SW02

Prince,Prince

15^{th} May 2023

1 Task 1:

In this first three lines of turtle document, prefixes are defined. These name-spaces will be used as abbreviations through out the document. @prefix dbo , this sets dbo as shortcut for DBpedia ontology. This ontology defines number of classes like country and capital, properties like founding year, name etc. dbp is set as name-space for resources in dbpedia (germany, bielefeld). xsd is set name-space shortcut for XML schema definition, used for date type. This turtle document is providing two entities here: Germany and bielefeld, also providing some details about them. dbp:Germany a dbo:Country ; germany is type of country. dbo:capital dbp:Berlin, capital of germany is berlin. dbp:Bielefeld a dbo:Settlement, bielefeld is type of settlement, dbo:country dbp:Germany, bielefeld is in germany, dbo:name, this is providing two names for bielefeld (bielefeld and liebefeld) and "de" refers that these names are in german. dbo:population, indicates that population of bielefeld is 327199. dbo:foundingYear, specifies that Bielefeld was founded on the date 1214-01-01. The xsd:date signifies that value is date signifies according to xml.

Final Natural language content from this document can be:

1. Germany is identified as a country.
2. The capital of Germany is specified as Berlin.
3. Bielefeld is identified as a settlement.
4. Bielefeld is located in the country of Germany.
5. The names for Bielefeld in German are "Bielefeld" and "Liebefeld".
6. The population of Bielefeld is given as 327,199.
7. The founding date of Bielefeld is stated to be January 1, 1214.

2 Task 2:

```
@prefix ex: <http://example.org/> .

# Animals
ex:Elephant a ex:Animal .
ex:Lion a ex:Animal .
```

```
# Zoo
ex:Zoo1 a ex:Zoo ;
  ex:hasAnimal ex:Elephant ;
  ex:hasAnimal ex:Lion .

# Visitors
ex:Visitor1 a ex:Visitor ;
  ex:visit ex:Zoo1 .

ex:Visitor2 a ex:Family ;
  ex:visit ex:Zoo1 ;
  ex:use ex:FamilyTicket .

# Enclosures and Cages
ex:Enclosure1 a ex:Enclosure ;
  ex:contains ex:Elephant .

ex:Cage1 a ex:Cage ;
  ex:contains ex:Lion ;
  ex:isPartOf ex:Enclosure1 .
```

In this model as you can see in the plot 1:

Elephant and Lion are defined as Animals. Zoo1 is a Zoo that has these two Animals. Visitor1 is a general Visitor who visits Zoo1. Visitor2 is a Family that visits Zoo1 and uses a FamilyTicket. Enclosure1 is an Enclosure that contains the Elephant. Cage1 is a Cage that contains the Lion and is part of Enclosure1.

Describe where you are not satisfied with the modeling capabilities. What cannot be (easily) modeled in a machine-understandable way?

1. Cardinality Constraints are there in RDF, we cant say that one family uses only on family ticket. we cant say that zoo must have atleast one animal there.

2. we can not include negative constraints in the model. We cant say that a visitor can be a general visitor and a family at the same time.

3. quantativae relationships or comparisons can not be define with RDF, like we can not say that in zoo there must be more animals than zoo keepers or we can not say that in enclosure there must be less animal than its capacity.

5. No easy way to explain heirarichal relationships, we can say cage is part of enclosue using RDF property, but it still does not inherently make this clear.

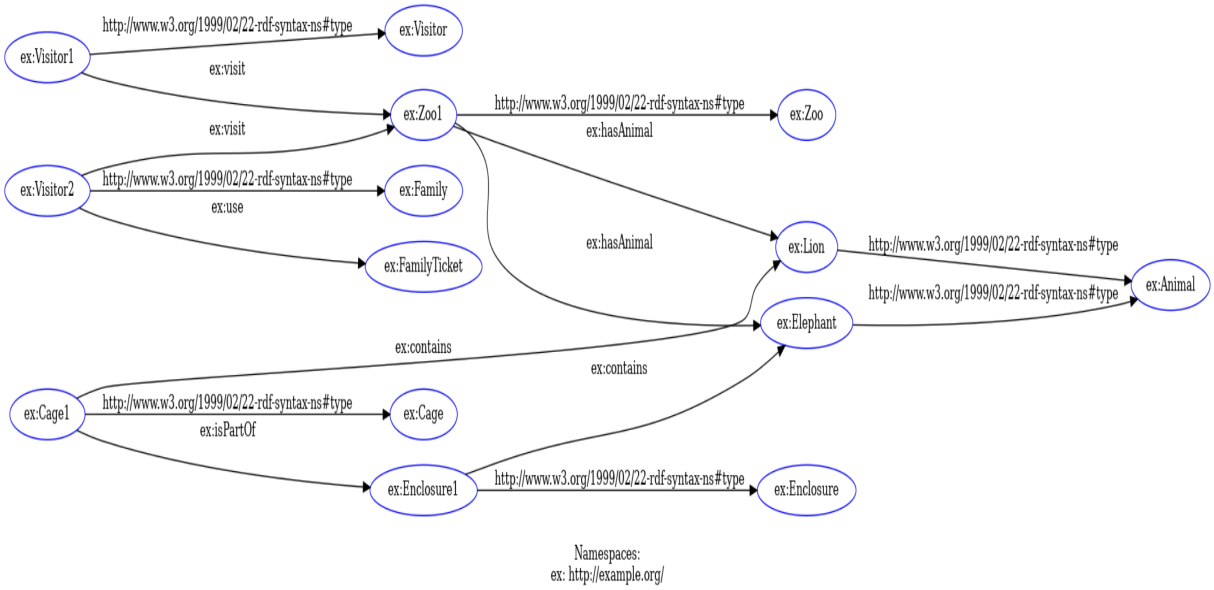

Figure 1: *RDF graph for task 2*

3 Task 3:

Reification is used when we want to express statements about statements. So, i am using predicate here then there will be four parts: subject, predicate, object and statement itself.

1. Charly says that Flipper is a mammal.

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

ex:Statement1 a rdf:Statement ;
  rdf:subject ex:Flipper ;
  rdf:predicate rdf:type ;
  rdf:object ex:Mammal .

ex:Charly ex:says ex:Statement1 .
```

In this tutrle document, I created a statement (Statement1) that says Flipper is a mammal. Then I am saying that Charly says this statement.

2. Charly says that his father says that Flipper is a fish.

```
ex:Statement1 a rdf:Statement ;
  rdf:subject ex:Flipper ;
  rdf:predicate rdf:type ;
```

```
  rdf:object ex:Fish .

ex:CharlyFather ex:says ex:Statement1 .

ex:Statement2 a rdf:Statement ;
  rdf:subject ex:CharlyFather ;
  rdf:predicate ex:says ;
  rdf:object ex:Statement2 .

ex:Charly ex:says ex:Statement2 .
```

In this second Turtle document, I first created a statement (Statement1) that says Flipper is a fish. Then I said that Charly's father says this statement (creating Statement2). Finally, I am saying that Charly says Statement2.

4 Task 4:

RDFS provides procedure to describe groups of related resources and relationship between resources. But it is less expressive than OWl.

1.Every convertible is a car: This can be satisfactorily modeled in RDFS using the "subClassOf" property. if we define convertible as subclass of car then it will mean that every convertible is a car.

2.Every car has a steering wheel: This cannot be satisfactorily modeled in RDFS. RDFS does not have built-in capabilities for defining universal quantities.

3. Every car has at least 1 door: this can not be statisfactorily modeled in RDFS, because of cardinality- constraints (number of instance of property that a resource can have).

4. Everything having a license plate is a vehicle: here we can define a domain "vehicle" for "hasLicensePlate", then it would mean everything that have license plat is vehicle. RDFS can satisfactorily model this.

5. No car can run upside down: this can be well defined by rule based languages like owl, not rdfs because it doesnot have way to provide negative constraints or assertions.

5 Task 5:

Based on the graph we have, puttign the list of explicit triples that we have:

```
<p:timbl> <foaf:made> <l:im> .
<p:douge> <rdf:type> <foaf:Person> .
<foaf:Person> <rdfs:subClassOf> <foaf:Agent> .
<foaf:made> <rdfs:domain> <foaf:Agent> .
<foaf:made> <rdfs:range> <foaf:Document> .
```

using rdfs entailment rule I could only derive 3 additional triples, which are entailed triples, from above 5 explicit one. Also, putting the entailment rule applied to derive these 3 triples:

```
<p:timbl> <rdf:type> <foaf:Agent>
```

rule used rdfs2: which states if E contains aaa (foaf:made) rdfs:domain xxx (foaf:Agent) and uuu (p:timbl) aaa (foaf:made) yyy then add uuu (p:timbl) rdf:type xxx (foaf:Agent).

```
<p:douge> <rdf:type> <foaf:Agent>
```

rule used rdfs9: which states if E contains uuu (foaf:Person) rdfs:subClassOf xxx (foaf:Agent) and vvv (p:douge) rdf:type uuu (foaf:Person) then add vvv (p:douge) rdf:type xxx (foaf:Agent).

```
<l:im> <rdf:type> <foaf:Document>
```

rule used rdfs3: which states if E contains aaa (foaf:made) rdfs:range xxx (foaf:Document) and uuu (p:timbl) aaa (foaf:made) vvv (l:im), then add vvv (l:im) rdf:type xxx (foaf:Document).