Family Tree Contradiction Exercise

Contents

Exercise

Use inferencing rules in SPARQL for detecting **contradictions** in the family tree dataset.

Find the Turtle file with contradictions on the Wiki.

Test the rules with GraphBD or Protégé.

Possible Solutions

Scenario 1

Qualify as *:Contradiction* any *instance* having more than one mothers. Two-step approach with INSERT and CONSTRUCT. We first infer the relation "*hasMother*" and then infer the contradiction.

Step 1: Every woman that has a child, is mother of that child.

Insert the relation *"hasMother"* from *?y* to *?x*, for every *?y* that is child of *?x*, where *?x* is also a Woman.

```
PREFIX : <http://laurenzi.ch#>
INSERT { ?y :hasMother ?x.}
where { ?x a :Woman;
        :hasChild ?y.}
```

Step 2: Raise contradiction whenever a child has two different mothers.

Infer the relation *rdf:type* between an instance *?a* and the class *:Contradiction*, where *?a* has mother *?b* AND has mother *?c* AND *?b* and *?c* are different.

```
PREFIX : <http://laurenzi.ch#>
CONSTRUCT
{?a a :Contradiction. }
where
```

```
{          ?a :hasMother ?b;
              :hasMother ?c.
       FILTER (?b != ?c)     }
```

Drawbacks of this solution: A child can have two mothers.

However, one can assume that three mothers for one child is still improbable. Therefore, the following solution as an **alternative to Step 2:**

Step 2: Raise contradiction whenever a child has three different mothers (having 2 mothers would not infer contradiction).

Infer the relation *rdf:type* between an instance *?a* and the class *:Contradiction*, where *?a* has mother *?b* AND has mother *?c* AND has mother *?d* AND *?b* and *?c* are different AND *?c* and *?d* are different AND *?d* and *?a* are different.

```
CONSTRUCT {?a a :Contradiction. }
where
{          ?a :hasMother ?b;
              :hasMother ?c;
             :hasMother ?d.
       FILTER (?b != ?c && ?c != ?d && ?b != ?d)   }
```

What if we want to raise contradiction without first inferring the “hasMother” relation?

Infer the relation *rdf:type* between an instance *?a* and the class *:Contradiction*, where *?a* is child of *?b* AND is child of *?c* AND is child of *?d* AND *?b* and *?c* are different AND *?c* and *?d* are different AND *?d* and *?a* are different.

```
CONSTRUCT {?a a :Contradiction. }
where
{          ?b :hasChild ?a.
           ?c :hasChild ?a.
          ?d :hasChild ?a.
          FILTER (?b != ?c && ?c != ?d && ?b != ?d)   }
```

Scenario 2:

Raise contradiction whenever a child has more than one incoming “hasChild” relation (without inferring the *hasMother* relationship beforehand).

Infer the relation *rdf:type* between an instance *?a* and the class *:Contradiction*, where *?a* is child of *?b* and *?b* is more than 1.

```
CONSTRUCT { ?a a :Contradiction }
WHERE
{
  {
    SELECT ?a
    WHERE { ?b :hasChild ?a. }
    GROUP BY ?a
    HAVING(COUNT(?b) > 1)
  }
}
```

Potential drawback: if we add the relation “hasChild” also for fathers, each targeting node of the relation “hasChild” will have 2 incoming “hasChild” relations. To overcome this issue, the alternative is the following:

Raise contradiction whenever a child has more than 2 parents (without infer the *hasMother* relationship beforehand).

Infer the relation *rdf:type* between an instance *?a* and the class *:Contradiction*, where *?a* is child of *?b* and *?b* is more than 2.

```
CONSTRUCT { ?a a :Contradiction }
WHERE
{
  {
    SELECT ?a
    WHERE { ?b :hasChild ?a. }
    GROUP BY ?a
    HAVING(COUNT(?b) > 2)
  }
}
```

Scenario 3:

Raise contradiction whenever a child has more than one mother **OR** more than one father.

Infer the relation *rdf:type* between an instance *?a* and the class *:Contradiction*, where

- *?a* is child of ?b AND ?b is a woman, AND ?b is more than 1,
- **OR**
- *?a* is child of ?b AND ?b is a man, AND ?b is more than 1.

```
CONSTRUCT { ?a a :Contradiction }
WHERE
{
  {
    SELECT ?a
    WHERE { ?b a :Woman; :hasChild ?a. }
    GROUP BY ?a
    HAVING(COUNT(?b) > 1)
  }
  UNION
  {
    SELECT ?a
    WHERE { ?b a :Man; :hasChild ?a. }
    GROUP BY ?a
    HAVING(COUNT(?b) > 1)
  }
}
```

More information about SPARQL and its language constructs

Visit: https://euclid-project.eu/modules/chapter2.html