UNDERSTANDING OWL `MIN` VS MAX VS EXACTLY PROPERTY RESTRICTIONS

HENRIETTE HARMSE

The **open world assumption** trips people up in many ways. In this post I will be looking at how the open world assumption affects the semantics of the property restrictions `min`, `max` and `exactly`.

The example I will use throughout this post is that of a product that may have no price, 1 price, exactly 1 price, or many prices. We will firstly assume a product must have at least 1 price, then that it can have a maximum of 1 price, and finally we will assume that it must have exactly 1 price. We therefore assume that we have a `hasPrice` data property that is defined as follows:

```
DataProperty: hasPrice
  Range:
    xsd:decimal
```

The example ontology can be found on GitHub. To be able to distinguish the different product types having different rules regarding how many prices they have, I created 3 different classes called `ProductWith_Min_1_Price`, `ProductWith_Max_1_Price` and `ProductWith_Exactly_1_Price` respectively.

Before we look at examples and the semantics of the `min`, `max`, `exactly` property restrictions, let us briefly recall what is meant by the open world assumption.

1. Open World Assumption versus Closed World Assumption

OWL has been designed with the explicit intention to be able to deal with incomplete information. Consequently, OWL intentionally does not make any assumptions with regards to information that is not known. In particular, no assumption is made about the truth or falsehood of facts that cannot be deduced from the ontology. This approach is known as the **open world assumption**. This approach is in contrast with the **closed world assumption** typically used in information systems. With the closed world assumption facts that cannot be deduced from a knowledge base (i.e. database) are implicitly understood as being false [1, 2, 3].

As an example, in a database when a product does not have a price, the general assumption is that the product does not have price. Moreover, in a database if a product has 1 price, the assumption is that the product has only 1 price and no other prices. This in stark contrast to OWL. If an OWL ontology defines a product for which no explicit price is given, the assumption is not that the product has no price. Rather, no assumption is made as to whether the product has a price, has many prices or whether it has no price. Futhermore, if a product has a price, the assumption is not that this is necessarily the only price for that product. Rather, it allows for the possibility that no other price may exist, or that many other prices may exist, which is merely not known. The only information that holds in an

Date: 23rd February 2019.

ontology is information that is either explicitly stated, or that can be derived form explicit information.

2. The min Property Restriction

To define a product that must have at least 1 price we define it as follows:

```
Class: ProductWith_Min_1_Price
  SubClassOf:
    hasPrice min 1 xsd:decimal

Individual: productWithoutPrice
  Types:
    ProductWith_Min_1_Price
```

If we now create an individual of type `ProductWith_Min_1_Price`, say `productWithoutPrice`, which has no price information, we will find that the reasoner will not give an inconsistency. The reason for this is that the reasoner has no information with regards to whether `productWithoutPrice` has any price information. Hence, it is possible that `productWithoutPrice` has a price that is merely unknown. To make explicit that `productWithoutPrice` has no price information we can define it as follows:

```
Individual: productWithoutPrice
  Types:
    ProductWith_Min_1_Price,
    hasPrice max 0 xsd:decimal
```

This revised definition of `productWithoutPrice` will now result in the reasoner detecting an inconsistency. Note that `ProductWith_Min_1_Price` allows for products that have more than 1 price. Hence, the following will not result in an inconsistency.

```
Individual: productWithManyPrices
  Types:
    ProductWith_Min_1_Price
  Facts:
    hasPrice 2.5,
    hasPrice 3.25
```

3. The max Property Restriction

To define a product that cannot have more than 1 price, we can define it as follows:

```
Class: ProductWith_Max_1_Price
  SubClassOf:
    hasPrice max 1 xsd:decimal
```

If we now define an individual `productWithMoreThan1Price` with more than 1 price (as shown in the example below), the reasoner will give an inconsistency.

```
Individual: productWithMoreThan1Price
  Types:
    ProductWith_Max_1_Price
  Facts:
    hasPrice 2.5,
    hasPrice 3.25
```

Note that individuals of type `ProductWith_Max_1_Price` can also have no price information without resulting in the reasoner giving an inconsistency. I.e., if we define the individual `productWithoutPrice` as

```
Individual: productWithoutPrice
  Types:
    ProductWith_Max_1_Price,
    hasPrice max 0 xsd:decimal
```

it will not give an inconsistency.

4. The exactly Property Restriction

Let us now define `ProductWith_Exactly_1_Price` with the individual `productWithExactly1Price` as follows:

```
Class: ProductWith_Exactly_1_Price
  SubClassOf:
    hasPrice exactly 1 xsd:decimal

Individual: productWithExactly1Price
  Types:
    ProductWith_Exactly_1_Price
  Facts:
    hasPrice 7.1
```

The `exactly` property is essentially syntactical shorthand for specifying both the `min` and `max` restrictions using the same cardinality. Thus, we could just as well have defined `ProductWith_Exactly_1_Price` as:

```
Class: ProductWith_Exactly_1_Price
  SubClassOf:
     hasPrice min 1 xsd:decimal,
     hasPrice max 1 xsd:decimal
```

or, given the classes we have already defined in the ontology, we can define it as:

```
Class: ProductWith_Exactly_1_Price
  SubClassOf:
    ProductWith_Max_1_Price,
    ProductWith_Min_1_Price
```

5. Prefer exactly

Given that the `exactly` property restriction is syntactical sugar, should we prefer using the combination of `min` and `max` directly as shown above? My answer to this is **no**. My motivation for this is that the semantics of `exactly` is only equivalent to the intersection of `min` and `max` if the cardinalities are the same and the data/object types are the same. As such specifying

```
Class: ProductWith_Exactly_1_Price
  SubClassOf:
    hasPrice exactly 1 xsd:decimal
```

has less opportunities for mistakes than specifying

```
Class: ProductWith_Exactly_1_Price
  SubClassOf:
    hasPrice min 1 xsd:decimal,
    hasPrice max 1 xsd:decimal
```

6. Conclusion

In this post I explained some of the ways in which the `min`, `max` and `exactly` property restrictions can trip people up due to the open world assumption. Please feel free to leave a comment if you have questions or suggestions about this post.

References

1. F. Baader and W. Nutt, *Basic Description Logics*, The Description Logic Handbook: Theory, Implementation and Applications (F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. Patel-Schneider, eds.), Cambridge University Press, New York, USA, 2007, pp. 45–104.
2. M. Krötzsch, F. Simančík, and I. Horrocks, *A Description Logic Primer*, Computing Research Repository (CoRR) **abs/1201.4089** (2012).
3. S. Rudolph, *Foundations of Description Logics*, Proceedings of the 7th International Conference on Reasoning Web: Semantic Technologies for the Web of Data (A. Polleres, C. d'Amato, M. Arenas, S. Handschuh, P. Kroner, S. Ossowski, and P. F. Patel-Schneider, eds.), Lecture Notes in Computer Science, vol. 6848, Springer, 2011, pp. 76–136.