EXERCISES INF3580 SPRING 2010 WEEK 4

Martin G. Skjæveland

08 March 2010

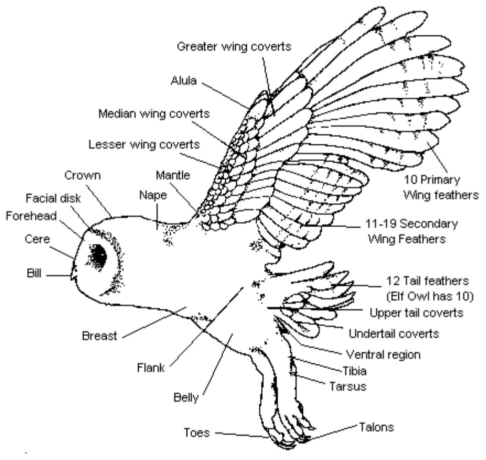

Figure 1: A bird.

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/`

4 SPARQL

Read

- Semantic Web Programming: chapter 6.
- Foundations of Semantic Web Technologies: chapter 7.

4.1 Query engine

In this exercise you are asked to make a SPARQL query engine.

4.1.1 Exercise

Write a java program which reads an RDF graph and a SPARQL query from file, queries the graph and outputs the query results as a table. Your program should accept `SELECT` queries, `CONSTRUCT` queries and `ASK` queries. A messages should be given if the query is of a different type.

Tip If I query the Simpsons RDF graph (`simpsons.rdf`) we wrote in a previous exercise with my SPARQL query engine and the `SELECT` query

```
PREFIX sim: <http://www.ifi.uio.no/INF3580/v10/ex/simpsons.rdf#>
PREFIX fam: <http://www.ifi.uio.no/INF3580/v10/ex/family.n3#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?s ?p ?o
WHERE{ ?s ?p ?o }
LIMIT 1
```

I get[6] To get the nicely formatted output I use the class ResultSetFormatter.

```
----------------------------------------
| s        | p             | o         |
========================================
| sim:Bart | fam:hasFather | sim:Homer |
----------------------------------------
```

Executing with the `ASK` query

```
ASK{ ?s ?o ?p }
```

gives me

```
true
```

Executing with the `CONSTRUCT` query

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX fam: <http://www.ifi.uio.no/INF3580/v10/ex/family.n3#>
PREFIX sim: <http://www.ifi.uio.no/INF3580/v10/ex/simpsons.rdf#>
CONSTRUCT{ sim:Bart rdfs:label ?name }
WHERE{ sim:Bart fam:hasName ?name }
```

[6] Note that your results may be different according to how your Simpsons RDF file looks like.

gives me

```
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix sim:     <http://www.ifi.uio.no/INF3580/v10/ex/simpsons.rdf#> .
@prefix fam:     <http://www.ifi.uio.no/INF3580/v10/ex/family.n3#> .

sim:Bart
      rdfs:label "Bart Simpson"^^xsd:string .
```

4.2 Query The Simpsons

In these exercises we will query the Simpsons RDF file we wrote in a previous exercise.

For each for the exercises below write the correct SPARQL query and execute it on the Simpsons RDF graph produced in the RDF exercise using your java SPARQL query engine. Explain the results. Are they what you expected? Pay special attention to blank nodes in the query outputs.

See W3C's SPARQL Query Language for RDF[7] for definitions and examples. The SPARQLer Validator[8] might also come in handy.

4.2.1 Exercise

Query: Find all Persons and order them by identifier.

Tip This is a simple query, where you will need to use `SELECT`, `WHERE` and `ORDER BY`. It is also important to set namespaces correctly in order for the query to work.

If you do not get any results, try the query

```
SELECT  ?s ?o ?p
WHERE  {?s ?o ?p}
```

and see if your get the expected results and namespaces. This query lists *all* triples in the graph.

4.2.2 Exercise

Query: Find all persons with a name and age. Output name and age. Order by name.

4.2.3 Exercise

Query: Find everyone which has a sibling and list both the person and the siblings. Order by person and sibling.

[7] http://www.w3.org/TR/rdf-sparql-query/
[8] http://www.sparql.org/validator.html

4.2.4 Exercise

Query: Find everyone that has a mother. List also their father and other parents, if there is information about this. Order results by person identifier.

4.2.5 Exercise

Query: Find everyone with a name with ‘M’ as first letter.

4.2.6 Exercise

Query: Find all of Maggie’s grandmothers.

4.2.7 Exercise

Query: Find everyone older than 10. Order by age, oldest first. Output name and age.

4.2.8 Exercise

Query: Is Herb the brother of Homer?

Tip Use `ASK`.

4.3 DBpedia

DBpedia is, according to DBpedia,

> a community effort to extract structured information from Wikipedia and to make this information available on the Web. DBpedia allows you to ask sophisticated queries against Wikipedia, and to link other data sets on the Web to Wikipedia data.

We will use their SPARQL endpoint, http://dbpedia.org/sparql , to extract some information.

There is also a more fancy GUI available, if you can get anything out of it: http://dbpedia.org/isparql/ .

Links:

- http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSSparqlProtocol

Note that DBpedia can be slow to respond.

4.3.1 Exercise

Extend your query engine so that it can query a SPARQL endpoint and not just a file.

A simple solution to differentiate between if the source to query is a file or a SPARQL endpoint is to let the program read three arguments, where the first argument specifies the source, e.g., running

```
java_program file simpsons.rdf sparql_query.rq
```

results in running the query `sparql_query.rq` on the file `simpsons.rdf`, just like the query engine you have already written in a previous exercise, while running

```
java_program endpoint http://some.sparql/endpoint/ sparql_query.rq
```

returns the result of querying the endpoint `http://some.sparql/endpoint/` with the query `sparql_query.rq`.

4.3.2 Exercise

Find all Simpsons characters in DBpedia and list their name, gender and relatives. Let gender and relatives be optional and list only names in English. Assume all characters have a `skos:subject` relation to Category:The_Simpsons_characters[9] , i.e.,

```
?character skos:subject
    <http://dbpedia.org/resource/Category:The_Simpsons_characters> .
```

Browse DBpedia to find the correct resource and property identifiers to use in your query.

4.3.3 Exercise

Make a `CONSTRUCT` query which creates a graph based on the `SELECT` query you made above. Type a character as `fam:Person` and use the properties

- `dbpfam:hasName`
- `dbpfam:hasGenderResource`
- `dbpfam:hasGenderLiteral`
- `dbpfam:hasRelative`
- `rdfs:label`

to relate the person to its name, gender and relatives. Use `hasGenderLiteral` if the value of gender is a literal, and `hasGenderResource` if the value is a resource[10]. The object values for `fam:hasRelationshipTo` must be resources and not literals. This means that you should ignore values of relatives given as literals. `rdfs:label` shall hold the name of the character.

[9]`http://dbpedia.org/resource/Category:The_Simpsons_characters`

[10]We create this odd construction because we are going to re-use this ontology later, and in OWL DL object properties and datatype properties are disjoint.

Tip Assume the table

Person	**Name**	**Gender**	**Relative**
`dbp:Marge_Simpson`	"Marge Simpson"	"Female"	`dbp:Maggie_Simpson`
`dbp:Lisa_Simpson`	"Lisa Simpson"	`dbp:Female`	"Father: Homer"

is the result of running your `SELECT` query. Then your `CONSTRUCT` query should produce the following RDF graph

```
dbp:Marge_Simpson a fam:Person ;
                  fam:hasName "Marge Simpson" ;
                  fam:hasGenderLiteral "Female" ;
                  fam:hasRelative dbp:Maggie_Simpson ;
                  rdfs:label "Marge Simpson" .

dbp:Lisa_Simpson a fam:Person ;
                 fam:hasName "Lisa Simpson" ;
                 fam:hasGenderResource dbp:Female ;
                 rdfs:label "Lisa Simpson" .
```

4.3.4 Exercise

Explain what a `DESCRIBE` SPARQL query is. Make an example using the DBpedia SPARQL endpoint.