValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                        </rdf:first>
                        <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
                    </rdf:rest>
                </rdf:rest>
            </owl:intersectionOf>
        </owl:subclassOf>
    </rdf:Description>

    <rdf:Description rdf:about="http://www.polymtl.ca#Personne">
        <owl:equivalentClass rdf:parseType="Resource">
            <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
            <owl:intersectionOf rdf:parseType="Resource">
                <rdf:first rdf:parseType="Resource">
```

```
                    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Restriction"/>
                    <owl:onProperty rdf:resource="http://www.polymtl.ca#nom"/>
                    <owl:someValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                </rdf:first>
                <rdf:rest rdf:parseType="Resource">
                    <rdf:first rdf:parseType="Resource">
                        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
                        <owl:unionOf rdf:parseType="Resource">
                            <rdf:first rdf:resource="http://www.polymtl.ca#Femme"/>
                            <rdf:rest rdf:parseType="Resource">
                                <rdf:first rdf:resource="http://www.polymtl.ca#Homme"/>
                                <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
                            </rdf:rest>
                        </owl:unionOf>
                    </rdf:first>
                    <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
                </rdf:rest>
            </owl:intersectionOf>
        </owl:equivalentClass>
    </rdf:Description>

    <owl:Class rdf:about="http://www.polymtl.ca#ProduitCulturel">
    </owl:Class>

    <owl:ObjectProperty rdf:about="http://www.polymtl.ca#auteur">
    </owl:ObjectProperty>

    <owl:DatatypeProperty rdf:about="http://www.polymtl.ca#nom">
    </owl:DatatypeProperty>

    <owl:DatatypeProperty rdf:about="http://www.polymtl.ca#titre">
    </owl:DatatypeProperty>
</rdf:RDF>
```

9.8 Exercices

■ Exercice 9.1

Voici en OWL une ébauche d'ontologie pour décrire des données géographiques :

```
Prefix(rdf:   = <http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Prefix(xsd:   = <http://www.w3.org/2001/XMLSchema#>)
Prefix(rdfs:  = <http://www.w3.org/2000/01/rdf-schema#>)
Prefix(owl:   = <http://www.w3.org/2002/07/owl#>)
Prefix(ex:    = <http://www.polymtl.ca#>)

Ontology(
 Declaration(Class(ex:Location))
 Declaration(Class(ex:PoliticalFunction))

 SubClassOf(ex:City ex:PoliticalLocation)
 SubClassOf(ex:Continent ex:GeographicalLocation)
 SubClassOf(ex:Country  ex:LargePoliticalLocation)
 SubClassOf(ex:GeographicalLocation ex:Location)
 SubClassOf(ex:Governor ex:HeadOfPoliticalEntity)
 SubClassOf(ex:HeadOfPoliticalEntity ex:PoliticalFunction)
 SubClassOf(ex:Lake ex:GeographicalLocation)
 SubClassOf(ex:LargePoliticalLocation ex:PoliticalLocation)
```

■ **Exercice 9.8**
a) On dit que le langage OLW Lite correspond à la famille $\mathcal{SHIF}(D)$. Expliquez pourquoi et dites aussi pourquoi la correspondance n'est pas exacte.

Références

[1] F. Baader and W. Nutt. Basic description logics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors, *The Description Logic Handbook : Theory, Implementation, and Applications*, pages 47–100. Cambridge University Press, 2003.

[2] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. *OWL Web Ontology Language Reference*. World Wide Web Consortium, http ://www.w3.org/TR/owl-ref/, 2004.