DATATYPES: ... DECIMAL DOES NOT YET WORK

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix : <foo:bla#>.
:value a owl:FunctionalProperty; a owl:DatatypeProperty;
       rdfs:range xsd:decimal.
## OK: then it complains about more than one value:
#:foo :value "2"^^xsd:decimal; :value "1"^^xsd:decimal.
#next two cases: does not complain and returns three values:
:foo :value "2"^^xsd:decimal; :value "1.0"^^xsd:decimal.
:foo :value "2"^^xsd:decimal; :value "2.3"^^xsd:decimal.
## only when adding this it complains about more than one value:
#:foo :value "2.3E13"^^xsd:decimal.
```

[Filename: RDF/decimal.n3]

```
prefix : <foo:bla#>
select ?X ?Y
from <file:decimal.n3>
where {?X :value ?Y}
```

[Filename: RDF/decimal.sparql]

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: owl:disjointUnionOf and negative Assertions objectPropertyAssertion vs. negativeObjectPropertyAssertion.
- User-defined datatypes (like XML Schema simple types).
- $\mathcal{SROIQ}$ 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.
- $\mathcal{SROIQ}(D)$ 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/sroiq-tr.pdf`.

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>
```

[Filename: RDF/disjointunion.xml]

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

[RDF/disjointunion.sparql]

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 a owl:Class; owl:equivalentClass
 [a owl:Restriction; owl:onProperty :hasAnimal; owl:minCardinality 2].
:HasTwoCats a owl:Class; owl:equivalentClass
 [a owl:Restriction; owl:onProperty :hasAnimal; owl:minCardinality 2; owl11:onClass :Cat].
:HasTwoDogs a owl:Class; owl:equivalentClass
 [a owl:Restriction; owl:onProperty :hasAnimal; owl:minCardinality 2; owl11:onClass :Dog].
```

[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 a owl:Class; owl:equivalentClass
 [a owl:Restriction; owl:onProperty :hasAnimal; owl:minCardinality 1].
:HasCat a owl:Class; owl:equivalentClass
 [a owl:Restriction; owl:onProperty :hasAnimal; owl:minCardinality 1; owl11:onClass :Cat].
:HasDog a owl:Class; owl:equivalentClass
 [a owl:Restriction; owl:onProperty :hasAnimal; owl:minCardinality 1; owl11:onClass :Dog].
```

[Filename: RDF/hasanimals.n3]

- export class tree:
  hasCat and hasDog are (non-disjoint) subclasses of hasAnimal.
- "owl:minCardinality 1 & owl11:onClass X" 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 dataypes 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>
```

[RDF/coordinates.xsd]

... and now use the datatypes ...

```
<!DOCTYPE rdf:RDF [
  <!ENTITY mon "http://www.semwebtech.de/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.de/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"/>
 <!-- ****** USE THEM IN A RESTRICTION ************************************ -->
 <owl:Restriction rdf:about="&mon;EasternHemisphere">
  <owl:onProperty rdf:resource="&mon;longitude"/>
  <owl:someValuesFrom rdf:resource="&Coords;#easternLongitude"/>
 </owl:Restriction>

 <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>
```

[RDF/coordinates.rdf]

... and now to the query:

```
prefix : <http://www.semwebtech.de/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 PELLET
- 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:allValuesFrom :atLeast18]).
:Child owl:intersectionOf (:Person
  [ owl:complementOf :Adult ]).
```

[Filename: RDF/adult.n3]

```
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.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:allValuesFrom :atLeast18]).
:HasTwoAdultChildren a owl:Restriction; owl:onProperty :child;
  owl:minCardinality 2; owl11:onClass :Adult.
```

[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]

DataRange 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.de/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.
```

[Filename: RDF/coordinates2.n3]

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

[Filename: RDF/coordinates2.sparql]

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.

```
@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.de/mondial/10/meta#>.
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.
@prefix : <foo://bla/>.
mon:population a owl:FunctionalProperty. ## all cities are different.
mon:population rdfs:range xsd:int.
: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;
     owl:minCardinality 2; owl11:onClass :BigCity]).
[owl:intersectionOf (mon:Country ## with 2 big cities, no provinces ## europe: empty
  [a owl:Restriction; owl:onProperty mon:hasCity;
      owl:minCardinality 2; owl11:onClass :BigCity]);
  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;
     owl:minCardinality 2; owl11:onClass :ProvinceWithBigCity]);
  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/inhabitants.n3]

Example: Cont'd

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

[Filename: RDF/inhabitants.sparql]

ENUMERATED DATATYPES [OWL:DATAONEOF NOT SUPPORTED]

```
<!DOCTYPE rdf:RDF [
  <!ENTITY mon "http://www.semwebtech.de/mondial/10/meta#">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">]>
<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.de/mondial/10/meta#">

  <owl:DataOneOf rdf:about="#Grades">
   <owl:Constant>2.0</owl:Constant>
   <owl:Constant>2.3</owl:Constant>
   <owl:Constant>2.7</owl:Constant>
   <owl:Constant>3.0</owl:Constant>
  </owl:DataOneOf>
</rdf:RDF>
```

[Filename: RDF/grades-enum.rdf]

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

[Filename: RDF/grades-strings.sparql]

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

ENUMERATED DATATYPES [OWL:ENUMERATION NOT SUPPORTED]

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

[Filename: RDF/grades-strings.n3]

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

[Filename: RDF/grades-strings.sparql]

- changing the grade to 2.5 results in an inconsistency.

- note: “3” and “3.0” are different strings.

ENUMERATION – NOT SUPPORTED

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

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

[RDF/names.sparql]

[Filename: RDF/names.n3]

- complains about use of literals as individuals.
- if the name “Mary” is not used in the KB, it complains even more.

7.4 OWL 1.1: Properties

- $\mathcal{SHIQ}$/OWL-DL concentrate on concept definitions,
- $\mathcal{SHOIQ}/\mathcal{SHOIQ}(D)$ add Nominals and datatypes,
- The $\mathcal{H}$ allows for a hierarchy of properties as already provided by RDFS, the $\mathcal{I}$ allows for inverse.
- $\mathcal{SROIQ}$ 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

- 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 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

- 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].
owl:AllDifferent owl:distinctMembers (:a :b :c).
: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 [Not Supported]

just a simple 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 : <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:disjointObjectProperty :hasDog. ########
```

[Filename: RDF/disjointproperties.n3]

```
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.sparql]

EXAMPLE: WIN-MOVE-GAME IN OWL

[Filename: RDF/winmove-graph1.rdf]

```
<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:bla="foo://bla/"
   xmlns="foo://bla/"
   xml:base="foo://bla/">
<owl:AllDifferent>
 <owl:distinctMembers rdf:parseType="Collection">
  <Node bla:out="2" rdf:about="a">
   <edge rdf:resource="b"/>
   <edge rdf:resource="f"/>
  </Node>
  <Node bla:out="3" rdf:about="b">
   <edge rdf:resource="c"/>
   <edge rdf:resource="g"/>
   <edge rdf:resource="k"/>
  </Node>
  <Node bla:out="2" rdf:about="c">
   <edge rdf:resource="d"/>
   <edge rdf:resource="l"/>
  </Node>
  <Node bla:out="1" rdf:about="d">
   <edge rdf:resource="e"/>
  </Node>
  <Node bla:out="1" rdf:about="e">
   <edge rdf:resource="a"/>
  </Node>
  <Node bla:out="0" rdf:about="f"/>
  <Node bla:out="2" rdf:about="g">
   <edge rdf:resource="i"/>
   <edge rdf:resource="h"/>
  </Node>
  <Node bla:out="1" rdf:about="h">
   <edge rdf:resource="m"/> <!-- tests: m,j,i,g -->
  </Node>
  <Node bla:out="1" rdf:about="i">
   <edge rdf:resource="j"/>
  </Node>
  <Node bla:out="0" rdf:about="j"/>
  <Node bla:out="0" rdf:about="k"/>
  <Node bla:out="1" rdf:about="l">
   <edge rdf:resource="d"/>
  </Node>
  <Node bla:out="0" rdf:about="m"/>
 </owl:distinctMembers>
</owl:AllDifferent>
</rdf:RDF>
```

[Filename: RDF/winmove-axioms.rdf]

```
<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/"
   xml:base ="foo://bla/">

<owl:Class rdf:about="WinNode">
 <owl:equivalentClass>
  <owl:Restriction>
   <owl:onProperty rdf:resource="edge"/>
   <owl:someValuesFrom rdf:resource="LoseNode"/>
  </owl:Restriction>
 </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:about="LoseNode">
 <owl:equivalentClass>
  <owl:Restriction>
   <owl:onProperty rdf:resource="edge"/>
   <owl:allValuesFrom rdf:resource="WinNode"/>
  </owl:Restriction>
 </owl:equivalentClass>
</owl:Class>

<owl:Class rdf:about="DeadEndNode">
 <rdfs:subClassOf rdf:resource="Node"/>
 <owl:equivalentClass>
  <owl:Restriction>
   <owl:onProperty rdf:resource="out"/>
   <owl:hasValue>0</owl:hasValue>
  </owl:Restriction>
 </owl:equivalentClass>
 <owl:equivalentClass>
  <owl:Restriction>
   <owl:onProperty rdf:resource="edge"/>
   <owl:cardinality>0</owl:cardinality>
  </owl:Restriction>
 </owl:equivalentClass>
 <!-- <rdfs:subClassOf rdf:resource="LoseNode"/>
        redundant, since implicit in the def of LoseNode-->
</owl:Class>

<owl:Class rdf:about="OneExitNode">
 <rdfs:subClassOf rdf:resource="Node"/>
 <owl:equivalentClass>
  <owl:Restriction>
   <owl:onProperty rdf:resource="out"/>
   <owl:hasValue>1</owl:hasValue>
  </owl:Restriction>
 </owl:equivalentClass>
 <owl:equivalentClass>
  <owl:Restriction>
   <owl:onProperty rdf:resource="edge"/>
   <owl:cardinality>1</owl:cardinality>
  </owl:Restriction>
 </owl:equivalentClass>
</owl:Class>

<owl:Class rdf:about="TwoExitsNode">
 <rdfs:subClassOf rdf:resource="Node"/>
 <owl:equivalentClass>
  <owl:Restriction>
   <owl:onProperty rdf:resource="out"/>
   <owl:hasValue>2</owl:hasValue>
```

```
prefix : <foo://bla/>
select ?W ?L ?D
from <file:winmove-graph1.rdf>
from <file:winmove-axioms.rdf>
where {{?W a :WinNode} UNION
       {?L a :LoseNode} UNION
       {?D a :DrawNode}}
```

[Filename: RDF/winmove.sparql]

```
prefix : <foo://bla/>
select ?W ?L ?X
from <file:winmove-graph1.rdf>
from <file:winmove-axioms.rdf>
where {{?W a :WinNode} UNION
       {?L a :LoseNode} UNION
       {:e :edge ?X}}
```

[Filename: RDF/winmove1.sparql]

```
prefix : <foo://bla/>
select ?DE ?W ?L ?D ?X # ?One
from <file:winmove-graph1.rdf>
from <file:winmove-axioms.rdf>
where {{?DE a :DeadEndNode} UNION
#        {?One a :OneExitNode} UNION
       {?W a :WinNode} UNION
       {?L a :LoseNode} UNION
       {?D a :DrawNode} UNION
       {:e :edge ?X}}
```

[Filename: RDF/winmove2.sparql]

FURTHER FEATURES OF OWL 1.1

(where the OWL syntax is not yet implemented, or I don't know it)

- Negated Assertions: negativeObjectPropertyAssertion(:x :rel :y) asserts that (:x :rel :y) does not hold.
- cross-property restrictions/role-value maps:
  - 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 (knows $\sqsubseteq$ 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 (shoeSize$>$IQ)).
- 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.

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:
  $KB \models C \equiv \bot \;\Leftrightarrow\; KB \cup \{C(a)\}$ for a new constant a is unsatisfiable.
- concept containment:
  $KB \models C \sqsubseteq D \;\Leftrightarrow\; KB \models (C \sqcap \neg D) \equiv \bot$.
- 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 $R(s,x_1),\ldots,R(s,x_n),C(x_1),\ldots,C(x_n)$ 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 :age ?Y}", but pruning the search space by "{?X 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).

¬ hasChild(john,_X)

$\square\{X_1 \leftarrow$ alice$\}$

$\square\{X_2 \leftarrow$ 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).

¬(hasChild(john,X) ∧ name(X,N))

¬(hasChild(john,X)) | ¬name(X,N)

Try $\square\{X_1 \leftarrow$ alice$\}$ | for X_1 and X_2:

Try $\square\{X_2 \leftarrow$ bob$\}$ | X_1 | X_2

¬ name(alice,N) | ¬ name(bob,N)

$N_1 \leftarrow$"Alice" | $N_2 \leftarrow$"Bob"

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

DL Tableaux: Examples

Consider the "Only female children" example from Slide 333.

TwoChildrenParent(sue)
child(sue,ann)
Female(ann)
child(sue,barbara)
Female(barbara)
ann $\neq$ barbara
TwoChildrenParent $\sqsubseteq \exists 2$ child.$\top$
OnlyFemaleChildrenParent $\sqsubseteq$ Person $\sqcap$ $\forall$child.Female

Query: ?- OnlyFemaleChildrenParent(X).

[$\neg$ OnlyFemaleChildrenParent(X)]
$\neg$(Person $\sqcap$ $\forall$child.Female(X))

$\neg$ Person(X) | $\neg$ $\forall$child.Female(sue)
try $\Box\{X \leftarrow$ sue$\}$ | $\exists$ child.($\neg$ Female)(sue)

child(sue,y)
$\neg$ Female(y)
count Sue's children=3: ann,barbara,y

ann=barbara | ann=y | barbara=y
$\Box$ | $\Box$ $\neg$ Female(ann) | $\Box$ $\neg$ 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 360).

(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)

Query: ?- higher(X,Y).

[$\neg$ higher(X,Y)]

from (2) for a:

$\neg$ Corner(a) — $\square$

(8) $\exists 1$ higher.$\top$(a)

from (8): (9) higher(a,d)

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

(10) 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 concistency, 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)

[$\neg$ higher(X,Y)]

from (2) for a:

$\neg$ Corner(a) (8) $\exists 1$ higher.$\top$(a)

$\Box$ from (8): (9) higher(a,d)

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

(10) higher(a,b)

from (2) for b:

$\neg$ Corner(b) (11) $\exists 1$ higher.$\top$(b)

$\Box$ 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)

$\Box$ (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: Corner $\sqsubseteq \exists 1$ higher.$\top$
(4) range: $\top \sqsubseteq \forall$ higher.Corner
(5) AntiSymmetric(higher)
(6) Irreflexive(higher)
(7) higher(a,b)

[$\neg$ higher(X,Y)]

(13) higher(b,e)

(14) Corner(e)

e must be one of a,b,c

e=a — higher(b,a) — $\square$ (Antisymm.)

e=b — higher(b,b) — $\square$ (Irrefl.)

e=c — (15) higher(b,c) — Answer: (b,c)

continue with (2) for c

$\neg$ Corner(c) — $\square$

(16) $\exists 1$ higher.$\top$(b)

from (16): (17) higher(c,f)

next: range of “higher” derives that f is a corner:

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

$\neg\top$(c) — $\square$

(18) $\forall$ 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)

[$\neg$ higher(X,Y)]

⋮

(15) higher(b,c)
(17) higher(c,f)
(19) Corner(f)
f must be one of a,b,c

f=a	f=b	f=c
higher(c,a)	higher(c,b)	higher(c,c)
Answer: (c,a)	□ (Antisymm.)	□ (Irrefl.)
[1]		

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 concident 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:i
:Person owl:subclassOf
 [a owl:Restriction; owl:onProper
:Parent owl:equivalentClass
 [a owl:Restriction; owl:onProper
:Grandparent owl:equivalentClass
 [a owl:Restriction; owl:onProper
```

[Filename: RDF/infinite-parents.n3]

```
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.sparql]

EXERCISE

Write RDF/OWL instances:

- John has two children in school, they are in the 3nd 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 daugther 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: REWERSE I5]
- policies
- verification

Applications

- Bioinformatics