EXERCISES INF3580 SPRING 2010 WEEK 6

Martin G. Skjæveland

12 March 2010

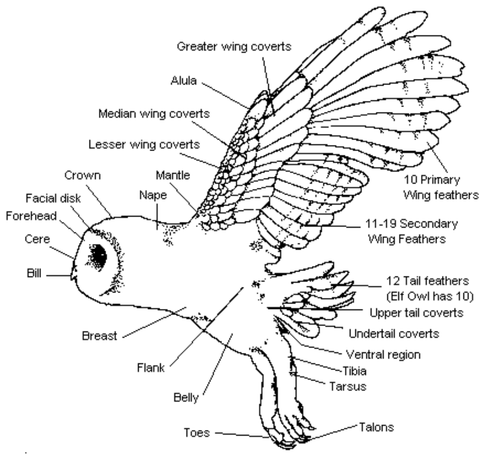

Figure 1: A bird.

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/`

6 RDFS

Read

- Semantic Web Programming: chapter 4, 5.
- Foundations of Semantic Web Technologies: chapter 2, 3.

6.1 RDFS vocabulary

We have already made an RDF representation of some of the members of the Simpson family. In this exercise we will use RDFS to define the classes and properties that we used to describe the Simpson family and also the classes and relationships that were missing from the vocabulary we were allowed to use in the RDF exercise.

We will also see how the added semantics for RDFS and the power of reasoners help us to discover implicit information.

Use RDFS to implement the following exercises. The syntax definition of RDFS is found at W3C's RDF Vocabulary Description Language 1.0: RDF Schema[6] .

6.1.1 Exercise

Create a new RDF file. Add all work in this week's exercises to this file. Let `fam` be the prefix for the namespace

```
http://www.ifi.uio.no/INF3580/v10/ex/family.n3#
```

6.1.2 Exercise

Create the classes and properties that we used in the RDF exercise, i.e., create the `rdfs:Class`-es

- `fam:Family`
- `fam:Person`

and the `rdf:Property`-es

- `fam:hasFamilyMember`
- `fam:hasBrother`
- `fam:hasSister`
- `fam:hasParent`
- `fam:hasMother`
- `fam:hasFather`
- `fam:hasSpouse`

[6] `http://www.w3.org/TR/rdf-schema/`

- `fam:hasName`

Tip This simply means that you shall state that `fam:Family` is an `rdfs:Class` and that `fam:hasBrother` is an `rdf:Property`:

```
fam:Family rdf:type rdfs:Class .
fam:hasBrother rdf:type rdf:Property .
```

6.1.3 Exercise

Add more classes and properties.

Classes:

- `fam:Man`
- `fam:Gender`
- `fam:Woman`

Properties:

- `fam:hasAunt`
- `fam:hasChild`
- `fam:hasDaugther`
- `fam:hasGender`
- `fam:hasGrandParent`
- `fam:hasHusband`
- `fam:hasSibling`
- `fam:hasSon`
- `fam:hasUncle`
- `fam:hasWife`
- `fam:hasAge`

6.1.4 Exercise

State that `fam:Female` and `fam:Male` are of type `fam:Gender`.

6.1.5 Exercise

Add an `rdf:Property` `fam:hasRelationshipTo` which has `fam:Person` as both domain and range. This is intended as a general property which holds between two individuals related by any family relationship.

6.1.6 Exercise

Set the domain and range of `fam:hasName` to respectively `fam:Person` and `xsd:string`, and the domain and range of `fam:hasAge` to `fam:Person` and `xsd:int` respectively.

6.1.7 Exercise

For each class you now have created add the correct `rdfs:subClassOf` property assertions. For each property add the correct `rdfs:subPropertyOf` property assertions and the correct `rdfs:domain` and `rdfs:range` assertions.

For the model we have created we assume the "obvious" semantics, e.g., that the class `Father` is the class of all fathers and `hasFather` is the relationship from a person to its father. If you are in doubt, choose the interpretation you think is correct.

Tip If we interpret the `hasSibling` property as a relationship between siblings and the `hasSister` property as a relationship from a sibling to a sister of the sibling, and interpret the classes `Person` and `Woman` in a way that fits this, then these relationships should look like the following in RDFS—this time in RDF/XML:

```
<rdf:Property rdf:about="#hasSibling">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Person"/>
</rdf:Property>

<rdf:Property rdf:about="#hasSister">
  <rdfs:subPropertyOf rdf:resource="#hasSibling"/>
  <rdfs:range rdf:resource="#Woman"/>
</rdf:Property>
```

6.2 The Simpson family

Now we will use the family vocabulary as a schema for the Simpsons data we have produced earlier, by opening both files in Protégé. Even though Protégé is an OWL editor it is quite safe to also load RDFS models as Protégé interprets them as OWL ontologies. This is not always the case, so it is best to use OWL if you do not need the meta-modelling capabilities of RDFS.

6.2.1 Exercise

Open Protégé and choose create a new OWL ontology. Give it the ontology URI

```
http://www.ifi.uio.no/INF3580/v10/ex/simpsons.owl
```

Save it to a file of your choice, and choose the format you would link to use. What format you choose will not be directly visible in the user interface of Protégé, but is used when saving the ontology to file.

6.2.2 Exercise

Import the Simpsons RDF file you wrote in the RDF week exercises and the Family RDFS file you have written in this week's exercises. (This is done by clicking the plus sign in the "Ontology Imports" pane on the starting page of Protégé and importing "an ontology contained in a specific file" for each of the RDF files.) Note that Protégé seems to have a problem *importing* files which are not in RDF/XML format, while *opening* files in different formats works better. If you have written your files in a format different from RDF/XML and you have problems with this exercise, try converting your files to RDF/XML with, e.g., RDF Validator and Converter[7] .

Then both files are successfully imported you should see the hierarchy of classes, properties and individuals under the tabs Classes, Object Properties and Data Properties respectively and Individuals. See also if your domain and range assertions look correct.

6.2.3 Exercise

Find the class Person in the Classes pane and see that it has quite a few members, while the classes Man and Woman have no members.

Now apply reasoning by choosing a reasoner, e.g., Pellet, in the Reasoner menu, and click Classify. . . in the same menu. Record any error messages that appear, you should get none.

If reasoning was successful, and assuming you have modelled classes and properties the same way as I have, you should see that the classes Man and Woman now have members.

Questions:

- What is the added statements about Bart after reasoning?
- In the Individuals list there might appear new individuals labelled genid1, genid2,. . . Explain what they are.
- Which individuals do not get any added information after reasoning?

6.3 RDFS metrics

6.3.1 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,

[7] `http://www.rdfabout.com/demo/validator/`

- 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.

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.

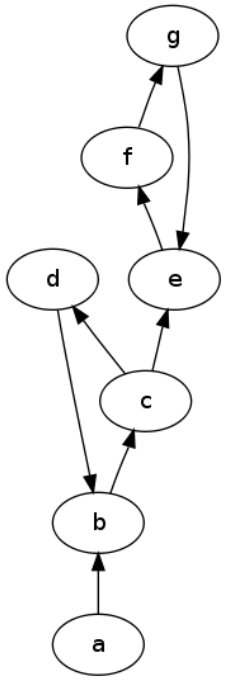

Figure 2: Two loops.

6.3.2 Exercise

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