TP1 - Un traducteur de RDF/Turtle vers RDF/Ntriples

Vous devez rendre au moins la partie 2 de ce TP (ainsi que la partie 3 si vous l'avez faite) au plus tard pour le début de la troisième séance de TP (semaine du 17 au 21 octobre).

1 Cadre et préparation du TP

Le Web sémantique est basé sur RDF, un modèle pour la représentation de données interopérables. L'unité de base en RDF est le triplet, correspondant à une phrase simple de la forme "sujet verbe complément" (le verbe est appelé *prédicat* et le complément est appelé *objet*). Le sujet et le prédicat sont des *entités* (noms entre chevrons `<...>`) et l'objet est soit une entité soit un *texte* (entre guillemets `"..."`). Il existe plusieurs notations pour RDF et on souhaite ici effectuer la conversion depuis (un fragment de) Turtle (extension `.ttl`) vers Ntriples (extension `.nt`). Le problème va être présenté par un exemple et vous devrez proposer une solution générale.

Voici un texte Turtle décrivant l'entité `<poly117>` comme étant un poly (type), ayant deux auteurs (Ridoux et Ferre) et ayant pour titre "Compilation". L'entité `<Ridoux>` est défini comme étant une personne et un professeur.

```
<poly117>
  <type> <poly> ;
  <auteur> <Ridoux>, <Ferre> ;
  <titre> "Compilation" .
<Ridoux> <type> <personne>, <professeur> .
```

Ntriples n'autorise qu'une représentation à plat, un triplet par ligne. Le même exemple peut ainsi être représenté comme suit.

```
<poly117> <type> <poly> .
<poly117> <auteur> <Ridoux> .
<poly117> <auteur> <Ferre> .
<poly117> <titre> "Compilation" .
<Ridoux> <type> <personne>.
<Ridoux> <type> <professeur> .
```

On voit donc qu'en Turtle, le point-virgule (;) permet de factoriser le sujet entre plusieurs triplets, et la virgule (,) permet de factoriser le sujet et le prédicat entre plusieurs triplets. Ces séparateurs jouent le même rôle que la conjonction de coordination 'et' en français.

> **Pour démarrer ce travail, nous vous demandons de proposer une grammaire hors-contexte pour spécifier le sous-ensemble du langage Turtle utilisé ici. On demande que la grammaire soit : (1) non étendue (sans les symboles $*$ ou ?), (2) LL(1), et (3) reflètent la structure sémantique de Turtle.**

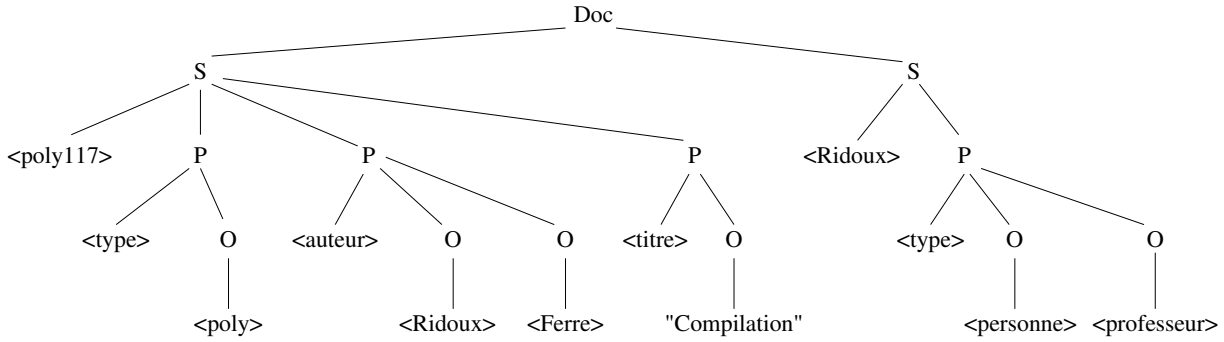

FIGURE 1 – AST de l'exemple.

2 Réalisation du traducteur de Turtle vers Ntriples

Dans le cadre du premier devoir à la maison, vous avez eu l'occasion de manipuler et modifier une grammaire ANTLR ou OCaml. Pendant les deux premières séances de TP Compilation, vous allez réaliser un traducteur de fichiers Turtle vers Ntriples. En vous appuyant sur votre grammaire de Turtle (voir section 1), vous allez construire le traducteur demandé en utilisant ANTLR ou OCaml.

2.1 Les différentes étapes du travail à réaliser

Vous allez construire deux versions de ce petit traducteur :

1. une première version où l'analyseur syntaxique ne produit pas d'arbre de syntaxe abstraite (AST) mais directement du Ntriples ;
2. une seconde version avec production de l'AST (voir Figure 1 pour l'exemple), ce qui nécessitera de construire un "arpenteur d'arbre" pour la production du Ntriples. L'intérêt de l'AST est de permettre plusieurs traduction ou calculs sans dupliquer le code de l'analyseur syntaxique. Ici, on souhaite calculer le nombre de descriptions d'un fichier Turtle (2 dans l'exemple ci-dessus).

   On aura donc deux arpenteurs d'abres, un pour la production du Ntriples, et un pour le calcul du nombre de descriptions Turtle.

Quelques consignes importantes pour l'écriture des analyseurs syntaxiques, en tant qu'implémentation des grammaires attribuées :

— bien séparer l'analyse lexicale de l'analyse syntaxique. Cette séparation conceptuelle doit apparaitre dans la structure de votre code.
— utiliser exclusivement des attributs (synthétisés et hérités). Il est interdit d'utiliser des variables globales car elles sont contraires au principe de compositionalité.
— comme la syntaxe est la base des calculs sémantiques, il est important de tenir compte de la sémantique lors de la conception de la grammaire. Ainsi, la grammaire doit se rapprocher le plus possible de la syntaxe abstraite visée.

Vous disposez de deux fichiers Turtle exemples, `test1.ttl` et `test2.ttl`, pour tester vos analyseurs.

2.2 Environnement ANTLR/Java

Dans le répertoire `/share/m1info/COMP/antlr/`, vous disposez de la version 3.5.2 de ANTLR, ainsi que de différents scripts pour la compilation de vos analyseurs ANTLR :

README : fichier avec des instructions détaillées sur les fichiers mentionnés ci-après.

g2java.sh : génère les fichiers Java à partir du fichier `.g` fourni en paramètre, en utilisant ANTLR. S'il y a plusieurs fichiers `.g`, il faut appeler ce script sur chacun d'eux, successivement.

javac.sh : compile l'ensemble des fichiers Java du répertoire courant avec ANTLR pour obtenir des fichiers `.class`.

run.sh : exécute la grammaire compilée avec ANTLR. Prend comme paramètres la classe principale (celle contenant la méthode `main`) puis les paramètres attendus par celle-ci.

Makefile : fichier Makefile pour le TP1 qui effectue les mêmes tâches que les scripts précédents.

Plusieurs environnements graphiques sont disponibles pour ANTLR. L'utilisation via ligne de commande reste cependant la solution conseillée, pour être la plus stable et assurer une bonne visibilité des messages d'erreur.

Si vous voulez utiliser ANTLRWorks, des fichiers JAR sont disponibles dans `/share/m1info/COMP/antlr`. Cette interface n'est cependant pas libre d'erreurs. Faire surtout attention aux messages d'erreur affichés à chaque compilation.

Vous pouvez aussi utiliser le *plug-in* Eclipse ANTLR IDE (disponible sur `antlrv3ide.sourceforge.net`). Attention, le *plug-in* n'a pas été mis à jour depuis quelque temps et son utilisation est relativement instable selon la version d'Eclipse. Les instructions pour l'installation sont indiquées sur le site Internet du *plug-in*.

2.3 Environnement OCaml

En début de chaque session, vous devez exécutez la commande

```
source /usr/local/env/opam_config_env.sh
```

de façon à positionner correctement les variables d'environnement pour utiliser OCaml. Alternativement, vous pouvez recopier son contenu dans votre fichier `~/.bash_profile`, lequel est exécuté au début de chaque session.

L'analyseur syntaxique et les différents arpenteurs d'arbres peuvent être placés dans un seul fichier ou plusieurs. La commande pour produire un exécutable à partir d'un fichier utilisant les *stream parsers* est de la forme :

```
ocamlc -pp camlp4o -o <exec-file> <source1.ml> ... <sourceN.ml>
```

Pour définir l'analyseur lexical, vous prendrez modèle sur le fichier `/share/m1info/COMP/dm/expr-with-custom-lexer.ml`, qui reprend l'exemple du DM avec un analyseur lexical spécifique.

3 Extensions possibles

S'il vous reste du temps, voici deux extensions possibles intéressantes : traiter les noeuds anonymes de Turtle et produire une version XML de Turtle. Ces deux extensions sont indépendantes.

3.1 Noeuds anonymes

À la place d'un sujet ou d'un objet, on peut avoir un noeud anonyme, donc sans nom mais avec éventuellement des couples propriété-valeurs associés. Cela permet par exemple de décrire les 2 versions du poly de compilation, sans avoir à nommer chaque version.

```
<poly117> <version>
   [ <annee> "2007" ;
     <nbpages> "147" ] ,
   [ <annee> "2011" ;
     <nbpages> "163" ] .
```

La traduction en Ntriples suppose de générer des identificateurs uniques de noeuds anonymes, de la forme _: *id*. La traduction Ntriples de l'exemple ci-dessus est la suivante.

```
<poly117> <version> _:v1 .
_:v1 <annee> "2007" .
_:v1 <nbpages> "147" .
<poly117> <version> _:v2 .
_:v2 <annee> "2011" .
_:v2 <nbpages> "163" .
```

3.2 Production de RDF/XML

Il existe aussi un format XML pour RDF. On propose de rajouter un arpenteur d'arbre pour le produire. La version XML du premier exemple Turtle est comme suit.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xml:base="http://mydomain.org/myrdf/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<rdf:Description rdf:about="poly117">
    <rdf:type rdf:resource="poly"/>
    <auteur rdf:resource="Ridoux"/>
    <auteur rdf:resource="Ferre"/>
    <titre>Compilation</titre>
</rdf:Description>
<rdf:Description rdf:about="Ridoux">
    <rdf:type rdf:resource="personne"/>
```

```
    <rdf:type rdf:resource="professeur"/>
</rdf:Description>
</rdf:RDF>
```

La version XML du second exemple avec noeuds anonymes est comme suit.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xml:base="http://mydomain.org/myrdf/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<rdf:Description rdf:about="poly117">
    <version rdf:parseType="Resource">
        <annee>2007</annee>
        <nbpages>147</annee>
    </version>
    <version rdf:parseType="Resource">
        <annee>2011</annee>
        <nbpages>163</annee>
    </version>
</rdf:Description>
</rdf:RDF>
```