EXERCISES INF3580 SPRING 2010 WEEK 7

Martin G. Skjæveland

10 March 2010

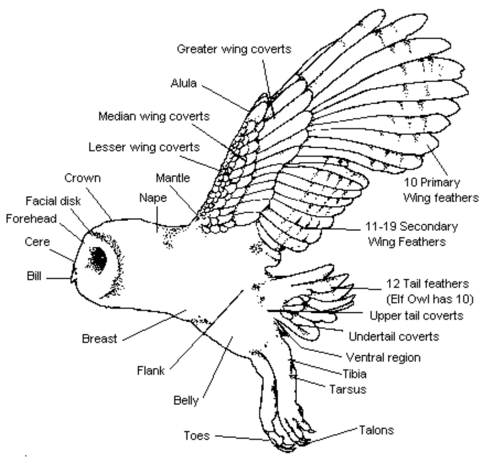

Figure 1: A bird.

This document contains exercises made for INF3580. Please send any comments, errors, bug or improvement reports to this exercise set to martige@ifi.uio.no[1] . Feedback is most welcome! Alphabetically thanks to Audun Stolpe, Espen H. Lian, Martin Giese and Rune Dahl for feedback.

The main curriculum for INF3580 spring 2010 is *Semantic Web Programming* by John Hebeler et al., Wiley Publishing, 2009. They have a website with additional articles and all source code used in the book at http://semwebprogramming.org/ . Auxiliary curriculum is the book *Foundations of Semantic Web Technologies* by Hitzler, Krützsch, Rudolph, CRC Press 2009.

Keep all the work you do for these exercises in a safe place. Setting up a version control system like `cvs`, `svn` or `git` for the work you do is smart. You can create a svn repository on IfI's svn server[2] , see their help section[3] for more information. There is also a walk-through[4] from old INF3120 on how to set up a `svn` repository and connect it to Eclipse, but news is that you'll need the plug-in subclipse[5] to make it work. Please contact me if you have any smart tips to share.

[1]`mailto:martige@ifi.uio.no`
[2]`https://wwws.ifi.uio.no/system/svn/`
[3]`https://wwws.ifi.uio.no/system/svn/help.cgi`
[4]`http://www.uio.no/studier/emner/matnat/ifi/INF3120/h06/studentarbeider/Prosjektoppgave/SVN_i_Eclipse.pdf`
[5]`http://subclipse.tigris.org/`

7 Reasoning

7.1 Querying with reasoning

7.1.1 Exercise

Write a program which extends the previous query program with the ability to read one or more models from file and query them with or without RDFS reasoning. The first parameter should be either `endpoint` or `file`, indicating if the model to be queried is an endpoint or file(s). If the first parameter given is `endpoint`, then the program should behave just as the previous program. If the first parameter is `file` then the program should treat all following parameters given, except the two last parameter, as URIs to RDF files, and collect them to one model, which is to be queried. The second to last parameter is the location of the SPARQL query. The last parameter is either `true` or `false` and indicated whether RDFS reasoning should be applied to the collected model prior to reasoning.

Running

```
java your_java_program file schema.rdf individuals.rdf query.rq true
```

should give you the answer of running the `query.rq` on the RDFS combined and inferred model of the files `schema.rdf` and `individuals.rdf`.

Running

```
java your_java_program file schema.rdf individuals.rdf query.rq false
```

should give you the answer of running the `query.rq` on the combined model of the files `schema.rdf` and `individuals.rdf`, but with no reasoning.

7.1.2 Exercise

Run the query “Find everyone that has a mother. . . ” which you have created in an earlier exercise on the Simpsons RDF file with and without reasoning. Compare and explain the two outputs.

7.1.3 Exercise

Run the query “Find Maggie’s grandmothers” on the Simpsons RDF file with and without reasoning. Compare and explain the two outputs.

7.1.4 Exercise

Run the query “Is Herb the brother of Homer” on the Simpsons RDF file with and without reasoning. Compare and explain the two outputs.

7.1.5 Exercise

Write a SPARQL query which answers the question “Who has Bart a family relationship to?”

Run the query both with and without reasoning and explain the results.

7.1.6 Exercise

Write a SPARQL query that lists all men and women. Run the query both with and without reasoning and explain the results.

7.1.7 Exercise

In the output in the previous exercise, from the query with reasoning enabled, is there someone missing, i.e., is there a person which is not classified as either a man or a woman? Why is that?

Is it possible to write a SPARQL query which lists all persons which are not either a man or a woman? Why / why not?

7.1.8 Exercise

In these exercises the output results for `SELECT` queries with reasoning enabled almost always returns more results then when reasoning is disabled. For which query/queries is this not the case? Why is it so?

7.2 Entailment calculation

7.2.1 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`.

7.2.2 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.

7.3 Entailment checker

7.3.1 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*.

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
```

7.3.2 Exercise

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

7.3.3 Exercise

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

7.3.4 Exercise

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

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.