

# WHY DOES THE REASONER IGNORE MY CONSTRAINT?

HENRIETTE HARMSE

A most frustrating problem often encountered by people, with experience in relational databases when they are introduced to ontologies, is that ontology reasoners seem to ignore constraints. In this post I give examples of this problem, explain why they happen and I provide ways to deal with each example.

## 1. AN EXAMPLE

A typical example encountered in relational databases is that of modeling orders with orderlines, which can be modeled via `Orders` and `Orderlines` tables where the `Orderlines` table has a foreign key constraint to the `Orders` table. A related ontology is given in Figure 1. It creates as expected `Order` and `Orderline` classes with a `hasOrder` object property. That individuals of `Orderline` are necessarily associated with **one** order is enforced by `Orderline` being a subclass of `hasOrder` exactly 1 `owl:Thing`.

## 2. TWO PROBLEMS

Two frustrating and most surprising errors given the Order ontology are: (1) if an `Orderline` individual is created for which no associated `Order` individual exists, the reasoner will not give an inconsistency, and (2) if an `Orderline` individual is created for which two or more `Order` individuals exist, the reasoner will also not give an inconsistency.

**2.1. Missing Association Problem.** Say we create an individual `orderline123` of type `Orderline`, which is not associated with an individual of type `Order`, in this case the reasoner will not give an inconsistency. The reason for this is due to the **open world assumption**. Informally it means that the only inferences that

---

*Date:* 9th March 2018.

```
ObjectProperty: hasOrder
  Domain:
    Orderline
  Range:
    Order

Class: Order

Class: Orderline
  SubClassOf:
    hasOrder exactly 1 owl:Thing

Class: owl:Thing
```

FIGURE 1. An ontology modeling orders and orderlines

```

Individual: orderline123
Types:
  Orderline,
  hasOrder max 0 owl:Thing

```

FIGURE 2. Enforcing that `orderline123` is not associated via `hasOrder`

```

Individual: order1
Types:
  Order

Individual: order2
Types:
  Order

Individual: orderline123
Types:
  Orderline
Facts:
  hasOrder order1,
  hasOrder order2

```

FIGURE 3. `orderline123` associated with two individuals of type `Order`

the reasoner can make from an ontology is based on explicit information stated in the ontology or what can be derived from explicit stated information.

When you state `orderline123` is an `Orderline`, there is **no explicit information** in the ontology that states that `orderline123` is **not** associated with an individual of `Order` via the `hasOrder` property. To make explicit that `orderline123` is not in such a relation, you have to define `orderline123` as in Figure 2. `hasOrder max 0 owl:Thing` states that it is known that `orderline123` is not associated with an individual via the `hasOrder` property.

**2.2. Too Many Associated Individuals Problem.** Assume we now change our definition of our `orderline123` individual to be associated via `hasOrder` to two individuals of `Order` as shown in Figure 3. Again, most frustratingly the reasoner does not find that the ontology is inconsistent. The reason for this is that OWL does not make the **unique name assumption**. This means that individuals with different names can be assumed by the reasoner to represent a single individual. To force the reasoner to see `order1` and `order2` as necessarily different, you can state `order1` different from `order2` by adding `DifferentFrom:order2` to `order1` (or similarly for `order2`).

### 3. CONCLUSION

In this post I explained problems newcomers (particularly with relational database experience) to ontologies may experience and how to deal with them.

### REFERENCES