Foundations of Semantic Web Technologies

Tutorial 6

Dörthe Arndt

WS 2023/24

Exercise 6.1 (OWL-DL)**.** For our final OWL exercise, we will look at more complex entailments involving existentials, disjunctions and counting. For this we will use Pellet[1]: an OWL 2 DL reasoner based on a tableau rather than rules. Pellet will find all possible entailments and will halt on any valid input, but (unlike the previous rule-based reasoner) may reject ontologies with features that may lead to undecidability. These questions will be a little more open-ended to avoid giving too many hints, so you may have to add some common-sense definitions.

Unfortunately, I did not find an online tool for owl-dl reasoning which outputs turtle. Therefore, we use owl-cli, an ontology tool-kit which contains Pellet for this exercise. Download the jar file at `https://github.com/atextor/owl-cli/releases/download/snapshot/owl-cli-snapshot.jar`. You can run Pellet in your terminal (assuming that you have java installed) using `java -jar owl-cli-snapshot.jar infer in.ttl >out.ttl` where `in.ttl` is the file which contains your ontology and the additions you add and `out.ttl` is the file you produce. In case you have problems running the reasoner, you can alternatively also solve the exercise by writing down the solution on paper. We will discuss and test jointly in class in that case.

```
@prefix ex:  <http://ex.org/>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

ex:sire a owl:ObjectProperty .
ex:dam a owl:ObjectProperty .
ex:twin a owl:ObjectProperty .

ex:Zeus a ex:Zonkey ; ex:twin ex:Zev .

ex:Zach a ex:Zebroid ; ex:dam ex:Lea ; ex:sire ex:Marty .
ex:Zia ex:dam ex:Lea ; ex:sire ex:Marty .

ex:Zeb a ex:Zorse ; ex:dam ex:Lea ; ex:sire ex:Zamba .
ex:Zab ex:dam ex:Lea ; ex:sire ex:Zamba .

ex:Trip a ex:Horse ; ex:dam ex:Canela ; ex:sire ex:Jupiter .
ex:Hannah ex:dam ex:Zeta ; ex:sire ex:Jupiter .
ex:Zeta a ex:Zebra .

ex:dam rdfs:subPropertyOf ex:parent .
ex:sire rdfs:subPropertyOf ex:parent .

ex:Zebra rdfs:subClassOf ex:Equine .
ex:Donkey rdfs:subClassOf ex:Equine .
ex:Horse rdfs:subClassOf ex:Equine .
ex:Zebroid rdfs:subClassOf ex:Equine .
ex:Zorse rdfs:subClassOf ex:Equine .
ex:Hebra rdfs:subClassOf ex:Equine .
```

[1]https://github.com/stardog-union/pellet

```
        ex:Zonkey rdfs:subClassOf ex:Equine .

        [ a owl:AllDisjointClasses ;
        owl:members ( ex:Zebroid ex:Zebra ex:Donkey ex:Horse ) ] .
```

You can simply write your solutions at the bottom of the data (there is only one input form this time).

Note that `ex:dam` denotes an equine mother and `ex:sire` an equine father. The final axiom denotes that the listed classes are pairwise disjoint. To these data, you should add RDFS/OWL definitions and axioms that allow to infer the given data. Other inferences not listed may also arise. Your answers should not mention any specific individual like `ex:Zach`, `ex:Lea`, etc., but rather should only define axioms on classes and properties. For each question you may add one or more axioms; **you may have to add axioms not described by the question** but that intuitively reflect reality (e.g., defining that all equines have exactly two parents reflects reality but defining that they have only one parent does not reflect reality). Axioms should accumulate from question to question.

You can invent new classes or new properties, but this reasoner requires that classes (that do not have explicit instances in the data) are declared as such, and that properties are declared as datatype properties (taking literal values), or object properties (taking IRI or blank node values). So (though not necessary) if you wish to add a class such as `ex:NonZebraEquine` and a new (object) property such as `ex:child`, be sure to add:

```
        ex:NonZebraEquine a owl:Class .
        ex:child a owl:ObjectProperty .
```

Let's start!

(a) Noting that the twin of a *zonkey* is a *zonkey*, add axioms to infer that:

```
                ex:Zev a ex:Zonkey .
```

(b) Noting that a *zebroid* is any child of a zebra and a non-zebra equine, add axioms to infer that:

```
                ex:Zia a ex:Zebroid .
```

(c) Noting that a *zorse* is any child of a zebra sire and a horse dam, add axioms to infer that:

```
                ex:Zab a ex:Zorse .
                ex:Zach a ex:Zorse .
                ex:Zia a ex:Zorse .
```

(d) Noting that a *hebra* is any child of a zebra dam and a horse sire, add axioms to infer that:

```
                ex:Hannah a ex:Hebra .
```

If you look through the results, you'll find some other interesting conclusions: that Marty is a zebra, that zorse and hebra are both sub-classes of zebroid, etc. Note that if you try your solutions in RDF Playground, you will be missing inferences! This is because the (OWL 2 RL/RDF) rules that RDF Playground uses are incomplete.