9.10 Open World and Closed World: OWL/DL/Tableaux/Logic and SPARQL

- OWL/DL reasoning: OWA.
  Everything that can neither be proven nor disproven is unknown
- SPARQL queries/algebraic evaluation: CWA
  BGPs that do not match (not proven to be true, i.e. false or unknown) are considered as "no answer"

SPARQL CWA AND OWL OWA: POSSIBLE – IF NOT IMPOSSIBLE

⇒ Use SPARQL to check what cannot be *proven* by using FILTER NOT EXISTS { *query* }:
If the negation of some formula φ cannot be proven – then φ is at least possible, i.e, there exists a model that makes φ true.

- Limited expressiveness $\neg\varphi$ must be OWL-DL-expressible.
  (means: wrt. stable models [Deductive Databases Lecture], where any possible solution can be described)
- Consider again Slide 398 for an earlier example.

SPARQL NOT EXISTS { $\neg\varphi$ } for checking possibility of φ

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#>.
:Childless owl:intersectionOf (:Person
  [ a owl:Restriction; owl:onProperty :hasChild; owl:maxCardinality 0]).
:Parent owl:intersectionOf (:Person
  [ a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 1]).
:john a :Person; :hasChild :alice, :bob.
:alice a :Person.  :bob a :Person.
```

[Filename: RDF/childless-small.n3]

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix : <foo://bla#>
select ?X ?C
from <file:childless-small.n3>
where { ?X a :Person . ?C a owl:Class; rdfs:subClassOf :Person
        FILTER NOT EXISTS {?X a ?C}}
```

[Filename: RDF/childless-small.sparql]

- John: only possible that he is a parent;
  for alice and bob, it is possible to be a parent or to be childless.

SPARQL CWA AND OWL OWA: POSSIBLE – IF NOT IMPOSSIBLE: A SCENARIO

- three rooms: bedroom, livingroom, guestroom
- some furniture: beds, a wardrobe, tables, chairs
- specification how many of these furniture can be placed in the rooms
- task: find out what can be placed where

Scenario

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://rooms#>.
:in a owl:ObjectProperty, owl:FunctionalProperty; owl:inverseOf :has;
  rdfs:domain :Furniture; rdfs:range :Room.
:Room owl:oneOf (:bedroom :livingroom :guestroom).
[] a owl:AllDifferent; owl:members (:bedroom :livingroom :guestroom).
:bedroom a :Room,
  [a owl:Restriction; owl:onProperty :has; owl:onClass :Bed; owl:qualifiedCardinality 1],
  [a owl:Restriction; owl:onProperty :has; owl:onClass :Wardrobe; owl:qualifiedCardinality 1],
  [a owl:Restriction; owl:onProperty :has; owl:onClass :Chair; owl:qualifiedCardinality 1],
  [a owl:Restriction; owl:onProperty :has; owl:onClass :Table; owl:maxQualifiedCardinality 0].
# :bedroom  :has :bed1 . # comment in or out ...
:guestroom a :Room,
  [a owl:Restriction; owl:onProperty :has; owl:onClass :Table; owl:maxQualifiedCardinality 0].
:livingroom a :Room,
  [a owl:Restriction; owl:onProperty :has; owl:onClass :Bed; owl:maxQualifiedCardinality 0],
  [a owl:Restriction; owl:onProperty :has; owl:onClass :Chair; owl:qualifiedCardinality 4].

:Furniture a owl:Class;
  owl:disjointUnionOf (:Bed :Wardrobe :Table :Chair);
  owl:equivalentClass [a owl:Restriction; owl:onProperty :in; owl:cardinality 1].
```

```
:Bed owl:oneOf (:bed1 :bed2 :bed3).  ### one in bedroom, none in livingr. -> two in guestroom
[] a owl:AllDifferent; owl:members (:bed1 :bed2 :bed3).
:Wardrobe owl:oneOf (:wr1).
:Table owl:oneOf (:t1).   ### only one. must be in livingroom -> no in bedroom.
:Chair owl:oneOf (:c1 :c2 :c3 :c4 :c5).
[] a owl:AllDifferent; owl:members (:c1 :c2 :c3 :c4 :c5).
  ### one must be in bedroom, 4 in livingroom, no one remains for guestroom

:InBedroom a owl:Class; owl:equivalentClass [ a owl:Restriction;
   owl:onProperty :in; owl:hasValue :bedroom ].
:InGuestroom a owl:Class; owl:equivalentClass [ a owl:Restriction;
   owl:onProperty :in; owl:hasValue :guestroom ].
:InLivingroom a owl:Class; owl:equivalentClass [ a owl:Restriction;
   owl:onProperty :in; owl:hasValue :livingroom ].
:NotInBedroom a owl:Class; owl:equivalentClass
   [ owl:intersectionOf (:Furniture [ owl:complementOf :InBedroom])].
:NotInLivingroom a owl:Class; owl:equivalentClass
   [ owl:intersectionOf (:Furniture [ owl:complementOf :InLivingroom])].
:NotInGuestroom a owl:Class; owl:equivalentClass
   [ owl:intersectionOf (:Furniture [ owl:complementOf :InGuestroom])].

## for the queries:
:RoomWithChair owl:equivalentClass
  [a owl:Restriction; owl:onProperty :has; owl:someValuesFrom :Chair].
## :guestroom a :RoomWithChair.   ## makes it inconsistent
:RoomWithoutChair owl:equivalentClass [a owl:Restriction;
  owl:onProperty :has; owl:onClass :Chair; owl:maxQualifiedCardinality 0].
:RoomWithTwoBeds owl:equivalentClass [a owl:Restriction;
  owl:onProperty :has; owl:onClass :Bed; owl:qualifiedCardinality 2].
```

[Filename: RDF/rooms.n3]

Scenario (Cont'd)

```
prefix : <foo://rooms#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix owl: <http://www.w3.org/2002/07/owl#>
select ?X ?Room ?InR ?Y ?InterpretAsMaybeInR
from <file:rooms.n3>
where {{ ?X :in ?Room}  UNION
  { ?X a :Furniture, ?InR . ?InR rdfs:subClassOf :Furniture  .
       FILTER contains(str(?InR),"In")}
    UNION
  { ?Y a :Furniture . ?NotInR rdfs:subClassOf :Furniture .
       FILTER contains(str(?NotInR),"NotIn") .
    FILTER NOT EXISTS { ?Y a ?NotInR .}
    bind (?NotInR as ?InterpretAsMaybeInR)
  }}
order by ?R ?InR ?NotInR
```

[Filename: RDF/rooms.sparql]

- The table must be in the livingroom,
- among bed1, bed2, bed3, one is in the bedroom and two are in the guestroom.

Scenario (Cont'd)

- are there two beds in the guestroom? (yes)
- is it possible that there is some chair in the guestroom?
  (no - it can be derived that the guestroom is a room without chair)
- is it possible that chair1 is in the bedroom? (yes)

```
prefix : <foo://rooms#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix owl: <http://www.w3.org/2002/07/owl#>
select ?A1
from <file:rooms.n3>
where { { bind('2BedsInGuestroom' AS ?A1) . :guestroom a :RoomWithTwoBeds }
  UNION { bind('NoChairInGuestroom' AS ?A1) . :guestroom a :RoomWithoutChair }
  UNION { bind('c1InBedroom' AS ?A1) . { :c1 :in :bedroom }}
  UNION { bind('maybeC1InBedroom' AS ?A1) .
          FILTER NOT EXISTS { :c1 :in :bedroom }}}
```

[Filename: RDF/rooms2.sparql]

9.11 Rules in DL: Hybrid Reasoning

- Early Approaches: Donini, Lenzerini et al 1991; Levy, Rousset 1996 (CARIN):
  rather disappointing safety and decidability results:
  roughly, due to objects implicitly (existentially) assured by DL specifications.
- Newer investigations in Semantic Web context:
  DLV (Eiter et al 2004), DL+log (Rosati 2006); Motik, Sattler, Studer 2005; Lukasiewicz 2007: more detailed syntactical and structural constraints.
- SWRL (Semantic Web Rule Language; 2004):
  - Full Power of OWL-DL, allows for specifying undecidable settings, high computational complexity,
  - building upon the basic RULE-ML ontology for describing rules (rule; head, body; different kinds of atoms),
  - DL-safe rules (decidable) supported by Pellet: restriction in syntax and in semantics variables only applied to named resources (prunes the tableau; roughly ignoring all only existentially known objects).
- recall that SPARQL also returns only answers bound to explicitly known nodes (cf. Slide 417).

SIMPLE RULE EXAMPLE: UNCLE

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix swrl: <http://www.w3.org/2003/11/swrl#>.
@prefix : <foo://bla#>.
:sue :hasChild :barbara; :hasSibling :john.
:john :name "John"; :hasChild :alice, :bob;  :hasSibling :sue.
:x a swrl:Variable.
:y a swrl:Variable.
:z a swrl:Variable.
 :uncleAuntRule a swrl:Imp;
   swrl:head ([ a swrl:IndividualPropertyAtom;
                swrl:propertyPredicate :hasUncleAunt;
                swrl:argument1 :x ;   swrl:argument2 :z ]);
   swrl:body ([ a swrl:IndividualPropertyAtom;
                swrl:propertyPredicate :hasChild;
                swrl:argument1 :y ;   swrl:argument2 :x ]
              [ a swrl:IndividualPropertyAtom;
                swrl:propertyPredicate :hasSibling;
                swrl:argument1 :y ;   swrl:argument2 :z ]).
```

[Filename: RDF/uncle-rule-swrl.n3]

```
prefix : <foo://bla#>
select ?X ?U
from <file:uncle-rule-swrl.n3>
where {?X :hasUncleAunt ?U}
```

[Filename: RDF/uncle-rule-swrl.sparql]

DL-SAFE RULES CONSIDER ONLY NAMED RESOURCES

- analogous to SPARQL queries and owl:hasKey (cf. Slide 404)
- only positive atoms in the body (then OWA vs. CWA does not play any role)

⇒ work only on a finite instantiated subgraph of the whole DL model

⇒ does not interfere with the blocking, and

⇒ does not break decidability.

Comparison: SWRL Rule, Property Chain, SPARQL, DL

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix swrl: <http://www.w3.org/2003/11/swrl#>.
@prefix : <foo://bla#>.
:john :hasChild :bob; :hasSibling :paul, [].
:hasSibling a owl:SymmetricProperty.
:paul a [a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 1].

:uncleRule a swrl:Imp;
   swrl:head ([ a swrl:IndividualPropertyAtom; swrl:propertyPredicate :uncle1;
                swrl:argument1 :y ; swrl:argument2 :z ]);
   swrl:body ([ a swrl:IndividualPropertyAtom; swrl:propertyPredicate :hasChild;
                swrl:argument1 :x ; swrl:argument2 :y ]
              [ a swrl:IndividualPropertyAtom; swrl:propertyPredicate :hasSibling;
                swrl:argument1 :x ; swrl:argument2 :z ]).
   :x a swrl:Variable.    :y a swrl:Variable.   :z a swrl:Variable.
[ owl:propertyChain ([owl:inverseOf :hasChild] :hasSibling) ] rdfs:subPropertyOf :uncle2.
:Uncle owl:equivalentClass [a owl:Restriction; owl:onProperty :hasSibling;
  owl:someValuesFrom [a owl:Restriction; owl:onProperty :hasChild;
                      owl:minCardinality 1]].
```

[Filename: RDF/uncle-comparison.n3]

DL-Safe Rules consider only named Resources (cont'd)

```
prefix : <foo://bla#>
select ?N ?U1 ?U2 ?isU
from <file:uncle-comparison.n3>
where {{?N :uncle1 ?U1} union {?N :uncle2 ?U2}
       union {?isU a :Uncle}}
```

[Filename: RDF/uncle-comparison.sparql]

- blank nodes are considered. Paul and a bnode (John's other brother) are Bob's uncles.
- implicitly known nodes are not considered: John's brother Paul has a child (so John is also an uncle) which is only implicitly known.

BUILT-IN SWRL ATOMS

SWRL provides some built-in atoms for owl:sameAs, owl:differentFrom etc.

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix swrl: <http://www.w3.org/2003/11/swrl#>.
@prefix : <foo://bla#> .

:name a owl:FunctionalProperty; a owl:DatatypeProperty.
:john a :Person; :name "John"; :hasChild :alice, :bob.
:alice a :Person; :name "Alice".
:bob a :Person; :name "Bob".
  :x a swrl:Variable.   :y a swrl:Variable.  :z a swrl:Variable.
:r a swrl:Imp;
   swrl:head ([ a swrl:IndividualPropertyAtom; swrl:propertyPredicate :hasSibling;
                swrl:argument1 :y ; swrl:argument2 :z ]);
   swrl:body ([ a swrl:IndividualPropertyAtom; swrl:propertyPredicate :hasChild;
                swrl:argument1 :x ; swrl:argument2 :y ]
              [ a swrl:IndividualPropertyAtom; swrl:propertyPredicate :hasChild;
                swrl:argument1 :x ; swrl:argument2 :z ]
              [ a swrl:DifferentIndividualsAtom;
                swrl:argument1 :y ; swrl:argument2 :z ]).
```

[Filename: RDF/sibling-rule.n3]

Built-In SWRL atoms (Cont'd)

```
prefix : <foo://bla#>
select ?X ?CH ?SIB
from <file:sibling-rule.n3>
where {{?X :hasChild ?CH} union {?X :hasSibling ?SIB}}
```

[Filename: RDF/sibling.sparql]

EVALUATION OF DL REASONING VS SWRL RULES

- DL Reasoning considers implicitly known resources (= graph nodes) and handles them in the tableau via blocking:
  - Structurally identical graph fragments are not further explored. Due to DL's locality principle and tree structure of the model, the model can be kept finite.
- Rules are also incorporated into the tableau, but since they do not have the tree property, blocking would not be sufficient for keeping the model finite when implicitly known resources are considered.

$\Rightarrow$ if something "important" about an implicitly known node can only be derived by a rule, it is not discovered (cf. next example).

DL-SAFETY: SWRL RULES DO NOT CONSIDER IMPLICIT RESOURCES

(see Turtle fragment next slide)

- Rule: all persons believe in God,
- jack has a blank node child _:b0 who is a parent,
- _:b0 is a believer (by the rule),
- as the grandchild is a person, application of the rule would result in the fact that it believes in God, i.e. it is a believer, which should make _:b0 a ParentOfBeliever.
- how to show that the grandchild is not considered by the rule: add a statement that _:b0 is not parent of any believer.
- run "classify" for the n3:
  - the ontology is consistent,
  - _:b0 is accepted to be a :NotParentOfBeliever.

SWRL Rules and DL-Safety: Example

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix swrl: <http://www.w3.org/2003/11/swrl#>.
@prefix : <foo://bla#> .
:jack a :Person; :hasChild [a :Person; a :Parent; a :NotParentOfBeliever].
:Parent owl:equivalentClass
  [a owl:Restriction; owl:onProperty :hasChild; owl:someValuesFrom :Person].
:Believer owl:equivalentClass
  [a owl:Restriction; owl:onProperty :believes; owl:minCardinality 1].
:ParentOfBeliever owl:equivalentClass
  [a owl:Restriction; owl:onProperty :hasChild; owl:someValuesFrom :Believer].
:NotParentOfBeliever owl:equivalentClass [a owl:Restriction;
     owl:onProperty :hasChild; owl:onClass :Believer; owl:cardinality 0].

:x a swrl:Variable.
:r a swrl:Imp;
   swrl:head ([ a swrl:IndividualPropertyAtom;
                swrl:propertyPredicate :believes;
                swrl:argument1 :x ; swrl:argument2 :god ]);
   swrl:body ([ a swrl:ClassAtom; swrl:classPredicate :Person;
                swrl:argument1 :x ]).
```

[Filename: RDF/hidden-prop.n3]

SWRL Rules and DL-Safety: Example (Cont'd)

```
prefix : <foo://bla#>
select ?B ?PB ?NPB
from <file:hidden-prop.n3>
where {{?B a :Believer} union {?PB a :ParentOfBeliever}
                          union {?NPB a :NotParentOfBeliever}}
```

[Filename: RDF/hidden-prop.sparql]

RULE EXAMPLE: BIG CITIES

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix mon: <http://www.semwebtech.org/mondial/10/meta#>.
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.
@prefix swrl: <http://www.w3.org/2003/11/swrl#>.
@prefix : <foo://bla/>.

mon:population  rdfs:range xsd:int; a owl:FunctionalProperty. ## all cities are different.
_:Million a rdfs:Datatype; owl:onDatatype xsd:int; owl:withRestrictions (_:m1).
_:m1 xsd:minInclusive 1000000 .

:ProvinceWithBigCity a owl:Class.   # otherwise sparql answer empty.
:ProvinceWithTwoBigCities a owl:Class.   # otherwise sparql answer empty.
:CountryWithTwoBigCities a owl:Class.   # otherwise sparql answer empty.

:HasBigPopulation owl:equivalentClass [a owl:Restriction;
  owl:onProperty mon:population; owl:someValuesFrom _:Million].
:BigCity owl:intersectionOf (mon:City :HasBigPopulation).
```

[Filename: RDF/bigcities-base.n3]

First: the simplest rule: a country where no provinces are contained in the database:

$\mathsf{Cw2BCs}(X) :- \mathsf{Country}(X) \wedge \mathsf{BigCity}(Y) \wedge \mathsf{BigCity}(Z) \wedge \mathsf{hasCity}(X, Y) \wedge \mathsf{hasCity}(X, Z) \wedge Y \neq Z.$

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix mon: <http://www.semwebtech.org/mondial/10/meta#>.
@prefix swrl: <http://www.w3.org/2003/11/swrl#>.
@prefix : <foo://bla/>.

:CountryNoProvsRule a swrl:Imp;
 swrl:head ([ a swrl:ClassAtom; swrl:classPredicate :CountryWithTwoBigCities;
              swrl:argument1 :x]);
 swrl:body ([ a swrl:ClassAtom; swrl:classPredicate mon:Country; swrl:argument1 :x ]
            [ a swrl:ClassAtom; swrl:classPredicate :BigCity; swrl:argument1 :y ]
            [ a swrl:ClassAtom; swrl:classPredicate :BigCity; swrl:argument1 :z ]
            [ a swrl:IndividualPropertyAtom; swrl:propertyPredicate mon:hasCity;
              swrl:argument1 :x ; swrl:argument2 :y ]
            [ a swrl:IndividualPropertyAtom; swrl:propertyPredicate mon:hasCity;
              swrl:argument1 :x ; swrl:argument2 :z ]
            [ a swrl:DifferentIndividualsAtom; swrl:argument1 :y ; swrl:argument2 :z ]).
```

[Filename: RDF/bigcities-country-noprovs-rule.n3]

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix mon: <http://www.semwebtech.org/mondial/10/meta#>.
@prefix swrl: <http://www.w3.org/2003/11/swrl#>.
@prefix : <foo://bla/>.

:x a swrl:Variable.   :y a swrl:Variable.     :z a swrl:Variable.
:ProvBigCityRule a swrl:Imp;
 swrl:head ([ a swrl:ClassAtom; swrl:classPredicate :ProvinceWithBigCity; swrl:argument1 :x]);
 swrl:body ([ a swrl:ClassAtom; swrl:classPredicate mon:Province; swrl:argument1 :x ]
            [ a swrl:ClassAtom; swrl:classPredicate :BigCity; swrl:argument1 :y ]
            [ a swrl:IndividualPropertyAtom; swrl:propertyPredicate mon:hasCity;
              swrl:argument1 :x ; swrl:argument2 :y ]).

:TwoProvsRule a swrl:Imp;
 swrl:head ([ a swrl:ClassAtom; swrl:classPredicate :CountryWithTwoBigCities; swrl:argument1 :x]);
 swrl:body ([ a swrl:ClassAtom; swrl:classPredicate mon:Country; swrl:argument1 :x ]
            [ a swrl:ClassAtom; swrl:classPredicate :ProvinceWithBigCity; swrl:argument1 :y ]
            [ a swrl:ClassAtom; swrl:classPredicate :ProvinceWithBigCity; swrl:argument1 :z ]
            [ a swrl:IndividualPropertyAtom; swrl:propertyPredicate mon:hasProvince;
              swrl:argument1 :x ; swrl:argument2 :y ]
            [ a swrl:IndividualPropertyAtom; swrl:propertyPredicate mon:hasProvince;
              swrl:argument1 :x ; swrl:argument2 :z ]
            [ a swrl:DifferentIndividualsAtom; swrl:argument1 :y ; swrl:argument2 :z ]).
```

[Filename: RDF/bigcities-2provs-rules.n3]


```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix mon: <http://www.semwebtech.org/mondial/10/meta#>.
@prefix swrl: <http://www.w3.org/2003/11/swrl#>.
@prefix : <foo://bla/>.

:Prov2BigCitiesRule a swrl:Imp;
 swrl:head ([ a swrl:ClassAtom; swrl:classPredicate :ProvinceWithTwoBigCities; swrl:argument1 :x]);
 swrl:body ([ a swrl:ClassAtom; swrl:classPredicate mon:Province; swrl:argument1 :x ]
            [ a swrl:ClassAtom; swrl:classPredicate :BigCity; swrl:argument1 :y ]
            [ a swrl:ClassAtom; swrl:classPredicate :BigCity; swrl:argument1 :z ]
            [ a swrl:IndividualPropertyAtom; swrl:propertyPredicate mon:hasCity;
              swrl:argument1 :x ; swrl:argument2 :y ]
            [ a swrl:IndividualPropertyAtom; swrl:propertyPredicate mon:hasCity;
              swrl:argument1 :x ; swrl:argument2 :z ]
            [ a swrl:DifferentIndividualsAtom; swrl:argument1 :y ; swrl:argument2 :z ]).

:Prov2CRule a swrl:Imp;
 swrl:head ([ a swrl:ClassAtom; swrl:classPredicate :CountryWithTwoBigCities; swrl:argument1 :x]);
   swrl:body ([ a swrl:ClassAtom; swrl:classPredicate mon:Country; swrl:argument1 :x ]
              [ a swrl:ClassAtom; swrl:classPredicate :ProvinceWithTwoBigCities; swrl:argument1 :y ]
              [ a swrl:IndividualPropertyAtom; swrl:propertyPredicate mon:hasProvince;
                swrl:argument1 :x ; swrl:argument2 :y ]).
```

[Filename: RDF/bigcities-prov-2bigcities-rules.n3]

Rule Example: Big Cities (Cont'd)

```
prefix : <foo://bla/>
prefix mon: <http://www.semwebtech.org/mondial/10/meta#>
select ?C ?BC ?P1 ?P2 ?X
from <file:bigcities-base.n3>
from <file:bigcities-2provs-rules.n3>
from <file:bigcities-prov-2bigcities-rules.n3>
from <file:bigcities-country-noprovs-rule.n3>
#from <file:dummy-cities.n3>     ## a small test setting
from <file:mondial-europe.n3>    ## europe is more than sufficient =:(
from <file:mondial-meta.n3>
where {# {?BC a :BigCity} UNION
       {?X a mon:Country; mon:carCode ?C; mon:hasCity ?BC . ?BC a :BigCity} UNION
       {?P1 a :ProvinceWithBigCity} UNION
       {?P2 a :ProvinceWithTwoBigCities} UNION
       {?X a :CountryWithTwoBigCities}}
```

[Filename: RDF/bigcities-by-rules.sparql]