Reasoning with Jena

1 Entailment calculation

1.1 Exercise

Create a program which reads a model, applies RDFS reasoning and outputs only the new entailed triples, and not including the RDFS axiomatic triples, e.g., like
`rdfs:Class rdf:type rdfs:Resource`.

1.1.1 Tip

One way to solve the exercise is to create an RDFS model from the read model, create an empty RDFS model (this model have only the axiomatic RDFS triples), and then remove the triples in the latter model from the former model using `Model.difference`.

1.2 Exercise

Use your program to find the RDFS inferred triples from the Animal RDF graph in the exercises week 3. Use the graph given in the solution of this exercise.

2 Entailment checker

2.1 Exercise

Write a java program which reads two RDF graphs and checks if the first graph entails the second by RDFS entailment. The program should return true/false if the first graph entails / does not entail the second graph.

Note that you need to consider blank nodes with special care. Explain why. See tip and ponder on the meaning of *(un)sound* and *(in)complete*.

2.1.1 Tip

Blank nodes can be treated differently, either by

1. ignoring them (bad solution: this will make your program logically *unsound* and *incomplete*),

2. outputting an "I don't know" message when appropriate (better: your program will be *sound*, but still *incomplete*),
3. or by the strategy explained below, or an equivalent one (perfect! your program is both sound and complete with respect to RDFS semantics).

The strategy of my program is to "manually" apply the two simple entailment rules, se1 and se2, to the a model containing the statements of `entailments.n3`, but with the additional restriction that the blank nodes to be added are collected from the statement to be checked for entailment. This extra restriction ensures that the process of adding blank nodes terminates—which is probably the reason why these rules are not included in Jena RDFS reasoning. Then create an RDFS model from this model and check if all the triples in the statement to be checked for entailment is contained in the RDFS model.

You may want to go about solving this exercise in steps, first the bad solution, when the better one, and of course, finish with the perfect one.

Running my program with the `entailments.n3` graph introduced in an earlier exercise as the first graph and

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix : <http://example.org#> .
:Father rdfs:subClassOf :Person .
```

as second graph, gives me the output:

```
true
```

Changing the second graph to

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix : <http://example.org#> .
:Father rdfs:subClassOf [ rdfs:subClassOf :Person ] .
```

gives me:

```
true
```

2.2 Exercise

Explain why the three proposed blank node strategies are respectively

1. unsound and incomplete,
2. sound and incomplete,
3. sound and complete.

2.3 Exercise

Use your program to check if the answers from your manual entailment calculation from earlier exercises are correct.

2.4 Exercise

Change your entailment checker program to check for OWL entailment, instead of RDFS entailment.

2.5 Exercise

Run your OWL entailment checker on the same input as the as you did with the RDFS entailment checker. Are there any differences?

2.5.1 Tip

You might want to replace all instances of `rdfs:Class` in `entailments.n3` with `owl:Class`. It seems that OWL reasoners do not reason correctly with RDFS ontologies without some preprocessing.

3 RDFS metrics

3.1 Exercise

Make a program which loads an RDF(S) file and outputs

- the number of named `rdfs:Class`-es,
- the number of named `rdfs:Property`-es,
- the number of `rdfs:domain` assertions,
- the number of `rdfs:range` assertions,
- the number of `rdfs:subClassOf` axioms,
- the number of `rdfs:subPropertyOf` axioms,
- optionally, the maximum depth of the subclass hierarchy and
- optionally, the maximum depth of the subproperty hierarchy.

The number of classes and properties should also include classes which are not explicitly declared as an `rdfs:Class` or `rdf:Property` (see Tip below), and not include classes or properties which are part of the RDF or RDFS vocabulary, e.g., `rdf:type` and `rdfs:subClassOf`.

The number of subclass and subproperty axioms should only include the axioms explicitly declared in the input file.

The maximum depth of the subclass hierarchy should count the maximum number of consecutive `rdfs:subClassOf` steps it is possible to make in the model, without stepping to an equivalent class. (If `A` is a subclass of `B` and `B` is subclass of `A`, then they are equivalent.) This means that you have to watch out for loops in the graph.

3.1.1 Tip

You should be able to make use of the RDF metrics program created in an earlier exercise. You will need to use an RDFS reasoner for some of the problems.

Running your metrics program on the following graph

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix : <http://example.org/> .
  :a a rdfs:Class ;
     rdfs:subClassOf :b .
  :b a rdfs:Class ;
     rdfs:subClassOf :c .
```

should give you an output similar to this:

```
Named classes: 3
Named properties: 0
Domain axioms: 0
Range axioms: 0
Subclass axioms: 2
Subproperty axioms: 0
Max. depth of class tree: 2
Max. depth of property tree: 0
```

Note that even though `:c` is not explicitly typed as `rdfs:Class`, the class count returns 3. The depth of the subclass hierarchy is 2, since `:a` is a subclass of `:b`, which is a subclass of `:c`.

Running the following graph through your metrics program

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix : <http://example.org/> .
  :a rdfs:subClassOf :b .
  :b rdfs:subClassOf :c .
  :c rdfs:subClassOf :d .
  :d rdfs:subClassOf :b .
  :c rdfs:subClassOf :e .
  :e rdfs:subClassOf :f .
  :f rdfs:subClassOf :g .
  :g rdfs:subClassOf :e .

  :a :relA :b .
  :a :relB :c .

  :relA rdfs:subPropertyOf :relB .

  :relA rdfs:range :b .
  :relB rdfs:domain :a .
```

should give you results similar to this:

```
Named classes: 7
Named properties: 2
Domain axioms: 1
Range axioms: 1
Subclass axioms: 8
Subproperty axioms: 1
Max. depth of class tree: 2
Max. depth of property tree: 1
```

Note that there are two loops in the subclass hierarchy of this graph, `:b` – `:c` – `:d` – `:b` and `:e` – `:f` – `:g` – `:e`, which can "interfere" with the maximum depth calculation of the subclass hierarchy, if you are not careful.

3.2 Exercise

Find the metrics of your family RDFS file. Are the results as expected? Why / why not?