Easter eggs

1 Logical puzzles

1.1 Exercise

According to Albert Einstein only 2 % of us are able to solve the Zebra Puzzle. Well, with a little help of OWL, it is not so difficult. Here it is:

1. There are five houses.
2. The Englishman lives in the red house.
3. The Spaniard owns the dog.
4. Coffee is drunk in the green house.
5. The Ukrainian drinks tea.
6. The green house is immediately to the right of the ivory house.
7. The Old Gold smoker owns snails.
8. Kools are smoked in the yellow house.
9. Milk is drunk in the middle house.
10. The Norwegian lives in the first house.
11. The man who smokes Chesterfields lives in the house next to the man with the fox.
12. Kools are smoked in a house next to the house where the horse is kept.
13. The Lucky Strike smoker drinks orange juice.
14. The Japanese smokes Parliaments.
15. The Norwegian lives next to the blue house.

Now, who drinks water? Who owns the zebra? In the interest of clarity, it must be added that each of the five houses is painted a different color, and their inhabitants are of different national extractions, own different pets, drink different beverages and smoke different brands of American cigarets [sic]. One other thing: in statement 6, right means your right.

The exercise is to encode the Zebra Puzzle in an ontology, so that the answer (Now, who drinks water? Who owns the zebra?) is computed by the reasoner.

1.1.1 Tip

The perhaps most difficult to do in this exercise is to express all the implicit information in the puzzle, e.g., that there are five houses on a row—where some houses are next to each other and some are not, that there are exactly five persons and that each person has exactly one favorite drink, and that the five drinks mentioned are the only ones considered. Remember that OWL ontologies and reasoning abides by the open world assumption and the none unique name assumption.

1.1.2 Solution

First I define an ontology to keep the rules of the puzzle, the implicit information given. The fifteen "explicit" statements listed in the puzzle text will be placed in a seperate ontology.

The high-level model of my ontology is that a *person* (or nationality) has exactly one *drink*, one *pet*, one *smoke* (cigarette brand) and lives in one *house*. A house has exactly one *color*.

Note that there are more than one way to model this, and that this solution might not be the "optimal". This may is ment to show some of the capabilities of OWL in a clear manner.

The file starts with the regular namespace declarations and that it is an ontology.

```
1  @prefix : <http://folk.uio.no/martige/what/2012/09/04/zebra_puzzle.ttl#> .
2  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3  @prefix owl: <http://www.w3.org/2002/07/owl#> .
4  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
6
7  <http://folk.uio.no/martige/what/2012/09/04/zebra_puzzle.ttl> rdf:type owl:Ontology .
```

All classes are defined as pairwise disjoint.

```
8  [ rdf:type owl:AllDisjointClasses ;
9    owl:members ( :Color :Drink :House :Person :Pet :Smoke ) ] .
```

Each class is also defined as equivalent to the set of its members. This is done to ensure that they cannot have other members (remember the OWA).

```
10  :Color rdf:type owl:Class ;
11         owl:equivalentClass [
12    rdf:type owl:Class ;
13    owl:oneOf ( :Blue :Yellow :Ivory :Green :Red ) ] .
14  :Drink rdf:type owl:Class ;
15         owl:equivalentClass [
16    rdf:type owl:Class ;
17    owl:oneOf ( :OrangeJuice :Water :Tea :Milk :Coffee ) ] .
18  :Pet rdf:type owl:Class ;
19       owl:equivalentClass [
20         rdf:type owl:Class ;
21         owl:oneOf ( :Horse :Zebra :Fox :Dog :Snails ) ] .
```

```
22  :Smoke rdf:type owl:Class ;
23         owl:equivalentClass [
24    rdf:type owl:Class ;
25    owl:oneOf
26    ( :Kools :Chesterfield :LuckyStrike :OldGold :Parliament ) ] .
27  :House rdf:type owl:Class ;
28         owl:equivalentClass [
29    rdf:type owl:Class ;
30    owl:oneOf ( :House5 :House4 :House2 :House3 :House1 ) ] .
31  :Person rdf:type owl:Class ;
32   owl:equivalentClass [
33     rdf:type owl:Class ;
34     owl:oneOf ( :Norwegian :Spaniard :Ukrainian :Japanese :Englishman ) ] .
```

The relationships between Person and Drink, Pet, Smoke and House, and between House and Color are defined by setting the correct domain and range. Additionally, all these ObjectProperties are defined as functional and inverse functional. This is to ensure that e.g., a person drinks at most one drink and that a drink is drunk by at most one person.

```
35  :drinks rdf:type owl:FunctionalProperty ,
36    owl:InverseFunctionalProperty ,
37    owl:ObjectProperty ;
38   rdfs:range :Drink ;
39   rdfs:domain :Person .
40
41  :hasColor rdf:type owl:FunctionalProperty ,
42      owl:InverseFunctionalProperty ,
43      owl:ObjectProperty ;
44     rdfs:range :Color ;
45     rdfs:domain :House .
46
47  :hasPet rdf:type owl:FunctionalProperty ,
48    owl:InverseFunctionalProperty ,
49    owl:ObjectProperty ;
50   rdfs:domain :Person ;
51   rdfs:range :Pet .
52
53  :livesIn rdf:type owl:FunctionalProperty ,
54     owl:InverseFunctionalProperty ,
55     owl:ObjectProperty ;
56    rdfs:range :House ;
57    rdfs:domain :Person .
58
59  :smokes rdf:type owl:FunctionalProperty ,
60    owl:InverseFunctionalProperty ,
61    owl:ObjectProperty ;
62   rdfs:domain :Person ;
63   rdfs:range :Smoke .
```

So far we have stated that a person drinks at most one drink. To make sure that a person does not drink zero drinks, we state that the class Person is subclass of the class that has one drinks relationship to a drink—and repeat an equivalent approach to the other classes that a person is related to.

```
64  :Person rdf:type owl:Class ;
```

```
65   rdfs:subClassOf
66      [ rdf:type owl:Class ;
67        owl:intersectionOf ( [ rdf:type owl:Restriction ;
68       owl:onProperty :drinks ;
69       owl:someValuesFrom owl:Thing
70     ]
71     [ rdf:type owl:Restriction ;
72       owl:onProperty :hasPet ;
73       owl:someValuesFrom owl:Thing
74     ]
75     [ rdf:type owl:Restriction ;
76       owl:onProperty :livesIn ;
77       owl:someValuesFrom owl:Thing
78     ]
79     [ rdf:type owl:Restriction ;
80       owl:onProperty :smokes ;
81       owl:someValuesFrom owl:Thing
82     ]
83   )
84   ] .
```

To say that there are five houses on a row is a statement with lots of implicit facts and can be broken down to the following statements, including some necessary vocabulary and a translation to OWL in mind:

1. There are five houses.
2. They are all different.
3. The second house lies to the right of the first house, the third to the right of the second house, and so on, but no house lies to the right of the fifth house and no house is left of the first house.
4. If a house A lies right of a house B, then B is left of A—and vice versa.
5. If a house A is to the right or to the left of a house B, then A is also next to B.
6. If a house A is next to a house B, then B is also next to the A.
7. No house is next to more than two houses.
8. No house is next to itself.

Now to OWL syntax. Each code block below is introduced with a reference to which of the statements above the block expresses.

Statement 6 is stated by defining isNextTo as a symmetric property.

Statement 8 is stated by defining the isNextTo as irreflexive.

Statement 5: isRightTo and isLeftTo are subPropertiesOf isNextTo.

Statement 4: isRightTo (is the) inverseOf isLeftTo, by which it follows that isLeftTo is the inverseOf isRightTo.

```
85  :isNextTo rdf:type owl:IrreflexiveProperty ,
86      owl:ObjectProperty ,
87      owl:SymmetricProperty ;
88     rdfs:domain :House ;
89     rdfs:range :House .
90
91  :isRightTo rdf:type owl:FunctionalProperty ,
92       owl:InverseFunctionalProperty ,
93       owl:ObjectProperty ;
94      owl:inverseOf :isLeftTo ;
95      rdfs:subPropertyOf :isNextTo .
96
97  :isLeftTo rdf:type owl:ObjectProperty ;
98     rdfs:subPropertyOf :isNextTo .
```

Statement 7: adding a cardinality restriction of max 2 isNextTo relations to the class House.

In the snippet below we also say that a house has one color and define the class as equivalent to the set of our five houses, just like we did for the Person and the relationships to Drink, Pet, House and Smoke.

```
 99  :House rdf:type owl:Class ;
100         rdfs:subClassOf [ rdf:type owl:Restriction ;
101    owl:onProperty :isNextTo ;
102    owl:onClass :House ;
103    owl:maxQualifiedCardinality "2"^^xsd:nonNegativeInteger
104          ] ,
105          [ rdf:type owl:Restriction ;
106    owl:onProperty :hasColor ;
107    owl:someValuesFrom owl:Thing
108          ] .
```

Statements 1 and 2.

```
109  [ rdf:type owl:AllDifferent ;
110    owl:distinctMembers ( :House2 :House1 :House4 :House3 :House5 )
111  ] .
```

Statement 3. To ensure that our five houses may not be placed in a circle, we state that the two houses on each end of the row, House1 and House5, are not next to each other. This is done with a NegativePropertyAssertion.

```
112  :House1 :isLeftTo :House2 .
113  :House2 :isLeftTo :House3 .
114  :House3 :isLeftTo :House4 .
115  :House4 :isLeftTo :House5 .
116
117  [ rdf:type owl:NegativePropertyAssertion ;
118    owl:targetIndividual :House1 ;
119    owl:sourceIndividual :House5 ;
120    owl:assertionProperty :isLeftTo
121  ] .
```

Finally, we define all the members of each class as distinct individuals. Note that we only define the members of each class as different. We need not state that Coffee is different from Blue since they are declared as members of classes which are are disjoint.

```
122  [ rdf:type owl:AllDifferent ;
123    owl:distinctMembers ( :Green :Ivory :Red :Yellow :Blue )
124  ] .
125  [ rdf:type owl:AllDifferent ;
126    owl:distinctMembers ( :Spaniard :Norwegian :Englishman :Ukrainian :Japanese )
127  ] .
128  [ rdf:type owl:AllDifferent ;
129    owl:distinctMembers ( :Tea :OrangeJuice :Water :Coffee :Milk )
130  ] .
131  [ rdf:type owl:AllDifferent ;
132    owl:distinctMembers ( :Dog :Zebra :Fox :Snails :Horse )
133  ] .
```

Next we create an ontology holding the fifteen statements, actually we only need fourteen, listed in the puzzle text. This ontology uses the vocabulary defined in the ontology above, and thus imports this ontology.

All statements are marked below. They are quite straight-forward, using blank nodes when a person or a house is not identified in the statement.

The first statement, *There are five houses*, is left out since it is already expressed in the other ontology.

```
 1  @prefix owl: <http://www.w3.org/2002/07/owl#> .
 2  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
 3  @prefix : <http://folk.uio.no/martige/what/2012/09/04/zebra_puzzle.ttl#> .
 4
 5  <http://folk.uio.no/martige/what/2012/09/04/zebra_puzzle_statements.ttl>
 6     rdf:type  owl:Ontology ;
 7     owl:imports  <http://folk.uio.no/martige/what/2012/09/04/zebra_puzzle.ttl> .
 8
 9  ### statement 2
10  :Englishman :livesIn [ :hasColor :Red ] .
11
12  ### statement 3
13  :Spaniard :hasPet :Dog .
14
15  ### statement 4
16  [] :drinks :Coffee ;
17     :livesIn [ :hasColor :Green ] .
18
19  ### statement 5
20  :Ukrainian :drinks :Tea .
21
22  ### statement 6
23  [] :hasColor :Green ;
24     :isRightTo [ :hasColor :Ivory ] .
25
26  ### statement 7
27  [] :smokes :OldGold ;
28     :hasPet :Snails .
29
30  ### statement 8
31  [] :smokes :Kools ;
32     :livesIn [ :hasColor :Yellow ] .
33
34  ### statement 9
35  [] :drinks :Milk ;
36     :livesIn :House3 .
37
38  ### statement 10
```

```
39  :Norwegian :livesIn :House1 .
40
41  ### statement 11
42  [] :smokes :Chesterfield ;
43     :livesIn [ :isNextTo _:x11 ] .
44  [] :livesIn _:x11 ;
45     :hasPet :Fox .
46
47  ### statement 12
48  [] :smokes :Kools;
49     :livesIn [ :isNextTo _:x12 ] .
50  [] :livesIn _:x12 ;
51     :hasPet :Horse .
52
53  ### statement 13
54  [] :smokes :LuckyStrike ;
55     :drinks :OrangeJuice .
56
57  ### statement 14
58  :Japanese :smokes :Parliament .
59
60  ### statement 15
61  :Norwegian :livesIn [ :isNextTo [ :hasColor :Blue ] ] .
```

Dr. Denis Ponomaryov has a different solution to the same puzzle, under the name Einstein's riddle, on his homepage.

2 Sudoku

Use SPARQL to solve a game of Sudoku.

2.1 Exercise

This is the same exercise as 7.6 page 302 in Foundations of Semantic Web Technologies.

Consider a small game of sudoku 4x4, which has the following setup

			3
			4
2			
3			

In (this game of) Sudoku the aim is to place the numbers 1–4 in the cells so that no number is duplicated for any row, any column or any of the four 2x2 squares that make up each corner of the game.

Create an RDF document and a SPARQL query that solves this game.

2.1.1 Tip

Use `FILTER` to state that all values on a row are distinct. Repeat this process so that it reflects the rules of the game.

2.1.2 Solution

Thanks to `http://semantic-web-grundlagen.de/` for the solution—borrowed under the Creative Commons Attribution-Noncommercial 3.0 Unported licence.

First we need an RDF document to collect the values to be placed on the game. This document is simply a holder for the values we need.

```
@prefix ex: <http://example.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
ex:square ex:allowed
 "1"^^xsd:int, "2"^^xsd:int, "3"^^xsd:int, "4"^^xsd:int .
```

The "SPARQL game board" is made up by 4x4 variables that represent the cells in the board. The cell variables are labelled to indicate their row and column placement, e.g., `?F22` is the cell in row 2 and column 2.

The SPARQL query collects the values from the document and continues to pose a series of restrictions by way of `FILTER`. Comments are interleaved in the query.

The reason I am using a `CONSTRUCT` query, and not a `SELECT` query which the other solution does, is simply to make the results fit inside a A4 pdf export.

```
PREFIX ex: <http://example.org/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
CONSTRUCT{
ex:square ex:F11 ?F11;
   ex:F12 ?F12;
   ex:F13 ?F13;
   ex:F14 ?F14;
   ex:F21 ?F21;
   ex:F22 ?F22;
   ex:F23 ?F23;
   ex:F24 ?F24;
   ex:F31 ?F31;
   ex:F32 ?F32;
   ex:F33 ?F33;
   ex:F34 ?F34;
   ex:F41 ?F41;
   ex:F42 ?F42;
   ex:F43 ?F43;
   ex:F44 ?F44.
}
WHERE {
ex:square ex:allowed ?F11, ?F12, ?F13, ?F14,
      ?F21, ?F22, ?F23, ?F24,
      ?F31, ?F32, ?F33, ?F34,
      ?F41, ?F42, ?F43, ?F44.

## intial game setup
FILTER ( ?F14 = "3"^^xsd:int )
FILTER ( ?F24 = "4"^^xsd:int )
FILTER ( ?F31 = "2"^^xsd:int )
FILTER ( ?F41 = "3"^^xsd:int )

## all values on the first row must be distinct
```

```
34  FILTER ( ?F11 != ?F12 ) FILTER ( ?F11 != ?F13 )
35  FILTER ( ?F11 != ?F14 ) FILTER ( ?F12 != ?F13 )
36  FILTER ( ?F12 != ?F14 ) FILTER ( ?F13 != ?F14 )
37
38  ## second row distinct
39  FILTER ( ?F21 != ?F22 ) FILTER ( ?F21 != ?F23 )
40  FILTER ( ?F21 != ?F24 ) FILTER ( ?F22 != ?F23 )
41  FILTER ( ?F22 != ?F24 ) FILTER ( ?F23 != ?F24 )
42
43  ## third row distinct
44  FILTER ( ?F31 != ?F32 ) FILTER ( ?F31 != ?F33 )
45  FILTER ( ?F31 != ?F34 ) FILTER ( ?F32 != ?F33 )
46  FILTER ( ?F32 != ?F34 ) FILTER ( ?F33 != ?F34 )
47
48  ## fourth row distinct
49  FILTER ( ?F41 != ?F42 ) FILTER ( ?F41 != ?F43 )
50  FILTER ( ?F41 != ?F44 ) FILTER ( ?F42 != ?F43 )
51  FILTER ( ?F42 != ?F44 ) FILTER ( ?F43 != ?F44 )
52
53  ## all values on the first column must be distinct
54  FILTER ( ?F11 != ?F21 ) FILTER ( ?F11 != ?F31 )
55  FILTER ( ?F11 != ?F41 ) FILTER ( ?F21 != ?F31 )
56  FILTER ( ?F21 != ?F41 ) FILTER ( ?F31 != ?F41 )
57
58  ## second column distinct
59  FILTER ( ?F12 != ?F22 ) FILTER ( ?F12 != ?F32 )
60  FILTER ( ?F12 != ?F42 ) FILTER ( ?F22 != ?F32 )
61  FILTER ( ?F22 != ?F42 ) FILTER ( ?F32 != ?F42 )
62
63  ## third column distinct
64  FILTER ( ?F13 != ?F23 ) FILTER ( ?F13 != ?F33 )
65  FILTER ( ?F13 != ?F43 ) FILTER ( ?F23 != ?F33 )
66  FILTER ( ?F23 != ?F43 ) FILTER ( ?F33 != ?F43 )
67
68  ## forth column distinct
69  FILTER ( ?F14 != ?F24 ) FILTER ( ?F14 != ?F34 )
70  FILTER ( ?F14 != ?F44 ) FILTER ( ?F24 != ?F34 )
71  FILTER ( ?F24 != ?F44 ) FILTER ( ?F34 != ?F44 )
72
73  ## the diagonals in the 2x2 boxes that make up the
74  ## board must be distinct. The other restrictions
75  ## for these boxes are covered by the FILTERS above
76  FILTER ( ?F11 != ?F22 ) FILTER ( ?F12 != ?F21 )
77  FILTER ( ?F13 != ?F24 ) FILTER ( ?F14 != ?F23 )
78  FILTER ( ?F31 != ?F42 ) FILTER ( ?F32 != ?F41 )
79  FILTER ( ?F33 != ?F44 ) FILTER ( ?F34 != ?F43 )
80  }
```

2.1.3 Results

The result of running this through my query program is:

```
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix ex:      <http://example.org/> .

ex:square
      ex:F11  "4"^^xsd:int ;
      ex:F12  "2"^^xsd:int ;
      ex:F13  "1"^^xsd:int ;
      ex:F14  "3"^^xsd:int ;
      ex:F21  "1"^^xsd:int ;
```

```
ex:F22  "3"^^xsd:int ;
ex:F23  "2"^^xsd:int ;
ex:F24  "4"^^xsd:int ;
ex:F31  "2"^^xsd:int ;
ex:F32  "4"^^xsd:int ;
ex:F33  "3"^^xsd:int ;
ex:F34  "1"^^xsd:int ;
ex:F41  "3"^^xsd:int ;
ex:F42  "1"^^xsd:int ;
ex:F43  "4"^^xsd:int ;
ex:F44  "2"^^xsd:int .
```