Université de Paris **Bases de données spécialisées**
M2 Info et Maths-Info **Année 2023-2024**

TP 3 b - Devoir maison

Inférence RDFS et SPARQL

Le langage SPARQL est conçu pour interroger un ensemble explicite de triplets RDF, aucune inférence n'est faite par le moteur de requêtes. (par exemple la commande `sparql` répond aux requêtes sur la base des seuls triplets présents dans le graphe source).

Pour obtenir des réponses qui tiennent compte des triplets dérivés par les contraintes, un système de raisonnement (*reasoner*) doit être installé entre les données et le moteur de requête.

Un tel système ne matérialise pas nécessairement le graphe saturé avec toutes les conséquences, mais produit un modèle qui est une "abstraction" du graphe saturé. C'est-à-dire les requêtes exécutées sur ce modèle retournent le même résultat que si elles étaient exécutées sur le graphe saturé.

Jena fournit un tel système de raisonnement, avec la possibilité d'y attacher plusieurs ensembles de règles d'inférence (différents fragments de RDFS, OWL, ainsi que des reasoners externes et des règles custom).

Toutefois il n'y a pas d'outil en ligne de commande qui permette d'intégrer le système de raisonnement avec le moteur de requêtes. Il faut passer par la programmation Java et utiliser l'API Jena. Notamment vous allez utiliser les packages qui permettent de manipuler les données RDF, ceux pour définir et utiliser un système de raisonnement, et ceux pour interagir avec le système d'interrogation SPARQL.

Vous devez donc vous familiariser avec ces parties de l'API Jena.

1 Lecture

Lire la documentation Jena sur les aspects suivants

1. La **RDF API**. Il s'agit des classes pour représenter et manipuler en Java les triplets et graphes RDF. Elles sont fournies par le package `org.apache.jena.rdf.model`. Vous utiliserez surtout les classes `RDFNode`|, `Resource`, `Property`, `Literal` pour la représentation des noeuds, la classe `Statement` pour la représentions des triplets, et l'interface `Model` pour la représentation des graphes RDF, ainsi que la classe `ModelFactory` pour la génération de modèles.

   Pour l'utilisation de ces classes faites référence à la Core Jena Javadoc.

2. Les **méthodes `read` et `write` de l'interface `Model`** pour lire et écrire du RDF. (Pour aller plus loin sur les techniques de I/O de Jena vous pouvez parcourir aussi la documentation I/O RDF).

3. L'**Inference API**. Il s'agit des classes pour la mise en place et configuration d'un système de raisonnement (*reasoner*) en Jena. Les classes plus utiles se trouvent principalement dans les packages `org.apache.jena.rdf.model` et `org.apache.jena.reasoner.rulesys`. Ces classes et leur méthodes permettent de dériver de nouvelles conséquences à partir d'un graphe RDF de base (instance de type `Model`) et d'une ontologie, en utilisant un système de règles d'inférence. Jena permet de "brancher" plusieurs systèmes de règles d'inférence sur les mêmes données, dont les plus courants sont RDFS et OWL. Vous allez utiliser en particulier les classes `InfModel` pour representer les graphes RDF complétés

par les nouvelles dérivations (Il est peu probable que vous ayez besoin d'utiliser la classe `OntModel`).

Pour le but de ce devoir, concentrez-vous sur la section RDFS Reasoner. Il n'est pas nécessaire de se renseigner sur les reasoners OWL ou génériques).

Pour l'utilisation des classes de l'Inference API faites référence à la Core Jena Javadoc.

4. La **Query API (ARQ)**, pour l'interrogation SPARQL des données RDF. Les classes principales de cette API se trouvent dans le package `org.apache.jena.query`. Vous utiliserez en particulier les classes `Query`, `QueryExecution`, `QuerySolution`, `ResultSet` et leur variantes.

   Pour l'utilisation de ces classes faites référence à la ARQ Javadoc.

Autres liens utiles :
— Le **tutoriels Jena**
— La **documentation complète de Jena**.
— La **Javadoc complète** de Jena

Classpath. Pour utiliser les packages (JAR) de la librairie Jena, ne pas oublier de modifier le *classpath*. Ces packages se trouvent dans le sous-répertoire `lib/` du répertoire d'extraction de Jena. Rappelez-vous qu'au TP1 vous avez défini la variable d'environnement `JENAROOT` égale au répertoire d'exportation de Jena. Utilisez donc cette commande pour modifier le *classpath* :

```
export CLASSPATH=$CLASSPATH:$JENAROOT/lib/*
```

2 Devoir : développement d'un outil `rdfssparql` pour combiner interrogation SPARQL et inférence RDFS

Obligatoire. Ecrire un programme Java utilisant Jena qui prend en input
— un fichier contenant des faits RDF
— un fichier contenant un schéma RDFS
— un fichier contenant une requête SPARQL
et affiche le résultat de l'exécution de la requête sur le graphe obtenu par saturation des faits et du schéma d'input par les règles d'inférence RDFS.

Le programme doit pouvoir traiter les requêtes de type `SELECT`, `CONSTRUCT` et `ASK`.

De plus le programme prend en input au moins les options suivantes :
— Le fragment RDFS à utiliser pour le calcul des conséquences (full / default / simple / pas d'inférence ; cf. `https://jena.apache.org/documentation/inference/#rdfs`). En l'absence de cette option le fragment "default" sera utilisé.
— Le format de l'output (RDF/XML, TURTLE, etc.) dans le cas de requêtes CONSTRUCT. En l'absence de cette option le format utilisé sera TURTLE.
— L'option "newfacts" qui permet d'évaluer la requête uniquement sur les nouveaux triplets dérivés (on exclut donc les triplets présents dans la base de faits RDF, ainsi que dans le schéma RDFS d'input).

Facultatif. Implémenter d'autres options utiles au choix. S'inspirer des options de la commande `sparql` (taper `sparql --help`).

Rendu. Rendre les sources Java de votre programme, ainsi qu'un rapport en format .pdf décrivant le fonctionnement de votre programme et toutes les options implémentées. Dans ce rapport fournir des instructions précises sur comment compiler et comment exécuter le programme (en

particulier la syntaxe des options). Fournir également deux ou trois input de test (chaque test comportant données, schéma et requête). Ces fichiers doivent contenir peu de triplets, pour illustrer le comportement correct du programme.

Fournir également un exemple de test utilisant des données plus volumineuses, obtenues depuis un SPARQL endpoint (DBPedia, Wikidata, etc). Ne pas inclure ces données dans votre rendu, mais donner des instructions précises pour les obtenir.