

7.3 OWL 1.1 (Work in Progress)

- OWL 1.1 notions belong to the namespace `http://www.w3.org/2006/12/owl11#` (usually denoted by `owl11`).
- Syntactic Sugar: `owl11:disjointUnionOf` and negative assertions: `ObjectPropertyAssertion` vs. `NegativeObjectPropertyAssertion`
- User-defined datatypes (like XML Schema simple types).
- *SROIQ* Qualified cardinality restrictions (only for non-complex properties), local reflexivity restrictions (individuals that are related to themselves via the given property), reflexive, irreflexive, symmetric, and anti-symmetric properties (only for non-complex properties), disjoint properties (only for non-complex properties), Property chain inclusion axioms (e.g., `SubObjectPropertyOf(SubObjectPropertyChain(owns hasPart) owns)` asserts that if x owns y and y has a part z , then x owns z).
- *SROIQ(D)* is decidable.
The Even More Irresistible SROIQ. Ian Horrocks, Oliver Kutz, and Ulrike Sattler. In Principles of Knowledge Representation and Reasoning (KR 2006). AAAI Press, 2006. Available at www.cs.man.ac.uk/~sattler/publications/sroi-q-tr.pdf.

NEGATIVE ASSERTIONS

- Assert that something is known *not* to hold:
NegativeObjectPropertyAssertion and NegativeDataPropertyAssertion

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@prefix : <foo://bla/names#> .
@prefix person: <foo://bla/persons/> .
person:john a :Person.
[ rdf:type owl11:NegativeObjectPropertyAssertion;
  rdf:subject person:john;
  rdf:predicate :lives;
  rdf:object :germany].
:German owl:equivalentClass [ a owl:Restriction;
  owl:onProperty :lives; owl:hasValue :germany ].
:NonGerman owl:complementOf :German.
```

[Filename: RDF/nongerman.n3]

```
prefix : <foo://bla/names#>
select ?P
from <file:nongerman.n3>
where {?P a :NonGerman}
```

[Filename: RDF/nongerman.sparql]

- John is derived to be a Non-German.

Comment on Negative Assertions

... are just syntactic sugar for a construct using complement classes (and actually implemented in the reasoner by this):

Any owl11:NegativeObjectPropertyAssertion $\neg(x r y)$ is encoded as

- a restriction $R(r, y)$ based on owl:hasValue:
 $R(r, y) = \{x | (x r y)\}$
(above: $R(\text{lives,germany}) = \text{:German}$)
- its complement $CompR(r, y) := \top \setminus R(r, y)$
(above: $CompR(\text{lives,germany}) = \text{:NonGerman}$)
- and the assertion that $x \in CompR(r, y)$.
(above: $\text{assert } (\text{:john a } \text{:NonGerman})$)

OWL: DISJOINT UNION

... syntactic sugar for owl:unionOf and owl:disjointWith:

(only a simple test and syntax example)

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:owl11="http://www.w3.org/2006/12/owl11#"
  xmlns:f="foo://bla/"
  xml:base="foo://bla/">
  <owl:Class rdf:about="Person">
    <owl11:disjointUnionOf rdf:parseType="Collection">
      <owl:Class rdf:about="Male"/>
      <owl:Class rdf:about="Female"/>
    </owl11:disjointUnionOf>
  </owl:Class>
  <f:Male rdf:about="John"/>
  <f:Female rdf:about="John"/>
</rdf:RDF>
```

```
prefix f: <foo://bla/>
select ?X
from <file:disjointunion.xml>
where {?X a f:Person}
```

[Filename:
RDF/disjointunion.sparql]

[Filename: RDF/disjointunion.xml]

EXAMPLE: PARRICIDES IN GREEK MYTHODOLOGY

(from ESWC'07 SPARQL tutorial by Marcelo Arenas et al)

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@prefix : <foo:greek#>.
:Person owl11:disjointUnionOf (:Parricide :Non-parricide).
  :iokaste a :Person; :child :oedipus.
  :oedipus a :Person, :Parricide; :married-to :iokaste; :child :perineikes.
  :perineikes a :Person; :child :thesandros.
  :thesandros a :Person; a :Non-parricide.
:Parent-of-parricide owl:equivalentClass [ a owl:Restriction;
  owl:onProperty :child; owl:someValuesFrom :Parricide ].
:Parent-of-non-parricide owl:equivalentClass [ a owl:Restriction;
  owl:onProperty :child; owl:someValuesFrom :Non-parricide ].
:Parent-of-parricide-Grandparent-of-non-parricide owl:intersectionOf
  ([a owl:Restriction; owl:onProperty :child; owl:someValuesFrom :Parricide]
  [a owl:Restriction;
  owl:onProperty :child; owl:someValuesFrom :Parent-of-non-parricide]).
```

[Filename: RDF/parricide.n3]

EXAMPLE (CONT'D)

- have a short look on the results:

```
prefix : <foo:greek#>
prefix owl: <http://www.w3.org/2002/07/owl#>
select ?P ?NP ?PP ?PNP ?X
from <file:parricide.n3>
where {{?P a :Parricide} UNION
       {?NP a :Non-parricide} UNION
       {?PP a :Parent-of-parricide} UNION
       {?PNP a :Parent-of-non-parricide} UNION
       {?X a :Parent-of-parricide-Grandparent-of-non-parricide} UNION
       { :Parent-of-parricide-Grandparent-of-non-parricide
         owl:equivalentClass owl:Nothing}}
```

[Filename: RDF/parricide.sparql]

- No *X* reported, but also no extra line (which would mean “yes” for the last question)!

Example (Cont'd)

- ask Zeus whether Parent-of-parricide-Grandparent-of-non-parricide is really non-empty

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.      [Filename:
@prefix : <foo:greek#>.                                  RDF/parricide2.n3]
:zeus :knows :iokaste, :oedipus, :perineikes, :thesandros.
:Result owl:equivalentClass [ a owl:Restriction; owl:onProperty :knows;
  owl:someValuesFrom :Parent-of-parricide-Grandparent-of-non-parricide ].
```

```
prefix : <foo:greek#>
select ?R ?X
from <file:parricide.n3>
from <file:parricide2.n3>
where {{?R a :Result} UNION                                [Filename: RDF/parricide2.sparql]
      {?X a :Parent-of-parricide-Grandparent-of-non-parricide}}
```

- Zeus is in R , i.e., he knows such a person (explicitly: he knows a person who must be a P.o.p.G.o.n.p),
- but neither SPARQL, nor Zeus know who that person is,
- it can be either Oedipus or Perineikes.

QUALIFIED ROLE RESTRICTIONS

- extends owl:Restriction, owl:onProperty, {min/max}Cardinality (int value) with owl11:onClass as result class.

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@prefix : <foo://bla/names#> .
:alice :name "Alice"; :hasAnimal :pluto, :struppi.
:john :name "John"; :hasAnimal :garfield, :nermal, :odie.
:sue :hasAnimal :grizabella.
:pluto a :Dog; :name "Pluto".
:struppi a :Dog; :name "Struppi".
:garfield a :Cat; :name "Garfield".
:nermal a :Cat; :name "Nermal".
:odie a :Dog; :name "Odie".
:grizabella :name "Grizabella".
:name a owl:FunctionalProperty.
:Dog a owl:Class.   :Cat a owl:Class.
:Cat owl:disjointWith :Dog.
:HasTwoAnimals owl:equivalentClass
  [a owl:Restriction; owl:onProperty :hasAnimal; owl:minCardinality 2].
:HasTwoCats owl:equivalentClass
  [a owl:Restriction; owl:onProperty :hasAnimal; owl11:onClass :Cat; owl:minCardinality 2].
:HasTwoDogs owl:equivalentClass
  [a owl:Restriction; owl:onProperty :hasAnimal; owl11:onClass :Dog; owl:minCardinality 2].
```

[Filename: RDF/cats-and-dogs.n3]

```
prefix : <foo://bla/names#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select ?X ?Y ?Z ?C
from <file:cats-and-dogs.n3>
where {{?X a :HasTwoCats} UNION
       {?Y a :HasTwoDogs} UNION
       {?C rdfs:subClassOf :HasTwoAnimals} UNION
       {?Z a :Cat}}
```

[Filename: RDF/cats-and-dogs.sparql]

QUALIFIED ROLE RESTRICTIONS – ANOTHER TEST

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@prefix : <foo://bla/names#> .
:alice :name "Alice"; :hasAnimal :pluto, :struppi.
:john :name "John"; :hasAnimal :garfield, :nermal, :odie.
:sue :hasAnimal :grizabella.           :grizabella :name "Grizabella".
:pluto a :Dog; :name "Pluto".         :struppi a :Dog; :name "Struppi".
:garfield a :Cat; :name "Garfield".   :nermal a :Cat; :name "Nermal".
:odie a :Dog; :name "Odie".
:name a owl:FunctionalProperty.
:Dog a owl:Class.   :Cat a owl:Class.   :Cat owl:disjointWith :Dog.
:HasAnimal owl:equivalentClass
  [a owl:Restriction; owl:onProperty :hasAnimal; owl:minCardinality 1].
:HasCat owl:equivalentClass
  [a owl:Restriction; owl:onProperty :hasAnimal; owl11:onClass :Cat; owl:minCardinality 1].
:HasDog owl:equivalentClass
  [a owl:Restriction; owl:onProperty :hasAnimal; owl11:onClass :Dog; owl:minCardinality 1].
```

[Filename: RDF/hasanimals.n3]

- export class tree:
hasCat and hasDog are (non-disjoint) subclasses of hasAnimal.
- “owl11:onClass X & owl:minCardinality 1” is equivalent to “owl:someValuesFrom X”.
- “owl:minCardinality 1” alone is equivalent to “owl:someValuesFrom owl:Thing”.

DEFINING OWN DATATYPES

Two possibilities:

- use XML Schema `xsd:simpleType` definitions on the Web:
 - OWL reasoners parse+understand XML Schema `simpleType` declarations
 - adopt the DAML+OIL solution: datatype URI is constructed from the URI of the XML schema document and the local name of the simple type.
- OWL vocabulary to do the same as in XML Schema `simpleTypes`.

EXAMPLE: USING XSD DATATYPES

- Define simple datatypes in an XML Schema file:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="file:coordinates.xsd">
  <xs:simpleType name="longitudeT">
    <xs:restriction base="xs:int">
      <xs:minExclusive value="-180"/>
      <xs:maxInclusive value="180"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="easternLongitude">
    <xs:restriction base="xs:int">
      <!-- note: base="longitudeT" would be nicer, but is not allowed when parsing from RDF -->
      <xs:minExclusive value="0"/>
      <xs:maxInclusive value="180"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="latitudeT">
    <xs:restriction base="xs:int">
      <xs:minInclusive value="-90"/>
      <xs:maxInclusive value="90"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

[Filename: RDF/coordinates.xsd]

... and now use the datatypes ...

```
<!DOCTYPE rdf:RDF [ <!ENTITY mon "http://www.semwebtech.org/mondial/10/meta#">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY Coords "file:coordinates.xsd"> ]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:mon="http://www.semwebtech.org/mondial/10/meta#">

  <!-- ***** IMPORTANT: ALL DATATYPES MUST BE MENTIONED TO BE PARSED ***** -->
  <rdfs:Datatype rdf:about="&Coords;#longitudeT"/>
  <rdfs:Datatype rdf:about="&Coords;#easternLongitude"/>
  <rdfs:Datatype rdf:about="&Coords;#latitudeT"/>
  <owl:Class rdf:about="&mon;EasternHemisphere">
  <owl:equivalentClass> <!-- again: don't give a uri to an owl:Restriction! -->
    <owl:Restriction>
      <owl:onProperty rdf:resource="&mon;longitude"/>
      <owl:someValuesFrom rdf:resource="&Coords;#easternLongitude"/>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>

  <mon:City mon:name="Berlin">
    <mon:longitude rdf:datatype="&xsd;#int">13</mon:longitude>
    <mon:latitude rdf:datatype="&xsd;#int">52</mon:latitude> </mon:City>
  <mon:City mon:name="Lisbon">
    <mon:longitude rdf:datatype="&Coords;#longitudeT">-9</mon:longitude>
    <mon:latitude rdf:datatype="&Coords;#latitudeT">38</mon:latitude> </mon:City>
</rdf:RDF>
```

[Filename: RDF/coordinates.rdf]

... and now to the query:

```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
select ?N
from <file:coordinates.rdf>
where {?X :name ?N . ?X a :EasternHemisphere}
```

[Filename: RDF/coordinates.sparql]

Comments

- DOES NOT WORK CURRENTLY
- TO BE CHANGED TO DECIMAL WHEN SUPPORTED BY PELLETT
- the RDF file must “define” all used rdf:Datatypes to be parsed from the XML Schema file. (if `<rdfs:Datatype rdf:about="&Coords;#easternLongitude"/>` is omitted, the result is empty)
- if a prohibited value, e.g. longitude=200 is given in the RDF file, it is rejected.
- the rdf:Datatype for mon:longitude and mon:latitude must be given, otherwise it is not recognized as a number (but it does not matter if xsd:int or coords:longitude is used).
- specifying rdfs:range for longitude and latitude *without* rdf:Datatype for mon:longitude and mon:latitude is even inconsistent!

OWL 1.1 DATATYPES

- use the XML Schema built-in types as resources (int and string must be supported; Pellet does not yet support decimal)
- [owl:DataRange](#): cf. simple Types in XML schema; derived from the basic ones (e.g. xsd:int is an owl:DataRange)
- specified by
 - owl:onDataRange: from what owl:DataRange they are derived,
 - the facets as in XML Schema:
{max/min}{In/Ex}clusive (as in XML Schema).

DATA RANGES: ADULTS

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix : <foo://bla/names#> .
:kate :name "Kate"; :age 62; :child :john.
:john :name "John"; :age 35; :child [:name "Alice"], [:name "Bob"; :age 8].
:child rdfs:domain :Person; rdfs:range :Person.
:age a owl:FunctionalProperty; a owl:DatatypeProperty; rdfs:range xsd:int.
:name a owl:FunctionalProperty; a owl:DatatypeProperty; rdfs:range xsd:string.
:atLeast18 a owl:DataRange; owl11:onDataRange xsd:int; owl11:minInclusive 18.
:Adult owl:intersectionOf (:Person
  [ a owl:Restriction;
    owl:onProperty :age;
    owl:someValuesFrom :atLeast18])).
:Child owl:intersectionOf (:Person
  [ owl:complementOf :Adult ]).
```

```
prefix : <foo://bla/names#>
select ?AN ?CN ?X ?Y
from <file:adult.n3>
where {{?A a :Adult; :name ?AN} UNION
       {?C a :Child; :name ?CN} UNION
       {?X :age ?Y}}
```

[Filename: RDF/adult.n3]

[Filename: RDF/adult.sparql]

AN EXAMPLE WITH TWO QRRs

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix : <foo://bla/names#> .

:kate :name "Kate"; :age 62; :child :john, :sue.
:sue :name "Sue"; :age 32; :child [:name "Barbara"].
:john :name "John"; :age 35;
      :child :alice, [:name "Bob"; :age 8], [:name "Alice"; :age 10].
:frank :name "Frank"; :age 40; :child [:age 18], [:age 13].
:child rdfs:domain :Person; rdfs:range :Person.
:age a owl:FunctionalProperty; a owl:DatatypeProperty; rdfs:range xsd:int.
:name a owl:FunctionalProperty; a owl:DatatypeProperty; rdfs:range xsd:string.
:atLeast18 a owl:DataRange; owl11:onDataRange xsd:int; owl11:minInclusive 18.
:Adult owl:intersectionOf (:Person
  [ a owl:Restriction; owl:onProperty :age; owl:someValuesFrom :atLeast18]).
:HasTwoAdultChildren owl:equivalentClass [ a owl:Restriction;
  owl:onProperty :child; owl11:onClass :Adult; owl:minCardinality 2 ].
```

[Filename: RDF/adultchildren.n3]

```
prefix : <foo://bla/names#>
select ?AN ?N
from <file:adultchildren.n3>
where {{?A a :Adult; :name ?AN} UNION
       {?X a :HasTwoAdultChildren; :name ?N}}
```

[Filename: RDF/adultchildren.sparql]

DATA RANGE RESTRICTION OF XSD:INT FOR COORDINATES

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
```

```
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
```

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
```

```
@prefix mon: <http://www.semwebtech.org/mondial/10/meta#>
```

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
```

```
@prefix : <foo://bla/>.
```

```
### workaround as long as only one restricting facet is allowed per item:
```

```
:Longitude1 a owl:DataRange; owl11:onDataRange xsd:int; owl11:minExclusive -180.
```

```
:Longitude a owl:DataRange; owl11:onDataRange :Longitude1; owl11:maxInclusive 180.
```

```
:Latitude1 a owl:DataRange; owl11:onDataRange xsd:int; owl11:minInclusive -90.
```

```
:Latitude a owl:DataRange; owl11:onDataRange :Latitude1; owl11:maxInclusive 90.
```

```
:EasternLongitude a owl:DataRange; owl11:onDataRange :Longitude; owl11:minInclusive 0.
```

```
:EasternHemispherePlace owl:equivalentClass [a owl:Restriction;  
  owl:onProperty mon:longitude; owl:someValuesFrom :EasternLongitude].
```

```
mon:longitude rdfs:range :Longitude.
```

```
mon:latitude rdfs:range :Latitude.
```

```
:Berlin a mon:City; :name "Berlin"; mon:longitude 13; mon:latitude 52.
```

```
#:Atlantis a mon:City; :name "Atlantis"; mon:longitude -200; mon:latitude 100.
```

```
:Lisbon a mon:City; :name "Lisbon"; mon:longitude -9; mon:latitude 38.
```

```
prefix : <foo://bla/>  
select ?N  
from <file:coordinates2.n3>  
where {?X :name ?N .  
       ?X a :EasternHemispherePlace}
```

[Filename: RDF/coordinates2.sparql]

[Filename: RDF/coordinates2.n3]

QUALIFIED ROLE RESTRICTIONS: EXAMPLE

Example: Country with at least two cities with more than a million inhabitants.

- define “more than a million” as a owl:DataRange
- search for all BigCities (= more than 1000000 inhabitants)
- check -via Provinces- which countries have two such cities.

Example: Cont'd

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@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 : <foo://bla/>.

mon:population rdfs:range xsd:int; a owl:FunctionalProperty. ## all cities are different.
:Million a owl:DataRange; owl11:onDataRange xsd:int; owl11:minInclusive 1000000.
:HasBigPopulation owl:equivalentClass [a owl:Restriction;
  owl:onProperty mon:population; owl:someValuesFrom :Million].
:BigCity owl:intersectionOf (mon:City :HasBigPopulation).
:ProvinceWithBigCity owl:intersectionOf (mon:Province ## CORRECT
  [a owl:Restriction; owl:onProperty mon:hasCity; owl:someValuesFrom :BigCity]).
:ProvinceWithTwoBigCities owl:intersectionOf (mon:Province ## europe: empty
  [a owl:Restriction; owl:onProperty mon:hasCity; owl11:onClass :BigCity; owl:minCardinality 2]).
[owl:intersectionOf (mon:Country ## with 2 big cities, no provinces ## europe: empty
  [a owl:Restriction; owl:onProperty mon:hasCity; owl11:onClass :BigCity; owl:minCardinality 2]);
  rdfs:subClassOf :CountryWithTwoBigCities].
[owl:intersectionOf (mon:Country ## with 2 provs with big cities ## TR,GB,E,R,UA,D,I,NL
  [a owl:Restriction; owl:onProperty mon:hasProvince; owl11:onClass :ProvinceWithBigCity; owl:minCardinality 2]);
  rdfs:subClassOf :CountryWithTwoBigCities].
[owl:intersectionOf (mon:Country ## with a prov with 2 big cities ## europe: empty
  [a owl:Restriction; owl:onProperty mon:hasProvince; owl:someValuesFrom :ProvinceWithTwoBigCities]);
  rdfs:subClassOf :CountryWithTwoBigCities].
[owl:intersectionOf (mon:Country ## Test: with a prov with 1 big city - + PL,A,F,CZ,H,RO
  [a owl:Restriction; owl:onProperty mon:hasProvince; owl:someValuesFrom :ProvinceWithBigCity]);
  rdfs:subClassOf :CountryWithBigCity].
```

[Filename: RDF/bigcities.n3]

Example: Cont'd

```
prefix : <foo://bla/>
prefix mon: <http://www.semwebtech.org/mondial/10/meta#>
select ?BC ?P1 ?P2 ?X1 ?X2
from <file:bigcities.n3>
from <file:mondial-europe.n3>
from <file:mondial-meta.n3>
where {# {?BC a :BigCity} UNION
      # {?P1 a :ProvinceWithBigCity} UNION
      # {?P2 a :ProvinceWithTwoBigCities} UNION
      {?X1 a :CountryWithBigCity} UNION
      {?X2 a :CountryWithTwoBigCities}}
```

[Filename: RDF/bigcities.sparql]

BIGCITIES: ALTERNATIVE

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@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 : <foo://bla/>.
```

```
:grmny a mon:Country; :hasProvOrCity :bln, :mch.
:bln a :BigCity; mon:population 3500000.
:mch a :BigCity; mon:population 1500000.
:frc a mon:Country; mon:hasProvince :ile, :prov.
:prs a mon:City; mon:cityIn :ile; mon:population 2000000.
:mrs a mon:City; mon:cityIn :prov; mon:population 1500000.
```

```
mon:hasProvince owl:inverseOf mon:isProvinceOf.
mon:isProvinceOf rdfs:subPropertyOf mon:belongsTo.
mon:cityIn rdfs:subPropertyOf mon:belongsTo.
mon:belongsTo a owl:TransitiveProperty; owl:inverseOf :hasProvOrCity. ## bridge country-prov-city
mon:population rdfs:range xsd:int; a owl:FunctionalProperty. ## all cities are different.
:Million a owl:DataRange; owl11:onDataRange xsd:int; owl11:minInclusive 1000000.
:HasBigPopulation owl:equivalentClass [a owl:Restriction;
  owl:onProperty mon:population; owl:someValuesFrom :Million].
:BigCity owl:intersectionOf (mon:City :HasBigPopulation).

:CountryWithTwoBigCities owl:intersectionOf (mon:Country
  [a owl:Restriction; owl:onProperty :hasProvOrCity; owl11:onClass :BigCity; owl:minCardinality 2]).
:CountryWithTwoBigCities2 owl:intersectionOf (mon:Country
  [a owl:Restriction; owl:onProperty :hasProvOrCity; owl11:onClass :BigCity; owl:minCardinality 2]).
```

```
prefix : <foo://bla/>
prefix mon:
  <http://www.semwebtech.org/mondial/10/meta#>
select ?BC ?X ?Y ?Z1 ?Z2
from <file:bigcities2.n3>
where {{?BC a :BigCity} UNION
       {?X :hasProvOrCity ?Y} UNION
       {?Z1 a :CountryWithTwoBigCities} UNION
       {?Z2 a :CountryWithTwoBigCities2}}
```

[Filename: RDF/bigcities2.sparql]

[Filename: RDF/bigcities2.n3]

ANOTHER TEST

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix mon: <http://www.semwebtech.org/mondial/10/meta#>.
@prefix : <foo://bla/>.
```

```
:grmny a mon:Country; :hasProvOrCity :bln.
:bln a mon:City; mon:population 3500000.
:frc a mon:Country; mon:hasProvince :ile.
:ile a mon:Province.
:prs a mon:City; mon:cityIn :ile; mon:population 2000000.
```

```
mon:hasProvince owl:inverseOf mon:isProvinceOf.
mon:isProvinceOf rdfs:subPropertyOf mon:belongsTo.
mon:cityIn rdfs:subPropertyOf mon:belongsTo.
```

```
mon:belongsTo a owl:TransitiveProperty;
  owl:inverseOf :hasProvOrCity. ## bridge country-prov-city
mon:population a owl:FunctionalProperty. ## all cities are different.
```

```
:CountryWithCity owl:intersectionOf (mon:Country
  [a owl:Restriction; owl:onProperty :hasProvOrCity; owl11:onClass mon:City; owl:minCardinality 1]).
:CountryWithProvince owl:intersectionOf (mon:Country
  [a owl:Restriction; owl:onProperty :hasProvOrCity; owl11:onClass mon:Province; owl:minCardinality 1]).
:CountryWithCity2 owl:intersectionOf (mon:Country
  [a owl:Restriction; owl:onProperty :hasProvOrCity; owl11:onClass owl:Thing; owl:minCardinality 1]).
```

```
prefix : <foo://bla/>
prefix mon:
  <http://www.semwebtech.org/mondial/10/meta#>
select ?C ?X ?Y ?Z1 ?Z2 ?Z3
from <file:cwc.n3>
where {{?C a mon:City} UNION
      {?X :hasProvOrCity ?Y} UNION
      {?Z1 a :CountryWithCity} UNION
      {?Z2 a :CountryWithProvince} UNION
      {?Z3 a :CountryWithCity2}}
```

[Filename: RDF/cwc.sparql]

[Filename: RDF/cwc.n3]

ENUMERATED DATATYPES

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix uni: <foo://uni/>.
uni:graded a owl:FunctionalProperty;
  a owl:DatatypeProperty; rdfs:range uni:Grades.
uni:Grades a owl:DataRange; owl11:onDataRange xsd:string;
  owl:oneOf ("1.0" "1.3" "1.7" "2.0" "2.3" "2.7" "3.0" "3.3" "3.7" "4.0").
[ a uni:Thesis; uni:author <foo://bla/john>;
  uni:graded "3.0"].
```

[Filename: RDF/grades-one-of.n3]

```
prefix : <foo://uni/>
select ?X ?G
from <file:grades-one-of.n3>
where {?X :graded ?G}
```

[Filename: RDF/grades-one-of.sparql]

- changing the grade to 2.5 results in an inconsistency.
- note: “3” and “3.0” are different strings.

ONEOF ON DATARANGE

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix : <foo://bla/names#>.
:Person owl11:disjointUnionOf (:Male :Female).
:MaleNames a owl:DataRange; owl11:onDataRange xsd:string;
    owl:oneOf ("John"^^xsd:string "Bob"^^xsd:string).
:FemaleNames a owl:DataRange; owl11:onDataRange xsd:string;
    owl:oneOf ("Mary"^^xsd:string "Alice"^^xsd:string).
:Male a owl:Class; owl:equivalentClass [owl:intersectionOf ( :Person
    [a owl:Restriction; owl:onProperty :name; owl:someValuesFrom :MaleNames])].
:Female a owl:Class; owl:equivalentClass [owl:intersectionOf ( :Person
    [a owl:Restriction; owl:onProperty :name; owl:someValuesFrom :FemaleNames])].
:FemaleNames owl:disjointWith :MaleNames.
:name a owl:FunctionalProperty; a owl:DatatypeProperty.
:john a :Person; :name "John"^^xsd:string.
:mary a :Person; :name "Mary"^^xsd:string.
```

[Filename: RDF/names.n3]

```
prefix : <foo://bla/names#>
select ?C ?N
from <file:names.n3>
where { :john a ?C ; :name ?N }
```

[Filename: RDF/names.sparql]

REIFICATION

Reification: treat a class (or a property or a statement) as a thing:

- Male and Female are both classes and instances of class Sex

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix : <foo://bla/names#>.
:Person owl11:disjointUnionOf (:Male :Female).
:Male a :Sex.
:Female a :Sex.
:MaleNames a owl:DataRange; owl11:onDataRange xsd:string;
    owl:oneOf ("John"^^xsd:string "Bob"^^xsd:string).
:FemaleNames a owl:DataRange; owl11:onDataRange xsd:string;
    owl:oneOf ("Mary"^^xsd:string "Alice"^^xsd:string).
:Male a owl:Class; owl:equivalentClass [owl:intersectionOf ( :Person
    [a owl:Restriction; owl:onProperty :name; owl:someValuesFrom :MaleNames])].
:Female a owl:Class; owl:equivalentClass [owl:intersectionOf ( :Person
    [a owl:Restriction; owl:onProperty :name; owl:someValuesFrom :FemaleNames])].
:FemaleNames owl:disjointWith :MaleNames.
:name a owl:FunctionalProperty; a owl:DatatypeProperty.
:john a :Person; :name "John"^^xsd:string.
:mary a :Person; :name "Mary"^^xsd:string.
```

[Filename: RDF/reification-class.n3]

```
prefix : <foo://bla/names#>
select ?P ?N ?S
from <file:reification-class.n3>
where {{?S a :Sex .
        ?P a :Person ; a ?S ; :name ?N}}
```

[Filename: RDF/reification-class.sparql]

7.4 OWL 1.1: Properties

- *SHIQ*/OWL-DL concentrate on concept definitions,
- *SHOIQ*/*SHOIQ(D)* add Nominals and datatypes,
- The *H* allows for a hierarchy of properties as already provided by RDFS, the *I* allows for inverse.
- *SROIQ* provides more expressiveness around properties.

SYMMETRIC PROPERTIES (OWL 1.0)

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/names#> .
:germany :borders :austria, :switzerland.
:borders a owl:SymmetricProperty.
```

[Filename: RDF/symmetricborders.n3]

```
prefix : <foo://bla/names#>
select ?X ?Y
from <file:symmetricborders.n3>
where {?X :borders ?Y}
```

[Filename: RDF/symmetricborders.sparql]

REFLEXIVE PROPERTIES (OWL 1.1)

```
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@prefix : <foo://bla/names#> .
:john a :Person; :knows :mary; :child :alice.
:knows a owl11:ReflexiveProperty.
:germany a :Country.
```

[Filename: RDF/reflexive.n3]

```
prefix : <foo://bla/names#>
select ?X ?Y
from <file:reflexive.n3>
where {?X :knows ?Y}
```

[Filename: RDF/reflexive.sparql]

- only applied to individuals, but ... to all of them:
John knows John, Alice knows Alice, and Germany knows Germany.

IRREFLEXIVE PROPERTIES

- $\text{irreflexive}(rel): \forall x : \neg rel(x, x).$
- acts as constraint,
- but can also induce that two things must be different:

$$\forall x, y : rel(x, y) \rightarrow x \neq y$$

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@prefix : <foo://bla/names#> .
:john :hasAnimal :pluto, :garfield.
#:hasAnimal a owl:FunctionalProperty.
:pluto :bites :garfield.
:bites a owl11:IrreflexiveProperty.
:HasTwoAnimals owl:equivalentClass
  [ a owl:Restriction;
    owl:onProperty :hasAnimal; owl:minCardinality 2 ].
```

[Filename: RDF/irreflexive.n3]

```
prefix : <foo://bla/names#>
select ?X ?Y ?Z
from <file:irreflexive.n3>
where {{?X :bites ?Y} UNION
       {?X :bites ?X} UNION
       {?Z a :HasTwoAnimals}}
```

[Filename: RDF/irreflexive.sparql]

- Pluto cannot be the same as Garfield.

ANTISYMMETRY

- $\text{antisymmetric}(rel): \forall x, y : (rel(x, y) \wedge rel(y, x)) \rightarrow x = y.$
- acts as a constraint.

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@prefix : <foo://bla/names#>.
owl:AllDifferent owl:distinctMembers (:a :b).
:rel a owl11:AntisymmetricProperty.
:a :rel :b.
:b :rel :a.
```

[Filename: RDF/antisymmetry.n3]

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <foo://bla/names#>
select ?X ?Y ?A ?B
from <file:antisymmetry.n3>
where {{?X :rel ?Y} UNION {?A owl:sameAs ?B}}
```

[Filename: RDF/antisymmetry.sparql]

IRREFLEXIVE AND ANTISYMMETRIC PROPERTIES

- Motivated by the “Ascending, Descending” graphics by M.C.Escher
http://en.wikipedia.org/wiki/Ascending_and_Descending

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix : <foo://bla/names#>.

:Corner owl:oneOf (:a :b :c); rdfs:subClassOf
  [a owl:Restriction; owl:onProperty :higher; owl:cardinality 1].
:higher rdfs:domain :Corner; rdfs:range :Corner.
#:higher a owl:FunctionalProperty. ## redundant, note cardinality 1
#:higher a owl:InverseFunctionalProperty. ## also redundant
:higher a owl11:AntisymmetricProperty.
:higher a owl11:IrreflexiveProperty.
:a :higher :b.
```

[Filename: RDF/escherstairs.n3]

```
prefix : <foo://bla/names#>
select ?X ?Y
from <file:escherstairs.n3>
where {?X :higher ?Y}
```

[Filename: RDF/escherstairs.sparql]

- Solution: $a > b, b > c, c > a$ is a valid model.
- note: can be extended to arbitrary n where every set of cycles through all corners is a solution!

DISJOINT PROPERTIES

- Syntax: (prop₁ owl11:disjointObjectProperties prop₂)

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix : <foo://bla/names#>.
```

```
:alice :name "Alice"; :hasDog :pluto, :struppi.
:john :name "John"; :hasCat :garfield, :nermal; :hasDog :odie.
:sue :hasCat :grizabella.
:sue :hasDog :grizabella.   ### test #####
:pluto a :Dog; :name "Pluto".
:struppi a :Dog; :name "Struppi".
:garfield a :Cat; :name "Garfield".
:nermal a :Cat; :name "Nermal".
:odie a :Dog; :name "Odie".
:grizabella :name "Grizabella".
:name a owl:FunctionalProperty.
:Cat owl:disjointWith :Dog.

:hasCat rdfs:subPropertyOf :hasAnimal.
:hasDog rdfs:subPropertyOf :hasAnimal.
:hasCat owl11:disjointObjectProperties :hasDog.
```

```
prefix : <foo://bla/names#>
select ?A ?B ?C ?D ?E ?F
from <file:disjointproperties.n3>
where {{?A :hasCat ?B} UNION
       {?C :hasDog ?D} UNION
       {?E :hasAnimal ?F}}
```

[Filename: RDF/disjointproperties.n3]

[Filename: RDF/disjointproperties.sparql]

AT THE DECIDABILITY BORDER

Some advanced constructs in DL that are proposed for OWL 1.1 are not even decidable

- \mathcal{ALC}_{reg} with transitivity, composition and union is EXPTIME-complete
- the same when inverse roles and even cardinalities for *atomic* roles (\mathcal{ALCQI}_{reg}) are added (recall that inverse and transitive closure are important concepts in ontologies).
- The combination of *non-atomic* roles with cardinalities is in general undecidable.
- The same holds for Role-Value-Maps. Decidability is obtained only for Role-Value-Maps over *functional* roles.

CARDINALITIES ON ATOMIC ROLES

- a city can be the capital of at most one country (but also of one or more provinces)

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@prefix : <http://www.semwebtech.org/mondial/10/meta#>.
@prefix mon: <http://www.semwebtech.org/mondial/10/>.

:City a owl:Class; owl:equivalentClass
  [a owl:Restriction; owl:onProperty :isCapitalOf;
    owl11:onClass :Country; owl:maxCardinality 1 ].

:name a owl:FunctionalProperty.
mon:C-Oslo a :City;
  :isCapitalOf mon:Norway, mon:P-Akershus, mon:P-Oslo.
mon:P-Akershus a :Province; :name "Akershus".
mon:P-Oslo a :Province; :name "Oslo".
mon:Norway a :Country; :name "Norway".
# mon:C-Oslo :isCapitalOf :foo. :foo a :Country; :name "Foo".
```

[Filename: RDF/one-capital.n3]

ACROSS THE DECIDABILITY BORDER

- Cardinality restrictions on complex (e.g. transitive) properties are not allowed (undecidable)

Every city can be located in several provinces, but these must belong to the same country.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@prefix : <http://www.semwebtech.org/mondial/10/meta#>.
@prefix mon: <http://www.semwebtech.org/mondial/10/>.

# Countries, Provinces, Cities:
:cityIn rdfs:subPropertyOf :belongsTo; rdfs:range :Province.
:isProvinceOf a owl:FunctionalProperty; rdfs:range :Country; rdfs:subPropertyOf :belongsTo.
:belongsTo a owl:TransitiveProperty; owl:inverseOf :hasProvOrCity.

:City a owl:Class; owl:equivalentClass
  [a owl:Restriction; owl:onProperty :belongsTo;
   owl11:onClass :Country; owl:maxCardinality 1].

:name a owl:FunctionalProperty.
mon:C-Oslo a :City; :cityIn mon:P-Akershus, mon:P-Oslo.
mon:Norway a :Country; :name "Norway".
mon:P-Akershus a :Province; :isProvinceOf mon:Norway; :name "Akershus".
mon:P-Oslo a :Province; :isProvinceOf mon:Norway; :name "Oslo".
# mon:C-Oslo :isCapitalOf :foo. :foo a :Country; :name "Foo".      [Filename: RDF/one-country.n3]
```

Detection of Potentially Undecidable Situations

Pellet does not accept combinations that can potentially be undecidable

The ontology is rejected by Pellet:

- Unsupported axiom: Ignoring transitivity axiom due to an existing cardinality restriction for property `http://www.semwebtech.org/mondial/10/meta#belongsTo`

- It is also rejected if

`:cityIn a owl:FunctionalProperty.`

`:isProvinceOf a owl:FunctionalProperty.`

is added (which guarantees decidability).

FURTHER FEATURES OF OWL 1.1

- Role Chains/Property Chains: `SubObjectPropertyOf(SubObjectPropertyChain(owns hasPart) owns)`
asserts that if x owns y and y has a part z , then x owns z .
`SubObjectPropertyOf(SubObjectPropertyChain(parent brother) uncle)`
asserts that the relationship “uncle” is a superset of “parent \circ brother”, i.e., the brothers of my parents are my uncles.
- Cross-property restrictions/role-value maps:
(cf. draft at <http://www.w3.org/Submission/owl111-overview/>)
 - `ObjectAllValuesFrom(likes knows =)` describes the class of individuals who like all people they know (in DL syntax: the concept defined by the role value map $(X.knows \sqsubseteq X.likes)$).
 - `DataSomeValuesFrom(shoeSize IQ greaterThan)`
describes the class of individuals whose shoeSize is greater than their IQ (in DL syntax: the concept defined by the role value map $(X.shoeSize > X.IQ)$).

ROLE CHAINS

- $\text{uncle} \equiv \text{brotherOf} \circ \text{child}$
- if a list is a `owl:subProperty` of something, it is interpreted as a role chain definition.
- **note: it is required to declare uncle as `owl:ObjectProperty`**

[Filename: RDF/uncle.n3]

```
prefix : <foo://bla/names#>
select ?U ?X
from <file:uncle.rdf>
where {?U :uncleOf ?X}
```

[Filename: RDF/uncle.sparql]

Syntax Role Chains in XML/RDF

... as expected: an rdf:List which is an owl:subPropertyOf another property.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns="foo://bla/names#"
  xml:base="foo://bla/names">
  <owl:ObjectProperty rdf:ID="uncleOf"/> <!-- !!!!! required !!!!! -->
  <rdf:List>
    <rdfs:subPropertyOf rdf:resource="#uncleOf"/>
    <rdf:rest rdf:parseType="Collection">
      <owl:ObjectProperty rdf:about="#child"/>
    </rdf:rest>
    <rdf:first rdf:resource="#brotherOf"/>
  </rdf:List>
  <Person rdf:ID="sue">
    <child rdf:resource="#anne"/>
    <child rdf:resource="#barbara"/>
    <brother rdf:resource="#john"/>
  </Person>
  <Person rdf:ID="john">
    <brotherOf rdf:resource="#sue"/>
  </Person>
</rdf:RDF>
```

```
prefix : <foo://bla/names#>
select ?U ?X
from <file:uncle.rdf>
where {?U :uncleOf ?X}
```

[Filename: RDF/uncle2.sparql]

[Filename: RDF/uncle.rdf]

Undecidable: Role Chains and Cardinalities

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/names#> .
@prefix person: <foo://bla/persons/> .
```

```
:uncleOf a owl:ObjectProperty.   ### required !!!!!!!!!!!
[ ] a rdf:List; rdfs:subPropertyOf :uncleOf;
    rdf:first :brotherOf ; rdf:rest (:child).
person:john a :Person; :brotherOf person:sue.
person:sue a :Person; :child person:anne, person:barbara.
```

```
:name a owl:FunctionalProperty.
:anne :name "Anne".   :barbara :name "Barbara".
:UncleOfMore a owl:Class; owl:equivalentClass
[a owl:Restriction; owl:onProperty :uncleOf; owl:minCardinality 2].
```

[Filename: RDF/uncleOfMore.n3]

```
prefix : <foo://bla/names#>
select ?U ?X
from <file:uncleOfMore.n3>
where {{?U :uncleOf ?X} UNION
      {?U a :uncleOfMore}}
```

[Filename: RDF/uncleOfMore.sparql]

- pellet: Definition of uncle is ignored; result empty.
WARN [main] (OWLLoader.java:352) - Unsupported axiom: Ignoring complex sub property axiom due to an existing cardinality restriction for property
foo://bla/names#uncleOf

SELF RESTRICIONS: $\{x \mid x r x\}$

- buggy. “a” is not detected to be cyclic.
(:b :to :b) is correctly derived.

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix owl11: <http://www.w3.org/2006/12/owl11#>.
@prefix : <foo://bla/>.
:Cyclic a owl:Class;
  owl:intersectionOf
  ( :Node [a owl11:SelfRestriction; owl:onProperty :to] ).
:b a :Cyclic.
:a a :Node; :to :a, :b.
```

[Filename: RDF/cyclic.n3]

```
prefix : <foo://bla/>
select ?N ?N2
from <file:cyclic.n3>
where {{?N a :Cyclic} UNION
       {?N :to ?N2}}
```

[Filename: RDF/cyclic.sparql]

7.5 DL and OWL Proving and Query Answering

- Tableau provers use refutation techniques:

Given an ontology formalization Φ ,

prove $\Phi \models \varphi$ by starting a tableau over $\Phi \wedge \neg\varphi$ and trying to close it.

For that, it is well-suited for *testing* if something holds:

- consistency of a concept definition:

$KB \models C \equiv \perp \Leftrightarrow KB \cup \{C(a)\}$ for a new constant a is unsatisfiable.

- concept containment:

$KB \models C \sqsubseteq D \Leftrightarrow KB \models (C \sqcap \neg D) \equiv \perp$.

- concept equivalence:

$KB \models C \equiv D \Leftrightarrow KB \models C \sqsubseteq D$ and $KB \models D \sqsubseteq C$.

- concept membership (for a given individual a):

$KB \models C(a) \Leftrightarrow KB \cup \{\neg C(a)\}$ is unsatisfiable.

TABLEAU EXPANSION RULES FOR DL

- DL: use tableau without free variables. Expansion of universally quantified formulas takes only place for constants that are actually introduced.
- makes it more similar to Model Checking
- actually, not the tableau is generated completely, but branches are investigated by backtracking.

$(C \sqcap D)(s)$	Add $C(s)$ and $D(s)$ to the branch.
$(C \sqcup D)(s)$	Add two branches, one with $C(s)$, the other with $D(s)$.
$\exists R.C(s)$	Add $R(s, x)$ and $C(x)$ where x is new.
$\forall R.C(s)$	Add $C(t)$ whenever $R(s, t)$ is on the tableau (requires bookkeeping).
$\geq nR.C(s)$	Add $R(s, x_1), \dots, R(s, x_n), C(x_1), \dots, C(x_n)$ and $x_i \neq x_j$ where x_i are new.
$\leq nR.C(s)$	Bookkeeping about $\{x \mid R(s, x)\}$. Whenever more than n , then add branches with all combinations $x_i = x_j$. Continue bookkeeping.
$C \sqsubseteq D$	For each s recursively add two branches with $\neg C(s)$ and $D(s)$.
Closure	Close a branch whenever $A(s)$ and $\neg A(s)$ occur.

QUERY ANSWERING IN DL AND OWL

Query answering requires to find all answer bindings to variables.

- find all X such that $KB \models C(X)$.
- find all D such that $KB \models D \sqsubseteq C$.

Start a tableau and collect substitutions that close branches:

- start with $KB \cup \{\neg C(X)\}$.
- collect substitutions for X for which the tableau closes.
- without free variables: generate a new $\neg C(s)$ whenever any rule introduces a constant s .
(= check if that s is an answer)
- harder to implement.
Not always all answers are found by the current implementations.
- help the system by not only asking “ $\{?X : \text{age } ?Y\}$ ”, but pruning the search space by “ $\{?X : \text{a :Person; :age } ?Y\}$ ”.

DL TABLEAUX: EXAMPLES

Who are John's children?

hasChild(kate, john)
 name(john, "John")
 hasChild(john, alice)
 name(alice, "Alice")
 hasChild(john, bob)
 name(bob, "Bob")

Query: ?- hasChild(john, X).

\neg hasChild(john, X)
 $\square\{X_1 \leftarrow \text{alice}\}$
 $\square\{X_2 \leftarrow \text{bob}\}$

What are the names of John's children?

hasChild(john, alice)
 hasChild(john, bob)
 name(john, "John")
 name(alice, "Alice")
 name(bob, "Bob")

Query: ?- hasChild(john, X), name(X , N).

$\neg(\text{hasChild}(\text{john}, X) \wedge \text{name}(X, N))$
 $\neg(\text{hasChild}(\text{john}, X)) \quad \neg\text{name}(X, N)$
 Try $\square\{X_1 \leftarrow \text{alice}\}$ for X_1 and X_2 :
 Try $\square\{X_2 \leftarrow \text{bob}\}$ X_1 X_2
 $\neg \text{name}(\text{alice}, N)$ $\neg \text{name}(\text{bob}, N)$
 $N_1 \leftarrow \text{"Alice"}$ $N_2 \leftarrow \text{"Bob"}$

- Note: one could try close the right branch with $X_0 \leftarrow \text{john}$ and $N_0 \leftarrow \text{"John"}$, but for that, the left branch will not close.

DL TABLEAUX: EXAMPLES

Consider the “Only female children” example from Slide 341.

TwoChildrenParent(sue)
 child(sue,ann)
 Female(ann)
 child(sue,barbara)
 Female(barbara)
 ann ≠ barbara

TwoChildrenParent $\sqsubseteq \exists 2 \text{ child. } \top$

OnlyFemaleChildrenParent $\sqsubseteq \text{Person} \sqcap \forall \text{child.Female}$

Query: ?- OnlyFemaleChildrenParent(X).

[$\neg \text{OnlyFemaleChildrenParent}(X)$]

$\neg(\text{Person} \sqcap \forall \text{child.Female}(X))$

$\neg \text{Person}(X) \quad \neg \forall \text{child.Female}(sue)$

try $\square \{X \leftarrow sue\}$ $\exists \text{child.}(\neg \text{Female})(sue)$

child(sue,y)

$\neg \text{Female}(y)$

count Sue's children=3: ann,barbara,y

ann=barbara ann=y barbara=y

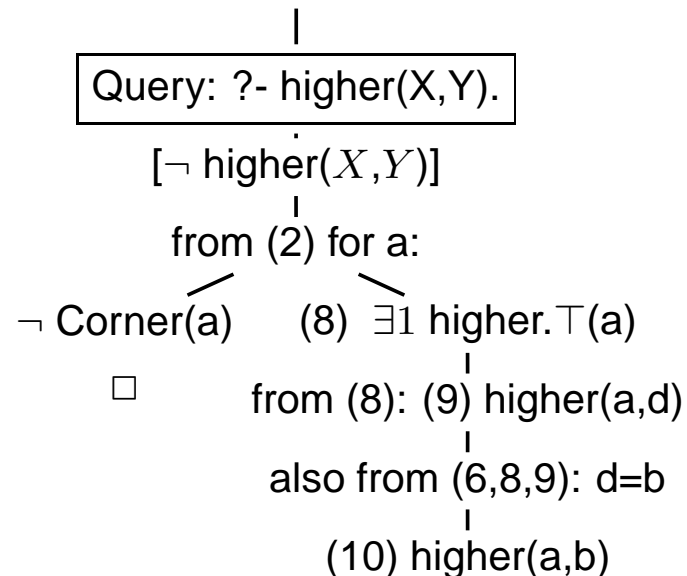
\square $\square \neg \text{Female}(ann)$ $\square \neg \text{Female}(barbara)$

- the negated query can be used for leading the expansion, but not for closing the tableau.
- Instead of X , all other persons are also tried to derive answers:
 John: tableau does not close (Alice)
 Kate: tableau does not close (Sue)

DL TABLEAUX: A MORE INVOLVED EXAMPLE

Consider again the Escher Stairs example (Slide 385).

- (1) Corner = AllDifferent(a,b,c)
- (2) cardinality: Corner $\sqsubseteq \exists 1$ higher. \top
- (3) domain: Corner $\sqsupseteq \exists$ higher. \top
- (4) range: $\top \sqsubseteq \forall$ higher.Corner
- (5) AntiSymmetric(higher)
- (6) Irreflexive(higher)
- (7) higher(a,b)



- the negated query can be used for leading the expansion, but not for closing the tableau. The first answer is higher(a,b) – which was given in the input. Try to find additional ones ...
- (2) can be applied for any constant, i.e., a, b, c, but also for e.g., john, germany etc. But the latter will not close the left branch.
- ... so choose “a” since it is already used in another fact.
- (10) (a,b) has already been reported and is ignored. As a fact, it belongs to the model of this branch. Continue the branch to check its consistency, and search for further answers in this model.
- how to continue? – Apply (2) again, for b.

Escher stairs tableau: continue with (2) for b

- (1) Corner = AllDifferent(a,b,c)
- (2) cardinality: Corner \sqsubseteq $\exists 1$ higher. \top
- (4) range: $\top \sqsubseteq \forall$ higher.Corner
- (5) AntiSymmetric(higher)
- (6) Irreflexive(higher)
- (7) higher(a,b)

$[\neg \text{higher}(X,Y)]$

from (2) for a:

$\neg \text{Corner}(\bar{a})$ (8) $\exists 1$ higher. $\top(a)$
 \square from (8): (9) higher(a,d)

also from (6,8,9): d=b

(10) higher(a,b)

from (2) for b:

$\neg \text{Corner}(\bar{b})$ (11) $\exists 1$ higher. $\top(b)$

\square from (11): (12) higher(b,e)

next: range of "higher" derives that e is a corner:

(4) $\top \sqsubseteq \forall$ higher.Corner

$\neg \top(b)$ (13) \forall higher.Corner(b)

\square (14) Corner(e)

e must be one of a,b,c – next: three branches ...

Escher stairs tableau: continued

- (1) Corner = AllDifferent(a,b,c)
- (2) cardinality: $\text{Corner} \sqsubseteq \exists 1 \text{ higher.}\top$
- (4) range: $\top \sqsubseteq \forall \text{ higher.Corner}$
- (5) AntiSymmetric(higher)
- (6) Irreflexive(higher)
- (7) higher(a,b)

$[\neg \text{higher}(X,Y)]$

(13) higher(b,e)

(14) Corner(e)

e must be one of a,b,c

e=a

higher(b,a)

□ (Antisymm.)

e=b

higher(b,b)

□ (Irrefl.)

e=c

(15) higher(b,c)

Answer: (b,c)

continue with (2) for c

$\neg \text{Corner}(\bar{c})$ (16) $\exists 1 \text{ higher.}\top(b)$

□ from (16): (17) higher(c,f)

next: range of "higher" derives that f is a corner:

(4) $\top \sqsubseteq \forall \text{ higher.Corner}$

$\neg \top(c)$ (18) $\forall \text{ higher.Corner}(c)$

□ (19) Corner(f)

Escher stairs tableau continued

- (1) Corner = AllDifferent(a,b,c)
- (2) cardinality: Corner $\sqsubseteq \exists 1$ higher. \top
- (4) range: $\top \sqsubseteq \forall$ higher.Corner
- (5) AntiSymmetric(higher)
- (6) Irreflexive(higher)
- (7) higher(a,b)

$[\neg \text{higher}(X,Y)]$

⋮

(15) higher(b,c)

(17) higher(c,f)

(19) Corner(f)

f must be one of a,b,c

f=a

higher(c,a)

Answer: (c,a)

[1]

f=b

higher(c,b)

□ (Antisymm.)

f=c

higher(c,c)

□ (Irrefl.)

The branch [1] cannot be closed. All formulas on this branch are consistent and describe a model. The answers to ?- higher(X,Y) in this model are (a,b), (b,c), and (c,d).

REQUIREMENTS ON (NOT ONLY DL) TABLEAU STRATEGIES

- select most promising formula to be expanded next
 - based on coincident symbols
 - “selectivity” of conditions
 - α -rules non-branching before β -rules (branching)
- non-closing branches: know when to stop and return answer matches
 - “saturated” branches: expansion does not add new formulas
 - do not expand irrelevant formulas at all

DL TABLEAUX: SO FAR, SO GOOD ...

Consider the axiom

$$\text{Person} \sqsubseteq \exists \text{hasParent. Person}$$

The tableau generation does not terminate.

Blocking

- a constant s_2 is introduced as an existential filler from expanding a fact about constant s_1 ,
- the knowledge about s_1 and s_2 is *saturated* (i.e., nothing new about them can be derived),
- and the same facts are known about s_1 and s_2 except the above existential chain,
- then *block* s_2 from application of the existential formula (which would just create another same thing).
- Such blocking can be done for every existentially introduced thing, and it has only to be dropped if differences between it and its “predecessor” are derived.
- Such ontologies can be used. Queries only return instances in the “relevant” finite portion.

BLOCKING

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix : <foo://bla/names#>.
:kate a :Person; :name "Kate"; :child :john.
:john a :Person; :name "John"; :child :alice.
:alice a :Person; :name "Alice".
:child rdfs:domain :Parent; owl:inverseOf :Parent owl:subClassOf
[a owl:Restriction; owl:onProperty :child owl:hasValue :Parent] owl:equivalentClass
[a owl:Restriction; owl:onProperty :child owl:hasValue :Grandparent] owl:equivalentClass
[a owl:Restriction; owl:onProperty :child owl:hasValue :kate] owl:equivalentClass

prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix : <foo://bla/names#>
select ?A ?B ?X
from <file:infinite-parents.n3>
where {{?A a :Parent} UNION
       {?B a :Grandparent} UNION
       {:parent rdfs:range :Parent} UNION
       {:kate :parent ?X}} # kate has no parent?!
```

[Filename: RDF/infinite-parents.n3]

[Filename: RDF/infinite-parents.sparql]

EXERCISE

Write RDF/OWL instances:

- John has two children in school, they are in the 3rd and 5th year. Children in the first year are 6 years old, those in the 2nd year are 7 years old, and so on. There are 12 years in school.
- Alice is a daughter of John. She is 8 years old.
- an “ideal family” consists of a father, a mother, and they have 2 children, a son and a daughter, and a dog.
- John’s family is an “ideal family”.
- Bob is John’s son.

Feed them into the Jena tool, activate the reasoner.

- How old is Bob?
- which of the above information can be omitted without losing information how old Bob is?

Chapter 8

Conclusion and Outlook

What should have been learnt:

- Formal Logic: interpretations, model theory, first-order logic
- Deductive systems: Datalog, minimal model semantics
- reasoning: tableau calculi
- RDF as a special, simple data model; URIs
representations: N3 and RDF/XML
- DL as another logic, Open World
- “database” vs. “knowledge base”
- OWL as “DL alive”

SEMANTIC WEB DATA: XML; RDF AND OWL

In contrast to XPath/XQuery, XSLT, XML Schema, XLink etc., RDF and OWL are *not* languages “*inside*” the XML world, but are concepts of their own that have - incidentally- also an XML syntax.

The combination of XML data and RDF/RDFS/OWL concepts is the base for the *Semantic Web*.

A Semantic Web application e.g. exists of

- a “central” portal that uses the following things:
- a set of ontological (OWL, RDFS) sources,
- a set of RDF sources,
- reasoning (using OWL and RDFS information),
- a semantical description of itself for allowing others to use it.

DL + (DEDUCTIVE) RULES

- Carin: DL + Horn Rules [Levy+Rousset 1996]
- \mathcal{AL} -log: Datalog with Description Logics [Donini+Lenzerini 1998]
- Semantic Web Rule Language (SWRL): OWL+RuleML [Horrocks+Patel-Schneider etc. 2004]
- DL+log [Rosati 2005]
- Closed World vs. Open World, Safety, Decidability, ...

SEMANTIC WEB SERVICES

- Ontologies for describing Web Services
(lifting the WSDL, UDDI stuff to a semantic level)
- different current proposals
OWL-S, WSMO (Web Services Modeling Ontology)
- semantic matchmaking between tasks and services

OTHER ISSUES

- trust, recommender systems, personalization
“Web 2.0”: semantic wikis, semantic blogs
- dynamics [DBIS: REVERSE I5]
- policies
- verification

APPLICATIONS

- Bioinformatics