

Logique descriptive et OWL

Michel Gagnon

Table des matières

1	Introduction	3
2	Langage de base \mathcal{AL}	4
2.1	Syntaxe du langage \mathcal{AL}	4
2.2	Sémantique du langage \mathcal{AL}	6
2.3	Exercices	8
3	Langages de la famille \mathcal{AL}	8
3.1	Les constructeurs de la famille \mathcal{AL}	8
3.1.1	Constructeur d'union \mathcal{U}	8
3.1.2	Quantification existentielle complète \mathcal{E}	9
3.1.3	Constructeur de fonctions \mathcal{F}	9
3.1.4	Constructeur d'inversion \mathcal{I}	9
3.1.5	Restriction de cardinalité \mathcal{N}	10
3.1.6	Restriction de cardinalité qualifiée \mathcal{Q}	11
3.1.7	Négation sans restriction (\mathcal{C})	11
3.1.8	Remarques supplémentaires	12
3.2	Exercices	13
4	La ABox et la TBox	14
4.1	Terminologies	14
4.2	Description du monde	15
4.3	Déclaration d'individus dans une terminologie	16
5	Inférence	17
5.1	Réduction à l'insatisfaisabilité	18
5.2	Élimination de la TBox	18
5.3	Inférence avec la ABox	19
5.4	Monde fermé et monde ouvert	19

6	Algorithme de déduction par tableau	21
6.1	Description de l'algorithme	21
6.2	Description formelle des règles	23
6.3	Exercices	27
7	Inférence avec axiomes d'inclusion	31
7.1	Exercices	32
8	Famille \mathcal{SH}	33
9	Le langage OWL	33
9.1	Ontologies en OWL 2	35
9.2	Axiomes de base pour décrire des ontologies simples	36
9.3	Descriptions de classes complexes	39
9.4	Descriptions des propriétés	39
9.5	Déclaration de faits	40
9.6	Les différents profils de OWL 2	42
9.6.1	OWL Lite	42
9.6.2	OWL EL	45
9.6.3	OWL QL	45
9.6.4	OWL RL	46
9.7	Traduction en RDF	46
9.8	Exercices	54

1 Introduction

Ce que nous entendons par logique descriptive est en fait une famille de formalismes pour représenter une base de connaissances d'un domaine d'application. Plus spécifiquement, ces logiques permettent de représenter des concepts (aussi appelés *classes*) d'un domaine et les relations (aussi appelées *rôles*) qui peuvent être établies entre les instances de ces classes. Par exemple, on pourrait les utiliser pour représenter les concepts *humain*, *femme* et *mère*, et spécifier que toute mère est une femme qui est le parent d'au moins un humain. Comme toute logique, des mécanismes d'inférence y sont associés, permettant ainsi de déduire de nouveaux faits à partir d'une base de connaissances. Supposons par exemple qu'une base de connaissances définit le concept *mère* comme sous-classe du concept *humain*. Si on apprend qu'une entité est une mère, on peut automatiquement déduire qu'elle est aussi un humain. De même, avec la définition de mère ci-dessus, si un fait établit que qu'une femme a deux enfants, on peut déduire qu'elle est une mère.

Dans une base de connaissances en logique descriptive, on distingue deux composantes : la *TBox* et la *ABox*. La première contient tous les axiomes définissant les concepts du domaine, comme la définition du concept de mère présenté au paragraphe précédent. La *ABox* contient des assertions sur des individus, en spécifiant leur classe et leurs attributs. C'est dans la *ABox* qu'on indiquerait que Marie est une femme et qu'elle a deux enfants. Le type d'inférence réalisé avec la *TBox* diffère de celui réalisé avec la *ABox*.

Dans la *TBox*, on est généralement intéressé à savoir si tous les concepts définis sont consistants, c'est-à-dire si, pour chaque concept, il peut exister au moins un individu membre de cette classe. Par exemple, si on définit une classe comme étant à la fois une sous-classe des classes *homme* et *femme* et que la *TBox* spécifie aussi que ces deux classes sont disjointes (c'est-à-dire qu'aucune entité ne peut à la fois être un homme et une femme), on se retrouve alors avec un concept inconsistant. Un autre type d'inférence réalisé avec la *TBox* est la *subsumption*, qui consiste à déduire qu'une classe est une sous-classe d'une autre classe, même si cela n'est pas déclaré explicitement dans la base de connaissances. Par exemple si on spécifie que *humain* est une sous-classe de *animal*, on peut déduire qu'une mère est un animal.

Dans la *ABox*, on retrouve les assertions sur les individus. En d'autres mots, on y spécifie quelles sont les entités du monde et à quelle classe elles appartiennent. C'est dans la *ABox*, par exemple, qu'on spécifiera que MARIE est une mère, c'est-à-dire un individu qui est une *instance* de la classe *mère*. La *ABox* contient aussi des énoncés spécifiant les relations qui existent entre les individus. Ainsi, comme dans la *TBox* il sera spécifié qu'une mère doit avoir au moins un enfant, la *ABox* devra contenir au moins un autre individu, et une relation entre celui-ci et Marie indiquant qu'il est un des ses enfants.

Les inférences avec la *ABox* visent normalement à déterminer si un ensemble d'assertions est *consistant*, c'est-à-dire si un individu déclaré comme instance d'une classe peut réellement être une instance de cette classe et, similairement, si une relation déclarée entre deux individus est réellement possible. Supposons par exemple qu'une *TBox* déclare qu'un célibataire est une personne non mariée. La *ABox* sera inconsistante si elle contient un célibataire qui est marié avec une autre personne.

2 Langage de base \mathcal{AL}

Les langages de la logique descriptive sont déterminés par la forme des énoncés qui sont permis. La plupart des langages utilisés découlent du langage \mathcal{AL} (attributive language), dont l'expressivité est plutôt limitée.

2.1 Syntaxe du langage \mathcal{AL}

Dans ce langage, les axiomes sont construits à partir d'un ensemble de concepts. Ainsi, pour déclarer qu'un humain est un animal, on utiliserait les concepts atomiques **Humain** et **Animal** et on déclarerait l'axiome suivant :

$$\text{Humain} \sqsubseteq \text{Animal}$$

Mais ceci n'est pas très informatif. En fait, cet axiome ne fait que déclarer que les humains forment un sous-ensemble des animaux. Cela ne définit pas vraiment ce qu'est un humain. Pour ce faire, il faut citer d'autres caractéristiques. On sait par exemple qu'un humain est un animal qui raisonne. En définissant le concept **Raisonné**, nous pourrions donc définir le concept **Humain** de la manière suivante :

$$\text{Humain} \equiv \text{Animal} \sqcap \text{Raisonné}$$

Ici nous avons vraiment un énoncé plus précis, qui établit l'équivalence entre deux concepts : d'un côté le concept **Humain** qui représente l'ensemble des humains, et de l'autre le concept **Animal** \sqcap **Raisonné** qui représente l'ensemble des individus appartenant à la fois à la classe **Animal** et à la classe **Raisonné**.

Tout énoncé de la forme $C \equiv D$, tel que C est un concept atomique, est appelé *définition*. C'est ce type d'énoncé qu'on utilise pour créer une TBox.

En plus des concepts atomiques, que nous définissons nous-mêmes lorsque nous construisons une base de connaissances, le langage \mathcal{AL} contient deux classes spéciales : le concept universel \top qui représente tous les individus du monde représenté, et le concept impossible \perp . Par définition, pour tout concept C , on a l'axiome suivant :

$$C \sqsubseteq \top$$

Soit maintenant un concept C qui est impossible, c'est-à-dire qu'aucun individu ne peut appartenir à ce concept. Pour représenter cette situation, on utilise l'axiome suivant :

$$C \sqsubseteq \perp$$

Le langage \mathcal{AL} contient la négation, qui ne peut être appliquée qu'à un concept atomique. Ainsi, on peut représenter la classe des non humains, en écrivant $\neg\text{Humain}$, mais on ne peut pas

représenter la classe des individus qui ne sont pas des animaux raisonnables, c'est-à-dire la classe $\neg(\text{Animal} \sqcap \text{Raisnable})$.

Finalement, le langage \mathcal{AL} permet de définir un concept par des restrictions sur les relations. On peut par exemple définir la classe des individus dont tous les enfants sont des femmes, en utilisant l'énoncé $\forall a\text{Enfant.Femme}$. On peut définir la classe des individus qui ont au moins un enfant, en utilisant la formule suivante : $\exists a\text{Enfant}.\top$. Notez que le langage \mathcal{AL} ne permet pas de spécifier un concept avec le quantificateur existentiel. Par exemple, on ne peut pas définir la classe des individus qui ont au moins une fille, qui exigerait un énoncé de la forme suivante : $\exists a\text{Enfant.Femme}$. Seul le concept universel est permis avec le quantificateur existentiel. Voici un exemple plus complexe, utilisant des restrictions sur les relations, qui définit le concept d'un père qui n'a que des filles :

$$\text{Humain} \sqcap \exists a\text{Enfant}.\top \sqcap \forall a\text{Enfant.Femme}$$

Littéralement, ce concept représente l'ensemble des individus qui sont des humains ayant au moins un enfant et dont tous les enfants sont des femmes. À noter que l'énoncé suivant ne serait pas une bonne représentation :

$$\text{Humain} \sqcap \forall a\text{Enfant.Femme}$$

Le problème, c'est qu'un humain qui n'a pas d'enfants serait aussi une instance de ce concept. Supposons par exemple l'existence d'une personne qui n'a pas d'enfants. On ne peut pas dire qu'elle ne respecte pas la restriction. Pour ne pas la respecter, il faudrait trouver au moins un de ses enfants qui n'est pas une fille. Comme cette personne n'a pas d'enfants, on ne trouvera pas un tel contre-exemple, et la restriction est donc respectée.

Pour représenter une personne qui n'a pas d'enfants, nous devons restreindre la valeur de la relation $a\text{Enfant}$ au concept impossible :

$$\forall a\text{Enfant}.\perp$$

Pour appartenir à ce concept, un individu doit avoir tous ses enfants appartenant au concept impossible. Il ne peut donc pas avoir d'enfants.

En résumé, les descriptions possibles dans le langage \mathcal{AL} sont les suivantes (on suppose que A est un concept atomique et C et que D sont des concepts atomiques ou complexes) :

A	(concept atomique)
\top	(concept universel)
\perp	(concept impossible)
$\neg A$	(négation atomique)
$C \sqcap D$	(intersection de concepts)
$\forall R.C$	(restriction de valeur)
$\exists R.\top$	(quantification existentielle limitée)

2.2 Sémantique du langage \mathcal{AL}

La sémantique du langage \mathcal{AL} fait appel à la théorie des ensembles. Essentiellement, à chaque concept est associé un ensemble d'individus dénotés par ce concept. Une interprétation suppose donc l'existence d'un ensemble non vide Δ qui représente des entités du monde décrit. Nous avons une fonction d'interprétation \mathcal{I} qui associe à chaque description un sous-ensemble de Δ . Supposons que pour chaque concept atomique A , la fonction $\mathcal{I}(A)$ associe un sous-ensemble $A^{\mathcal{I}} \subseteq \Delta$, et pour chaque rôle atomique R , une relation binaire $R^{\mathcal{I}} \subseteq \Delta \times \Delta$. Voici comment est définie cette fonction d'interprétation pour les autres descriptions possibles :

$$\begin{aligned}\mathcal{I}(\top) &= \Delta \\ \mathcal{I}(\perp) &= \emptyset \\ \mathcal{I}(\neg A) &= \Delta \setminus A^{\mathcal{I}} \\ \mathcal{I}(C \sqcap D) &= \mathcal{I}(C) \cap \mathcal{I}(D) \\ \mathcal{I}(\forall R.C) &= \{a \in \Delta \mid \forall b.(a, b) \in \mathcal{I}(R) \rightarrow b \in \mathcal{I}(C)\} \\ \mathcal{I}(\exists R.\top) &= \{a \in \Delta \mid \exists b.(a, b) \in \mathcal{I}(R)\}\end{aligned}$$

On dit que deux concepts C et D sont équivalents, ce qui est dénoté par l'expression $C \equiv D$, si on a $\mathcal{I}(C) = \mathcal{I}(D)$, quelle que soit l'interprétation \mathcal{I} . Par exemple, il est assez simple de vérifier l'équivalence $\forall a \text{Enfant.Femme} \sqcap \forall a \text{Enfant.Médecin} \equiv \forall a \text{Enfant.}(\text{Femme} \sqcap \text{Médecin})$, c'est-à-dire que l'ensemble des personnes dont tous les enfants sont des femmes et dont tous les enfants sont des médecins est exactement le même que l'ensemble des personnes dont tous les enfants sont à la fois femme et médecin.

Il est à noter que, conformément à ces définitions, nous avons les équivalences suivantes :

$$\begin{aligned}\neg \top &\equiv \perp \\ \neg \perp &\equiv \top \\ C \sqcap \neg C &\equiv \perp \\ C \sqcup \neg C &\equiv \top\end{aligned}$$

Finalement, nous n'insisterons jamais suffisamment sur l'importance de bien comprendre la sémantique du quantificateur universel. En particulier, il ne faut jamais oublier que la description $\forall R.C$ représente aussi des entités qui ne participent pas du tout à la relation R . En effet, pour respecter une telle description, une entité i doit respecter une des deux conditions suivantes :

1. Pour tout objet j tel que cette entité est liée¹ à cet objet par la relation R , cet objet doit absolument appartenir à la classe C .
2. L'entité i n'est liée à aucun autre objet par la relation R .

Pour mieux comprendre la sémantique du langage \mathcal{AL} , considérons l'ensemble des définitions suivantes :

1. Ici, on considère que i est liée à j si on a (i, j) dans l'interprétation de R , en d'autres mot si on a une relation de i vers j .

$\text{Parent} \equiv \exists a \text{Enfant} . \top$

$\text{ParentDeFemme} \equiv \exists a \text{Enfant} . \top \sqcap \forall a \text{Enfant} . \neg \text{Homme}$

$\text{Célibataire} \equiv \forall \text{mariéAvec} . \perp$

$\text{HommeMarié} \equiv \text{Homme} \sqcap \exists \text{mariéAvec} . \top$

$\text{GrandParentChoyé} \equiv \forall a \text{Enfant} . (\exists a \text{Enfant} . \top)$

Voici une interprétation possible, qui réfère à un monde contenant sept entités (voir aussi la figure 1 :

$\Delta = \{a, b, c, d, e, f, g\}$

$\mathcal{I}(\text{Homme}) = \{a, b, c, g\}$

$\mathcal{I}(\text{aEnfant}) = \{(a, c), (b, d), (b, e), (c, g)\}$

$\mathcal{I}(\text{mariéAvec}) = \{(b, f), (f, b)\}$

Avec cette interprétation, on obtient les ensembles suivants pour chacune des classes définies :

$\text{Parent} : \{a, b, c\}$

$\text{ParentDeFemme} : \{b\}$

$\text{Célibataire} : \{a, c, d, e, g\}$

$\text{HommeMarié} : \{b\}$

$\text{GrandParentChoyé} : \{a, d, e, f, g\}$

On notera dans cette interprétation qu'on a plus d'individus dans la classe **GrandParentChoyé** que ce à quoi on s'attendait. C'est qu'il y a une erreur dans la description de ce concept.

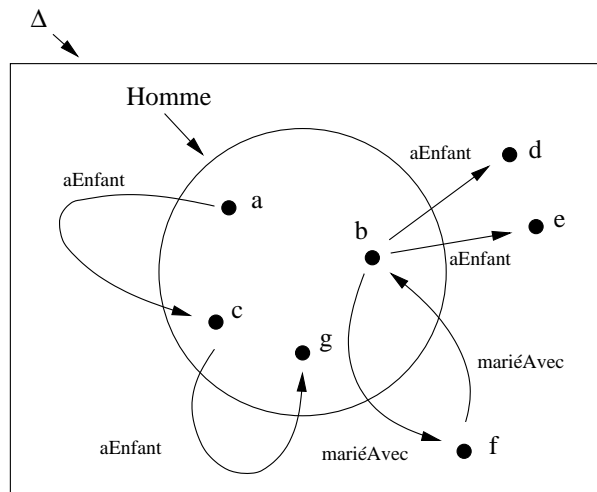


FIGURE 1 – Exemple d'interprétation pour le langage \mathcal{AL}

Voici maintenant une autre interprétation, qui n'est pas nécessairement ce que nous aurions en tête, mais qui est tout à fait correcte :

$$\begin{aligned}\Delta &= \{a\} \\ \mathcal{I}(\text{Homme}) &= \{\} \\ \mathcal{I}(\text{aEnfant}) &= \{\} \\ \mathcal{I}(\text{mariéAvec}) &= \{\}\end{aligned}$$

Il s'agit d'une interprétation qui correspond à un monde qui ne comprend qu'un seul individu qui est une femme célibataire n'ayant pas d'enfants.

2.3 Exercices

■ Exercice 3.1

Corrigez la description de `GrandParentChoyé` pour qu'elle soit plus conforme à la réalité.

3 Langages de la famille \mathcal{AL}

Le langage présenté dans la section précédente n'est généralement pas suffisamment expressif pour représenter les ontologies qu'on utilise en pratique. Il existe donc plusieurs autres constructeurs que l'on peut ajouter à notre langage pour le rendre plus expressif. Pour désigner chacun de ces constructeurs, on utilise un symbole.

3.1 Les constructeurs de la famille \mathcal{AL}

3.1.1 Constructeur d'union \sqcup

Le premier constructeur, indiqué par le symbole \sqcup , représente l'*union de concepts*. Ainsi, si on veut représenter l'ensemble des individus qui appartiennent soit à la classe C , soit à la classe D , on écrirait la description $C \sqcup D$ et dont l'interprétation est la suivante :

$$\mathcal{I}(C \sqcup D) = \mathcal{I}(C) \cup \mathcal{I}(D)$$

Ce type de construction est très pratique dans les restrictions de relations. Par exemple, on pourrait représenter un magasin qui ne vend que des chats et des chiens :

$$\text{Magasin} \sqcap \forall \text{vend.}(\text{Chat} \sqcup \text{Chien})$$

Selon la fonction d'interprétation telle que définie plus haut, pour appartenir à cette classe, une entité doit appartenir à la fois à l'ensemble des magasins et l'ensemble des entités telles que si elles vendent quelque chose, il s'agit nécessairement d'un chat ou d'un chien.

3.1.2 Quantification existentielle complète \mathcal{E}

Dans le langage \mathcal{AL} , on peut utiliser le quantificateur existentiel pour spécifier qu'une entité doit avoir au moins une relation avec un autre objet, mais on ne peut pas spécifier la classe de cet autre objet. Ceci limite fortement l'expressivité du langage. On ne peut pas, par exemple, définir la classe de ceux qui possèdent au moins un chien. Pour ce faire, il faut étendre le langage, en y ajoutant la *quantification existentielle complète* (représentée par le symbole \mathcal{E}), qui s'écrit $\exists R.C$, et dont l'interprétation est la suivante :

$$\mathcal{I}(\exists R.C) = \{a \in \Delta \mid \exists b. (a, b) \in \mathcal{I}(R) \wedge b \in \mathcal{I}(C)\}$$

On peut maintenant représenter la classe des gens qui possèdent au moins un chien :

$$\exists \text{possède.Chien}$$

3.1.3 Constructeur de fonctions \mathcal{F}

Un autre ajout utile (dénommé \mathcal{F}) est la possibilité de spécifier qu'un rôle est en fait une fonction, c'est-à-dire une relation telle que aucune entité ne peut être reliée à plus d'une autre entité par cette relation. Prenons par exemple la relation `mariéAvec`, qui nous permettrait de définir le concept `HommeMarié` :

$$\text{HommeMarié} \equiv \text{Homme} \sqcap \exists \text{mariéAvec}.\top$$

Si on ne spécifie rien de plus, rien n'empêche une instance de cette classe d'être mariée avec plus d'une personne. Pour empêcher cela, il nous faut une manière de spécifier que la relation `mariéAvec` en fait une fonction. On écrira donc un axiome de la forme $\text{Fun}(R)$ pour dire qu'un rôle R est une fonction.

3.1.4 Constructeur d'inversion \mathcal{I}

Dans plusieurs situations, on est intéressé à définir un rôle qui est l'inverse d'un autre rôle. Par exemple, si Maria regarde Paulo, on sait aussi que Paulo est regardé par Maria. Si on utilise deux relations `regarde` et `estRegardéPar`, on aimerait que tout fait de la forme `regarde(x, y)` implique nécessairement le fait `estRegardéPar(y, x)`. Pour ce faire, on utilisera le constructeur d'inversion (dénommée par \mathcal{I}) :

$$\text{estRegardéPar} \equiv \text{regarde}^-$$

Soit R un rôle, la sémantique du constructeur d'inversion est définie de la manière suivante :

$$\mathcal{I}(R^-) = \{(y, x) \mid (x, y) \in \mathcal{I}(R)\}$$

3.1.5 Restriction de cardinalité \mathcal{N}

Deux autres constructeurs, exprimant la *restriction de cardinalité* (dénommés \mathcal{N}), permettent de représenter des concepts comme *l'ensemble de ceux qui possèdent au moins deux chiens*, ou encore *l'ensemble de ceux qui possèdent au plus quatre chiens*. Ils ont les formes $\leq n R$ et $\geq n R$, et sont interprétés de la manière suivante :

$$\mathcal{I}(\geq n R) = \{a \in \Delta \mid |\{b \mid (a, b) \in \mathcal{I}(R)\}| \geq n\}$$

$$\mathcal{I}(\leq n R) = \{a \in \Delta \mid |\{b \mid (a, b) \in \mathcal{I}(R)\}| \leq n\}$$

Voici par exemple comment on peut décrire le concept de père qui a exactement deux enfants :

$$\text{Homme} \sqcap \geq 2 \text{aEnfant} \sqcap \leq 2 \text{aEnfant}$$

Il est intéressant de remarquer que si on limite les valeurs de cardinalité aux seules valeurs 0 ou 1, on se retrouve avec une logique \mathcal{ALCF} . Notons d'abord les équivalences suivantes :

$$\geq 1 R \equiv \exists R. \top$$

$$\geq 0 R \equiv \top$$

$$\leq 0 R \equiv \forall R. \perp$$

Pour signifier qu'une relation R est fonctionnelle, il suffit d'écrire l'axiome suivant :

$$\top \sqsubseteq \leq 1 R$$

Cet axiome spécifie que, quelle que soit l'entité du monde décrit, elle ne peut être liée à plus d'une entité par la relation R , ce qui impose donc que R soit une fonction (partielle). Ainsi, pour indiquer que la relation **mariéAvec** est fonctionnelle, on écrira l'axiome suivant :

$$\top \sqsubseteq \leq 1 \text{mariéAvec}$$

Remarquez que si on n'énonce pas l'axiome précédent, la description suivante n'implique pas que la relation **mariéAvec** soit une fonction :

$$\text{HommeMarié} \equiv \text{Homme} \sqcap \exists \text{mariéAvec}. \top \sqcap \leq 1 \text{mariéAvec}$$

Cet axiome spécifie qu'un homme marié doit être marié avec exactement une personne. Mais il s'agit d'une restriction qui ne s'applique qu'à cette description. On pourrait, par exemple, définir le concept de femme mariée de manière à permettre qu'une femme ait plusieurs maris, et ce en utilisant la même relation :

$$\text{FemmeMariée} \equiv \text{Femme} \sqcap \exists \text{mariéAvec}. \text{Homme}$$

Ainsi, rien n'empêcherait d'ajouter les assertions suivantes dans la ABox :

FemmeMariée(Ana)
 mariéAvec(Ana,Paulo)
 mariéAvec(Ana,Eduardo)
 Paulo \neq Eduardo

3.1.6 Restriction de cardinalité qualifiée \mathcal{Q}

Les deux constructeurs pour la restriction de cardinalité que nous venons de voir ont une limitation importante. En effet, comme la restriction s'applique à la relation, on ne peut pas imposer le nombre minimal ou maximal d'entités d'une *classe spécifique* auquel on peut être lié par une relation. Par exemple, si on a la relation *possède*, on ne pourra pas, dans une logique \mathcal{ALN} , décrire la classe des gens qui possèdent plus de deux chiens. Pour cela, il faut plutôt avoir accès à un constructeur de restriction de cardinalité *qualifiée* (dénommée \mathcal{Q}) :

$$\mathcal{I}(\geq n R.C) = \{a \in \Delta \mid |\{b \mid (a,b) \in \mathcal{I}(R) \wedge b \in \mathcal{I}(C)\}| \geq n\}$$

$$\mathcal{I}(\leq n R.C) = \{a \in \Delta \mid |\{b \mid (a,b) \in \mathcal{I}(R) \wedge b \in \mathcal{I}(C)\}| \leq n\}$$

Avec ces constructeurs à notre disposition, nous sommes maintenant en mesure de décrire la classe des gens qui possèdent deux chiens ou plus :

≥ 2 possède.Chien

3.1.7 Négation sans restriction (\mathcal{C})

Finalement, on peut étendre le langage en ajoutant la possibilité d'utiliser la négation sans aucune restriction. Cette caractéristique est désignée par le symbole \mathcal{C} (pour *complément*). Avec cet ajout il est devenu plus facile de décrire la classe des gens qui n'ont pas d'enfants de sexe masculin :

$\neg \exists a$ Enfant.Homme

On pourrait aussi représenter la classe de ceux qui n'ont aucun fils qui est marié :

$\forall a$ Enfant. \neg (Homme $\sqcap \geq 1$ mariéAvec)

Il est à noter que cette dernière extension du langage ne le rend pas plus expressif. En effet, on peut toujours tout ramener à la négation du langage \mathcal{AL} . Ainsi, les concepts précédents sont

équivalents aux concepts suivants :

$$\begin{aligned} & \forall a \text{Enfant}. \neg \text{Homme} \\ & \forall a \text{Enfant}. (\neg \text{Homme} \sqcup \forall \text{mariéAvec}. \perp) \end{aligned}$$

3.1.8 Remarques supplémentaires

Nous voici donc maintenant avec plusieurs nouveaux langages de la logique descriptive, selon la combinaison de constructeurs que l'on ajoute au langage \mathcal{AL} . Par exemple, si on ajoute l'union de concepts, on se retrouve avec le langage \mathcal{ALU} . Si on ajoute la quantification existentielle complète et la restriction de cardinalité, on obtient le langage \mathcal{ALEN} . Mais il est à noter que ces constructeurs ne sont pas tous indépendants. On a déjà vu que la négation étendue n'ajoute rien à l'expressivité, mais on peut aussi démontrer que \mathcal{ALC} et \mathcal{ALUE} sont équivalents en terme d'expressivité. Ceci revient à dire que la quantification existentielle complète et l'union de concepts ne sont pas nécessaires si on a la négation étendue.

Pour bien comprendre cela, il faut d'abord savoir qu'il y a plusieurs équivalences logiques dans ces langages :

$$\begin{aligned} \neg(C \sqcap D) & \equiv \neg C \sqcup \neg D \\ \neg(C \sqcup D) & \equiv \neg C \sqcap \neg D \\ \neg \exists R.A & \equiv \forall R. \neg A \\ \neg \forall R.A & \equiv \exists R. \neg A \\ \neg(\geq n R) & \equiv \leq (n-1) R \\ \neg(\leq n R) & \equiv \geq (n+1) R \\ \neg \neg C & \equiv C \end{aligned}$$

Pour mieux comprendre les troisième et quatrième équivalences, imaginez d'abord la classe des parents qui n'ont aucune fille, qui serait représentée de la manière suivante : $\neg \exists a \text{Enfant}. \text{Femme}$. Il est clair que pour tout membre de cette classe, il faut que chacun de ses enfants ne soit pas une femme : $\forall a \text{Enfant}. \neg \text{Femme}$. Si on considère maintenant la classe de ceux dont les enfants ne sont pas tous des femmes, soit $\neg \forall a \text{Enfant}. \text{Femme}$, on voit qu'on peut de manière équivalente la décrire comme l'ensemble des personnes qui ont au moins un enfant qui n'est pas une femme : $\exists a \text{Enfant}. \neg \text{Femme}$.

Ayant ces équivalences, il est aisé de voir qu'on peut exprimer l'union de concepts en n'utilisant que la négation et l'intersection : $C \sqcup D \equiv \neg(\neg C \sqcap \neg D)$. De même, la quantification existentielle n'est pas nécessaire si on a la quantification universelle : $\exists R.C \equiv \neg \forall R. \neg C$. On remarque par contre que l'on obtient des formules plus complexes et moins intuitives.

À noter qu'il nous reste à voir deux constructeurs importants (ceux de la famille \mathcal{O}), qui permettent d'introduire des individus dans des descriptions de concepts. Ils sont présentés à la section suivante.

3.2 Exercices

■ Exercice 3.1

Représentez les concepts suivants en logique descriptive :

- Ceux qui ne possèdent ni chien ni chat.
- Les hommes végétariens qui habitent en campagne.
- Les femmes qui n’aiment pas les chats.

■ Exercice 3.2

Écrivez un axiome pour indiquer que si un individu appartient à la classe A, il appartient aussi à la classe B ou C, mais jamais aux deux en même temps.

■ Exercice 3.3

Soient deux classes A et B. Écrivez en logique descriptive un axiome de subsomption pour indiquer que ces deux classes sont disjointes.

■ Exercice 3.4

Marie est une personne qui n’aime que les personnes qui n’aiment pas le fromage. Lequel (ou lesquels) des axiomes suivants représente correctement ce fait en logique descriptive :

- a) $(\text{Personne} \sqcap \forall \text{aime} . (\text{Personne} \sqcap \neg \forall \text{aime} . \text{Fromage}))(\text{MARIE})$
- b) $(\text{Personne} \sqcap \forall \text{aime} . (\text{Personne} \sqcap \forall \text{aime} . \neg \text{Fromage}))(\text{MARIE})$
- c) $(\text{Personne} \sqcap \forall \text{aime} . (\text{Personne} \sqcap \exists \text{aime} . \neg \text{Fromage}))(\text{MARIE})$

■ Exercice 3.5

Soit la description suivante en logique descriptive :

$$A \equiv \forall R . A \sqcap \exists R . A \sqcap \neg B$$

a) Dites, pour chacune des interprétations suivantes, si elle satisfait cette description (justifiez vos réponses) :

Interprétation 1 :

- $\mathcal{I}(A) = \{a, b, c, d\}$
- $\mathcal{I}(B) = \{e, f\}$
- $\mathcal{I}(R) = \{(a, b), (b, c), (c, d), (d, a), (e, a), (f, a)\}$

Interprétation 2 :

- $\mathcal{I}(A) = \{a, b, c\}$
- $\mathcal{I}(B) = \{e\}$
- $\mathcal{I}(R) = \{(a, a), (a, b), (a, c), (b, a), (b, c), (c, a), (e, e)\}$

b) Dites en termes simples ce que signifient les axiomes suivants :

1. $\top \sqsubseteq \forall R.C$
2. $\exists R.(\exists R.C) \sqsubseteq \exists R.C$

4 La ABox et la TBox

Nous avons déjà vu, dans l'introduction, qu'une base de connaissances en logique descriptive est composée d'une TBox, qui représente la terminologie d'un domaine, et d'une ABox qui déclare des individus particuliers dans ce domaine. Nous allons maintenant aborder plus formellement ces deux composantes.

4.1 Terminologies

Jusqu'à maintenant, nous avons vu plusieurs langages qui permettent d'écrire des formules complexes décrivant des concepts. Nous allons maintenant définir le concept de *terminologie*. Mais il nous faut d'abord définir le concept d'*axiome terminologique*, qui est en fait toute formule de la forme suivante :

$$C \sqsubseteq D \text{ ou } C \equiv D$$

La première forme déclare que toute entité de la classe C appartient aussi à la classe D , alors que la seconde indique que les concepts C et D sont équivalents, c'est-à-dire que si on appartient à la classe C , on appartient nécessairement à la classe D et vice versa. La sémantique de ces deux axiomes est sans surprise :

$$\begin{aligned}\mathcal{I}(C \sqsubseteq D) &= \text{vrai si } \mathcal{I}(C) \subseteq \mathcal{I}(D) \\ \mathcal{I}(C \equiv D) &= \text{vrai si } \mathcal{I}(C) = \mathcal{I}(D)\end{aligned}$$

Une *définition* est un axiome de la forme $C \equiv D$ où C est un concept atomique. Une définition sert en fait à associer un nom à un concept complexe. Nous avons déjà vu un exemple de définition, lorsque nous avons décrit le concept de *humain* :

$$\text{Humain} \equiv \text{Animal} \sqcap \text{Raisnable}$$

Nous pouvons donc maintenant définir une terminologie, ou TBox, comme un ensemble de définitions tel qu'un symbole n'apparaît jamais plus d'une fois du côté gauche (autrement dit, aucun concept n'est défini plus d'une fois). Et on dira qu'une interprétation \mathcal{I} *satisfait* une terminologie \mathcal{T} si elle est vraie pour toutes les définitions contenues dans \mathcal{T} . Dans ce cas, on dira aussi que \mathcal{I} est un *modèle* de \mathcal{T} .

Il est parfois difficile de définir un concept. Prenons par exemple les concepts **Homme** et **Femme**. Il n'est pas évident de définir exactement ces concepts. Ainsi, dans bien des applications, il peut être suffisant de dire qu'il sont tous les deux des sous-classes du concept **Humain**, ce qui

nous amènera plutôt à utiliser les axiomes suivants :

Homme \sqsubseteq Humain

Femme \sqsubseteq Humain

De tels axiomes sont appelés *axiomes de spécialisation*. Il est important de noter que ces axiomes, bien qu'utiles pour représenter nos ontologies, ne sont pas nécessaires. En effet, on pourrait très bien utiliser un concept arbitraire pour exprimer ce qui nous manque pour décrire entièrement un concept et l'ajouter à la description. Supposons par exemple le concept Homme, qui résume toutes les caractéristiques d'un homme que l'on ne veut pas ou ne sait pas décrire. On peut maintenant transformer notre axiome ci-dessus en une définition :

Homme \equiv Humain \sqcap Homme

Voici un exemple de terminologie, inspirée de [?] :

Femme \equiv Personne \sqcap Feminin

Homme \equiv Personne \sqcap \neg Femme

Mère \equiv Femme \sqcap $\exists a$ Enfant.Personne

Père \equiv Homme \sqcap $\exists a$ Enfant.Personne

Parent \equiv Père \sqcup Mère

Grandmère \equiv Mère \sqcap $\exists a$ Enfant.Parent

MèreDeFamilleNombreuse \equiv Mère \sqcap ≥ 3 aEnfant

MèreSansFille \equiv Mère \sqcap $\forall a$ Enfant. \neg Femme

Épouse \equiv Femme \sqcap \exists mariéAvec.Homme

Célibataire \equiv Personne \sqcap \forall mariéAvec. \perp

4.2 Description du monde

La seconde composante d'une base de connaissances, que nous avons appelée ABox, fournit une *description du monde*. La ABox introduit des *individus* en spécifiant leurs noms, les concepts auxquels ils appartiennent, et leurs relations avec d'autres individus. Soit par exemple des individus dont les noms sont a, b et c . On écrit $C(a)$ pour indiquer que a est un individu appartenant à l'ensemble dénoté par le concept C . On écrit $R(b, c)$ pour indiquer que b et c sont liés par la relation R ou, en d'autres mots, que l'individu c remplit le rôle R pour l'individu b . Supposons, par exemple, un monde constitué de trois individus dont les noms sont PEDRO, RODRIGO et CLAUDIA. On écrira Père(PEDRO) pour exprimer le fait que Pedro est un père, et aEnfant(CLAUDIA,RODRIGO) pour indiquer que Rodrigo est le fils de Claudia.

Nous savons déjà que la sémantique de la TBox est composée d'un ensemble Δ , qui énumère les entités du monde décrit, et d'une fonction d'interprétation \mathcal{I} , qui associe un sous-ensemble de Δ à chaque concept de la TBox, et un ensemble de paires d'entités de Δ pour chaque relation de la TBox. Pour associer une sémantique à la ABox, il suffit d'associer un élément de Δ à chaque nom qui s'y trouve. Plus formellement, si a est un nom introduit dans la ABox, on a :

$$\mathcal{I}(a) \in \Delta$$

On dit qu'une interprétation \mathcal{I} *satisfait* une assertion $C(a)$ si $\mathcal{I}(a) \in \mathcal{I}(C)$, autrement dit, il est vrai que a appartient à la classe C si l'entité qui lui correspond dans le monde fait partie de l'ensemble des objets du monde dénoté par C . De même, on dit qu'une interprétation \mathcal{I} satisfait une assertion de la forme $R(a, b)$ si $(\mathcal{I}(a), \mathcal{I}(b)) \in \mathcal{I}(R)$. Une interprétation \mathcal{I} satisfait une ABox \mathcal{A} si elle satisfait toutes les assertions de \mathcal{A} . Si une interprétation \mathcal{I} satisfait une ABox \mathcal{A} , on dit que \mathcal{I} est un *modèle* de \mathcal{A} . Finalement, on dira qu'une interprétation \mathcal{I} satisfait une assertion α ou une ABox \mathcal{A} *relativement* à une TBox \mathcal{T} si, en plus d'être un modèle de α ou \mathcal{A} , elle est aussi un modèle de \mathcal{T} .

4.3 Déclaration d'individus dans une terminologie

Jusqu'à maintenant, nous avons fait une distinction claire entre une terminologie, soit la TBox, qui sert à représenter les concepts d'un domaine et les relations entre ces concepts, et une description du monde, où sont déclarés les individus qui constituent un monde spécifique dans le domaine décrit. Dans certaines situations, on aimerait pouvoir désigner des individus dans la terminologie. Prenons par exemple le concept de membre permanent du Conseil de sécurité de l'ONU. La seule manière vraiment naturelle de définir cette classe est d'énumérer explicitement la liste des cinq pays qui en font partie. Avec ce que nous avons présenté jusqu'à maintenant, nous ne pourrions pas le définir de cette manière, puisqu'on ne peut pas déclarer des individus dans une terminologie. Il nous faudra donc un nouveau constructeur, l'*énumération* (dénoté par le symbole \mathcal{O} dans la famille des langages) :

$$\{a_1, \dots, a_n\}$$

où a_1, \dots, a_n sont des noms d'individus

Sans surprise, la sémantique associe à cette description l'ensemble des entités du monde qui correspondent aux noms a_1, \dots, a_n :

$$\mathcal{I}(\{a_1, \dots, a_n\}) = \{\mathcal{I}(a_1), \dots, \mathcal{I}(a_n)\}$$

Nous avons donc maintenant ce qu'il nous faut pour définir le concept de membre permanent du Conseil de sécurité :

$$\text{MembrePermanentConseilSécurité} \equiv \{\text{USA}, \text{FRANCE}, \text{RUSSIE}, \text{UK}, \text{CHINE}\}$$

Supposons maintenant que nous voulions représenter le concept de citoyen canadien, qui est défini de la manière suivante : personne née au Canada ou naturalisée par le Canada. Difficile de définir ce concept sans faire référence explicitement à l'entité Canada. Il nous faut donc un autre constructeur permettant de décrire l'ensemble des individus qui sont reliés à un individu spécifique par une relation R :

$R : a$

La sémantique de ce constructeur est la suivante :

$$\mathcal{I}(R : a) = \{d \in \Delta \mid (d, \mathcal{I}(a)) \in \mathcal{I}(R)\}$$

Voici comment, avec ce constructeur, on peut définir le concept de citoyen canadien :

$\text{CitoyenCanadien} \equiv \text{lieuNaissance : CANADA} \sqcap \text{naturaliséPar : CANADA}$

5 Inférence

En logique descriptive, il y a quatre propriétés qu'on peut être intéressé à prouver pour une TBox \mathcal{T} :

Satisfaisabilité : Un concept C est *satisfaisable* relativement à \mathcal{T} s'il existe un modèle \mathcal{I} de \mathcal{T} tel que $\mathcal{I}(C) \neq \emptyset$. En d'autres mots, un concept est satisfaisable s'il existe au moins une entité du monde décrit qui peut appartenir à l'ensemble décrit par ce concept. Par exemple, le concept $\text{Homme} \sqcap \neg \text{Homme}$ est insatisfaisable, puisque l'intersection d'un ensemble avec son complément est toujours l'ensemble vide. Il est donc impossible qu'une entité du monde appartienne à cette classe.

Subsommation : Un concept C est *subsumé* par un concept D relativement à \mathcal{T} si $\mathcal{I}(C) \subseteq \mathcal{I}(D)$ pour tout modèle \mathcal{I} de \mathcal{T} . Dans ce cas, on écrira $\mathcal{T} \models C \sqsubseteq D$. Par exemple, si une terminologie \mathcal{T} contient l'axiome $\text{Mère} \equiv \text{Femme} \sqcap \exists a \text{Enfant.Personne}$, il est relativement aisé de démontrer la subsommation suivante : $\text{Mère} \sqsubseteq \text{Femme}$. Autrement dit, si une entité appartient à l'ensemble des mères, elle appartient aussi à l'ensemble des femmes.

Équivalence : Deux concepts C et D sont *équivalents* relativement à \mathcal{T} si $\mathcal{I}(C) = \mathcal{I}(D)$ pour tout modèle \mathcal{I} de \mathcal{T} . Dans ce cas, on écrira $\mathcal{T} \models C \equiv D$.

"Disjointness" : Deux concepts C et D sont *disjoints* relativement à \mathcal{T} si $\mathcal{I}(C) \cap \mathcal{I}(D) = \emptyset$ pour tout modèle \mathcal{I} de \mathcal{T} . On désignera ce fait par l'énoncé suivant : $\mathcal{T} \models C \sqcap D \sqsubseteq \perp$. Par exemple, si on considère la terminologie fournie à la fin de la section 4.1, on peut démontrer que les concepts Père et Mère sont disjoints. En effet, selon la définition du concept Homme , un homme fait partie du complément de l'ensemble des femmes. Ainsi, une entité ne peut être à la fois un homme et une femme, ce qui implique qu'il ne peut pas être à la fois un père et une mère, puisque par définition un père est aussi un homme, et une mère est aussi une femme.

5.1 Réduction à l'insatisfaisabilité

En principe, nous devrions avoir un mécanisme de preuve pour chacune des propriétés énumérées à la section précédente. Mais en pratique, cela n'est pas nécessaire, puisqu'on peut tout ramener à une preuve d'insatisfaisabilité. C'est-à-dire que pour prouver chacune de ces propriétés, il suffit de la réécrire sous une autre forme et prouver que la nouvelle formule obtenue est insatisfaisable. Voici donc comment on fait cela :

Pour deux concepts C et D , on a :

- i. C est subsumé par $D \Leftrightarrow C \sqcap \neg D$ est insatisfaisable
- ii. C et D sont équivalents $\Leftrightarrow C \sqcap \neg D$ et $\neg C \sqcap D$ sont insatisfaisables
- iii. C et D sont disjoints $\Leftrightarrow C \sqcap D$ est insatisfaisable

Essayons maintenant de comprendre chacune de ces équivalences. Considérons d'abord le cas de la subsumption. Si $C \sqcap \neg D$ est insatisfaisable, cela signifie qu'il est impossible d'identifier dans le monde décrit une entité qui appartient à l'ensemble représenté par C et qui n'appartient pas à l'ensemble représenté par D . Donc, soit l'ensemble $\mathcal{I}(C)$ est vide, soit toutes les entités qu'il contient font aussi partie de $\mathcal{I}(D)$. Dans le premier cas, on a $\mathcal{I}(C) \subseteq \mathcal{I}(D)$ puisque par définition l'ensemble vide est un sous-ensemble de tous les ensembles. Dans le second cas, il est évident qu'on a $\mathcal{I}(C) \subseteq \mathcal{I}(D)$, puisque toute entité de $\mathcal{I}(C)$ doit aussi appartenir à $\mathcal{I}(D)$. Donc, dans tous les cas, on a $\mathcal{I}(C) \subseteq \mathcal{I}(D)$, ce qui confirme la subsumption.

Pour comprendre le cas de l'équivalence, il suffit de savoir que $C \equiv D$ est équivalent à $C \sqsubseteq D \sqcap D \sqsubseteq C$. Le dernier cas, celui de deux concepts disjoints, est lui aussi simple à comprendre. Il est en effet évident que si deux concepts sont disjoints, il ne peut y avoir aucune entité qui appartient à la fois aux deux ensembles représentés par ces concepts. L'intersection des deux concepts est donc insatisfaisable.

5.2 Élimination de la TBox

Pour développer des procédures d'inférence, comme la méthode des tableaux présentée plus loin, il est pratique d'utiliser comme point de départ des formules indépendantes de toute terminologie. Autrement dit, nous voulons éliminer la TBox. Si la TBox ne contient aucun cycle dans les définitions (ce qui sera le cas la plupart du temps), on y arrivera tout simplement en remplaçant tous les termes de la formule par leur définition dans la terminologie. Évidemment, si un terme de la formule n'a aucune définition dans la terminologie, il demeure inchangé. On répète ce processus jusqu'à ce que la formule obtenue ne contienne aucun terme qui possède une définition dans la terminologie.

Prenons par exemple le concept $\text{Femme} \sqcap \text{Homme}$ dont on voudrait démontrer l'insatisfaisabilité par rapport à la terminologie présentée à la fin de la section 4.1. Il faudrait alors éliminer la TBox en effectuant les étapes suivantes :

- (1) $\text{Femme} \sqcap \text{Homme}$
- (2) $\text{Personne} \sqcap \text{Feminin} \sqcap \text{Personne} \sqcap \neg \text{Femme}$
- (3) $\text{Personne} \sqcap \text{Feminin} \sqcap \text{Personne} \sqcap \neg (\text{Personne} \sqcap \text{Feminin})$

On peut alors utiliser une technique de preuve, comme celle montrée plus loin, pour prouver l'insatisfaisabilité de la formule (3).

5.3 Inférence avec la ABox

Jusqu'à maintenant, nous avons abordé certaines propriétés intéressantes d'une terminologie que nous voudrions pouvoir inférer. Nous allons maintenant voir le genre d'inférence qu'on est appelé à réaliser lorsque nous augmentons notre base de connaissances en y ajoutant des individus. En d'autres mots, nous nous intéresserons au genre de raisonnement qui concerne la ABox.

La première chose qu'on aimerait vérifier est la *consistance* de la ABox. Autrement dit, on voudrait s'assurer que la ABox ne contient aucune assertion $C(a)$ telle qu'il est impossible que a appartienne à la classe C . Par exemple, en reprenant notre terminologie de la section 4.1, nous aurions une base de connaissances inconsistante si nous avions les deux assertions suivantes dans la ABox : $\{\text{Mère(ANA)}, \text{Père(ANA)}\}$. En effet, selon la TBox, il est impossible qu'un individu soit à la fois une mère et un père.

Un autre type d'inférence que l'on peut être appelé à effectuer avec une ABox consiste à déterminer si un individu appartient à une certaine classe. Plus formellement, on écrira $\mathcal{A} \models C(a)$ si à partir d'une ABox \mathcal{A} on peut déduire que a appartient au concept C . Évidemment, si la ABox contient l'assertion $C(a)$, cette inférence est triviale, mais dans plusieurs cas, elle nécessite un raisonnement plus complexe. En guise d'exemple simple d'inférence non triviale, supposons une TBox \mathcal{T} qui contient l'unique axiome $C \equiv D \sqcap E$ et une ABox \mathcal{A} qui contient l'assertion $C(a)$. On peut alors inférer l'assertion $D(a)$. Autrement dit, on a : $\mathcal{A} \models D(a)$ relativement à la TBox \mathcal{T} . Pour vérifier $\mathcal{A} \models \alpha$, il est suffisant de vérifier que $\mathcal{A} \cup \{\neg\alpha\}$ est inconsistant. C'est d'ailleurs ce principe qui est utilisé pour faire les inférences avec la méthode du tableau que nous verrons à la section 6.

5.4 Monde fermé et monde ouvert

Supposons qu'une ABox utilisant la terminologie de la section 4.1 ne contient que l'assertion suivante :

$a\text{Enfant}(\text{PAULO}, \text{RODRIGO})$

On sait que Paulo a un enfant. Mais peut-on déduire qu'il n'a qu'un seul enfant, puisqu'un seul enfant est déclaré dans la ABox ? La réponse à cette question dépend de si la sémantique de notre langage utilise l'hypothèse du monde fermé ou l'hypothèse du monde ouvert. Avec l'hypothèse du monde fermé, on considère que le monde se limite à ce qui est énoncé. Autrement dit, dans notre exemple, Paulo n'aurait qu'un seul enfant puisque aucun autre enfant n'est déclaré. C'est cette hypothèse qui est normalement adoptée dans les bases de données.

En logique descriptive, c'est plutôt l'hypothèse du monde ouvert qui prévaut. Dans notre exemple, rien n'exclut que Paulo ait d'autres enfants, même si cela n'est pas spécifié dans la ABox. Pour indiquer que Paulo n'a qu'un seul enfant, il faudrait ajouter l'assertion suivante dans la ABox :

$(\leq 1 \text{ aEnfant})(\text{PAULO})$

Cette hypothèse du monde ouvert a un impact important dans la manière de faire des inférences en logique descriptive. Pour illustrer cela, considérons la ABox suivante, que nous dénommerons \mathcal{A}_{oe} :

aEnfant(JOCASTE,OEDIPE)
aEnfant(OEDIPE,POLYNICE)
aEnfant(JOCASTE,POLYNICE)
aEnfant(POLYNICE,THERSANDRE)
parricide(OEDIPE)
 \neg parricide(THERSANDRE)

Supposons maintenant que l'on désire savoir si Jocaste a un enfant qui est un parricide et qui a lui-même un enfant qui n'est pas un parricide. Autrement dit on veut vérifier le fait suivant :

$\mathcal{A}_{oe} \models (\exists \text{aEnfant.}(\text{Parricide} \sqcap \exists \text{aEnfant.}\neg(\text{Parricide})))(\text{JOCASTE})$

Si on utilise l'hypothèse du monde fermé, on sera amené à faire le raisonnement suivant :

Jocaste a deux enfants : Oedipe et Polynice. Par l'hypothèse du monde fermé, elle n'a pas d'autres enfants. Donc, au moins un de ces deux enfants doit répondre à la description spécifiée, c'est-à-dire être un parricide et avoir un enfant qui n'est pas un parricide. Considérons d'abord Oedipe, qui est un parricide. La ABox spécifie que Oedipe est le père de Polynice. Comme rien ne permet de déterminer si Polynice est ou n'est pas un parricide, l'hypothèse du monde fermé nous amène à déduire qu'il n'en est pas un. On a donc trouvé un cas qui respecte la description spécifiée.

Avec l'hypothèse du monde ouvert, on ne peut rien affirmer sur le fait que Polynice soit un parricide ou non. Par contre, par le principe du tiers exclu normalement utilisé en logique, on sait qu'il est l'un ou l'autre. On peut dans un premier temps voir ce que l'on peut déduire si on suppose que Polynice est un parricide, et ensuite considérer le cas où il n'en est pas un. Supposons donc que Polynice est un parricide. On sait qu'il est le fils de Jocaste et qu'il est le père de Thersandre, qui n'est pas un parricide. La description est donc respectée. Supposons maintenant qu'il n'est pas un parricide. Comme il est le fils de Oedipe, qui est lui-même un parricide et fils de Jocaste, la description est encore respectée. Ainsi, que Polynice soit un parricide ou non, la description est consistante. La preuve est donc faite.

Ce que nous montre cet exemple est que les preuves sont plus complexes avec l'hypothèse du monde ouvert. On est souvent appelé à envisager plusieurs situations alternatives afin de faire la preuve.

Un autre aspect important de la logique descriptive est qu'elle ne suppose pas l'unicité des noms. C'est-à-dire que deux noms différents ne signifient pas nécessairement qu'on a affaire à deux entités distinctes du monde décrit. Ainsi, la ABox suivante n'implique pas que Paulo a deux enfants, parce que rien ne nous garantit que Francisco et Chico réfèrent à deux entités différentes :

aEnfant(PAULO,FRANCISCO)
aEnfant(PAULO,CHICO)

Pour être sûr que deux entités différentes sont représentées, il faudrait ajouter l'axiome suivant à la ABox :

FRANCISCO \neq CHICO.

6 Algorithme de déduction par tableau

6.1 Description de l'algorithme

La meilleure méthode pour faire des inférences avec la logique descriptive est la méthode du tableau. Supposons que nous voulions prouver $C \sqsubseteq D$. Le principe de la démonstration est le suivant. On nie le fait à prouver et on tente de démontrer que ce nouveau fait est insatisfaisable. Dans notre cas, cela revient à démontrer que $C \sqcap \neg D$ est insatisfaisable.

Supposons par exemple que l'on veut démontrer que la classe $\exists\text{possède.}(\text{Livre} \sqcap \text{Antiquité})$ est subsumée par la classe $(\exists\text{possède.Livre} \sqcap \exists\text{possède.Antiquité})$, c'est-à-dire :

$$\exists\text{possède.}(\text{Livre} \sqcap \text{Antiquité}) \sqsubseteq (\exists\text{possède.Livre} \sqcap \exists\text{possède.Antiquité})$$

En d'autres mots, on veut prouver que si quelqu'un possède un livre ancien, il appartient nécessairement à la classe de ceux qui possèdent au moins un livre et au moins une antiquité. Pour ce faire il faut prouver que la classe suivante est insatisfaisable :

$$\exists\text{possède.}(\text{Livre} \sqcap \text{Antiquité}) \sqcap \neg(\exists\text{possède.Livre} \sqcap \exists\text{possède.Antiquité})$$

La première étape avant de commencer la preuve consiste à normaliser la description de manière à que toutes les négations se trouvent immédiatement à côté d'un identificateur de classe. On y arrive en appliquant les règles d'équivalence présentées à la section 3 :

$$\begin{aligned} &\exists\text{possède.}(\text{Livre} \sqcap \text{Antiquité}) \sqcap \neg(\exists\text{possède.Livre} \sqcap \exists\text{possède.Antiquité}) \\ &\exists\text{possède.}(\text{Livre} \sqcap \text{Antiquité}) \sqcap (\neg\exists\text{possède.Livre} \sqcup \neg\exists\text{possède.Antiquité}) \\ &\exists\text{possède.}(\text{Livre} \sqcap \text{Antiquité}) \sqcap (\forall\text{possède.}\neg\text{Livre} \sqcup \forall\text{possède.}\neg\text{Antiquité}) \end{aligned}$$

Pour faire notre preuve, on suppose un individu arbitraire, que l'on dénomme a , et qui est une instance de cette classe :

$$(1) (\exists\text{possède.}(\text{Livre} \sqcap \text{Antiquité}) \sqcap (\forall\text{possède.}\neg\text{Livre} \sqcup \forall\text{possède.}\neg\text{Antiquité}))(a)$$

Il est clair qu'un individu qui appartient à la classe $C \sqcap D$ appartient aussi aux classes C et D . Par exemple, si un individu est un homme brésilien, c'est-à-dire qu'il est une instance de la classe $\text{Homme} \sqcap \text{Brésilien}$, on peut aussi dire qu'il est un homme, d'une part, et un brésilien d'autre

part. Revenant à notre exemple précédent, on peut donc ajouter les deux faits suivants :

- (2) $(\exists \text{possède.}(\text{Livre} \sqcap \text{Antiquité}))(a)$
- (3) $(\forall \neg \text{possède.} \text{Livre} \sqcup \forall \neg \text{possède.} \text{Antiquité})(a)$

On sait maintenant que notre individu a possède un livre qui est une antiquité. On ne sait pas quel est cet objet, mais on sait qu'il existe. Appelons b ce nouvel individu. On peut alors écrire :

- (4) $\text{possède}(a, b)$
- (5) $(\text{Livre} \sqcap \text{Antiquité})(b)$

Par un raisonnement similaire à ce que nous avons fait précédemment, on peut ajouter les faits suivants :

- (6) $\text{Livre}(b)$
- (7) $\text{Antiquité}(b)$

Considérons maintenant le fait (3) que nous avons ajouté plus tôt. Il nous indique que a appartient à la classe de ceux qui ne possèdent aucun livre ou qui ne possèdent aucune antiquité. Autrement dit, au moins un des deux énoncés suivants doit être vrai :

- (8) $(\forall \text{possède.} \neg \text{Livre})(a)$
- (9) $(\forall \text{possède.} \neg \text{Antiquité})(a)$

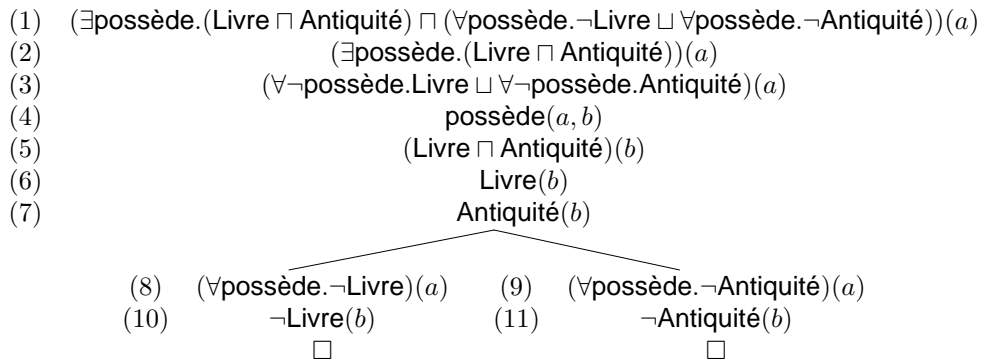
Supposons que (8) est vrai. On sait déjà que a possède b . L'objet ne peut donc pas être un livre :

- (10) $\neg \text{Livre}(b)$

Or, ceci contredit l'énoncé (6). Similairement, conformément à l'énoncé (9), on peut déduire :

- (11) $\neg \text{Antiquité}(b)$

Ceci contredit l'énoncé (7). Donc, quelle que soit la possibilité considérée, on arrive toujours à une contradiction, ce qui signifie que notre énoncé initial (1) est insatisfaisable. La subsomption est donc prouvée. Si l'on désigne par \square la contradiction, on peut représenter la preuve par un arbre, où chaque branchement représente des alternatives :



Ainsi, pour faire une preuve, on crée d'abord l'énoncé dont on veut montrer l'insatisfaisabilité. Ensuite, en appliquant une séquence de règles, on ajoute de nouveaux énoncés. On crée ainsi ce qu'on appelle un *tableau*, qui est en fait un arbre, à cause des branchements qui peuvent y apparaître, comme dans l'exemple précédent.

Lorsqu'on a ajouté la contradiction \square dans une branche, on dit qu'elle est fermée et on cesse d'en faire l'expansion. Lorsque toutes les branches d'un tableau sont fermées, on dit que le tableau est fermé et l'énoncé initial est considéré insatisfaisable. Si un tableau contient au moins une branche ouverte et qu'on ne réussit plus à appliquer des règles pour en faire l'expansion, le processus s'arrête et on conclut que l'énoncé initial n'est pas insatisfaisable.

6.2 Description formelle des règles

Soit un tableau et \mathcal{A} un chemin complet qui va de la racine de l'arbre à une feuille. Le tableau 1 fournit la liste des règles d'expansion pour la logique \mathcal{ALCN} .

Règle	Condition	Action
règle- \sqcap	\mathcal{A} contient $(C_1 \sqcap C_2)(x)$ et ne contient pas déjà les deux énoncés $C_1(x)$ et $C_2(x)$.	On ajoute à \mathcal{A} les énoncés $C_1(x)$ et $C_2(x)$
règle- \sqcup	\mathcal{A} contient $(C_1 \sqcup C_2)(x)$ et ne contient aucun des deux énoncés $C_1(x)$ et $C_2(x)$.	On ajoute à \mathcal{A} le branchement suivant : $\begin{array}{c} \diagup \quad \diagdown \\ C_1(x) \quad C_2(x) \end{array}$
règle- \exists	\mathcal{A} contient $(\exists R.C)(x)$ et il n'existe aucun individu z tel que $R(x, z)$ et $C(z)$ sont aussi dans \mathcal{A}	On ajoute $R(x, y)$ et $C(y)$ à \mathcal{A} , où y est un nom d'individu qui n'existe pas déjà dans \mathcal{A}
règle- \forall	\mathcal{A} contient $(\forall R.C)(x)$ et $R(x, y)$, mais ne contient pas $C(y)$	On ajoute $C(y)$ à \mathcal{A}
règle- \geq	\mathcal{A} contient $(\geq n R)(x)$ et il n'y a pas dans \mathcal{A} des individus z_1, \dots, z_n qui sont tous distincts (c'est-à-dire qu'on doit avoir explicitement dans \mathcal{A} l'énoncé $z_i \neq z_j$ pour chaque paire possible avec cet ensemble d'individus) et qui sont tels que \mathcal{A} contient la relation $R(x, z_i)$ pour tous ces individus ($1 \leq i \leq n$).	Soit un ensemble de n individus dénotés par y_1, \dots, y_n , qui sont des noms qui n'existent pas dans \mathcal{A} . On ajoute à \mathcal{A} les énoncés $y_i \neq y_j$ pour chaque paire possible avec cet ensemble, ainsi que les énoncés $R(x, y_i)$ pour ($1 \leq i \leq n$).
règle- \leq	\mathcal{A} contient $(\leq n R)(x)$ et les énoncés $R(x, y_1), \dots, R(x, y_{n+1})$. Il n'existe aucune identité $y_i = y_j$ dans \mathcal{A} pour ($1 \leq i \leq n+1$), ($1 \leq j \leq n+1$), $i \neq j$	Pour chaque paire possible (y_i, y_j) d'individus parmi y_1, \dots, y_{n+1} , on ajoute une nouvelle branche avec $y_i \neq y_j$.

TABLE 1 – Règles de tableau pour la logique \mathcal{ALCN}

Pour chaque règle appliquée, on ajoute les nouveaux énoncés à la fin à partir de la branche où on se trouve. On les ajoute généralement à la fin de tous les branchements à partir du point où on se trouve. Supposons par exemple le tableau de la figure 2a, où le traitement de l'énoncé A entraîne l'ajout de l'énoncé B. Il faudra alors étendre le tableau de la manière illustrée à la figure 2b. Similairement, lorsqu'on doit ajouter un branchement dans un tableau, on l'ajoute à toutes

les branches existantes à partir du point où la règle s'applique. Supposons par exemple que le traitement de l'énoncé A , dans le tableau de la figure 2a, entraîne l'ajout d'un branchement, avec l'énoncé B d'un côté et l'énoncé C de l'autre côté. Le tableau résultant sera celui illustré à la figure 2c.

On ajoute la contradiction \square à la fin d'un chemin \mathcal{A} lorsqu'une des situations suivantes se présente :

- \mathcal{A} contient à la fois un énoncé de la forme $C(x)$ et son complément $\neg C(x)$
- \mathcal{A} contient un énoncé de la forme $C(x)$, un énoncé de la forme $\neg C(y)$ et une des identités $x = y$ ou $y = x$.
- \mathcal{A} contient un énoncé de la forme $\perp(x)$ (qui, rappelons-le, signifie que l'individu x ne peut pas exister)

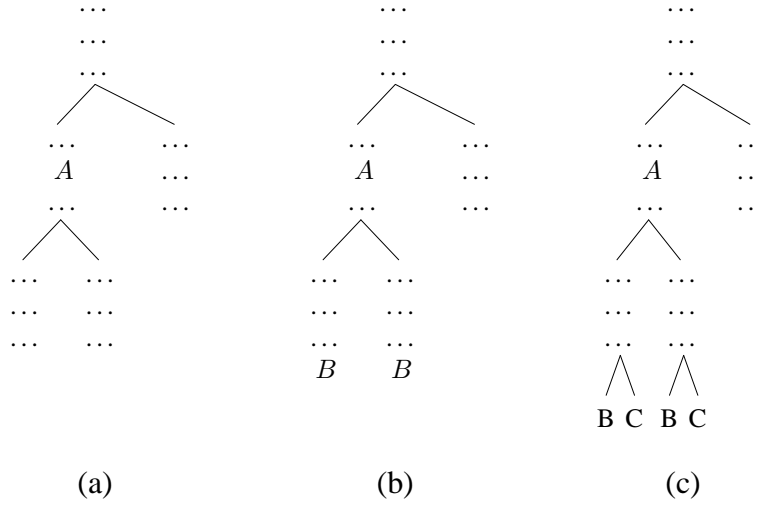


FIGURE 2 – Exemples d'expansion de tableau

Les quatre premières règles d'expansion de tableau ont déjà été vues dans l'exemple de preuve ci-haut. Il nous reste maintenant à voir des exemples d'application des deux dernières. Supposons d'abord une terminologie qui contient la définition suivante :

$$\text{Parent} \equiv \exists a \text{Enfant}. \top$$

Nous tenterons maintenant de démontrer que quelqu'un qui a plus de deux enfants est un parent, c'est-à-dire la subsomption suivante :

$$\geq 2 \text{ aEnfant} \sqsubseteq \text{Parent}$$

Pour ce faire, il faut d'abord écrire le fait dont on veut démontrer l'insatisfaisabilité :

$$\geq 2 \text{ aEnfant} \sqcap \neg \text{Parent}$$

Puis il faut éliminer la TBox :

$$\geq 2 \text{ aEnfant} \sqcap \neg(\exists \text{ aEnfant}.\top)$$

Finalement, il faut normaliser l'énoncé :

$$\geq 2 \text{ aEnfant} \sqcap \forall \text{ aEnfant}.\perp$$

On peut maintenant commencer la preuve par tableau qui, au moment de l'application de la règle- \geq , sera dans l'état suivant :

- (1) $(\geq 2 \text{ aEnfant} \sqcap \forall \text{ aEnfant}.\perp)(a)$
- (2) $(\geq 2 \text{ aEnfant})(a)$
- (3) $(\forall \text{ aEnfant}.\perp)(a)$

Ici, d'après l'énoncé (2), on sait que l'individu a doit avoir au moins deux enfants. Or, dans le tableau, ceux-ci ne sont pas explicités. C'est exactement ce que la règle- \geq nous permet de faire :

- (1) $(\geq 2 \text{ aEnfant} \sqcap \forall \text{ aEnfant}.\perp)(a)$
- (2) $(\geq 2 \text{ aEnfant})(a)$
- (3) $(\forall \text{ aEnfant}.\perp)(a)$
- (4) $\text{aEnfant}(a, b)$
- (5) $\text{aEnfant}(a, c)$
- (6) $b \neq c$

La prochaine étape d'expansion applique la règle- \forall à l'énoncé (3), ce qui entraîne l'ajout de deux énoncés spécifiant que les individus b et c ne peuvent pas exister, d'où l'introduction de la contradiction \square qui ferme le tableau :

- (1) $(\geq 2 \text{ aEnfant} \sqcap \forall \text{ aEnfant}.\perp)(a)$
- (2) $(\geq 2 \text{ aEnfant})(a)$
- (3) $(\forall \text{ aEnfant}.\perp)(a)$
- (4) $\text{aEnfant}(a, b)$
- (5) $\text{aEnfant}(a, c)$
- (6) $b \neq c$
- (7) $\perp(b)$
- (8) $\perp(c)$
- (9) \square

Supposons maintenant la terminologie suivante :

$$\text{Parent} \equiv \exists \text{ aEnfant}.\top$$

$$\text{ParentDeFamilleNombreuse} \equiv \text{Parent} \sqcap \geq 3 \text{ aEnfant}$$

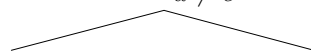
Nous voulons maintenant démontrer qu'un parent de quatre enfants ou plus est un parent de famille nombreuse, soit :

$$\geq 4 \text{ aEnfant} \sqsubseteq \text{ParentDeFamilleNombreuse}$$

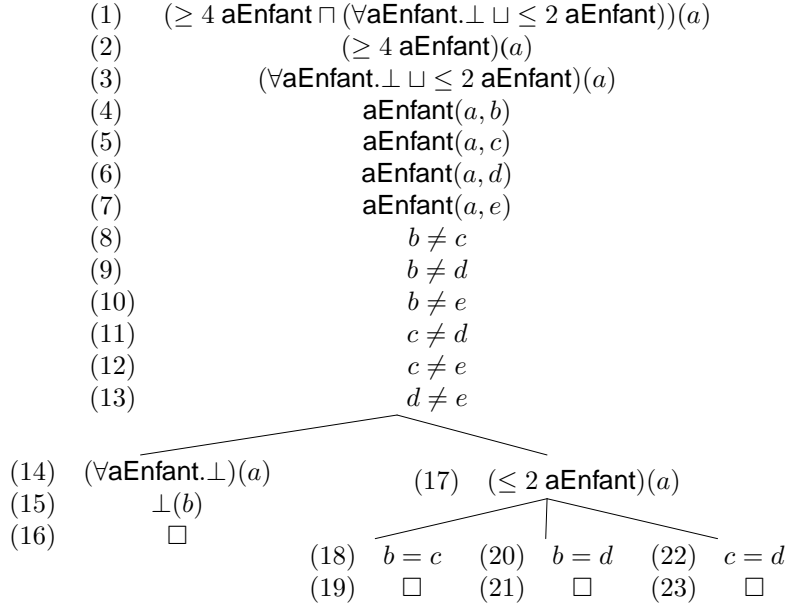
Pour ce faire, on démarre le tableau avec le fait $(\geq 4 \text{ aEnfant} \sqcap \neg \text{ParentDeFamilleNombreuse})(a)$ que l'on normalise et réécrit de la manière suivante après élimination de la TBox :

$$(\geq 4 \text{ aEnfant} \sqcap (\forall \text{aEnfant} . \perp \sqcup \leq 2 \text{ aEnfant}))(a)$$

Remarquez qu'ici nous avons appliqué la règle d'équivalence $\neg(\geq n R) \equiv \leq (n - 1) R$. Voici l'état du tableau au moment d'appliquer la règle- \leq :

(1)	$(\geq 4 \text{ aEnfant} \sqcap (\forall \text{aEnfant} . \perp \sqcup \leq 2 \text{ aEnfant}))(a)$
(2)	$(\geq 4 \text{ aEnfant})(a)$
(3)	$(\forall \text{aEnfant} . \perp \sqcup \leq 2 \text{ aEnfant})(a)$
(4)	$\text{aEnfant}(a, b)$
(5)	$\text{aEnfant}(a, c)$
(6)	$\text{aEnfant}(a, d)$
(7)	$\text{aEnfant}(a, e)$
(8)	$b \neq c$
(9)	$b \neq d$
(10)	$b \neq e$
(11)	$c \neq d$
(12)	$c \neq e$
(13)	$d \neq e$
	
(14)	$(\forall \text{aEnfant} . \perp)(a)$
(15)	$\perp(b)$
(16)	\square
(17)	$(\leq 2 \text{ aEnfant})(a)$

Considérons maintenant les individus b , c et d . La règle- \leq nous dit qu'il faut créer trois nouvelles branches, qui seront toutes rapidement fermées :



6.3 Exercices

■ Exercice 6.1

Soit la classe MonChien définie de la manière suivante :

$$\text{MonChien} \equiv \text{Chien} \sqcap \forall \text{ aime} . \forall \text{ aime} . \text{Chat} \sqcap \forall \text{ aime} . (\text{Chien} \sqcap \forall \text{ aime} . \perp)$$

Prouvez l'axiome de subsomption suivant :

$$\text{MonChien} \sqsubseteq \forall \text{ aime} . \forall \text{ aime} . \text{Chien} \sqcap \text{Chien} \sqcap \forall \text{ aime} . \text{Chien}$$

■ Exercice 6.2

Soit la base de connaissances suivante :

$$\begin{aligned} &R(a, b) \\ &(\forall R . (\exists S . C) \sqcap \leq 1 R)(a) \\ &(\forall R . (\forall S . D))(a) \end{aligned}$$

Prouvez que l'assertion suivante est une conséquence logique de cette base de connaissances :

$$(\exists S . (C \sqcap D))(b)$$

■ Exercice 6.3

Prouvez $\neg \exists R . \top \sqsubseteq \forall R . (\forall R . \top)$.

■ Exercice 6.4

Soit la terminologie suivante :

$$\begin{aligned}
A &\equiv B \sqcap C \\
B &\equiv D \sqcup E \\
C &\equiv F \sqcup \neg E
\end{aligned}$$

Prouvez l'axiome de subsomption suivant :

$$A \sqsubseteq D \sqcup F$$

■ Exercice 6.5

Soit la base de connaissances suivante :

$$\begin{aligned}
&(\exists R.D \sqcap \leq 2R)(a) \\
&R(a, b) \\
&R(a, c) \\
&U(b) \\
&(\neg U \sqcap \neg D)(c)
\end{aligned}$$

Prouvez que l'assertion suivante est une conséquence logique de cette base de connaissances :

$$D(b)$$

■ Exercice 6.6

Déterminez si les axiomes de subsomption suivants sont vrais :

- a) $\forall R.A \sqcap \forall R.B \sqcap \geq 1R \sqsubseteq \exists R.(A \sqcap B)$
- b) $\exists R.A \sqcap \exists R.B \sqcap \leq 1R \sqsubseteq \exists R.(A \sqcap B)$
- c) $\exists R.A \sqcap \exists R.B \sqsubseteq \exists R.(A \sqcap B)$
- d) $\exists R.A \sqcap \exists R.B \sqsubseteq \exists R.(A \sqcup B)$

■ Exercice 6.7

Soit la base de connaissances suivante :

$$\begin{aligned}
\text{Woman} &\equiv \text{Person} \sqcap \text{Female} \\
\text{Man} &\equiv \text{Person} \sqcap \neg \text{Female}
\end{aligned}$$

Prouvez que l'assertion suivante est une conséquence logique de cette base de connaissances :

$$\text{Person} \sqsubseteq \text{Woman} \sqcup \text{Man}$$

■ Exercice 6.8

Soit la base de connaissances suivante :

$\text{ProteinLoverPizza} \equiv \text{Pizza} \sqcap \forall \text{hasTopping}.(\text{Meat} \sqcap \text{Fish})$
 $\text{MeatyPizza} \equiv \text{Pizza} \sqcap \forall \text{hasTopping}.\text{Meat}$
 $\text{VegetarianPizza} \equiv \text{Pizza} \sqcap \forall \text{hasTopping}.(\neg \text{Meat} \sqcup \neg \text{Fish})$

a) Prouvez que l'assertion suivante est une conséquence logique de cette base de connaissances :

$\text{ProteinLoverPizza} \sqsubseteq \text{MeatyPizza}$

b) Supposons que l'axiome suivant est ajouté à la base de connaissances, pour indiquer que les classe Meat et Fish sont disjointes :

$\text{Meat} \sqcap \text{Fish} \sqsubseteq \perp$

Prouvez que l'assertion suivante est une conséquence logique de cette base de connaissances :

$\text{ProteinLoverPizza} \sqsubseteq \text{VegetarianPizza}$

c) Comment devrions-nous corriger la base de connaissances pour que l'assertion en b) soit pas une conséquence logique de la base de connaissances ?

■ Exercice 6.9

Soit la base de connaissances suivante :

$\text{aEnfant}(\text{JOCASTE}, \text{OEDIPE})$
 $\text{aEnfant}(\text{JOCASTE}, \text{POLYNICE})$
 $\text{aEnfant}(\text{OEDIPE}, \text{POLYNICE})$
 $\text{aEnfant}(\text{POLYNICE}, \text{THERSANDRE})$
 $\text{Parricide}(\text{OEDIPE})$
 $\neg \text{Parricide}(\text{THERSANDRE})$
 $(\geq 2 \text{ aEnfant} \sqcap \leq 2 \text{ aEnfant})(\text{JOCASTE})$
 $(\geq 1 \text{ aEnfant} \sqcap \leq 1 \text{ aEnfant})(\text{OEDIPE})$
 $(\geq 1 \text{ aEnfant} \sqcap \leq 1 \text{ aEnfant})(\text{POLYNICE})$

a) Prouvez que l'assertion suivante est une conséquence logique de cette base de connaissances :

$(\exists \text{aEnfant}.(\text{Parricide} \sqcap \exists \text{aEnfant}.\neg \text{Parricide}))(\text{JOCASTE})$

b) Prouvez que la base de connaissances devient inconsistante si on ajoute l'assertion suivante :

$(\exists \text{aEnfant}.(\neg \text{Parricide} \sqcap \exists \text{aEnfant}.\text{Parricide}))(\text{JOCASTE})$

■ Exercice 6.10

Prouvez l'axiome de subsomption suivant :

$$\forall R.A \sqsubseteq \neg \exists R.\top \sqcup \exists R.A$$

■ Exercice 6.11

Soit l'ontologie suivante :

$$\begin{aligned} A &\sqsubseteq B \sqcup C \sqcup D \\ B &\equiv \forall R.(E \sqcap F) \sqcap \geq 1R \\ C &\equiv \exists R.(E \sqcap \neg F) \\ D &\equiv \exists R.(F \sqcap \neg E) \end{aligned}$$

Prouvez l'axiome de subsomption suivant :

$$A \sqsubseteq \exists R.(E \sqcup F)$$

■ Exercice 6.12

Soit l'ontologie suivante :

$$\begin{aligned} A &\sqsubseteq B \\ B &\equiv C \sqcap \exists R.A \\ C &\equiv D \sqcap \exists R.B \end{aligned}$$

Prouvez l'axiome de subsomption suivant :

$$A \sqsubseteq D$$

Attention : Pour faire la preuve, il faut d'abord normaliser le premier axiome, c'est-à-dire le transformer en équivalence en ajoutant une classe arbitraire \bar{A} :

$$A \equiv B \sqcap \bar{A}$$

■ Exercice 6.13

Soit l'ontologie suivante, représentée en logique descriptive :

$$\begin{aligned} A &\equiv \neg B \sqcap C \\ D &\equiv \forall R.A \sqcup \forall S.B \\ E &\equiv D \sqcap \exists S.A \\ F &\equiv D \sqcap \exists S.B \end{aligned}$$

À l'aide de la méthode du tableau, déterminez si $E \sqsubseteq \neg F$.

■ Exercice 6.14

Prouvez que si A et B sont deux classes disjointes, cela implique logiquement l'axiome suivant :

$$\exists R.A \sqcap \exists R.B \sqsubseteq \geq 2R$$

■ Exercice 6.15

Soient les axiomes suivants :

$$\begin{aligned} A &\sqsubseteq B \\ A &\sqsubseteq C \end{aligned}$$

a) Montrez, seulement en considérant la sémantique, que l'axiome $A \sqsubseteq (B \sqcap C)$ est une conséquence logique de cette terminologie.

b) Faites la même démonstration, mais cette fois en utilisant la technique de preuve par tableau.

7 Inférence avec axiomes d'inclusion

Dans la méthode d'inférence présentée à la section précédente, on ne peut traiter que des axiomes d'équivalence. Or, une ontologie peut aussi contenir des axiomes d'inclusion. Prenons par exemple l'ontologie suivante, qui contient un axiome d'inclusion pour spécifier le domaine d'un rôle :

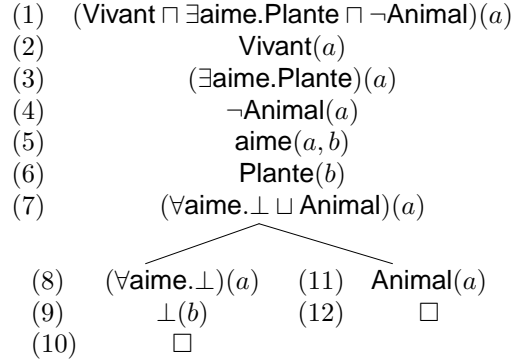
$$\begin{aligned} \text{Plante} &\equiv \text{Vivant} \sqcap \exists \text{aime.Plante} \\ \exists \text{aime.T} &\sqsubseteq \text{Animal} \end{aligned}$$

Selon la sémantique de la logique descriptive, l'axiome suivant est (bizarrement !) une conséquence logique de cette ontologie : $\text{Plante} \sqsubseteq \text{Animal}$. Autrement dit, les plantes sont des animaux ! Malheureusement, on n'arrivera jamais à prouver cela avec la technique de preuve par tableau vue précédemment.

Pour pouvoir raisonner avec ce type d'axiome, il faut faire un petit ajout au mécanisme d'inférence. Il faudra d'abord transformer tout axiome d'inclusion de la forme $C \sqsubseteq D$ en un axiome équivalent $\top \sqsubseteq \neg C \sqcup D$. Dans la preuve par tableau, on pourra alors, pour tout individu i qui y est introduit, ajouter le fait $(\neg C \sqcup D)(i)$. Avec cet ajout, on peut maintenant faire notre preuve par tableau. Voici d'abord l'ontologie transformée :

$$\begin{aligned} \text{Plante} &\equiv \text{Vivant} \sqcap \exists \text{aime.Plante} \\ \top &\sqsubseteq \forall \text{aime}.\bot \sqcup \text{Animal} \end{aligned}$$

Il nous faut donc maintenant prouver l'insatisfaisabilité de $\text{Plante} \sqcap \neg \text{Animal}$. Après élimination de la TBox, on produit la preuve suivante :



7.1 Exercices

■ Exercice 7.1

Soit l'ontologie suivante :

$$\begin{aligned} \text{Plante} &\equiv \text{Vivant} \sqcap \exists \text{aime.Plante} \\ \exists \text{aime}.\top &\sqsubseteq \text{Animal} \\ \text{Plante} \sqcap \text{Animal} &\sqsubseteq \perp \end{aligned}$$

Montrez que le concept **Plante** est insatisfaisable.

■ Exercice 7.2

Soit l'ontologie suivante :

$$\begin{aligned} A &\sqsubseteq B \\ B &\equiv C \sqcap \exists R.A \\ C &\equiv D \sqcap \exists R.B \end{aligned}$$

Prouvez l'axiome de subsomption suivant :

$$A \sqsubseteq D$$

■ Exercice 7.3

Soit l'ontologie suivante :

$$\begin{aligned} C_1 &\equiv C_2 \sqcap \exists R_1.(C_3 \sqcup C_4) \\ &\geq 1R_1 \sqsubseteq C_5 \\ \top &\sqsubseteq \forall R_1.C_6 \\ C_3 &\equiv C_0 \sqcap \forall R_2.C_0 \\ C_4 &\equiv C_0 \sqcap \exists R_2.C_0 \end{aligned}$$

En utilisant la méthode du de tableau démontrez l'axiome de subsomption suivant :

$$C_1 \sqsubseteq \exists R_1.(C_0 \sqcap C_6)$$

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 concerne 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 $\mathcal{I}(\mathbf{aEnfant})$, elle appartient aussi à $\mathcal{I}(\mathbf{aDescendant})$. 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 [?], 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

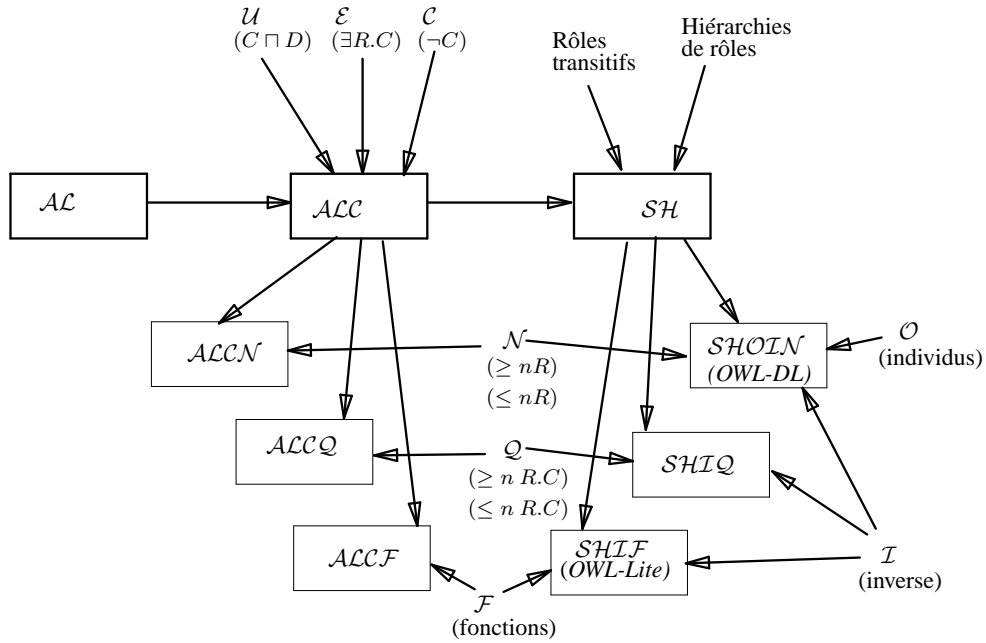


FIGURE 3 – Familles de logiques descriptives

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 subsumption 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 connaissance 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.

Similairement, 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 valeur est un littéral (DataProperty). Voici un exemple 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 `DataPropertyAssertion`. 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 individu 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 concept de classe en RDF, la distinction de deux types de propriétés. Nous allons maintenant aborder des éléments de OWL qui nous mènent nettement 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 entités, alors qu'on peut le faire en OWL. Voici, 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 individu 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

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 :

Célibataire \equiv **Personne** $\sqcap \leq 0$ **mariéAvec**
Végétarien \equiv **Personne** $\sqcap \forall$ **mange.Vegetal**

En OWL-Lite :

```
Prefix(local:=<http://www.polymtl.ca#>)
Ontology(
  EquivalentClasses(
    local:Celibataire
    ObjectIntersectionOf(
      local:Personne
      ObjectMaxCardinality(0 local:mariéAvec)))
  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 *SHIF* est un peu abusive. En effet, OWL-Lite ne contient pas tout le pouvoir expressif de la famille *ALC*, à partir de laquelle la famille *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 *ALC*, qui accepte aussi des descriptions complexes à cette position. Par exemple, l'axiome suivant ne pourrait pas être 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 : $(A \sqcap B) \sqsubseteq \perp$. 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 *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 $\sqsubseteq \exists \text{nom.xsd:string} \sqcap \exists \text{age.xsd:int}$
 Homme \sqsubseteq Personne
 Femme \sqsubseteq Personne
 Père \equiv Personne $\sqcap \exists \text{aEnfant.Personne}$
 PèreDeFilles \equiv Père $\sqcap \forall \text{aEnfant.Femme}$
 $\top \sqsubseteq \forall \text{aEnfant.Personne (image)}$
 $\exists \text{aEnfant.}\top \sqsubseteq$ Personne (domaine)
 $\text{aEnfant} \equiv \text{aParent}^-$ (rôle inverse)
 $\top \sqsubseteq \leq 1 \text{ estConjointDe}$ (rôle fonctionnel)
 $\top \sqsubseteq \leq 1 \text{ estConjointDe}^-$ (rôle injectif)
 $\text{estConjointDe} \equiv \text{estConjointDe}^-$ (rôle symétrique)
 $\top \sqsubseteq \forall \text{estConjointDe.Personne (image)}$
 $\exists \text{estConjointDe.}\top \sqsubseteq$ Personne (domaine)
 mariéAvec \sqsubseteq estConjointDe
 $\top \sqsubseteq \forall \text{age.xsd:int (image)}$
 $\exists \text{age.}\top \sqsubseteq$ Personne (domaine)
 $\top \sqsubseteq \forall \text{nom.xsd:string (image)}$
 $\exists \text{nom.}\top \sqsubseteq$ 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

À venir

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 ₁ ... 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 $item_1, item_2 \dots item_n$, la fonction $TLIST(item_1, item_2 \dots item_n)$ 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 C_2)	$T(C_1)$ rdfs:subClassOf $T(C_2)$.
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 $TLIST(individu_1, \dots individu_n)$.

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 $n \geq 2$	[] a owl:Restriction ; owl:onProperties <i>TLIST</i> (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 $n \geq 2$	[] a owl:Restriction ; owl:onProperties <i>TLIST</i> (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 ... DR _n)	[] a rdfs:Datatype ; owl:intersectionOf <i>TLIST</i> (DR ₁ , ... DR _n) .
DataUnionOf(DR ₁ ... DR _n)	[] a rdfs:Datatype ; owl:unionOf <i>TLIST</i> (DR ₁ , ... DR _n) .
DataComplementOf(DR)	[] rdfs:Datatype ; owl:datatypeComplementOf T(DR) .
DataOneOf(littéral ₁ ... littéral _n)	[] a rdfs:Datatype ; owl:oneOf <i>TLIST</i> (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 ... P _n) P)	T(P) owl:propertyChainAxiom <i>TLIST</i> (P ₁ ... 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 ... 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 ... P _n), où n > 2	[] a owl:AlldisjointProperties ; owl:members <i>TLIST</i> (P ₁ ... 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 <i>TLIST</i> (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  $\equiv \exists \text{nom.xsd:string} \sqcap (\text{Femme} \sqcup \text{Homme})$ 
Livre  $\sqsubseteq \text{ProduitCulturel} \sqcap \exists \text{auteur.Personne} \sqcap \exists \text{titre.xsd:string}$ 
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)

```

```

SubClassOf(ex:Mayor ex:HeadOfPoliticalEntity)
SubClassOf(ex:MountainRange ex:GeographicalLocation)
SubClassOf(ex:Ocean ex:GeographicalLocation)
SubClassOf(ex:PoliticalLocation ex:Location)
SubClassOf(ex:President ex:HeadOfPoliticalEntity)
SubClassOf(ex:PrimeMinister ex:HeadOfPoliticalEntity)
SubClassOf(ex:Province ex:LargePoliticalLocation)
SubClassOf(ex:State ex:LargePoliticalLocation)
)

```

Complétez et/ou modifiez cette ontologie pour qu'on puisse s'en servir afin de créer des bases de connaissances permettant de répondre aux questions suivantes :

- Qui est le premier ministre du Canada ?
- Quelle est la capitale de l'Ontario ?
- Quelles sont les villes des États-Unis qui sont situées sur la côte atlantique ?
- Quelles sont les villes du Brésil dont la population est supérieure à 3 millions d'habitants ?
- Quelles sont les plus belles plages du Brésil qui ne sont pas trop proches d'une grande ville ?
- Quelles sont les villes les plus intéressantes pour un amateur d'arts ?

■ Exercice 9.2

Soit l'ontologie suivante :

```

<?xml version="1.0"?>
<rdf:RDF
  xmlns="http://www.polymtl.ca/exemple#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.polymtl.ca/exemple">
  <owl:Ontology rdf:about="" />
  <rdfs:Class rdf:ID="A">
    <owl:unionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:someValuesFrom rdf:resource="#A"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="R1"/>
        </owl:onProperty>
      </owl:Restriction>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#R1"/>
        </owl:onProperty>
        <owl:allValuesFrom rdf:resource="#A"/>
      </owl:Restriction>
    </owl:unionOf>
  </rdfs:Class>
  <owl:ObjectProperty rdf:about="#R1">

```

```

    <a rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  </owl:ObjectProperty>
</rdf:RDF>

```

a) On remarque que cette ontologie n'est pas dans la famille OWL Lite. Elle a exactement une caractéristique qui la fait appartenir à la famille OWL DL Identifiez cette caractéristique.

b) Dessinez le graphe RDF correspondant à cette ontologie.

■ Exercice 9.3

Soit une ontologie pour représenter le domaine des livres, utilisée entre autres par les maisons d'édition pour divulguer sur le web leur catalogues. Voici, en syntaxe abstraite de OWL, un extrait de cette ontologie, pour représenter les romans :

```

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(foaf: = <http://xmlns.com/foaf/0.1/>)
Prefix(ex:   = <http://www.polymtl.ca/exemple#>)

Ontology( <http://www.polymtl.ca/livres>
  Import( <foaf:index.rdf>)

  Declaration(Class(foaf:Organization))
  Declaration(Class(foaf:Person))

  EquivalentClasses(
    ex:Document
    ObjectIntersectionOf(
      ObjectSomeValuesFrom(ex:hasAuthor foaf:Person)
      DataSomeValuesFrom(ex:hasTitle xsd:string))

  SubClassOf(ex:Novel ex:PublishedDocument)

  EquivalentClasses(
    ex:PublishedDocument
    ObjectIntersectionOf(
      ex:Document
      ObjectSomeValuesFrom(ex:hasPublisher ex:Publisher)))

  SubClassOf(ex:Publisher foaf:Organization)

  ObjectPropertyRange(ex:hasAuthor foaf:Person)
  ObjectPropertyRange(ex:hasPublisher ex:Publisher)

  DataPropertyRange(ex:hasTitle xsd:string)
  FunctionalDataProperty(ex:hasTitle)
)

```


Nous savons que le livre «Agaguk», écrit par Yves Thériault, a été publié aux Éditions Hexagone. Cette information pourrait être diffusée en ajoutant les faits suivants :

```
ClassAssertion(ex:Novel ex:AGAGUK)
ObjectPropertyAssertion(ex:hasAuthor ex:AGAGUK ex:Yves_Therriault)
ObjectPropertyAssertion(ex:hasPublisher ex:AGAGUK ex:Hexagone)
DataPropertyAssertion(ex:hasTitle ex:AGAGUK "Agaguk"^^xsd:string)
ClassAssertion(ex:Publisher ex:Hexagone)
ClassAssertion(foaf:Person ex:Yves_Therriault)
```

a) On sait que ce livre, comme bien d'autres, a été traduit en plusieurs langues. Modifiez l'ontologie pour que cette situation puisse être adéquatement représentée. Toute maison d'édition, à n'importe quel endroit dans le monde, devrait pouvoir utiliser cette ontologie pour représenter un livre traduit et publié dans une autre langue. En particulier, il faudrait que la maison d'édition Tashenbuch, qui le publie en allemand, puisse elle aussi le représenter sur son site. Il faudrait aussi que l'ontologie permette à un agent de retrouver facilement les informations suivantes :

- La traduction d'un livre dans une certaine langue, si cette traduction existe ;
- Le livre original, lorsqu'on consulte une information qui représente un livre traduit ;
- Le traducteur d'un livre.

b) À partir de cette ontologie, créez une base de connaissances qui contient toutes les instances requises pour représenter le livre «Agaguk» dans sa version originale en français et sa traduction en allemand. Assurez-vous que les instances de cette base de connaissances soient reliées par les propriétés adéquates.

■ Exercice 9.4

Choisissez un domaine et écrivez une ontologie la plus expressive possible tout en vous limitant à OWL 2 EL.

■ Exercice 9.5

Prenez l'ontologie de l'exercice précédent et modifiez-la pour la rendre la plus expressive possible tout en vous limitant à OWL 2 DL.

■ Exercice 9.6

On veut représenter un concept de quatuor à corde qui contient exactement quatre instruments à corde. Mais on aimerait spécifier qu'il ne s'agit pas de n'importe quelle combinaison d'instruments, mais plutôt celle-ci : deux violons, un violoncelle (cello) et un contrebasse (bass). Pour ce faire, on propose la description suivante :

```
<owl:ObjectProperty rdf:ID="composed_of_instruments"/>
<owl:Class rdf:ID="String_quartet">
  <rdfs:subClassOf>
    <owl:Restriction>
```

```

    <owl:onProperty rdf:resource="#composed_of_instruments"/>
    <owl:hasValue rdf:parseType="Collection">
      <Violin/>
      <Violin/>
      <Cello/>
      <Bass/>
    </owl:hasValue>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#Quartet"/>
</owl:Class>

```

- Dans quel sous-langage de OWL 2 cette ontologie est-elle écrite ?
- Dites si cette représentation exprime bien ce que l'on veut (justifiez votre réponse).

■ Exercice 9.7

Représentez en logique descriptive la base de connaissances suivante :

```

<?xml version="1.0"?>
<rdf:RDF xmlns="http://www.owl-ontologies.com/Ontology1166107887.owl#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#">

  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="A"/>
  <owl:Class rdf:ID="B">
    <rdfs:subClassOf rdf:resource="#A"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#R"/>
        <owl:allValuesFrom rdf:resource="#B"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#R"/>
        <owl:someValuesFrom>
          <owl:Restriction>
            <owl:onProperty rdf:resource="#R"/>
            <owl:someValuesFrom rdf:resource="#C"/>
          </owl:Restriction>
        </owl:someValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="#C"/>
  </owl:Class>
  <owl:Class rdf:ID="C">
    <rdfs:subClassOf rdf:resource="#A"/>
    <owl:disjointWith rdf:resource="#B"/>
  </owl:Class>
  <owl:ObjectProperty rdf:ID="R">
    <rdfs:domain rdf:resource="#A"/>
    <rdfs:range rdf:resource="#A"/>
  </owl:ObjectProperty>
</rdf:RDF>

```

■ Exercice 9.8

- 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.