Semantic Web – Lab 2&3

This lab module consists of 5 parts to be completed over 4 sessions (6 hours total). Below is a recommended timeline for each section. You may work **in pairs** (*binome*). You should submit a report including your results obtained in Part 3, 4, and 5, along with the code in Part 4.

Suggested Time Allocation

Part 1: <= 1.5 hours

Part 2 & 3: <=1.5 hours

Part 4: <= 2 hours (*additional time provided for Java beginners*)

Part 5: <= 1 hour

Part 1: Introduction to Protégé

Follow the instructions in "*introduction-protégé3.pdf*" and build your knowledge graph of pizza. The software is downdable here: https://protege.stanford.edu/

Don't forget to install the plug-ins PELLET, SnapSparql and SWRL.

The required files for this part are available in file *pizzas_owl.zip*.

Part 2: Reasoning and Query

The main goal is to design the family ontology, create individuals and infer new relations. Open the ontology file "*family_lab.owl*" with Protégé. This is an incomplete ontology and you need to complete it with the following instructions. Before starting your work, let's read the description of the classes and properties.

Description of the ontology

Classes and subclasses

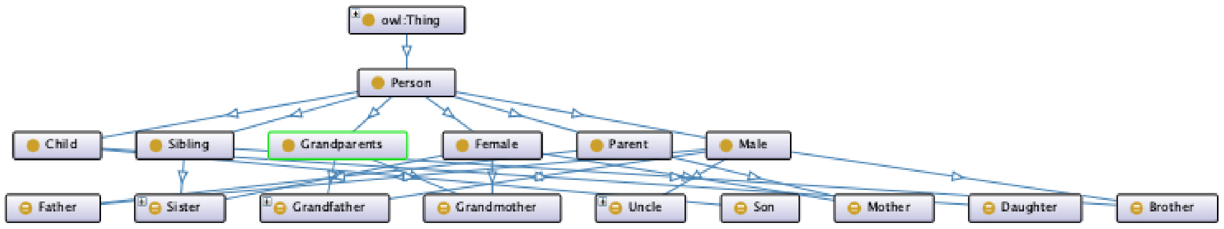

FIGURE 1:FAMILY ONTOLOGY

Class properties

1. Data properties

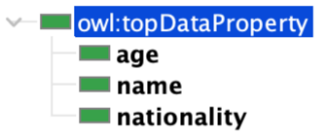

FIGURE 2.DATATYPE PROPERTIES

a. datatype property *name* with domain **Person** and range xsd:string
b. datatype property *age* with domain **Person** range xsd:int
c. datatype property *nationality* with domaine **Person** and range xsd:string

2. Object properties

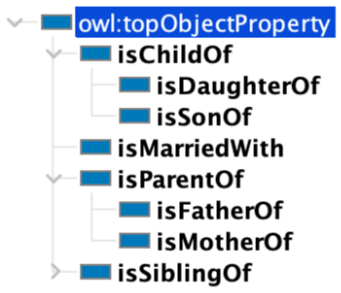

FIGURE 3. OBJECT PROPERTIES

a. object property *isMarriedWith* with **Person as** domain and range
b. A person is parent of another person. The object property *isParentOf* with **Person** as domain and range
c. A Male is father of person. The object property *isFatherOf* which is sub property of *isParentOf* with domain **Male** and range **Person**
d. A Female is mother of person. the object property *isMotherOf* which is sub property of *isParentOf* with domain **Female** and range **Person**
e. A person belongs to another person's siblings. The object property *isSiblingOf* with domain **Person** and range **Person**
f. A man is the brother of a person

i. the object property *isBrotherOf* which is sub property of *isSiblingOf* with domain **Male** and range **Person**

g. A Female is the sister of a person

   i. the object property *isSisterOf* which is sub property of *isSiblingOf* with domain **Female** and range **Person**

h. A person is a child of another person

   i. the object property *isChildOf* with domain **Person** and range **Person**

i. A Male is the son of a person

   i. the object property *isSonOf* which is sub property of *isChildOf* with domain **Male** and range **Person**

j. A woman is the daughter of a person

   i. the object property *isDaughterOf* which is sub property of *isChildOf* with domain **Female** and range **Person**

3. *Class and property restrictions.*

(Your work starts here!)

- Define the NECESSARY AND SUFFICIENT CONDITION :
  - An uncle has the restriction : *is brother of a parent*
  - A grandfather has the restriction : *is father of a parent*
  - A grandmother has the restriction : is mother of a parent
  - A father has the restriction : *isFatherOf* property has at least one instance
  - A mother has the restriction : *isMotherOf* property has at least one instance
  - A son has the restriction : *isSonOf* property has at least one instance
  - A daughter has the restriction : *isDaughterOf* property has at least one instance
  - A brother has the restriction : *isBrotherOf* property has at least one instance
  - A sister has the restriction : *isSisterOf* property has at least one instance
- Define the DISJOINTS CLASSES: (in Disjoint boxes)
  - Male and Female are disjoints
  - Father and Mother are disjoints
  - Son and Daughter are disjoints
  - GandFather and GrandMother are disjoints

4. *Assign types to properties*

- *MarriedWith* and *isSiblingOf* properties are symmetric
- *isSiblingOf* property is transitive
- *isChildOf* property is the inverse property of *isParentOf* property
- name, age and nationality are functional properties

5. *Individuals*

1. create individuals to Male class :

   a. Peter, 70, is*MarriedWith* Marie. He is French

   b. Thomas, 40, *isSonOf* Peter. He is French

   c. Paul, 38 *isSonOf* Peter

   d. John, 45, is italian

e. Pedro, 10, *isSonOf* John

f. Tom, 10, *isSonOf* Thomas and Alex

g. Michael, 5, *isSonOf* Thomas and Alex

2. create individuals to Female class :

a. Marie, 69, french

b. Sylvie, 30, *isDaughterOf* Marie and Peter

c. Chloe, 18, *isDaughterOf* Marie and Peter

d. Sylvie is*MarriedWith* Johne. Claude, 5, *isDaughterOf* Sylvie, french

f. Alex, 25, is*MarriedWith* Thomas

6. ***Check ontology consistency with PELLET***
   1. Check consistency using Pellet
   2. Infer instances of the following classes **Person**, **Uncle, Son, Sister, Grandparent, Grandfather, Grandmother;** check the instances generated automatically in the classes Person, Grandparent, Grandfather, Grandmother. No individual is generated in the classes Brother, Sister and Uncle. Why? How to infer for example the relations *isBrotherOf* or *isSister* starting from *childOf*?

In the next Lab (Lab 3), we will see how to express these kinds of restrictions using a rule language. Indeed, inference rules will solve this problem.

Part 3: SPARQL queries

- Use **SnapSparQl** plug-in for this lab
- Load the family ontology (owl) that you created in part 2.
- Write the following queries by checking with OWL inference:
  1. How old is Peter?
  2. Who are Sylvie's parents?
  3. The women over 30 years?
  4. What are the instances of Person?

Write and execute SPARQL queries to response to the following:

1. List the instances of the class Son
2. List the instances of the class Daughter
3. List the instances of the class Parent
4. List the instances of the class Father
5. List the instances of the class Mother
6. List the instances of the class Grandmother
7. List the instances of the class Grandfather
8. List the instances of the class Brother
9. List the instances of the class Sister

OPTIONAL: if the result is empty, rewrite a more complex SPARQL query, or an SWRL rule to return non empty result if possible

Write and execute SPARQL queries (several conditions in WHERE) to response to the following:

1. List (name, age) of children of Peter
2. List of persons whose father is more that 40 years old
3. List (name, age) of all French citizens. For each one, if he/she is married, display the name of his wife/her husband
4. List of the name of persons who are brother of someone
5. List of the name of persons who are daughter of someone
6. List of the name of persons who are uncle of someone
7. List of the name of persons who are married

Example of a SPARQL query:

```
PREFIX ns: <...#>
SELECT <variable>
WHERE{
        <triplet>.
        <triplet>.
        <fonction>.
}
```

Prefix defines the used namespaces.
? is used to denotes variables (example ?age)
Details about SPARQL can be found at:
http://jena.sourceforge.net/ARQ/Tutorial/

Part 4 Java programming using SPARQL with Jena

Download the file “semanticsJena.zip” and follow the instructions in “JenaWithMaven.pdf” to compile and execute the Java project with maven.

Here is the architecture of the project:

```
.
├── README.md
├── data
│   ├── family.owl.xml
│   ├── owlrules.txt
│   ├── query.txt
│   └── rules.txt
├── pom.xml
└── src
    ├── main
    │   ├── java
    │   └── resources
    └── test
        ├── java
```

Folder data consists of the ontology file family.owl.xml, SPARQL Query query.txt, SWRL rule rules.txt, and owlrules.txt importing the rule-based reasoning system, with details below:

File owlrules.txt

```
@include <OWLMicro>.
```

<OWLMicro> refers to a built-in rule set that implements a subset of OWL reasoning.

File ourrules.txt

```
@prefix ns: <http://www.owl-ontologies.com/unnamed.owl#>.

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.



[rule1: (?per rdf:type ns:Person) (?per ns:age ?age) greaterThan(?age, 60)->
(?per rdf:type ns:PersonneAge)]
```

This file defines a custom semantic rule for inference.

File query.txt

```
PREFIX ns: <http://www.owl-ontologies.com/unnamed.owl#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?user ?age

WHERE {

?user rdf:type ns:PersonneAge.

?user ns:age ?age.

}
```

Do the following jobs in the Java Project.

- Create query files that correspond to query from Part3 of this lab. Modify the Main class to refer to these query (one by one) and execute the Main program (compare the results with the ones obtained in Part 3)
- Add a new functionality in the JenaEngine class that allows to read the value of property of type ObjectType (see the javadoc of the Jena API)
- Add a new functionality in the JenaEngine class that allows to read the value of property of type DataType (see the javadoc of the Jena API)
- Create a new class in the application package that
  a. Read a name of a person

b. Display his/her parents, brothers and sisters.
c. If this person is married, display the name and age of her husband/his wife
d. If this person is not married Display all the persons
   ? whose gender is different,
   ? whose age is close (+/- 5 years) and
   ? is not married

Part 5: query dbpedia.org

Open the public SPARQL endpoint: http://dbpedia.org/snorql/ and test the following query to get information related to the resource http://dbpedia.org/resource/Thomas_Edison

```
SELECT ?property ?hasValue ?isValueOf
WHERE {
{ <http://dbpedia.org/resource/Thomas_Edison> ?property ?hasValu
e }
UNION
{ ?isValueOf ?property
<http://dbpedia.org/resource/Thomas_Edison> }
}
```

Based on the dbpedia page of Thomas Edison http://dbpedia.org/page/Thomas_Edison, the results of the previous query and using the following public SPARQL endpoint: http://dbpedia.org/snorql/, specify and test the following queries:

1. On what date was Thomas Edison Born?

2. What date did Edison Die?

Other queries:

1. US Presidents born in XXth Century

   Note: use ?President rdf:type <http://dbpedia.org/class/yago/WikicatPresidentsOfTheUnitedStates>.

2. US Presidents born in XXth Century and if available their death date.

3. List the American recipients of MacArthur Fellowships.