UNDERSTANDING OWL UNIVERSAL PROPERTY RESTRICTIONS

HENRIETTE HARMSE

In my previous post I explained existential property restrictions. In this post I want to deal with universal property restrictions. In designing ontologies existential property restrictions tend to be used more often than universal property restrictions. However, beginners in ontology design tend to prefer universal property restrictions, which can lead to inconsistencies that can be very difficult to debug, even for experienced ontology designers [1, 2].

We again start with a very simple ontolgy:

```
ObjectProperty: owns

Class: Cat
  SubClassOf: Pet

Class: Dog
  SubClassOf: Pet

Class: Person
  DisjointWith: Pet

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

We will use this ontology as basis for explaining universal property restrictions. We assume we want to extend this ontology with a `DogLover` class to represent persons owning only dogs. In this post

1. I will introduce the `DogLover` class which I will define as `Class: DogLover EquivalentTo: owns only Dog`,
2. I will show you how this definition can lead to inconsistencies that can be difficult to debug, and
3. I will explain how we can fix this error.

1. Class: DogLover EquivalentTo: owns only Dog

Let us start out by defining the `DogLover` class as

```
Class: DogLover
  EquivalentTo: owns only Dog
```

We can run the reasoner to ensure that our ontology is currently consistent. We can test our ontology by adding a `aDogOnlyOwner` individual owning a `Dog`.

```
Individual: aDog
  Types: Dog
```

Date: 11th May 2018.

FIGURE 1. Explanation for `DogLover` with a cat

```
Individual: aDogOnlyOwner
  Facts: owns aDog
```

If we run the reasoner it will **not** infer that `aDogOnlyOwner` is a `DogLover`, reason being that due to the open world assumption the reasoner has no information from which it can derive that `aDogOnlyOwner` owns only dogs. We can try and fix this by stating that

```
Individual: aDogOnlyOwner
  Types: owns max 0 Cat
```

Again the reasoner will **not** infer that `aDogOnlyOwner` owns only dogs. This is because it still allows for the possibility that `aDogOnlyOwner` can own, say a house. To exclude all other options we have to define `aDogOnlyOwner` as follows:

```
Individual: aDogOnlyOwner
  Types: owns max 0 (not Dog)
```

which enforces that `aDogOnlyOwner` owns nothing besides dogs. The reasoner will now infer that `aDogOnlyOwner` is a `DogLover`. We can also test that individuals of type `DogLover` cannot own cats, for example:

```
Individual: aCat
  Types: Cat

Individual: aDogLover
  Types: DogLover
  Facts: owns aCat
```

If we run the reasoner, it will give an inconsistency. As I said in my previous post, it is always a good idea to review the explanations for an inconsistency, even when you expect an inconsistency, as seen in Figure 1. It states that the inconsistency is due to

1. `aDogLover` owning a cat (`aDogLover owns aCat`),
2. a cat is not a dog (`Pet DisjointUnionOf Cat, Dog`),
3. an individual that loves dogs owns only dogs (`DogLover EquivalentTo owns only Dog`),
4. the `aDogLover` individual is of type `DogLover` (`aDogLover Type DogLover`), and
5. the `aCat` individual is of type `Cat` (`aCat Type Cat`).

At this point you may think our definition of `DogLover` is exactly what we need, but it contains a rather serious flaw.

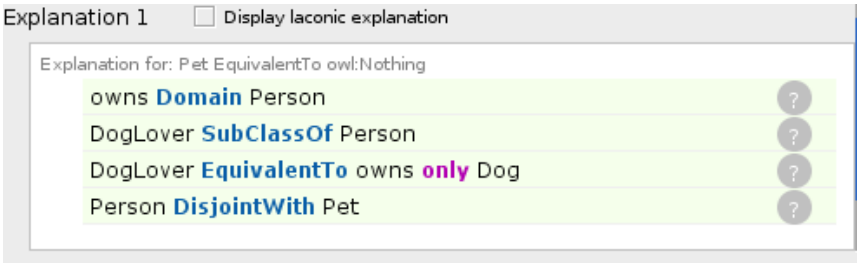

FIGURE 2. Explanation for `Pet` equivalent to `owl:Nothing`

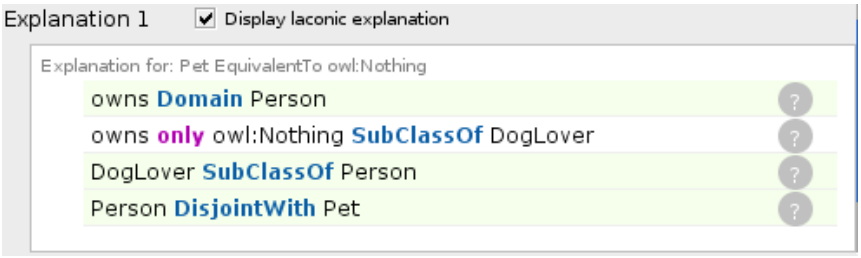

FIGURE 3. Laconic explanation for `Pet` equivalent to `owl:Nothing`

2. A Serious Flaw

To keep our ontology as simple as possible for this section, please remove any individuals you may have added to your ontology, but leave the `DogLover` class as we have defined it in the previous section. Just to ensure that our ontology is consistent, you run the reasoner to confirm that it is consistent. What we now want to do is to infer that when an individual owns something, that individual is a person. The way we can achieve this is by defining the domain for the `owns` property:

```
ObectProperty: owns
  Domain: Person
```

Now, this looks like a rather innocent change, but when you run the reasoner again you will find that `Pet`, `Cat` and `Dog` are all equivalent to `owl:Nothing` while `Person` is equivalent to `owl:Thing`. An explanation for why `Pet` is equivalent to `owl:Nothing` is given in Figure 2.

The explanation given in Figure 2 can be difficult to understand. Indeed, research has shown that there are explanations that are difficult to understand even for experienced ontology designers [1, 2]. In cases where it is hard to understand explanations, using laconic explanations can be helpful. Laconic justifications aim to remove subexpressions from axioms that do not contribute to explaining an entailment or inconsistency [1, 2]. Ticking the "Display laconic explanation" displays the laconic explanation in Figure 3.

The main difference between the explanations in Figures 2 and 3 are

```
DogLover EquivalentTo owns only Dog
```

versus

```
owns only owl:Nothing SubClassOf DogLover
```

Where does `owns only owl:nothing SubClassOf DogLover` come from? Recall that `A EquivalentTo B` is just syntactical sugar for the two axioms `A SubClassOf B` and `B SubClassOf A`. What this explanation is saying is that there is a problem with the `owns only Dog SubClassOf DogLover` part of our axiom (there is no problem with the `DogLover SubClassOf owns only Dog` part of our axiom). Furthermore, it states that `Dog` is inferred to be equivalent to `owl:Nothing`. For this we need to understand the meaning of `owns only Dog` better.

In Figure 4 I give an example domain where I make the assumption that for all individuals all ownership information is specified explicitly. Thus, individuals with no ownership links own nothing and individuals with ownership links own only what is specified and nothing else. The `owns only Dog` class includes all individuals that are known to own nothing besides dogs. However, a confusing aspect of the semantics of universal restrictions is that it also includes those individuals that owns nothing. To confirm this for yourself you can use the following ontology (from which I removed the domain restriction for the moment). This ontology will infer that `anIndividualOwningNothing` is of type `DogLover`.

```
ObjectProperty: owns

Class: Cat
  SubClassOf: Pet

Class: Dog
  SubClassOf: Pet

Class: Person
  DisjointWith: Pet

Class: Pet
  DisjointUnionOf: Cat, Dog
  DisjointWith: Person

Class: DogLover
  SubClassOf: Person
  owns only Dog SubClassOf: DogLover

Individual: anIndividualOwningNothing
  Types: owns only (not owl:Thing)
```

We are now ready to explain why `Pet` is equivalent to `owl:Nothing`:

1. `owns Domain Person` states that whenever an individual owns something, that individual is a `Person`.
2. `owns only Pet SubClassOf DogLover` includes saying that when an individual owns nothing at all, that individual is a `DogLover`.
3. Since `DogLover` is a subclass of `Person` it means `Person` now includes individuals that owns something and individuals that owns nothing, which means `Person` is equivalent to `owl:Thing`.
4. `Person` and `Pet` are disjoint, hence `Pet` must be equivalent to `owl:Nothing`.

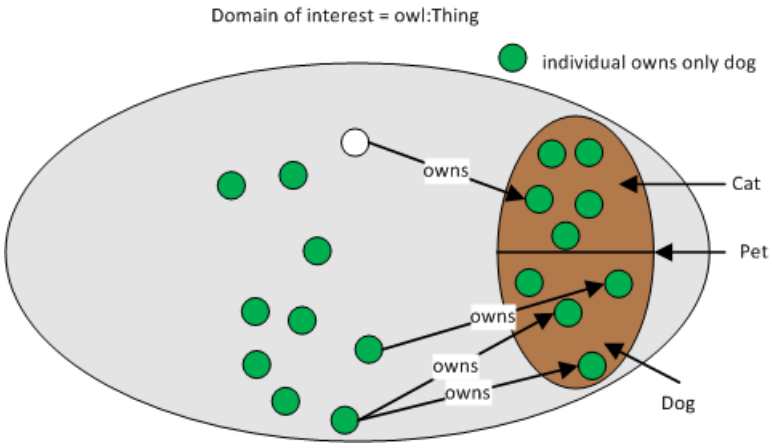

FIGURE 4. The meaning of `owns only Dog`

3. Conclusion

So how do we fix our ontology? Simple, we enforce that for someone to be a `DogLover`, they must own at least 1 dog and nothing else but dogs.

```
Class: DogLover
  EquivalentTo: owns some Dog and owns only Dog
```

The example ontologies of this post can be found in github.

References

1. M. Horridge, *Justification Based Explanation in Ontologies*, Ph.D. thesis, University of Manchester, 2011.
2. Matthew Horridge, Samantha Bail, Bijan Parsia, and Uli Sattler, *Toward cognitive support for OWL justifications.*, Knowl.-Based Syst. **53** (2013), 66–79.