Cadinality test – example

Ontology

Let the ontology will be defined as below:

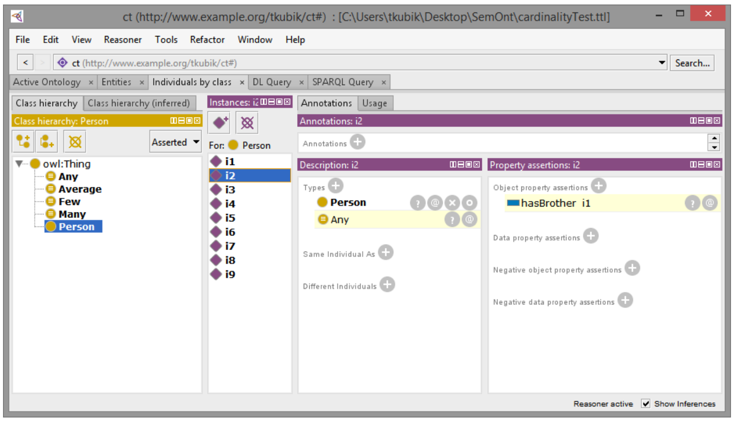
```
@prefix : <http://www.example.org/tkubik/ct#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://www.example.org/tkubik/ct#> a owl:Ontology .

:hasBrother a owl:SymmetricProperty .

:Any a owl:Class ;
        owl:equivalentClass [
        a owl:Restriction ;
        owl:onProperty :hasBrother ;
        owl:someValuesFrom owl:Thing .
        ] .

:Average a owl:Class ;
        owl:equivalentClass [
       a owl:Class ;
       owl:intersectionOf (
        [ a owl:Restriction ;
          owl:onProperty :hasBrother ;
          owl:minCardinality "3"^^xsd:nonNegativeInteger . ]
        [ a owl:Restriction ;
          owl:onProperty :hasBrother ;
          owl:maxCardinality "4"^^xsd:nonNegativeInteger . ]
       ) ;
    ] .

:Few a owl:Class ;
        owl:equivalentClass [
       a owl:Class ;
       owl:intersectionOf (
        [ a owl:Restriction ;
          owl:onProperty :hasBrother ;
          owl:minCardinality "1"^^xsd:nonNegativeInteger . ]
        [ a owl:Restriction ;
          owl:onProperty :hasBrother ;
          owl:maxCardinality "2"^^xsd:nonNegativeInteger . ]
       ) ;
    ] .

:Many a owl:Class ;
        owl:equivalentClass  [
      a owl:Class ;
      owl:intersectionOf (
        [ a owl:Restriction ;
          owl:onProperty :hasBrother ;
          owl:minCardinality "5"^^xsd:nonNegativeInteger . ]
       ) ;
    ] .

:Person a owl:Class .

:i1 a owl:NamedIndividual , :Person ;
        :hasBrother :i2 , :i3 .
:i2 a owl:NamedIndividual , :Person .
:i3 a owl:NamedIndividual , :Person .
:i4 a owl:NamedIndividual , :Person ;
        :hasBrother :i5 , :i6 , :i7, :i8, :i9 .
:i5 a owl:NamedIndividual , :Person .
:i6 a owl:NamedIndividual , :Person .
:i7 a owl:NamedIndividual , :Person .
:i8 a owl:NamedIndividual , :Person ;
        :hasBrother :i5 , :i6 , :i7 .
:i9 a owl:NamedIndividual , :Person .
```

Consider definitions of `Average`, `Few` and `Many` classes. They are defined with the use of `owl:equivalentClass` that allows put some restrictions on the class membership.

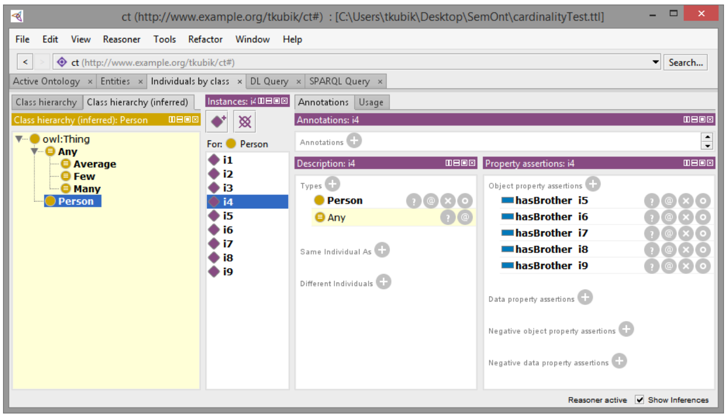
Such ontology, when processed with semantical reasoning turned, will be extended by inferred facts as follows.

1. `i1` is an instance of `Any` and has two brothers declared explicitly (these are `i2` and `i3`).
2. `i2` is an instance of `Any` class and `hasBrother i1` , but not `i3` . It comes from the declaration of `hasBrother` as a *Symmetric* property but not *Transitive*. In fact this property cannot be declared as *Transitive* due to restrictions declared on classes `Average`, `Few` and `Many`. This is covered by the OWL 2 Web Ontology Language, Structural Specification and Functional-Style Syntax (Second Edition) (https://www.w3.org/TR/owl2-syntax/#Global_Restrictions%20_on_Axioms_in_OWL_2_DL) which requires that only simple object properties may be asserted to be reflexive or symmetric. Asserting that a property is transitive automatically makes it composite. Then the reasoner would rise an exception:

```
Non-simple property '<http://www.example.org/tkubik/ct#hasBrother>' or its inverse appears in the cardinality
restriction 'ObjectMaxCardinality(4 <http://www.example.org/tkubik/ct#hasBrother> owl:Thing)'.
```



3. `i4` is an instance of `Any` class and has five brothers declared explicitly.



4. Because `i1` has two brothers and `i4` has five brothers one may think that both should belong to, respectively, `Few` and `Many` classes. But this is not the case. With the use of `owl:minCardinality` restriction one can set **sufficient** conditions for an instance to be a member of particular class (e.g. if the knowledge base includes information that someone has 2 brothers, this person can be assigned to `Few` class). But `owl:maxCardinality` restrictions are used to declare **necessary** conditions. Because of open world assumption it is difficult to say that someone has no more brothers that already recorded in the knowledge base (e.g. if the knowledge base includes information that someone has 3 brothers there is no guarantee that this person has no more brothers, but for sure this person cannot be assign to `Few` class because the max cardinality restriction would be broken). Combining necessary and sufficient conditions (by intersection operator) makes the resulting formula a **necessary** condition.
Fortunatelly there is a way of closing the world. The instances can be assign to the proper class by running SPARQL querry as in the example below (which result with a list list entities bolonging to `Few` class).

```
PREFIX cnt: <http://www.example.org/tkubik/ct#>
CONSTRUCT {?s a cnt:Few}
WHERE
{SELECT ?s (count(?p) AS ?numBrothers)
      WHERE
      {  ?s cnt:hasBrother ?p
      } GROUP BY ?s
     HAVING (?numBrothers>0 && ?numBrothers<=2)
}
```