Sémantique de RDF

Michel Gagnon

Table des matières

1 Introduction

Pour utiliser à bon escient une description RDF, il n'est pas suffisant de bien comprendre la syntaxe. Il faut aussi bien comprendre la sémantique de chaque construction utilisée, afin de s'assurer que ceux qui utilisent ces données aient bien la même interprétation de leur signification que ceux qui les ont créées.

Pour présenter la sémantique de RDF, nous procéderons en trois étapes. Dans un premier temps, nous verrons l'interprétation simple de RDF, c'est-à-dire une interprétation qui ne tient aucunement compte du vocabulaire utilisé. Ensuite, nous contraindrons plus la sémantique en expliquant comment l'interprétation prend en compte les éléments du vocabulaire RDF (c'est-à-dire les éléments `rdf:type`, `rdf:Property`, `rdf:List`, `rdf:Statement`, etc). Finalement, nous ajouterons l'interprétation du vocabulaire RDFS.

1.1 Interprétation

1.2 Interprétation simple

Pour interpréter un langage, il faut d'abord identifier les entités du monde auxquelles les constructions du langage se réfèrent. Dans le cas de RDF, cet univers comprend trois ensembles :

IR **:** Un ensemble non vide de ressources.

IP **:** Un ensemble de propriétés.

LV **:** Un sous-ensemble de IR qui représente les valeurs qui seront attribuées au littéraux simples.

La figure 1 illustre la relation entre ces trois ensembles. On remarque que puisque $\mathrm{LV} \subset \mathrm{IR}$, les valeurs des littéraux sont des entités qui existent, puisqu'elles sont des ressources. Mais à noter que ceci n'implique pas qu'un littéral puissent être identifié par un IRI. En fait, RDF ne permet pas cela. Du moins, pas de manière directe.

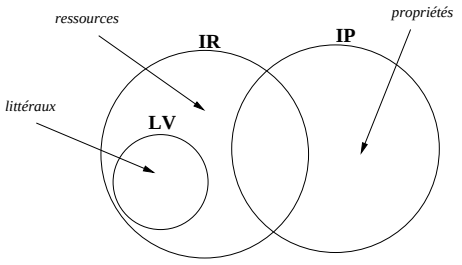

FIGURE 1 – Ensembles de l'univers pour interprétation simple d'un vocabulaire RDF

Une fois l'univers défini, il faut définir une interprétation I, qui fait le lien entre chaque construction d'un graphe RDF et les entités comprises dans cet univers. Toute interprétation supppose d'abord un vocabulaire V qui constitue l'ensemble des noms que l'on peut retrouver dans un graphe RDF, c'est-à-dire les IRI et les littéraux.

Toute interprétation suppose aussi l'existence des fonctions suivantes :

— $\mathrm{I}_{\mathrm{EXT}} : \mathrm{IP} \mapsto \mathcal{P}(\mathrm{IR} \times \mathrm{IR})$, qui associe à chaque propriété de IP l'ensemble des paires $\langle x, y \rangle$ telles que les ressources x et y appartiennent à la relation définie par la propriété en question.
— $\mathrm{I}_{\mathrm{S}} : \mathrm{IRI} \mapsto \mathrm{IR} \cup \mathrm{IP}$, qui associe chaque IRI à une ressource ou une propriété.
— $\mathrm{I}_{\mathrm{L}} : \mathcal{L} \mapsto \mathrm{IR}$ où $\mathcal{L}$ est l'ensemble des littéraux typés. Cette fonction associe chaque littéral typé à sa valeur.

Contrairement à ce que l'on pourrait imaginer, l'image I_{L} est l'ensemble des ressources, plutôt que l'ensemble LV. La raison de ce choix est qu'il est possible qu'on retrouve dans un graphe RDF un littéral mal formé selon le type qui est spécifié, comme ce serait le cas par exemple avec

l'expression `"chien"^^xsd:int`. Dans ce cas, la sémantique spécifie que la valeur associée soit une entité arbitraire qui n'est pas un littéral.

Avec ces définitions, nous pouvons maintenant fournir une interprétation d'un graphe RDF ne contenant aucun noeud vide. Soit V le vocabulaire utilisé dans le graphe RDF. Une interprétation retourne une valeur de vérité pour un graphe donné (la notation N-Triple est utilisée pour désigner les triplets du graphe) :

— Si E est un graphe ne contenant aucun noeud vide, $I(E) = faux$ si $I(E')$ est faux pour au moins un triplet E' contenu dans E. Sinon $I(E) = vrai$.
— Si E est un triplet `s p o .` ne contenant aucun noeud vide, alors $I(E) = vrai$ si `s`, `p` et `o` font partie de V, $I(p) \in IP$ et $\langle I(s), I(o)\rangle \in I_{EXT}(I(p))$. Sinon, $I(E) = faux$.
— Si E est un littéral simple `"aaa"` qui fait partie de V, alors $I(E) = aaa$.
— Si E est un littéral simple `"aaa"@ll` qui fait partie de V, où `ll` spécifie la langue, alors $I(E) = \langle aaa, ll\rangle$.
— Si E est un littéral typé et $E \in V$, alors $I(E) = I_L(E)$.
— Si E est un IRI et $E \in V$, alors $I(E) = I_S(E)$.

Avec cette définition d'interprétation, la valeur obtenue est $faux$ aussitôt qu'un graphe contient au moins un nom qui n'est pas dans le vocabulaire. Autrement dit, si un nom est utilisé dans graphe RDF, il faut absolument qu'il existe une entité correspondante dans l'univers. On remarquera aussi que la première condition implique qu'un graphe vide est trivialement vrai.

Pour terminer la présentation de l'interprétation simple, il ne nous reste plus qu'à voir comment on peut traiter les graphes qui contiennent au moins un noeud vide. L'idée consiste ici à traiter ces noeuds comme les variables existentielles en logique des prédicats. Ces noeuds vides doivent nécessairement désigner une entité de l'univers, sauf qu'on ne sait pas laquelle.

Pour interpréter un graphe contenant des noeuds vides, il suffit d'avoir une fonction A qui associe chaque noeud vide à une des ressources de l'ensemble IR. Ainsi, on peut définir une nouvelle interprétation $I + A$, qui est égale à l'interprétation I pour tout sauf les noeuds vides :

$$[I + A](E) = \begin{cases} A(E), & \text{si } E \text{ est un noeud vide,} \\ I(E), & \text{sinon.} \end{cases}$$

Pour généraliser notre définition d'interprétation à un graphe pouvant contenir des noeuds vides, il suffit d'ajouter le cas suivant :

— Si E est un graphe contenant des noeuds vides, alors $I(E) = vrai$ s'il existe une fonction A telle que $[I + A](E) = vrai$. Sinon, $I(E) = faux$.

Une des conséquences importantes de la manière dont est définie l'interprétation simple est qu'il est possible que deux propriétés aient la même extension, tout en étant distinctes. Supposons, par exemple, un univers comprenant les propriétés "posséder" et "acheter" et tel que tout objet possédé par quelqu'un a aussi été acheté par celui-ci. Les deux propriétés auront donc exactement

les mêmes paires dans leur extension. Mais cela ne signifie aucunement que ces deux propriétés sont équivalentes.

Une autre conséquence est que rien n'empêche une propriété d'être elle-même incluse dans son extension. Comme nous le verrons plus loin, c'est le cas avec la propriété `rdf:type`, qui peut être appliquée à elle-même.

1.3 Interprétation simple et conséquence logique

On dit qu'une interprétation I satisfait E si $I(E) = vrai$, et qu'un ensemble S de graphes RDF implique (simplement) un graphe E si toute interprétation qui satisfait tous les membres de S satisfait aussi E. À partir de cette notion de conséquence logique, on peut prouver plusieurs lemmes, que l'on retrouve dans le document officiel du W3C décrivant la sémantique de RDF. Ici, nous ne retiendrons que deux lemmes importants :

Lemme de la fusion : La fusion de tous les graphes d'un ensemble S de graphes est une conséquence logique de S.

Supposons maintenant que l'instance d'un graphe est définie comme étant le graphe obtenu en remplaçant un ou plusieurs noeuds vides de ce graphe par un autre noeud vide, un IRI ou un littéral. On peut maintenant présenter le prochain lemme :

Lemme de l'interpolation : Un ensemble de graphes S implique logiquement un graphe E si et seulement si un sous-graphe de S est une instance de E.

1.4 Rdf-interprétation

Nous avons vu à la section précédente comment interpréter un graphe RDF sans tenir compte de la sémantique de son vocabulaire. Mais nous savons que RDF a déjà un vocabulaire pré-défini. Si dans un graphe RDF nous utilisons ce vocabulaire, il est important de respecter la sémantique associée à ce vocabulaire. Ceci aura pour effet de restreindre les interprétations possibles pour un tel graphe. Nous parlerons alors d'une rdf-interprétation, définie formellement de la manière suivante :

Soit $rdfV$ le vocabulaire RDF, constitué des items suivants :

```
rdf:type
rdf:Property
rdf:XMLLiteral
rdf:nil
rdf:List
rdf:Statement
rdf:subject
rdf:predicate
rdf:object
```

```
rdf:first
rdf:rest
rdf:Seq
rdf:Bag
rdf:Alt
rdf:_1
rdf:_2
...
rdf:value
```

Une *rdf-interprétation* d'un vocabulaire V est une interprétation simple de $V \cup \texttt{rdf}V$ qui satisfait les deux conditions concernant le terme `rdf:XMLLiteral` telles que décrites dans la spécification du W3C ainsi que la condition et les axiomes qui suivent :

Condition sémantique de RDF :
Soit une entité x de l'univers. $x \in IP$ si et seulement si $\langle x, I(\texttt{rdf:Property})\rangle \in I_{EXT}(I(\texttt{rdf:type}))$.

Axiomes de RDF :

```
rdf:type rdf:type rdf:Property .
rdf:subject rdf:type rdf:Property .
rdf:predicate rdf:type rdf:Property .
rdf:object rdf:type rdf:Property .
rdf:first rdf:type rdf:Property .
rdf:rest rdf:type rdf:Property .
rdf:value rdf:type rdf:Property .
rdf:_1 rdf:type rdf:Property .
rdf:_2 rdf:type rdf:Property .
...
rdf:nil rf:type rdf:List .
```

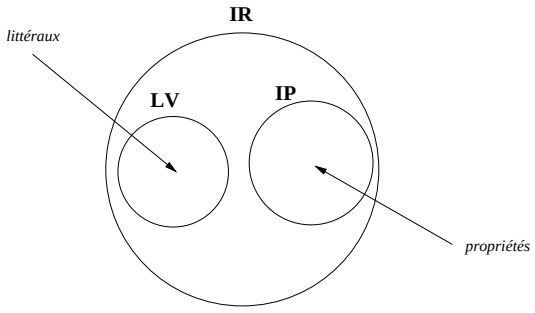

FIGURE 2 – Ensembles de l'univers pour rdf-interprétation d'un vocabulaire RDF

Pour comprendre l'effet de l'ajout de cette condition, supposons l'existence d'un triplet de la forme suivante :

```
ex:u rdf:type rdf:Property .
```

L'entité qui correspond à cette `ex:u` dans l'univers est une propriété. Supposons maintenant qu'un item `ex:p` représente une propriété, ce qui sera nécessairement le cas si on le retrouve comme prédicat dans un triplet. La condition permet alors de déduire le triplet suivant :

```
ex:p rdf:type rdf:Property .
```

Une conséquence importante de cette condition est que toute propriété peut apparaître comme sujet d'un triplet. Or, par définition de l'interprétation simple, le sujet d'un triplet est toujours une ressource. Ce qui signifie qu'avec la rdf-interprétation, on a $\mathrm{IP} \subseteq \mathrm{IR}$. L'univers ressemblera plutôt à celui illustré à la figure 2.

Quant aux axiomes, ils établissent clairement quels éléments du vocabulaire RDF sont des propriétés, donc que l'on s'attend à retrouver dans les triplets à la position du prédicat. Le dernier axiome ne fait qu'établir l'existence de la liste vide, nécessaire pour la construction de toute liste.

Il est important de remarquer que la sémantique de RDF ne va pas au-delà ce qui est spécifié ici. Ce qui signifie qu'elle n'ajoute aucune contrainte sur les conteneurs, les collections et la réification.

1.5 Rdfs-interprétation

Nous avons vu à la section précédente que l'ajout apporté par une rdf-interpretation, par rapport à une interprétation simple, est une spécification du concept de propriété. Une rdfs-interprétation ajoute à cela une spécification du concept de classe. Intuitivement, une classe est toute ressource qui peut se retrouver comme objet dans un triplet dont le prédicat est `rdf:type`. Nous définirons donc un sous-ensemble IC de l'ensemble des ressources pour représenter l'ensemble des classes de l'univers, tel qu'illustré à la figure 3.

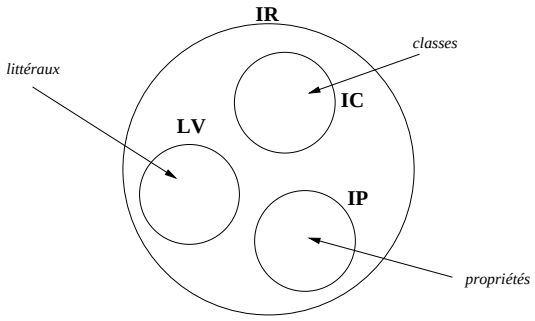

FIGURE 3 – Ensembles de l'univers pour rdfs-interprétation d'un vocabulaire RDF

Il nous faudra alors une nouvelle fonction d'interprétation $\mathrm{IC}_{\mathrm{EXT}}$, qui associe des éléments du vocabulaire à un élément de l'ensemble IC. Une rdfs-interprétation devra respecter les conditions

suivantes :

— $x \in IC_{EXT}(y)$ si et seulement si $\langle x, y \rangle \in I_{EXT}(I(\texttt{rdf:type}))$
— $IC = IC_{EXT}(I(\texttt{rdfs:Class}))$
— $IR = IC_{EXT}(I(\texttt{rdfs:Resource}))$
— $LV = IC_{EXT}(I(\texttt{rdfs:Literal}))$

La première condition spécifie que l'ensemble des classes dans l'univers correspond à toute ressource qui peut être l'objet de la propriété `rdf:type`. La deuxième condition peut être comprise de la manière suivante. Supposons l'existence d'un triplet de la forme suivante :

```
ex:u rdf:type ex:c .
```

Dans ce cas, `ex:c` appartient à l'ensemble des classes, par la première condition et, selon la deuxième condition, ce triplet implique logiquement le triplet suivant :

```
ex:c rdf:type rdfs:Class .
```

De même, par la définition de l'interprétation simple, `ex:u` et `ex:c` sont des ressources. Donc, par la troisième condition on peut déduire les triplets suivants :

```
ex:u rdf:type rdfs:Resource .
ex:c rdf:type rdfs:Resource .
```

La dernière condition indique que tout littéral appartient à la classe correspondant à `rdfs:Literal`. Mais étant donné qu'un littéral ne peut pas être le sujet d'un triplet, on ne peut pas spécifier directement sa classe, comme nous venons de le faire avec les classes et les ressources. Par contre, on pourrait dire que l'objet d'un triplet est un littéral, sans pouvoir spécifier ce littéral :

```
ex:u ex:prop _:x .
_:x  rdf:type rdfs:Literal .
```

À noter qu'une classe peut avoir une extension vide. Aussi, une conséquence des deux premières conditions est que le triplet suivant est toujours valide :

```
rdfs:Class  rdf:type rdfs:Class .
```

Voici maintenant d'autres conditions sur l'interprétation du vocabulaire de RDFS :

— Si $\langle x, y \rangle \in I_{EXT}(I(\texttt{rdfs:domain}))$ et $\langle u, v \rangle \in I_{EXT}(x)$, alors $u \in IC_{EXT}(y)$.

— Si $\langle x, y \rangle \in I_{EXT}(I(\texttt{rdfs:range}))$ et $\langle u, v \rangle \in I_{EXT}(x)$, alors $v \in IC_{EXT}(y)$.

— Si $\langle x, y \rangle \in I_{EXT}(I(\texttt{rdfs:subClassOf}))$, alors $x \in IC$, $y \in IC$ et $IC_{EXT}(x)$ est un sous-ensemble de $IC_{EXT}(y)$.

— $I_{EXT}(I(\texttt{rdfs:subClassOf}))$ est transitive et réflexive sur IC.

— Si $x \in IC$, alors $\langle x, I(\texttt{rdfs:Resource}) \rangle \in I_{EXT}(I(\texttt{rdfs:subClassOf}))$.

— $I_{EXT}(I(\texttt{rdfs:subPropertyOf}))$ est transitive et réflexive sur IP.

— Si $\langle x, y \rangle \in I_{EXT}(I(\texttt{rdfs:subPropertyOf}))$, alors $x \in IP$, $y \in IP$ et $I_{EXT}(x)$ est un sous-ensemble de $I_{EXT}(y)$.

La première de ces conditions spécifie que si on a un triplet de la forme suivante :

```
ex:p  rdfs:domain ex:c .
```

alors <ex:p> est une propriété telle que pour tout triplet de la forme

```
ex:a ex:p ex:b
```

on a nécessairement que toute interprétation doit associer à <ex:a> un élément de l'univers qui appartient à la classe représentée par l'item <ex:c>. En d'autres mots, cela signifie que nous pourrions déduire le triplet suivant :

```
ex:a rdf:type ex:c
```

La deuxième condition définit de manière similaire l'image d'une propriété. Selon la troisième condition, si une entité x est déclarée comme étant sous-classe d'une autre entité y, ces deux entités sont des classes et l'extension de x est un sous-ensemble de l'extension de y. Quant à la quatrième condition, elle spécifie que toute classe est sous-classe d'elle-même (réflexivité) et que si C est sous-classe de C' qui est elle-même sous-classe de C'', alors C est sous-classe de C'' (transitivité). La condition suivante indique que toute classe est sous-classe de la classe des ressources. Finalement, les deux dernières conditions caractérisent les propriétés.

Voilà qui termine cette brève présentation de la sémantique de RDF. Pour être plus complet, il faudrait aussi présenter la sémantique formelle des types de donnée, qui a été exclue du présent document pour ne pas l'alourdir. On pourra se référer au document officiel du W3C pour plus de détails sur la sémantique des types de données. À noter aussi que la rdfs-interprétation contient un ensemble de triplets axiomatiques, dont la liste est fournie à l'annexe A.

1.6 Exercices

■ Exercice 1.1
Soit le graphe RDF illustré à la figure 4. Dites laquelle ou lesquelles des interprétations suivantes

sont valides pour ce graphe :

Interprétation 1 :

IR : $\{p_1, p_2, r_1, r_2\}$
IP : $\{p_1, p_2\}$
I_{EXT} : $p_1 \rightarrow \{(p_1, r_1), (p_2, r_2)\}$
$p_2 \rightarrow \{(r_1, r_2)\}$
I_S : $\{(\texttt{ex:A}, p_1), (\texttt{ex:B}, r_1)\}$

Interprétation 2 :

IR : $\{p1, r_1, r_2\}$
IP : $\{p1\}$
I_{EXT} : $p_1 \rightarrow \{(p_1, p_1), (r_1, r_2)\}$
I_S : $\{(\texttt{ex:A}, p_1), (\texttt{ex:B}, p_1)\}$

FIGURE 4 – Graphe RDF

■ **Exercice 1.2**
a) Considérez les deux graphes RDF illustrés aux figures 5 et 6. Est-il possible de fournir une interprétation qui soit la même pour les deux graphes ? (Justifiez votre réponse)

b) En vous servant des axiomes et des règles d'inférence pour RDF et RDFS, démontrez que le graphe RDF illustré à la figure 7 est une conséquence logique du graphe 1 (indiquez bien la règle utilisée à chaque étape de votre preuve) :

■ **Exercice 1.3**

Soit la description suivante en RDF :

```
@prefix xmlns: <http://www.polymtl.ca/exemple#> .
@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 local:  <http://www.polymtl.ca/voc#> .
```

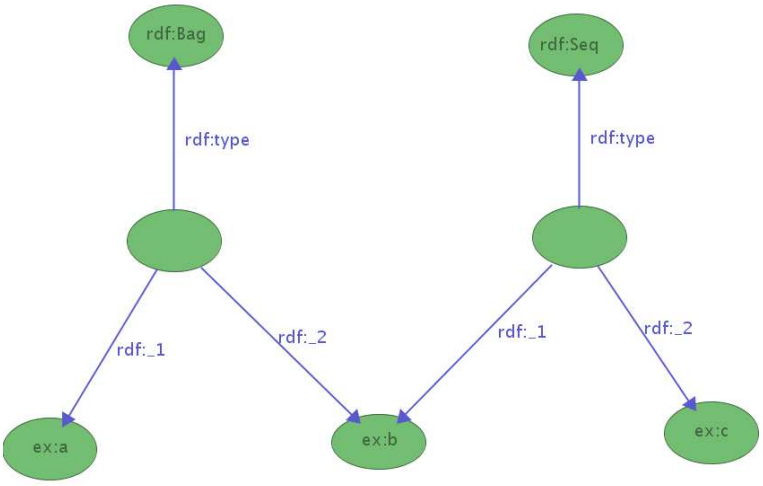

FIGURE 5 – Graphe 1

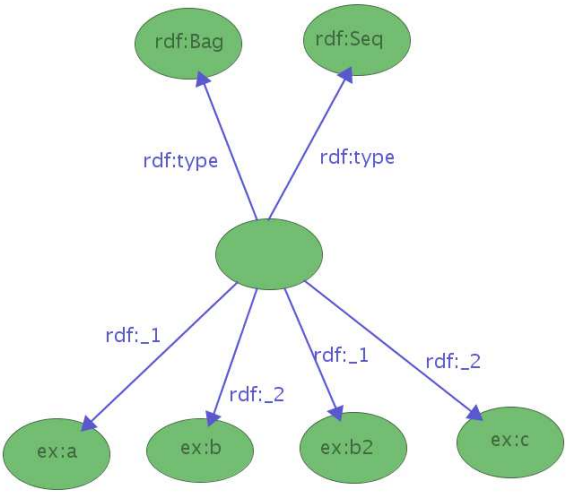

FIGURE 6 – Graphe 2

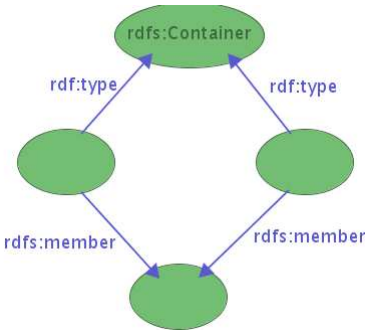

FIGURE 7 – Graphe 3

```
local:a rdf:type local:b .
local:b rdf:type local:c .
```

Identifiez, parmi les triplets suivants, ceux qui sont conséquence logique de cette description :

a) `local:a rdf:type local:c .`
b) `local:b rdf:type rdfs:Class .`
c) `local:b rdfs:domain local:a .`

Pour chacun de ces triplets, vous fournirez une preuve en utilisant les règles d'inférence, dans le cas où il est une conséquence logique. Dans le cas contraire, vous fournirez une interprétation du graphe original qui ne satisfait pas le triplet.

■ Exercice 1.4

Soit la description suivante en RDF :

```
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix local:  <http://www.polymtl.ca/voc#> .

local:BergerAllemand rdf:type local:Chien ;
                     rdfs:subClassOf local:Chien .
```

Dites s'il existe une interprétation possible pour ce graphe RDF. Si oui, fournissez une intepré-tation, sinon justifiez votre réponse.

2 Inférence

La définition de la sémantique de RDF est utile pour déterminer l'expressivité de ce modèle de donnée. Elle permet d'établir clairement ce que nous voulons signifier lorsque nous construisons

un graphe RDF. Nous allons maintenant aborder le problème de l'inférence. Plus précisément, nous voulons savoir si un graphe RDF est une conséquence logique d'un autre graphe RDF. De manière plus précise, supposons l'existence de deux graphes RDF G_1 et G_2. Supposons maintenant que pour toute interprétation telle que G_1 est vrai, la même interprétation rend nécessairement vrai le graphe G_2. On dira alors que G_2 est une conséquence logique de G_1 ou, de manière équivalente, que G_1 implique logiquement G_2. En d'autres mots, cela signifie que chaque fois qu'on aura le graphe G_1, on pourra déduire le graphe G_2.

L'importance de l'inférence se révèle particulièrement lorsqu'on consulte un graphe RDF pour vérifier si un fait est vrai selon ce graphe. Il est possible que les faits ne soient pas tous exprimés explicitement par un graphe. Prenons par exemple le graphe suivant :

```
ex:a rdf:type ex:C1 .
ex:C1 rdfs:subClassOf ex:C2 .
```

Selon ce graphe, le fait `ex:a rdf:type ex:C2` est vrai, bien qu'il ne soit pas exprimé explicitement.

En général, pour appliquer une procédure d'inférence, on n'utilise pas la sémantique du langage. On utilise plutôt des règles de déduction qui respectent la sémantique du langage, c'est-à-dire des règles qui nous permettent de déduire de nouveaux faits qui sont nécessairement des conséquences logiques des faits originaux.

Nous allons maintenant présenter les règles de déduction de RDF. Nous utiliserons la même convention que celle adoptée dans le document du W3C : *aaa, bbb, cccc*, etc. représentent un IRI, alors que *xxx, yyy, etc.* désignent un IRI ou un noeud vide. Un terme `lll` représente un littéral quelconque, et _ *:nnn* un noeud vide, où nnn est son identificateur. Nous pouvons maintenant énumérer les règles d'inférence. Toutes les règles ont la forme **Si** E **contient le triplet** T**, alors ajouter le triplet** T'. À partir d'un graphe E, on obtient alors un graphe $E' = E \cup \{T'\}$ qui est conséquence logique de E.

2.1 Inférence simple

Il n'y a pas de règle particulière pour l'inférence simple. Pour faire de l'inférence simple, il suffit d'appliquer le théorème d'interpolation. Plus précisément, pour déterminer que G_2 est conséquence logique de G_1, il suffit de vérifier qu'un sous-graphe de G_1 est une instance de G_2.
Prenons par exemple le graphe G_1 suivant :

```
ex:a ex:p ex:b .
ex:c ex:q ex:a .
```

on peut déduire le graphe G_2 suivant :

```
ex:c ex:q _:id1 .
```

En effet, on peut vérifier qu'en remplaçant le noeud vide `_:id1` par l'IRI `:a`, G_2 correspond au second triplet de G_1. On a donc un sous-graphe de G_1 qui est une instance de G_2 (le second triplet), ce qui permet d'affirmer $G_1 \models G_2$.
Considérons maintenant le graphe G_3 suivant, qui est aussi conséquence logique de G_1 :

```
_:id1 ex:p ex:b .
ex:c ex:q _:id2 .
```

Ici, en remplaçant les deux noeuds vides par l'IRI `:a`, on obtient le graphe G_1. Donc tout le graphe G_1 est une instance de G_2 et on peut dire que G_2 est conséquence logique de G_1.

On ne peut pas, à partir de G_1, déduire le graphe G_4 suivant, où l'on tente de réutiliser un même identificateur de noeud vide pour deux IRI différents :

```
_:id1 ex:p ex:b .
_:id1 ex:q ex:a .
```

Dans ce dernier cas, peu importe par quel IRI on remplace le noeud vide, il y aura toujours un triplet de G_4 qui ne se trouve pas dans G_1. Pour nous en convaincre, supposons que l'interprétation du graphe original est la suivante :

$$
\begin{array}{ll}
I_R = & \{1, 2, 3\} \\
I_P = & \{r1, r2\} \\
I_{EXT} = & r1 \rightarrow \{\langle 1, 2\rangle\} \\
 & r2 \rightarrow \{\langle 3, 1\rangle\} \\
I_S = & \texttt{ex:a} \rightarrow 1 \\
 & \texttt{ex:b} \rightarrow 2 \\
 & \texttt{ex:c} \rightarrow 3 \\
 & \texttt{ex:p} \rightarrow r1 \\
 & \texttt{ex:q} \rightarrow r2
\end{array}
$$

Il est clair que le gaphe inféré est faux sous cette interprétation, puisqu'on ne trouve aucune entité x telle que $\langle x, 2\rangle \in I_{EXT}(r1)$ et $\langle x, 1\rangle \in I_{EXT}(r2)$.

2.2 RDF-inférence

Les règles d'inférence RDF sont les suivantes :

Règle	**Si E contient**	**Alors ajouter (si le vocabulaire D est reconnu)**
rdfD1	xxx aaa "sss" ^^ddd . où ddd est un type de D	xxx aaa `_:nnn` . `_:nnn` rdf :type ddd .
rdfD2	uuu aaa yyy .	aaa `rdf:type rdf:Property.`

La seconde règle n'ajoute pas beaucoup de pouvoir au mécanisme d'inférence. Elle spécifie que la classe d'un prédicat est nécessairement `rdf:Property`,

La première est plus contraignante. Elle indique que si nous utilisons un type pour un littéral, alors l'entité qui correspond à ce littéral doit appartenir à ce type.

2.3 Inférence RDFS

Voici maintenant les règles d'inférence RDFS, qui complètent l'ensemble des règles requises pour notre mécanisme d'inférence :

Règle	**Si E contient**	**Alors ajouter (si le vocabulaire D est reconnu)**
rdfs1	Pour tout IRI aaa dans D	aaa `rdf:type rdfs:Datatype` .
rdfs2	aaa `rdfs:domain` xxx . uuu aaa yyy .	uuu `rdf:type` xxx .
rdfs3	aaa `rdfs:range` xxx . uuu aaa vvv .	vvv `rdf:type` xxx .
rdfs4a	uuu aaa xxx .	uuu `rdf:type rdfs:Resource` .
rdfs4b	uuu aaa vvv .	vvv `rdf:type rdfs:Resource` .
rdfs5	uuu `rdfs:subPropertyOf` vvv . vvv `rdfs:subPropertyOf` xxx .	uuu `rdfs:subPropertyOf` xxx .
rdfs6	uuu `rdf:type rdf:Property` .	uuu `rdfs:subPropertyOf` uuu .
rdfs7	aaa `rdfs:subPropertyOf` bbb . uuu aaa yyy .	uuu bbb yyy .
rdfs8	uuu `rdf:type rdfs:Class` .	uuu `rdfs:subClassOf rdfs:Resource` .
rdfs9	uuu `rdfs:subClassOf` xxx . vvv `rdf:type` uuu .	vvv `rdf:type` xxx .
rdfs10	uuu `rdf:type rdfs:Class` .	uuu `rdfs:subClassOf` uuu .
rdfs11	uuu `rdfs:subClassOf` vvv . vvv `rdfs:subClassOf` xxx .	uuu `rdfs:subClassOf` xxx .
rdfs12	uuu `rdf:type` `rdfs:ContainerMembershipProperty` .	uuu `rdfs:subPropertyOf rdfs:member` .
rdfs13	uuu `rdf:type rdfs:Datatype` .	uuu `rdfs:subClassOf rdfs:Literal` .

Spécification de la méthode pour déterminer si un graphe G_2 est conséquence logique de G_1

1. Ajouter à G_1 tous les triplets axiomatiques, sauf ceux qui concernent l'appartenance à un conteneur (`rdf:_1`, `rdf:_2`, etc.).
2. Pour tout triplet de G_1 utilisant une propriété d'appartenance à un conteneur, ajouter les axiomes qui le concernent.
3. En utilisant une règle d'inférence, ajouter un nouveau triplet au graphe.
4. Répéter l'étape précédente jusqu'à ce qu'on obtienne un sous-graphe qui est instance de G_2, ou qu'on aie épuisé toutes les possibilités.

Attention : cette méthode n'est pas complète. Certaines conséquences logiques ne pourront pas être démontrées. Prenons par exemple le graphe suivant :

```
ex:B324 ex:name "Michel"^^xsd:string .
ex:Z236 ex:name "Michel"^^xsd:string .
```

On ne peut pas déduire le graphe suivant, qui spécifie que les deux entités sont associées à une même chaîne de caractères :

```
ex:B324 ex:name _:b .
ex:Z236 ex:name _:b .
_:b rdf:type xsd:string .
```

On en pourra non plus faire une démonstration qui fait appel à une propriété qui n'est pas nommé explicitement. Par exemple, à partir du graphe suivant :

```
ex:a rdfs:subPropertyOf _:b .
_:b rdfs:domain ex:c .
 ex:d ex:a ex:e .
```

on ne pourra pas déduire sa conséquence logique suivante :

```
ex:d rdf:type ex:c .
```

À noter qu'il s'agit de situations un peu particulières qui ne devraient pas se produire fréquemment.

2.4 Exercices

■ Exercice 2.1
Soit le graphe suivant :

```
ex:a ex:p ex:b .
ex:p rdfs:subPropertyOf ex:q .
```

Montrez que l'énoncé `ex:q rdf:type rdf:Property .` est une conséquence logique de ce graphe.

■ Exercice 2.2
Soit le graphe suivant :

```
ex:Paul ex:connait ex:Jean .
ex:Paul rdf:type ex:Personne .
ex:Jean rdf:type ex:Personne .
ex:Paul ex:nom "Gagnon" .
ex:Jean ex:nom "Gagnon" .
```

Montrez qu'à partir de ce graphe, on peut déduire que deux personnes ont le même nom, c'est-à-dire le graphe suivant :

```
[] rdf:type ex:Personne ;
   ex:nom _:l1 .
[] rdf:type ex:Personne ;
   ex:nom _:l1 .
_:l1 rdf:type rdfs:Literal .
```

■ **Exercice 2.3**
Soit le graphe suivant :

```
ex:detester rdfs:subPropertyOf ex:avoirSentimentEnvers .
ex:Chat rdfs:subClassOf ex:Animal .
ex:Tom rdf:type  ex:Chat .
ex:Jerry rdf:type ex:Animal .
ex:Tom ex:detester ex:Jerry .
```

Montrez qu'à partir de ce graphe, on peut déduire qu'un animal a un sentiment envers un autre animal, c'est-à-dire le graphe suivant :

```
[] rdf:type ex:Animal ;
   ex:avoirSentimentEnvers
      [rdf:type ex:Animal] .
```

■ **Exercice 2.4**
Montrez que l'énoncé suivant est valide (c'est-à-dire vrai pour toutes les interprétations d'un graphe RDF) :

```
rdfs:Class rdf:type rdfs:Class .
```

3 Les graphes nommés

4 Annexe A - Triplets axiomatiques de RDFS

```
rdf:type rdfs:domain rdfs:Resource .
rdfs:domain rdfs:domain rdf:Property .
rdfs:range rdfs:domain rdf:Property .
rdfs:subPropertyOf rdfs:domain rdf:Property .
rdfs:subClassOf rdfs:domain rdfs:Class .
rdf:subject rdfs:domain rdf:Statement .
rdf:predicate rdfs:domain rdf:Statement .
rdf:object rdfs:domain rdf:Statement .
```

```
rdfs:member rdfs:domain rdfs:Resource .
rdf:first rdfs:domain rdf:List .
rdf:rest rdfs:domain rdf:List .
rdfs:seeAlso rdfs:domain rdfs:Resource .
rdfs:isDefinedBy rdfs:domain rdfs:Resource .
rdfs:comment rdfs:domain rdfs:Resource .
rdfs:label rdfs:domain rdfs:Resource .
rdf:value rdfs:domain rdfs:Resource .

rdf:type rdfs:range rdfs:Class .
rdfs:domain rdfs:range rdfs:Class .
rdfs:range rdfs:range rdfs:Class .
rdfs:subPropertyOf rdfs:range rdf:Property .
rdfs:subClassOf rdfs:range rdfs:Class .
rdf:subject rdfs:range rdfs:Resource .
rdf:predicate rdfs:range rdfs:Resource .
rdf:object rdfs:range rdfs:Resource .
rdfs:member rdfs:range rdfs:Resource .
rdf:first rdfs:range rdfs:Resource .
rdf:rest rdfs:range rdf:List .
rdfs:seeAlso rdfs:range rdfs:Resource .
rdfs:isDefinedBy rdfs:range rdfs:Resource .
rdfs:comment rdfs:range rdfs:Literal .
rdfs:label rdfs:range rdfs:Literal .
rdf:value rdfs:range rdfs:Resource .

rdf:Alt rdfs:subClassOf rdfs:Container .
rdf:Bag rdfs:subClassOf rdfs:Container .
rdf:Seq rdfs:subClassOf rdfs:Container .
rdfs:ContainerMembershipProperty rdfs:subClassOf rdf:Property .

rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso .

rdf:XMLLiteral rdf:type rdfs:Datatype .
rdf:XMLLiteral rdfs:subClassOf rdfs:Literal .
rdfs:Datatype rdfs:subClassOf rdfs:Class .

rdf:_1 rdf:type rdfs:ContainerMembershipProperty .
rdf:_1 rdfs:domain rdfs:Resource .
rdf:_1 rdfs:range rdfs:Resource .
rdf:_2 rdf:type rdfs:ContainerMembershipProperty .
rdf:_2 rdfs:domain rdfs:Resource .
```

```
rdf:_2 rdfs:range rdfs:Resource .
...
```