# RDFS and reasoning

Read

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

## 1 Entailment

In these exercises we will learn about entailment and decide the logical consequences of RDFS statements.

Let `entailments.n3` be the file listed below, where `rdf` and `rdfs` are the usual namespaces.

```
:Person    a                  rdfs:Class .
:Man       a                  rdfs:Class ;
    rdfs:subClassOf    :Person .
:Parent    a                  rdfs:Class ;
    rdfs:subClassOf    :Person .
:Father    a                  rdfs:Class ;
    rdfs:subClassOf    :Parent ;
    rdfs:subClassOf    :Man .
:Child     a                  rdfs:Class ;
    rdfs:subClassOf    :Person .
:hasParent a                  rdf:Property ;
    rdfs:domain        :Person ;
    rdfs:range         :Parent .
:hasFather a                  rdf:Property ;
    rdfs:subPropertyOf :hasParent ;
    rdfs:range         :Father .
:isChildOf a                  rdf:Property ;
    rdfs:domain        :Child ;
    rdfs:range         :Parent .
:Ann       a                  :Person ;
    :hasFather         :Carl .
:Carl      a                  :Man .
```

:CODE: :END:

### 1.1 Exercise

Is `entailments.n3` syntactically correct RDF(S)?

### 1.1.1 Solution

Yes.

## 1.2 Exercise

Assuming the RDFS statements in `entailments.n3` are syntactically correct, are they semantically correct, i.e., do they give an accurate description of "the real world"?

## 1.3 Exercise

Explain what it means for one set of statements to entail a (different) set of statements.

### 1.3.1 Solution

Quoting RDF semantics[1] :

> Entailment is the key idea which connects model-theoretic semantics to real-world applications. As noted earlier, making an assertion amounts to claiming that the world is an interpretation which assigns the value true to the assertion. If A entails B, then any interpretation that makes A true also makes B true, so that an assertion of A already contains the same "meaning" as an assertion of B; one could say that the meaning of B is somehow contained in, or subsumed by, that of A.

# 2 Manual entailment calculation

In the following exercises decide if `entailment.n3` entails the statement(s) given and explain why/why not? If the answer is "yes, the statement(s) is entailed by `entailments.n3`", then use the simple entailment rules (`se1`, `se2`) and the rdfs entailment rules (`rdfs1`, . . . , `rdfs12`) found at RDFS entailment rules[2] to prove your answer. If the answer is "no", then explain, informally or formally, why this is so.

There are quite a few of these exercises, but many of them are quite easy so they should be quick to do. If they are too easy, then skip to the last ones, which are perhaps a bit harder.

## 2.1 Exercise

First, to get the an overview of the statements in `entailments.n3`, draw a diagram.

### 2.1.1 Solution

**Legend:** Nodes are represented in the diagram as nodes. A resource of type `rdfs:Class` is depicted with a circle. A resource of type `rdf:Property` is depicted with a diamond. A box is a resource which is not of type `rdfs:Class` or `rdf:Resource`. The resources `rdfs:Class`,

[1] http://www.w3.org/TR/rdf-mt/#entail
[2] http://www.w3.org/TR/rdf-mt/

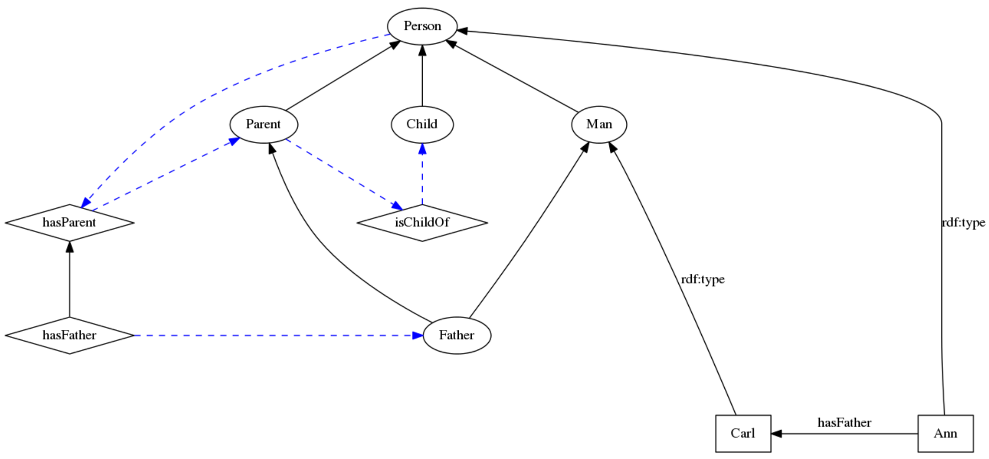

Figure 1: A diagram of the entailments.n3 graph.

`rdf:Property`, `rdf:type`, `rdfs:subClassOf` and `rdfs:subPropertyOf` are not marked by nodes in the diagram.

Triples are illustrated with edges, where the node which the edge points from is the subject, the edge itself is the predicate and the object is the node which the edge points to. A solid unlabelled edge represent an `rdfs:subClassOf` or a `rdfs:subPropertyOf` predicate. Blue, dashed edges show domain and range restrictions for properties. An edge pointing to a property represents the domain restriction for the property, a range restriction is marked with an outwards pointing edge.

## 2.2 Exercise

```
:Father rdfs:subClassOf :Person .
```

### 2.2.1 Solution

True. `:Father` is (transitively) a subclass of `:Person`. Rule rdfs11.

In the proof below each line is marked with "P" if the statement is a premise, i.e., exists in `entailments.n3`, or with the rule and the input statements to this rule by which the line in question is concluded.

Proof:

1. `:Father rdfs:subClassOf :Parent` — P
2. `:Parent rdfs:subClassOf :Person` — P
3. `:Father rdfs:subClassOf :Person` — 1, 2, rdfs11

Statements 1. and 2. are found in `entailments.n3` and are premises to the application of the entailment rule rdfs11 on line 3, which yields the statement we're after.

## 2.3 Exercise

```
:Man rdfs:subClassOf :Person .
```

### 2.3.1 Solution

True. This is explicitly stated in `entailments.n3`, so the entailment is trivial.

## 2.4 Exercise

```
:Carl a :Person .
```

### 2.4.1 Solution

True. `:Carl` is a `:Man`. `:Man` is a subclass of `:Person`. Thus, `:Carl` is a `:Person`. Rule rdfs9. Proof:

1. `:Man rdfs:subclassOf :Person` — P
2. `:Carl rdf:type :Man` — P
3. `:Carl rdf:type :Person` — 1, 2, rdfs9

## 2.5 Exercise

```
:Carl a :Parent .
```

### 2.5.1 Solution

True. Carl is in the range of `hasFather` (`:Ann :hasFather :Carl`). The range of `hasFather` is `:Father`. Thus, `:Carl` is a `:Father`. `:Father` is a subclass of `:Parent`, which makes `:Carl` a `:Parent`. Rule rdfs3 and rdfs9. Proof:

1. `:hasFather rdfs:range :Father` — P
2. `:Ann :hasFather :Carl` — P
3. `:Carl rdf:type :Father` — 1, 2, rdfs3
4. `:Father rdfs:subClassOf :Parent` — P
5. `:Carl rdf:type :Parent` — 4, 3, rdfs9

## 2.6 Exercise

```
:Carl :hasChild :Ann .
```

### 2.6.1 Solution

False. The predicate `:hasChild` does not exist in `entailments.n3`.

## 2.7 Exercise

```
:Carl a :Man .
```

### 2.7.1 Solution

True. Trivial. Statement is in `entailments.n3`.

## 2.8 Exercise

```
:Carl a :Father .
```

### 2.8.1 Solution

True. See `:Carl a :Parent`.

## 2.9 Exercise

```
:Child rdf:type rdfs:Resource .
```

### 2.9.1 Solution

True. Proof:

1. `:Child rdf:type rdfs:Class` — P
2. `:Child rdf:type rdfs:Resource` 1, rdfs4a

## 2.10 Exercise

```
:Ann a :Child .
```

### 2.10.1 Solution

False. The statements about Ann in `entailments.n3` are `:Ann a Person` and `:Ann :hasFather :Carl`. None of these statements force us to conclude that `:Ann` is a `:Child`.

## 2.11 Exercise

```
:Ann :isChildOf :Carl .
```

### 2.11.1 Solution

False. See `:Ann a :Child`.

## 2.12 Exercise

```
:Ann :hasParent :Carl .
```

### 2.12.1 Solution

True. `:Ann :hasFather :Carl` is a statement in `entailments.n3`. `:hasFather` is a subproperty of `:hasParent`. This means that all pairs related by the `:hasFather` property are also related by the `:hasParent` property. Rule rdfs7. Proof:

1. `:Ann :hasFather :Carl` — P
2. `:hasFather rdfs:subPropertyOf :hasParent` — P
3. `:Ann :hasParent :Carl` — 2, 1, rdfs7

## 2.13 Exercise

```
:Ann :hasParent _:x .
```

### 2.13.1 Solution

True. This follows from the results from the previous exercise. Since it is true that "Ann has a parent Carl", then it is true that "Ann has *some* parent". This is a conclusion by the application of a simple entailment rule. Proof:

1. `:Ann :hasParent :Carl` — P (from previous exercise)
2. `:Ann :hasParent _:x` — 1, se1

## 2.14 Exercise

```
:Ann :hasParent [ rdf:type :Person ] .
```

### 2.14.1 Solution

True. This follows from the results of the previous exercise and the fact that the fact that the range of the property `:hasParent` is `:Person`.

1. `:Ann :hasParent _:x` — P (from previous exercise)
2. `:hasParent rdfs:range :Parent` — P
3. `_:x rdf:type :Parent` — 2, 1, rdfs3
4. `:Parent rdfs:subClassOf :Person` — P
5. `_:x rdf:type :Person` — 4, 3, rdfs9

`:Ann :hasParent [ rdf:type :Person ]` is just an other form of writing the statements 1. and 5.

## 2.15 Exercise

```
:hasFather rdfs:domain :Person .
```

### 2.15.1 Solution

False. No RDFS entailment rule lets one derive a statement (see right-most column in the RDFS entailment rules table) about `rdfs:domain`.

With the "Extensional Entailment Rules" (`ext1`, ..., `ext9`) the entainment would be true, however, there rules are not included in the rdfs-rules, quoting the RDF Semantics document: #+begin$_{\text{quote}}$: The semantics given [above] is deliberately chosen to be the weakest 'reasonable' interpretation of the RDFS vocabulary. Semantic extensions MAY strengthen the range, domain, subclass and subproperty semantic conditions [. . . ]. #+end$_{\text{quote}}$

## 2.16 Exercise

```
rdfs:range rdf:type rdfs:Resource .
```

### 2.16.1 Solution

True. This statement is an axiomatic triple and is always satisfied in an RDFS model.

## 2.17 Exercise

```
:hasFather rdfs:range :Father .
```

### 2.17.1 Solution

True. Trivial. Statement is in `entailments.n3`.

## 2.18 Exercise

```
:hasFather rdfs:domain [ rdfs:subClassOf :Person ] .
```

### 2.18.1 Solution

False. See `:hasFather rdfs:domain :Person .` exercise.

## 2.19 Exercise

```
:Father rdfs:subClassOf [ rdfs:subClassOf :Person ] .
```

### 2.19.1 Solution

True. Proof:

1. `:Father rdfs:subClassOf :Parent` — P
2. `:Father rdfs:subClassOf _:n` — 1, se1

3. `:Parent rdfs:subClassOf :Person` — P
4. `_:n rdfs:subClassOf :Person` — 3, se2

Combine 2. and 4. to get `:Father rdfs:subClassOf [     rdfs:subClassOf :Person ]`.

## 3 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.

### 3.1 Exercise

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

`http://www.ifi.uio.no/INF3580/v16/simpsons.owl`

Save it to a file of your choice, and choose the format you would like 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.

#### 3.1.1 Solution

I chose Turtle. The saved file looks like this:

```
@prefix : <http://www.ifi.uio.no/INF3580/v16/simpsons.owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl2xml: <http://www.w3.org/2006/12/owl2-xml#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@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/simpsons#> .
@base <http://www.ifi.uio.no/INF3580/v16/simpsons.owl> .

  <http://www.ifi.uio.no/INF3580/v16/simpsons.owl> rdf:type owl:Ontology .
```

### 3.2 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[3] .

[3] `http://www.easyrdf.org/converter`

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.

### 3.2.1 Solution

```
<http://www.ifi.uio.no/INF3580/v16/simpsons.owl>
  owl:imports
    <http://www.ifi.uio.no/INF3580/simpsons> ,
    <http://www.ifi.uio.no/INF3580/family> .
```

## 3.3 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 named individuals do not get any added information after reasoning?

### 3.3.1 Solution

The added information about Bart after reasoning is that he

- hasRelationshipTo Marge, Homer
- hasParent Marge, Homer

The genid individuals are the blank nodes we already have met in the previous exercises. Notice that they also get added information after reasoning. In my case genid1 has Abraham as father. Now I know that Homer is the same individual as genid1, but this is not "discovered" by the reasoner. To make this happen we need to add restrictions so that they *necessarily* must be the same individuals.

The individuals that reasoning does not add any information to are Simpsons, Female, Male.