Sailly Steven

Fonctionnement et options implémentées

Fonctionnement

On utilisera `oaj` comme abbréviation de `org.apache.jena`.

Classe `Main`

Classe principale :

- Parsing et vérification des arguments (trivial)
- Calcul des conséquences si besoin : classe `ReasonerRunner`
- Exécution de la requête ; classe `QueryRunner`

Classe `ReasonerRunner`

Calcul des conséquences :

- Création d'une instance de `oaj.reasoner.Reasoner`, avec les paramètres demandés par l'utilisateur
- Instances de `oaj.rdf.model.Model` pour représenter les faits et le schéma
- Faits dérivés représentés dans une instance de `oaj.rdf.model.impl.InfModelImpl`
- Si nécessaire, on ne garde que les faits dérivés

  (`infered = infered.difference(rdf).difference(rdfs)`)
- Renvoi d'un `String` contenant les faits

Classe `QueryRunner`

Exécution de la requête :

- Instance de `oaj.query.Query` pour représenter la requête
- Instance de `oaj.query.Dataset` pour représenter les données sur lesquelles va être exécutée la requête
- Exécution de la requête selon son type et affichage du résultat

Options implémentées

Celles de la partie obligatoire du sujet

Compilation et exécution

Le programme a été testé avec Java 17 et 21.

```
export CLASSPATH=$CLASSPATH:$JENAROOT/lib/*
javac *java
java Main <ARGS>
```

`<ARGS>` peut contenir ces arguments :

- `rdf=<rdf_file>` : le fichier contenant les faits RDF
- `rdfs=<rdfs_file>` : le fichier contenant le schéma RDFS
- `sparql=<sparql_file>` : le fichier contenant la requête SPARQL
- `compliance=[full, default, simple, none]` : le fragment RDFS à utiliser pour le calcul des conséquences
- `output=<output>` : le format de l'output
- `newfacts` : évaluer la requête uniquement sur les nouveaux triplets
- `debug` : afficher des informations de débogage

Un fichier *jar* compilé avec Java 17 est également fourni :

```
java -jar rdfssparql.jar <ARGS>
```

Exécution sur les exemples

Fichiers fournis

Les fichiers fournis se trouvent dans le répertoire `data`. Par exemple, pour exécuter la requête contenue dans le fichier `data/construct.sparql` sur les faits contenus dans `data/rois.ttl` et le schéma contenu dans `data/rois_schema.ttl`, avec un output au format XML lisible, avec le fragment `full`, sur les triplets présents dans la base ainsi que sur ceux dérivés, et en affichant les informations de débogage, on pourra exécuter :

```
java Main rdf=data/rois.ttl rdfs=data/rois_schema.ttl sparql=data/construct.sparql
output=RDF/XML-ABBREV compliance=full debug
```

Deux autres exemples de requêtes sont fournis : `ask.sparql` et `select.sparql`.

SPARQL endpoint

```
curl https://data.bnf.fr/11915277/charles_maurras/rdf.n3
--output charles_maurras.ttl &&
sed -i '2408,2411d' charles_maurras.ttl
```

Retirer le type `foaf:Person` :

```
sed -e 's/a foaf:Person ;//g' -i charles_maurras.ttl
```

Création des fichiers de schéma et de requête :

```
echo "@prefix bio: <http://vocab.org/bio/0.1/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

bio:birth rdfs:domain foaf:Person" > charles_maurras_schema.ttl

echo "PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name WHERE {
    ?x a foaf:Person .
    ?x foaf:name ?name .
}" > charles_maurras_select.sparql
```

Puis exécution comme décrit précédemment.

Les fichiers `charles_maurras*` sont aussi disponibles, comme l'ensemble des autres fichiers, sur ce dépôt github.