A COMMON MISCONCEPTION REGARDING OWL PROPERTIES

HENRIETTE HARMSE

A misconception w.r.t. OWL properties that I come across from time-to-time is that people mistakenly think that properties express relations between classes rather than relations between individuals. In this post

1. I explain how this misconception fails,
2. I explain what is the meaning of OWL properties, and
3. I explain how you can model relations between classes if that is really what you want to do.

Throughout this post I will refer to object properties only, even though what I say applies to data properties as well, except of course that object properties relate objects to objects whereas data properties relate objects to data type values.

As an example ontology I will use the following:

```
Class: Employer
Class: Employee

ObjectProperty: employs
   Domain: Employer
   Range: Employee
```

1. How thinking that Properties express Relations between Classes fails

Thinking that properties express relations between classes fails in two different ways:

1. The first way in which the expectations of users fail under this misconception is that their assumption is that domain and range axioms behave as constraints. That is if we extend our `Employer` ontology with something nonsensical like

   ```
   Class: Cheese
   Class: Fridge

   Individual: companyFridge
       Types: Fridge

   Individual: blueCheese
      Types: Cheese
      Facts: employs companyFridge
   ```

   the reasoner will indicate that our ontology is inconsistent because the individual `blueCheese` is **not** of type `Employer` and neither is the individual `companyFridge` an instance of `Employee`. However, this is **not** the case.

Date: 23rd June 2018.

This ontology **is** in fact consistent because domain and range axioms do **not** behave as constraints.

2. The other way in which the assumption of users is shattered is that they tend to think that domain and range axioms mean that in our `Employer` ontology it means that instances of the `Employer` class must necessarily be linked via the `employs` property to an instance (or instances?) of `Employee`. Thus, the expectation is that if we have an instance of `Employer` that is not linked to an instance of `Employee` via the `employs` property, the reasoner should give an inconsistency. Again, this is **wrong**. This ontology will still be consistent.

2. The Real Meaning of OWL Object Properties

The OWL specification is very explicit about the meaning of object properties. Is states:

Object properties connect pairs of individuals.

So what is the meaning of domain and range axioms then? Domain and range axioms are **not** constraints to be checked, but rather they are axioms from which the reasoner can make inferences. What domain and range axioms state is that whenever two instances are linked via the `employs` property, for example, it means that the first instance is of type `Employer` and the second instance is of type `Employee`. Thus, in our cheese and fridge example, no matter how silly it is, the reasoner will infer that `blueCheese` is an instance of `Employer` and `companyFridge` is an instance of `Employee`.

Finally, the underlying mathematical formalization of object properties themselves does not enable linking of pairs of classes. An OWL object property `r` is represented as a role r in description logics. Stating that the role r has the domain C and the range D, where C and D are concepts, is achieved through the following axioms:

$$\{\exists a.\top \sqsubseteq C, \top \sqsubseteq \forall a.D\}$$

where $\top$ is the set representing the application domain.

An object property `r` between instances `a` and `b` is expressed as a role assertion

$$r(a,b)$$

in description logics. Note that this assertion contains no information regarding description logic concepts (i.e. classes in OWL).

3. If you really, really want to have a Link between Classes

So if you really, really want to express a relation between classes, how can this be done? For our `Employer` ontology we can state the following:

```
Class: Employer
   SubClassOf: employs some Employee
```

This states that the `Employer` class is a subclass of the class consisting of the instances that are linked to at least 1 instance of the `Employee` class. If we now have an instance of `Employer` that is not linked via `employs` to an instance of `Employee`, the reasoner will find that our ontology is inconsistent. To state that `acme` is an employer without employees, we state it as follows:

```
Individual: acme
   Types: Employer
   Facts: employs max 0 Employee
```

4. Conclusion

In this post I explained that object properties link individuals (**not** classes!) and that thinking otherwise can lead to various errors when designing an ontology.