UNDERSTANDING OWL EXISTENTIAL PROPERTY RESTRICTIONS

HENRIETTE HARMSE

For many starting out with OWL ontologies understanding the exact meaning of property restrictions can be challenging. In this post I will use visual representations to explain the meaning of existential property restrictions. For the purpose of this post, we start out with a simple ontology for modelling the relation between a person and their pets. You can model this in the free Protégé ontology editor.

```
ObjectProperty: owns

Class: Cat
  SubClassOf: Pet

Class: Dog
  SubClassOf: Pet

Class: Person
  DisjointWith: Pet

Class: Pet
  DisjointUnionOf: Cat, Dog
  DisjointWith: Person
```

I have specified this ontology using the OWL 2 Manchester syntax. Other syntaxes can be used as well, notably the functional-style syntax, which is used to define the direct semantics of OWL 2. In this post I use the Manchester syntax because it tends to be more intuitive to non-logicians, but it has some weaknesses which I will point when we get to it. Also for clarity and compactness I have ommitted the prefixes.

This ontology states that in our domain we have persons (`Class: Person`) and pets (`Class: Pet`). Persons cannot be pets, and pets cannot be persons (`Class: Pet DisjointWith: Person`). Cats (`Class: Cat SubclassOf: Pet`) and dogs (`Class: Dog SubclassOf: Pet`) are pets. We further assume that cats and dogs are the only pets in our domain, and a cat is not a dog and vice versa (`Class: Pet DisjointUnionOf: Cat, Dog`). I have also defined a `owns` object property, which will be the basis of our discussions on existential property restrictions. You can run the reasoner to confirm for yourself that this ontology is indeed consistent, that is, it contains no logical constradictions.

In this post I will explain through visual representations what is the meaning of

1. `owns some owl:Thing`,
2. `Class: AnimalLover SubClassOf: owns some owl:Thing`,

Date: 26th April 2018.

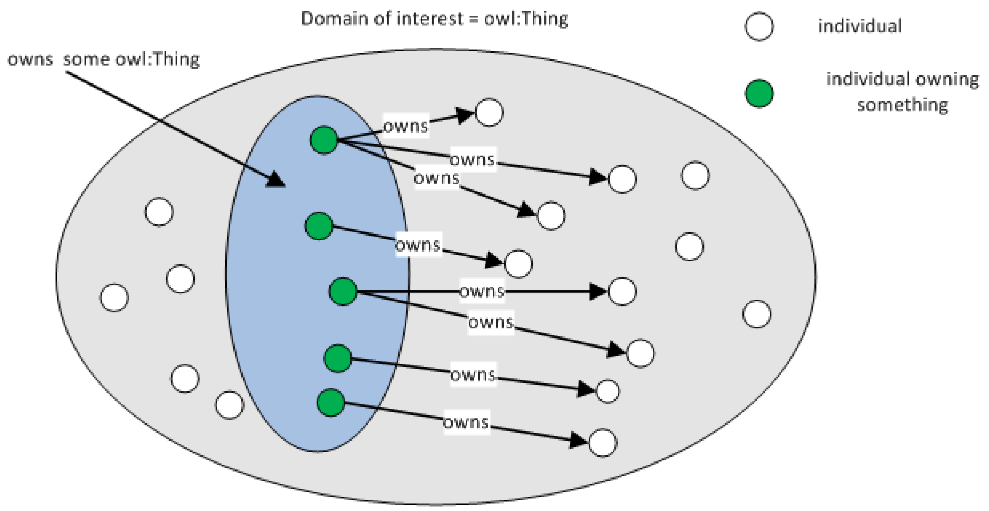


FIGURE 1. `owns some owl:Thing`

3. `owns some owl:Thing SubClassOf: AnimalLover`, and
4. `Class: AnimalLover EquivalentTo: owns some Pet`.

1. OWNS SOME OWL:THING

One of the first things one has to realize working with OWL ontologies is that OWL describes sets (in OWL called classes) and relations (in OWL called properties) between sets for some domain of interest. A domain of interest consists of elements (in OWL called individuals) that can belong to classes and/or form part of properties. The domain of interest represents the largest set of individuals that we are interested in, which in OWL is represented by `owl:Thing`. The smallest set we are interested in is the empty set, which in OWL is represented by `owl:Nothing`.

In Figure 1 I have illustrated an example domain consisting of some individuals (the small circles) and some `owns` properties (the arrows) that exist between individuals. At this stage I have not indicated the `Person` or `Pet` classes as yet. The meaning of `owns some owl:Thing` is that it represents the set of those individuals that we know owns something. In our example it consists of the individuals represented by the green circles (i.e. 5 individuals). Note that due to the `open world assumption`, we cannot assume that individuals that do not form part of the `owns` property, necessarily do not own anything. Rather, OWL reasoners assume that it is not know whether these individuals own something or do not own something. Because `owns some owl:Thing` represents a set OWL reasoners and ontology editors some times refer to an expression like `owns some owl:Thing` as an `anonymous class`. I.e. it is a class just like `Person`, but unlike `Person` it does not have name.

To refer specifically to owners of pets we have to define our existential restriction as `owns some Pet`. This is illustrated in Figure 2. As you can see `owns some Pet` is a subset (subclass) of `owns some owl:Thing`.

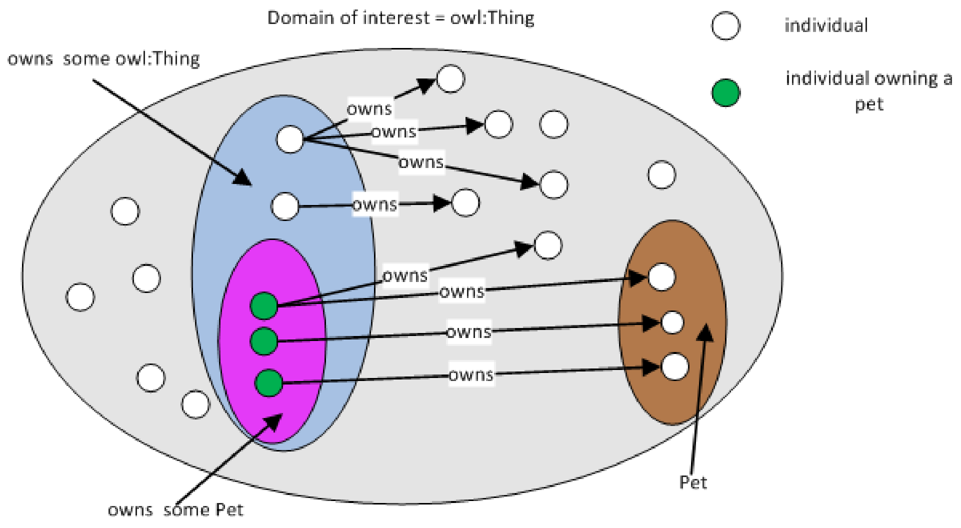


FIGURE 2. `owns some Pet`

2. CLASS: ANIMALLOVER SUBCLASSOF: OWNS SOME PET

Assume now we want to model a person that loves animals as someone who has at least 1 pet. We can model this as follows:

```
Class: AnimalLover
  SubClassOf:
    Person,
    owns some owl:Thing
```

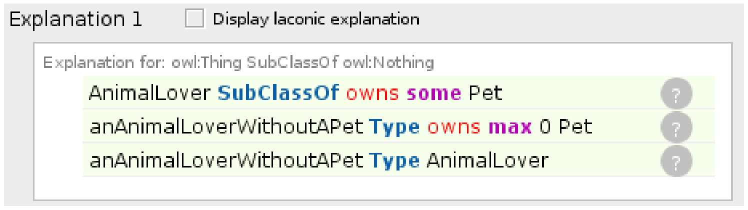
To test our ontology we create an individual without pet.

```
Individual: anAnimalLoverWithoutAPet
  Types: AnimalLover
```

Since our `anAnimalLoverWithoutAPet` individual is defined as belonging to the `AnimalLover` class and we have not stated that it owns a pet, we may expect that the reasoner will find our ontology inconsistent. However, this is not the case. The reason for this is again due to the open world assumption: there is nothing in our ontology that states that the `anAnimalLoverWithoutAPet` individual has no pets. To make explicit that `anAnimalLoverWithoutAPet` owns no pets, we change our definition of `anAnimalLoverWithoutAPet` as follows

```
Individual: anAnimalLoverWithoutAPet
  Types:
    AnimalLover,
    owns max 0 Pet
```

which states that `anAnimalLoverWithoutAPet` has a maximum of zero pets. If we now run the reasoner it will give an inconsistency. When your ontology gives an inconsistency when you expects it, it is good idea to check whether it gives an inconsistency for the correct reasons. This will help you to confirm whether you designed your ontology correctly for your desired outcomes. In this case the explanation for the inconsistency are given in Figure ??: It states that the inconsistency is due to the following reasons: (1) `anAnimalLoverWithoutAPet` ia an animal lover (`anAnimalLoverWithoutAPet Types AnimalLover`), (2) who does not own a pet



Explanation for: owl:Thing SubClassOf owl:Nothing

```
AnimalLover SubClassOf owns some Pet
anAnimalLoverWithoutAPet Type owns max 0 Pet
anAnimalLoverWithoutAPet Type AnimalLover
```

FIGURE 3. Explanation for an animal lover not owning a pet

`anAnimalLoverWithoutAPet Types owns max 0 owl:Thing`, (3) but the expectation is that an animal lover should own a pet (`AnimalLover SubClassOf owns some Pet`).

Explanations are minimal. That means that if you remove any 1 of the reasons given in an explanation, it is no longer an explanation for the inconsistency (or an entailment). Hence,

```
anAnimalLoverWithoutAPet Type AnimalLover
AnimalLover owns some Pet
```

is **not** an explanation for the inconsistency. This gives us a hint on how to remove an inconsistency from an ontology. If we can change our ontology such that any of the reasons given in an explanation for an inconsistency no longer holds, our ontology will be consistent. In this case we remove the `owns max 0 Pet` type from `anAnimalLoverWithoutAPet`:

```
Individual: anAnimalLoverWithoutAPet
  Types: AnimalLover
```

An incorrect assumption we may make based on the design of our ontology is that when ever an individual has a pet, the reasoner will infer that the individual is an `AnimalLover`. However, as I said, this assumption is incorrect. Thus, changing our ontology as follows

```
Individual: aCat
  Types: Cat

Individual: aPetOwner
  Facts:
    owns aCat
```

will not result in inferring that `aPetOwner` is an `AnimalLover`.

To understand this see Figure 4. Because we have defined `AnimalLover` as a subclass of `owns some Pet` it means the following possibilities exist:

1. There may be persons who have pets who do not love animals.
2. There may be individuals that have pets who are not persons.

Hence, from our current ontology, the reasoner can infer nothing more.

3. owns some Pet SubClassOf: AnimalLover

Now in this section, instead of defining `AnimalLover` as a subclass of `owns some Pet`, let us use a general class axiom and define `owns some Pet SubClassOf: AnimalLover` with `AnimalLover` defined as follows:

```
Class: AnimalLover
  SubClassOf: Person
```

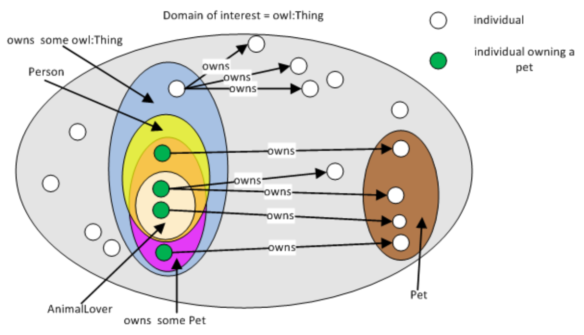


FIGURE 4. `AnimalLover SubClassOf: owns some Pet`

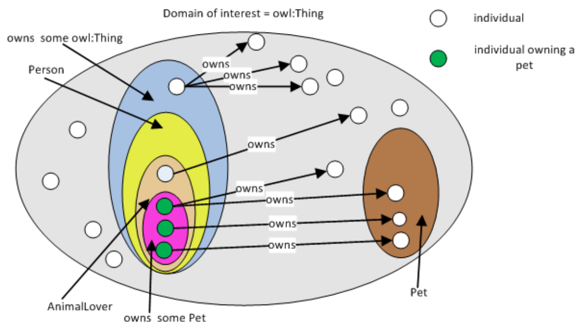


FIGURE 5. `owns some Pet SubClassOf: AnimalLover`

Note that if you now save your ontology in Manchester syntax, you will loose the `owns some owl:Thing SubClassOf: AnimalLover` general class axiom. Rather save it in say RDF/XML or OWL/XML syntax. We also add the following individuals:

```
Individual: aCat

Individual: aPetOwner
  Facts:
    owns aCat
```

Ensure that you have specified no type information for `aCat` and `aPetOwner`. If you run the reasoner it will not infer that `aCat` is of type `Cat`. This is because `aCat` is not of type `Cat`. If you state that `aCat` is of type `Cat`, it will infer that `aPetOwner` is an `AnimalLover`.

One reason why this design may not be ideal is that if we again consider the case where an individual of type `AnimalLover` wdoes not have a pet, it will not give an inconsistency.

```
Individual: anAnimalLoverWithoutAPet
  Types:
    AnimalLover,
    owns max 0 Pet
```

To understand the reason why the reasoner cannot make this inference, see Figure 5. Because we defined `owns some Pet SubClassOf: AnimalLover` the possibility exists that there are `AnimalLover`s who do not own any pets.

4. Class: AnimalLover EquivalentTo: owns some Pet

From the previous 2 sections we have noted that

5. Conclusion

github [?]

References