More OWL

Read

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

1 The Simpsons in OWL 2

In this exercise we will extend the family ontology by using OWL 2 vocabulary. OWL 2 adds expressivity to OWL DL without becoming undecidable. For an overview of the new features of OWL 2 see OWL 2 Web Ontology Language: New Features and Rationale.

1.1 Exercise

Create a new ontology which imports the family OWL ontology we made in last week's exercises.

1.1.1 Solution

```
 1  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
 2  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
 3  @prefix owl: <http://www.w3.org/2002/07/owl#> .
 4  @prefix owl2xml: <http://www.w3.org/2006/12/owl2-xml#> .
 5  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
 6  @prefix fam: <http://www.ifi.uio.no/INF3580/v18/family#> .
 7  @prefix : <http://www.ifi.uio.no/INF3580/v18/family.owl2.n3#> .
 8  @base <http://www.ifi.uio.no/INF3580/v18/family.owl2.n3> .
 9
10  <http://www.ifi.uio.no/INF3580/v18/family.owl2.n3> rdf:type owl:Ontology ;
11    owl:imports <http://www.ifi.uio.no/INF3580/v18/family.owl> .
```

1.2 Exercise

State that a person have exactly one gender.

1.2.1 Solution

To do this we use the `exactly` cardinality restriction.

```
hasGender exactly 1 Gender
```

In N3 this translates to:

```
12  foaf:Person rdfs:subClassOf
13    [ rdf:type owl:Restriction ;
14      owl:onProperty fam:hasGender ;
15      owl:onClass fam:Gender ;
16      owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger
17    ] .
```

1.3 Exercise

Explain what owl keys are.

How would you state that `foaf:name` is key for `Person`?

1.3.1 Solution

See `http://www.w3.org/2007/OWL/wiki/Easy_Keys`.

```
18  foaf:Person owl:hasKey (foaf:name) .
```

1.4 Exercise

Identify the correct characteristics of all properties in the ontology, i.e., which properties are

- asymmetric
- reflexive
- irreflexive

Explain what asymmetric, reflexive and irreflexive properties are. Use examples.

1.4.1 Tip

Note that the exercise says "identify" and not "add". The reason for this is that not all combinations of property characteristics are admissible in OWL 2. Adding more property characteristics carelessly will cause reasoners to report errors. For more information see the paper The even more irresistible SROIQ by Horrocks et al. SROIQ is the description logic which is the logical basis for OWL 2.

1.4.2 Solution

See wikipedia for explanations of the characteristics asymmetric, reflexive and irreflexive.

Assuming all relationships have Person as both domain and range, the following is an arguably common interpretation of their characteristics.

	Irreflexive	Asymmetric	Symmetric	Transitive
hasRelationshipTo	X		X	
hasSibling	x		X	X
hasBrother	x			X
hasSister	x			X
hasParent	x	X		
hasMother	x	X		
hasFather	x	X		
hasAunt	x	X		
hasUncle	x	X		
hasChild	x			
hasSon	x			
hasDaugther	x			
hasGrandParent	x			
hasSpouse	x		X	
hasHusband	x	X †)		
hasWife	x	X †)		

Notes:

- x: hasRelationshipTo is irreflexive, so all subproperties of it must also be.
- †) Assuming heteronormativity.

Relationships not considered:

- hasFamilyMember
- hasName
- hasAge
- hasGender

1.5 Exercise

Explain what it means for two properties to be disjoint. Add the natural disjoint axioms for properties in the ontology.

1.5.1 Solution

Formally, relations are sets of pairs of individuals, so stating that two relations R and S are disjoint is the same as stating that no pair (a, b) can be member of *both* R and S.

As an example, if `hasChild` and `hasSpouse` is disjoint, then the model is not accept that a parent may be the spouse of its own child—which seems quite fair. A more normal example is declaring `hasSister` and `hasBrother` as disjoint, but this statement is not necessary as the range of the two properties are respectively `Woman` and `Man`, which are already stated as disjoint.

This is an example of how to express disjointness of properties in OWL:

```
19  [ rdf:type owl:AllDisjointProperties ;
20    owl:members ( fam:hasFamilyMember
21   fam:hasGender
22   fam:isRelativeOf
23         ) ] .
```

1.6 Exercise

Explain what property chaining is. Using property chaining, define the properties

- `fam:hasAunt`
- `fam:hasGrandParent`
- `fam:hasUncle`

as equivalent to the correct property chain.

Create two new properties

- `:hasGrandMother`
- `:hasGrandFather`

in the same manner.

1.6.1 Solution

See the the book's homepage for an explanation of what property chaining is.

Property chaining is added in Protégé by selecting the property, e.g., `fam:hasAunt`, and adding the appropriate axiom in the slot "Property chains". The symbol for chaining is "o", e.g.,

```
hasParent o hasSister -> hasAunt
```

In Turtle it looks like this.

```
24  fam:hasAunt owl:propertyChainAxiom ( fam:hasParent fam:hasSister ) .
25  fam:hasUncle owl:propertyChainAxiom ( fam:hasParent fam:hasBrother ) .
26  fam:hasGrandParent owl:propertyChainAxiom ( fam:hasParent fam:hasParent ) .
27  fam:hasGrandMother owl:propertyChainAxiom ( fam:hasParent fam:hasMother ) .
28  fam:hasGrandFather owl:propertyChainAxiom ( fam:hasParent fam:hasFather ) .
```

1.7 Exercise

Define a class `Minor` as a `Person` which is under the age of 18.

Define a class `Juvenile` as a `Person` which is under the age of 16.

1.7.1 Solution

In Protégé `Minor` is defined by adding an axiom under Equivalent classes:

```
Person and hasAge some integer [< 18]
```

In the same manner `Juvenile` is defined as equivalent to

```
Person and hasAge some integer [< 16]
```

In Turtle syntax:

```
29  :Minor owl:equivalentClass
30    [ rdf:type owl:Class ;
31      owl:intersectionOf
32        ( foaf:Person
33          [ rdf:type owl:Restriction ;
34    owl:onProperty foaf:age ;
35    owl:someValuesFrom
36       [ rdf:type rdfs:Datatype ;
37         owl:onDatatype xsd:integer ;
38         owl:withRestrictions ( [ xsd:maxExclusive 18 ] )
39     ] ] ) ] .
40
41  :Juvenile owl:equivalentClass
42    [ rdf:type owl:Class ;
43      owl:intersectionOf
44        ( foaf:Person
45          [ rdf:type owl:Restriction ;
46    owl:onProperty foaf:age ;
47    owl:someValuesFrom
48       [ rdf:type rdfs:Datatype ;
49         owl:onDatatype xsd:integer ;
50         owl:withRestrictions ( [ xsd:maxExclusive 16 ] )
51     ] ] ) ] .
```

1.8 Exercise

Add a class `Job` and a object property `hasJob` with domain `Person` and range `Job`.

Define a class `Retiree` as a `Person` which is over 66 years old and has no `Job`.

1.8.1 Solution

```
52  :Job rdf:type owl:Class .
53
54  :hasJob rdf:type owl:ObjectProperty ;
55   rdfs:domain foaf:Person ;
56   rdfs:range :Job .
57
58  :Retiree owl:equivalentClass
59    [ rdf:type owl:Class ;
60      owl:intersectionOf
61      ( foaf:Person
62        [ rdf:type owl:Class ;
63          owl:complementOf
64          [ rdf:type owl:Restriction ;
65    owl:onProperty :hasJob ;
66    owl:someValuesFrom owl:Thing
67        ] ]
68        [ rdf:type owl:Restriction ;
69          owl:onProperty foaf:age ;
70          owl:someValuesFrom
71          [ rdf:type rdfs:Datatype ;
72    owl:onDatatype xsd:integer ;
73    owl:withRestrictions ( [ xsd:minExclusive 66 ] )
74        ] ]
75    ) ] .
```