Model Semantics

Read

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

1 Definitions

First, some notation and definitions collected from the lecture slides.

1.1 Syntax: Triple abbreviations

Triple pattern	Triple instance	Abbreviation
`indi prop indi .`	i_1 r i_2	$r(i_1, i_2)$
`indi rdf:type class .`	i_1 `rdf:type` C	$C(i_1)$
`class rdfs:subClassOf class .`	C `rdfs:subClassOf` D	$C \sqsubseteq D$
`prop rdfs:subPropertyOf prop .`	r `rdfs:subPropertyOf` s	$r \sqsubseteq s$
`prop rdfs:domain class .`	r `rdfs:domain` C	$\mathsf{dom}(r, C)$
`prop rdfs:range class .`	r `rdfs:range` C	$\mathsf{rg}(r, C)$

1.2 Interpretation

An *interpretation* $\mathcal{I}$ consists of:

- A set $\Delta^{\mathcal{I}}$, called the *domain* $\mathcal{I}$
- For each individual URI i, an element $i^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
- For each class URI C, a subset $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
- For each property URI r, a relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

1.3 Validity in Interpretations (RDF)

Given an interpretation $\mathcal{I}$, define $\models$ as follows:

- $\mathcal{I} \models r(i_1, i_2)$ iff $\langle i_1^{\mathcal{I}}, i_2^{\mathcal{I}} \rangle \in r^{\mathcal{I}}$
- $\mathcal{I} \models C(i)$ iff $i^{\mathcal{I}} \in C^{\mathcal{I}}$

1.4 Validity in Interpretations, cont. (RDFS)

Given an interpretation $\mathcal{I}$, define $\models$ as follows:

- $\mathcal{I} \models C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
- $\mathcal{I} \models r \sqsubseteq s$ iff $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$
- $\mathcal{I} \models \mathsf{dom}(r, C)$ iff $\mathsf{dom}\ r^{\mathcal{I}} \subseteq C^{\mathcal{I}}$
- $\mathcal{I} \models \mathsf{rg}(r, C)$ iff $\mathsf{rg}\ r^{\mathcal{I}} \subseteq C^{\mathcal{I}}$

2 Exercises

In these exercises use the notation and definitions above in your answers.

2.1 Exercise

Let Γ be the RDF graph below.

1. Create an interpretation $\mathcal{I}_1$ such that $\mathcal{I}_1 \models \Gamma$.
2. Create an interpretation $\mathcal{I}_2$ such that $\mathcal{I}_2 \not\models \Gamma$.
3. Create an interpretation $\mathcal{I}_3$ such that $\mathcal{I}_3 \models \Gamma$ and $|\Delta^{\mathcal{I}_3}| = 1$, i.e., the domain of the interpretation contains only one element.

```
@prefix : <http://www.example.org#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
:Tweety rdf:type   :Bird .
:Nixon  rdf:type   :Republican .
:Nixon  rdf:type   :Quacker .
:Nixon  :listensTo :Tweety .
:Tweety :likes     :Tux .
```

2.1.1 Solution

1. There are many interpretations that satisfy the statements in the RDF graph. This is one:
   - $\Delta^{\mathcal{I}} = \{1, 2, 3, 4, 5\}$
   - `:Tweety` $^{\mathcal{I}} = 1$, `:Nixon` $^{\mathcal{I}} = 3$ `:Tux` $^{I} = 4$
   - `:Bird` $^{\mathcal{I}} = \{1, 2, 4, 5\}$, `:Republican` $^{\mathcal{I}} = \{3, 5\}$, `:Quacker` $^{\mathcal{I}} = \{3, 5, 4\}$
   - `:listensTo` $^{\mathcal{I}} = \{\langle 3, 1\rangle\}$, `:likes` $^{\mathcal{I}} = \{\langle 1, 4\rangle, \langle 3, 4\rangle\}$
2. There are also many ways to construct an interpretation that does not satisfy the RDF graph. Here are some examples of how:
   - `:Tweety` $^{\mathcal{I}} \notin$ `:Bird` $^{\mathcal{I}}$
   - `:Nixon` $^{\mathcal{I}} \notin$ `:Republican` $^{\mathcal{I}}$
   - `:Nixon` $^{\mathcal{I}} \notin$ `:Quacker` $^{\mathcal{I}}$
   - $\langle$ `:Nixon` $^{\mathcal{I}}$, `:Tweety` $^{\mathcal{I}}\rangle \notin$ `:listensTo` $^{\mathcal{I}}$

- $\langle$ `:Tweety` $^{\mathcal{I}}$, `:Tux` $^{\mathcal{I}}\rangle \notin$ `:likes` $^{\mathcal{I}}$

3. Let
   - $\Delta^{\mathcal{I}} = \{b\}$ (some set with one element.)
   - `:Tweety` $^{\mathcal{I}} =$ `:Nixon` $^{\mathcal{I}} =$ `:Tux` $^{\mathcal{I}} = b$
   - `:Bird` $^{\mathcal{I}} =$ `:Republican` $^{\mathcal{I}} =$ `:Quacker` $^{\mathcal{I}} = \Delta^{\mathcal{I}}$
   - `:listensTo` $^{\mathcal{I}} =$ `:likes` $^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

2.2 Exercise

Let Γ be the RDFS graph listed below.

1. Create an interpretation $\mathcal{I}_1$ such that $\mathcal{I}_1 \models \Gamma$.
2. Create an interpretation $\mathcal{I}_2$ such that $\mathcal{I}_2 \not\models \Gamma$.

```
@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#> .
: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 .
```

2.2.1 Solution

1. *One* possible solution (i.e., there are many):
   - $\Delta^{\mathcal{I}} = \{A, B, C\}$
   - `:Ann` $^{\mathcal{I}} = A$, `:Carl` $^{\mathcal{I}} = C$
   - `:Father` $^{\mathcal{I}}$ = `:Man` $^{\mathcal{I}}$ = `:Parent` $^{\mathcal{I}} = \{C\}$, `:Person` $^{\mathcal{I}} = \{A, B, C\}$, `:Child` $^{\mathcal{I}} = \{B\}$,
   - `:hasParent` $^{\mathcal{I}} =$ `:hasFather` $^{\mathcal{I}} = \{\langle A, C\rangle\}$ `:isChildOf` $^{\mathcal{I}} = \emptyset$

2. Let, e.g.,:
    - `:Person` $^{\mathcal{I}} \subset$ `:Child` $^{\mathcal{I}}$
    - `:Ann` $^{\mathcal{I}} \notin$ `:Person` $^{\mathcal{I}}$
    - `:hasFather` $^{\mathcal{I}} \not\subseteq$ `:hasParent` $^{\mathcal{I}}$

2.3 Exercise

Let Γ be the RDFS graph `entailments.n3`. Show by way of model semantics the following claims:

1. $\Gamma \models$ `:Father rdfs:subClassOf :Person .`
2. $\Gamma \not\models$ `:Ann a :Child .`
3. $\Gamma \models$ `:Ann :hasParent :Carl .`
4. $\Gamma \models$ `:Carl a :Person .`
5. $\Gamma \not\models$ `:Carl :hasChild :Ann .`

2.4 Exercise

Let Γ be the RDFS graph `entailments.n3`. As we have seen in a previous week's exercises, using the standardised RDFS semantics the entailment $\Gamma \models$ `:hasFather rdfs:domain :Person.` does not hold. Does it hold in our simplified semantics?

3 Querying with reasoning

3.1 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.

3.1.1 Solution

The program is called EvenMoreIrresistibleQueryEngine and extends the previous query engine IrresistibleQueryEngine.

```
import org.apache.jena.rdf.model.*;
import org.apache.jena.query.*;
public class EvenMoreIrresistibleQueryEngine extends IrresistibleQueryEngine{
```

The method `collectFilesToModel` which takes an array of filenames `files[]`, a starting point `start` in the array, and the number `length` of filenames to read from the array. Each file is read into a model and added to the model `model`.

```
public Model collectFilesToModel(String files[], int start, int length){
  Model model = ModelFactory.createDefaultModel();
  for(int i = start; i < length; i++){
    model = model.add(readModel(files[i]));
  }
  return model;
}
```

It contains two methods, one where RDFS reasoning is applied before querying and one where no reasoning is applied.

```
public void queryModel(Model model, String query_file){
  Query query = QueryFactory.read(query_file);
  QueryExecution qexec = QueryExecutionFactory.create(query, model);
  writeQueryResult(query, qexec);
}
public void queryModelWithReasoning(Model model, String query_file){
  model = ModelFactory.createRDFSModel(model);
  queryModel(model, query_file);
}
```

`main` orchestrates the whole thing.

```
  public static void main(String args[]){
    EvenMoreIrresistibleQueryEngine dave = new EvenMoreIrresistibleQueryEngine();
    if(args[0].equals("endpoint")){
      dave.queryEndpoint(args[1], args[2]);
    } else if(args[0].equals("file")){
      Model model = dave.collectFilesToModel(args, 1, args.length-2);
      boolean reasoning = Boolean.valueOf(args[args.length-1]);
      if(reasoning){
dave.queryModelWithReasoning(model, args[args.length-2]);
      } else{
dave.queryModel(model, args[args.length-2]);
      }
    } else{ /* informative error message */ }
  }
}
```

3.2 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.

3.2.1 Solution

Note the results in the father and mother column are the same with and without reasoning. The difference is in the parents column. Since all mothers and fathers are also parents, both `fam:hasMother` and `fam:hasFather` is a subproperty of `fam:hasParent`, the "reasoned" results returns more rows.

The results with reasoning *disabled* is displayed first.

```
-----------------------------------------------
| person     | mother     | father     | parent |
===============================================
| _:b0       | sim:Mona   |            |        |
| sim:Bart   | sim:Marge  | sim:Homer  |        |
| sim:Lisa   | sim:Marge  | sim:Homer  | _:b1   |
| sim:Lisa   | sim:Marge  | sim:Homer  | _:b2   |
| sim:Maggie | sim:Marge  | sim:Homer  | _:b0   |
| sim:Maggie | sim:Marge  | sim:Homer  | _:b3   |
-----------------------------------------------
```

```
--------------------------------------------------
| person     | mother     | father     | parent    |
==================================================
| _:b0       | sim:Mona   |            | sim:Mona  |
| sim:Bart   | sim:Marge  | sim:Homer  | sim:Homer |
| sim:Bart   | sim:Marge  | sim:Homer  | sim:Marge |
| sim:Lisa   | sim:Marge  | sim:Homer  | _:b1      |
| sim:Lisa   | sim:Marge  | sim:Homer  | _:b2      |
| sim:Lisa   | sim:Marge  | sim:Homer  | sim:Homer |
| sim:Lisa   | sim:Marge  | sim:Homer  | sim:Marge |
| sim:Maggie | sim:Marge  | sim:Homer  | _:b0      |
| sim:Maggie | sim:Marge  | sim:Homer  | _:b3      |
| sim:Maggie | sim:Marge  | sim:Homer  | sim:Homer |
| sim:Maggie | sim:Marge  | sim:Homer  | sim:Marge |
--------------------------------------------------
```

3.3 Exercise

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

3.3.1 Solution

With the reasoner enabled Jackeline is listed as grandmother of Maggie twice. The reason for this may be found in the results of the previous exercise. The query asks for all mothers to a parent of Maggie. Without reasoning Maggie has only two parents, which are the two blank nodes `_:b0` and `_:b1`. With reasoning enabled Marge is added as a parent of Maggie, and Marge's mother is Jackeline.

This is clearer if we also output the parents linking Maggie to her grandmothers.

```
------------------------
| grandmother | parent |
========================
| sim:Mona    | _:b0   |
------------------------
```

```
------------------------
| grandmother | parent |
========================
| sim:Mona    | _:b0   |
------------------------
```

3.4 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.

3.4.1 Solution

In the Simpsons RDF file Herb is represented as (only) the brother of a parent of Lisa. There is no information or restrictions in the family RDF that forces this parent to be Homer. (The anonymous person who Herb is the brother of could be Marge and it can be a third person. There is nothing in the family ontology restricting a person to only have two parents.) So both results are `false`.

```
false
```

```
false
```

3.5 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.

3.5.1 Solution

Family relations are represented with the `fam:isRelativeOf` property.

```
SELECT ?person
WHERE{ sim:Bart fam:isRelativeOf ?person }
ORDER BY ?person
```

The Simpsons RDF file contains no instances of `fam:isRelativeOf`, so no results are returned when the reasoner is turned off. With the reasoner enabled all the individuals related to Bart by a subproperty of `fam:isRelativeOf`, e.g., `fam:hasSister`, `fam:hasBrother`, `fam:hasMother` and `fam:hasFather`, is returned.

```
----------
| person |
==========
----------
```

```
-------------
| person    |
=============
| sim:Homer |
| sim:Marge |
-------------
```

3.6 Exercise

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

3.6.1 Solution

To get only men and woman, we filter the type to be either `fam:Man` or `fam:Woman`.

```
SELECT ?person ?type
WHERE
  { ?person a ?type .
    FILTER(?type = fam:Man || ?type = fam:Woman)
  } ORDER BY ?type ?person
```

With the reasoner disabled no results are returned. This is because no individuals are typed as either `fam:Man` or `fam:Woman` in the Simpsons RDF file. With the reasoner enabled individuals are classified as a man or a woman due to the domain and range restrictions set on properties in the family RDFS file.

```
-----------------
| person | type |
=================
-----------------
```

```
---------------------------
| person      | type      |
===========================
| sim:Abraham | fam:Man   |
| sim:Herb    | fam:Man   |
| sim:Homer   | fam:Man   |
| sim:Marge   | fam:Woman |
| sim:Mona    | fam:Woman |
| sim:Patty   | fam:Woman |
| sim:Selma   | fam:Woman |
---------------------------
```

3.7 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?

3.7.1 Solution

Bart, Lisa and Maggie are missing. From the Simpsons RDF there is not enough information to conclude their gender. They are "just" the children of Homer and Marge, which does not entail any information about their gender.

Yes, it is possible to write this query in SPARQL. Remember that SPARQL has to live by the open world assumption, but SPARQL does support *negation by failure* using `OPTIONAL`, `FILTER` and `!bound`. This is quite an elaborate construct, however, there seem to be new features coming to W3C's SPARQL soon: http://www.w3.org/TR/sparql-features/.

Some SPARQL engines, e.g., ARQ, already have support for negation queries.

I have restricted the output to persons having a name to avoid the anonymous individuals.

```
SELECT ?name
WHERE{
  ?person a foaf:Person ;
   foaf:name ?name .
  OPTIONAL {
    ?person a ?type .
    FILTER(?type = fam:Man || ?type = fam:Woman)
  }
  FILTER (!bound(?type))
  } ORDER BY ?type ?person
```

which gives the results

```
--------------------
| name             |
====================
| "Bart Simpson"   |
| "Homer Simpson"  |
| "Lisa Simpson"   |
| "Maggie Simpson" |
| "Marge Simpson"  |
--------------------

--------------------
| name             |
====================
| "Bart Simpson"   |
| "Lisa Simpson"   |
| "Maggie Simpson" |
--------------------
```

3.8 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?

3.8.1 Solution

The only `SELECT` query that does not return more results with the reasoner enabled is the query using *negation by failure*.

The chance of a failure in a negation by failure is less with more information.