A Simple Guide to PAOGdA

PAGOdA v3.0

1 A Simple Example

```
1  QueryReasoner r = QueryReasoner.getInstance(ontology);
2  try  {
3      r.loadOntology(ontology);
4      r.importData(dataPath);
5      if (!r.preprocess()) return ;
6      AnswerTuples answers;
7      AnswerTuple answer;
8      for (String queryText: queryTexts) {
9          answers = r.evaluate(queryText);
10         int arity = answers.getArity();
11         for (; answers.isValid(); answers.moveNext()) {
12             answer = answers.getTuple();
13             for (int i = 0; i < arity; ++i)
14                 System.out.println(answer.getGroundTerm(i) + "␣");
15             System.out.println();
16             answers.dispose();
17         }
18     }
19 } finally {
20     r.dispose();
21 }
```

Given an `OWLOntology` ontology, in *Line 1*, `QueryReasoner.getInstance()` chooses an appropriate instance for it, since conjunctive query answering is much simpler in a certain fragment of OWL 2. For example, we provide specialised reasoners for ontologies in the OWL 2 RL profile or in OWL 2 EL with minor restriction. *Line 2 and Line 3* load an ontology and a data set into our reasoner. And then in *Line 4*, the reasoner computes all the materialisation and checks the satisfiability of the ontology and the dataset. It returns `false` if any inconsistency is detected. Given a conjunctive query in SPARQL syntax, PAGOdA evaluates a query in *Line 13*. `AnswerTuples` represents a set of answers and `AnswerTuple` has all the information for one answer. To traverse an set of answers, one has to check if there is any answer left in `AnswerTuples` by

Line 15 and retrieve an answer by *Line 16*. After the process of an answers, one can move to the next answer like *Line 20*.

2 Configuration

2.1 Use PAGOdA in a Java Project

Step 1. Include pagoda.jar into the project's classpath;

Step 2. Include the following jar packages in folder into the classpath, or if one is using Maven to manage the project, simply add the following dependencies in your pom.xml and then include a correct version of JRDFox.jar in the build path.

```
<dependencies>
  <dependency>
    <groupId>com.hermit-reasoner</groupId>
    <artifactId>org.semanticweb.hermit</artifactId>
    <version>1.3.8.1</version>
  </dependency>
  <dependency>
    <groupId>net.sourceforge.owlapi</groupId>
    <artifactId>owlapi-distribution</artifactId>
    <version>3.5.0</version>
  </dependency>
  <dependency>
    <groupId>org.openrdf.sesame</groupId>
    <artifactId>sesame-runtime</artifactId>
    <version>2.7.13</version>
  </dependency>
  <dependency>
    <groupId>org.apache.jena</groupId>
    <artifactId>jena-arq</artifactId>
    <version>2.12.0</version>
  </dependency>
  <dependency>
    <groupId>org.antlr</groupId>
    <artifactId>antlr-complete</artifactId>
    <version>3.5.2</version>
  </dependency>
  <dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
  </dependency>
  <dependency>
    <groupId>org.jgrapht</groupId>
```

```
      <artifactId>jgrapht-jdk1.5</artifactId>
      <version>0.7.3</version>
    </dependency>
  </dependencies>
```

2.2 Use PAGOdA from Command Line

Unzip the download from website and use the following command to run PAGOdA from command line. Currently, PAGOdA supports Turtle format datasets and conjunctive queries in SPARQL format.

```
java -jar pagoda.jar ONTOLOGY_PATH [DATA_PATH] QUERY_PATH [
    ANS_PATH]
```