7.3 OWL 2 (W3C Recommendation since October 2009)

- OWL2 notions belong to the OWL namespace
  (aside: development proposal owl11 used a separate namespace)
- 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., SubPropertyOf(PropertyChain(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:f="foo://bla/"
   xml:base="foo://bla/">
 <owl:Class rdf:about="Person">
  <owl:disjointUnionOf rdf:parseType="Collection">
   <owl:Class rdf:about="Male"/>
   <owl:Class rdf:about="Female"/>
  </owl:disjointUnionOf>
 </owl:Class>
 <f:Male rdf:about="John"/>
 <f:Female rdf:about="Mary"/>
 <!--<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}
```

[Filename: RDF/disjointunion.sparql]

Example: Parricides in Greek Mythology

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

A parricide is a person who killed his/her father.

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

[Filename: RDF/parricide.sparql]

- No X reported.

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 : <foo:greek#>.
:zeus :knows :iokaste, :oedipus, :perineikes, :thesandros.
:KnowsPoPGonP  owl:equivalentClass [ a owl:Restriction; owl:onProperty :knows;
  owl:someValuesFrom :Parent-of-Parricide-Grandparent-of-Non-Parricide ].
```

[Filename: RDF/parricide2.n3]

```
prefix : <foo:greek#>
select ?K ?X
from <file:parricide.n3>
from <file:parricide2.n3>
where {{?K a :KnowsPoPGonP} UNION
       {?X a :Parent-of-Parricide-Grandparent-of-Non-Parricide}}
```

[Filename: RDF/parricide2.sparql]

- Zeus is in K, 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 Iokaste or Oedipus (depending on whether Perineikes is a parricide, which nobody knows).

QUALIFIED ROLE RESTRICTIONS

- extends owl:Restriction, owl:onProperty, owl:{min/max}QualifiedCardinality (int value) with owl:on{Class/DataRange} as result class/type.

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/names#> .
:alice :name "Alice"; :hasAnimal :pluto, :struppi.
:john :name "John"; :hasAnimal :garfield, :odie.
:pluto a :Dog; :name "Pluto".
:struppi a :Dog; :name "Struppi".
:garfield a :Cat; :name "Garfield".
:odie a :Dog; :name "Odie".
: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; owl:onClass :Cat; owl:minQualifiedCardinality 2].
:HasTwoDogs owl:equivalentClass
 [a owl:Restriction; owl:onProperty :hasAnimal; owl:onClass :Dog; owl:minQualifiedCardinality 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
       {?Z a :HasTwoAnimals} UNION
       {?C rdfs:subClassOf :HasTwoAnimals}}
```

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

Qualified Role Restrictions – Another Test

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@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; owl:onClass :Cat; owl:minQualifiedCardinality 1].
:HasDog owl:equivalentClass
 [a owl:Restriction; owl:onProperty :hasAnimal; owl:someValuesFrom :Dog].
```

[Filename: RDF/hasanimals.n3]

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

Negative Assertions

- Assert that something is known *not* to hold:
  NegativeObjectPropertyAssertion and NegativeDataPropertyAssertion
- with owl:sourceIndividual, owl:assertionProperty, and owl:targetIndividual or owl:targetValue.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/names#> .
@prefix person: <foo://bla/persons/> .
person:john a :Person.
[ rdf:type owl:NegativePropertyAssertion;
  owl:sourceIndividual person:john;
  owl:assertionProperty :lives;
  owl:targetIndividual :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 owl: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(lives,germany) = :German)
- its complement $CompR(r, y) := \top \setminus R(r, y)$
  (above: CompR(lives,germany) = :NonGerman)
- and the assertion that $x \in CompR(r, y)$.
  (above: assert (:john a :NonGerman))

DATATYPES: HASVALUE WITH LITERAL VALUE

Characterize a class as the set of all things where a given property has a given value:

- all things in Mondial that have the name “Berlin”:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix mon: <http://www.semwebtech.org/mondial/10/meta#>.
@prefix : <foo:bla#>.
:Berlin owl:equivalentClass [ a owl:Restriction;
   owl:onProperty mon:name; owl:hasValue "Berlin"].
```

[Filename: RDF/has-literal-value.n3]

```
prefix : <foo:bla#>
select ?X
from <file:has-literal-value.n3>
from <file:mondial-europe.n3>
where {?X a :Berlin}
```

[Filename: RDF/has-literal-value.sparql]

- Often preferable: define an owl:DataRange (unary or enumeration), give it a url, and use some/allValuesFrom.

Enumerated Datatypes

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix uni: <foo://uni/>.
uni:graded a owl:FunctionalProperty;
 a owl:DatatypeProperty; rdfs:range uni:Grades.
uni:Grades a rdfs:Datatype;
 owl:equivalentClass [ a rdfs:Datatype;
  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 "2.5"].
```

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

- inconsistent: “2.5” does not belong to the allowed grades,
- note: “3” is also not allowed since “3” and “3.0” are different strings,
- see alternative next slide.

Enumerated Datatypes

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix uni: <foo://uni/>.
uni:graded a owl:FunctionalProperty;
 a owl:DatatypeProperty; rdfs:range [ a rdfs:Datatype;
  owl:oneOf (1 1.3 1.7 2.0 2.3 2.7 3 3.3 3.7 4) ] .
[ a uni:Thesis; uni:author <foo://bla/john>;
  uni:graded 2].
```

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

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

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

- grade 2.5 results in an inconsistency,
- internally (in case of an error message e.g.), the values are represented/handled as “2.3”^^xsd:decimal,
- parsing and output uses the default representation,
- both representations 2 and 2.0 are allowed.

OneOf on DataRange

```
@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#>.
:Male a owl:Class.    :Female a owl:Class.
:Person owl:disjointUnionOf (:Male :Female).
:MaleNames a rdfs:Datatype; owl:equivalentClass [ a rdfs:Datatype;
     owl:oneOf ("John"^^xsd:string "Bob"^^xsd:string) ] .
:FemaleNames a rdfs:Datatype; owl:equivalentClass [ a rdfs:Datatype;
    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])].
: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-
@prefix owl: <http://www.w3.org/2002/07/owl#>
@prefix : <foo://bla/names#>.
:Person owl:disjointUnionOf (:Male :Female).
:Male a :Sex.
:Female a :Sex.
:MaleNames owl:equivalentClass [ a rdfs:Datatype; owl:oneOf ("John" "Bob") ] .
:FemaleNames owl:equivalentClass [ a rdfs:Datatype; owl:oneOf ("Mary" "Alice") ].
: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])].
:name a owl:FunctionalProperty; a owl:DatatypeProperty.
:john a :Person; :name "John".
:mary a :Person; :name "Mary".
```

[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]

DATATYPES

- common built-ins from XML Schema: int, decimal, ..., date, time, datetime.
- "2"^^xsd:decimal is different from "2"^^xsd:int

```
@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:DatatypeProperty; rdfs:range xsd:decimal.
:foo :value "2"^^xsd:decimal; :value "1.0"^^xsd:decimal.
:foo :value "2.0"^^xsd:decimal; :value "2.3"^^xsd:decimal.
:foo :value "2"^^xsd:integer; :value "1"^^xsd:integer.
```

[Filename: RDF/decimal.n3]

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

[Filename: RDF/decimal.sparql]

- jena: returns 6 results: "2"^^xsd:decimal, 1.0, 2.0, 2.3, 1, 2
- pellet: returns 5 results: 1, 2, 2.3, 2.0, 1.0

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.

DATATYPES IN OWL

- use the XML Schema built-in types as resources (int and string must be supported; Pellet does also support decimal)
- rdfs:Datatype: cf. simple Types in XML schema; derived from the basic ones (e.g. xsd:int is an rdfs:Datatype)
- specified by
  - owl:onDatatype: from what datatype they are derived,
  - owl:withRestrictions is a list of restricting facets
  - facets as in XML Schema:
    xsd:{max/min}{In/Ex}clusive etc.
- similar to owl:Restrictions: define by
  myDatatypeName owl:equivalentClass [*datatypeSpec*].

DATA RANGES: ADULTS

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@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.
:atLeast18T owl:equivalentClass
  [a rdfs:Datatype; owl:onDatatype xsd:int; owl:withRestrictions ( _:x1 )].
_:x1 xsd:minInclusive 18 .
:Adult owl:intersectionOf (:Person
  [ a owl:Restriction;
    owl:onProperty :age;
    owl:someValuesFrom :atLeast18T]).
: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 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.
:atLeast18T owl:equivalentClass [a rdfs:Datatype;
  owl:onDatatype xsd:int; owl:withRestrictions ( [ xsd:minInclusive 18 ] ) ].
:Adult owl:intersectionOf (:Person
  [ a owl:Restriction; owl:onProperty :age; owl:someValuesFrom :atLeast18T]).
:HasTwoAdultChildren  owl:equivalentClass [ a owl:Restriction;
    owl:onProperty :child; owl: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]

DataRange Restriction for Geographical Coordinates

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@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/>.
:LongitudeT owl:equivalentClass [ a rdfs:Datatype; owl:onDatatype xsd:decimal;
   owl:withRestrictions ( [xsd:minExclusive -180] [xsd:maxInclusive 180] ) ] .
:LatitudeT owl:equivalentClass [ a rdfs:Datatype; owl:onDatatype xsd:decimal;
  owl:withRestrictions ( [ xsd:minInclusive -90] [xsd:maxInclusive 90] ) ] .
:EasternLongitudeT  owl:equivalentClass [a rdfs:Datatype;
  owl:onDatatype :LongitudeT; owl:withRestrictions ( [xsd:minInclusive 0] ) ] .
:EasternHemispherePlace owl:equivalentClass [a owl:Restriction;
  owl:onProperty mon:longitude; owl:someValuesFrom :EasternLongitudeT].
mon:longitude rdfs:range :LongitudeT.
mon:latitude rdfs:range :LatitudeT.
:Berlin a mon:City; :name "Berlin"; mon:longitude 13.3; mon:latitude 52.45 .
#:Atlantis a mon:City; :name "Atlantis"; mon:longitude -200; mon:latitude 100 .
:Lisbon a mon:City; :name "Lisbon"; mon:longitude -9.1; mon:latitude 38.7 .
```

[Filename: RDF/coordinates.n3]

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

[Filename: RDF/coordinates.sparql]

EXAMPLE: USING XSD DATATYPES

- [Does not work completely ...] Define simple datatypes in an XML Schema file:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="file:coordinates2.xsd">
 <xs:simpleType name="longitudeT">
  <xs:restriction base="xs:decimal">
   <xs:minExclusive value="-180"/>
   <xs:maxInclusive value="180"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="easternLongitude">
  <xs:restriction base="xs:decimal">
    <!-- note: base="longitudeT" would be nicer, but is not allowed when parsing from RDF -->
   <xs:minInclusive value="10"/>
   <xs:maxInclusive value="180"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="latitudeT">
  <xs:restriction base="xs:decimal">
   <xs:minInclusive value="-90"/>
   <xs:maxInclusive value="90"/>
  </xs:restriction>
 </xs:simpleType>
</xs:schema>
```

[Filename: RDF/coordinates2.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:coordinates2.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;EasternHemispherePlace">
 <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="&Coords;#longitudeT">13.3</mon:longitude>
  <mon:latitude rdf:datatype="&Coords;#latitudeT">52.45</mon:latitude> </mon:City>
 <mon:City mon:name="Lisbon">
  <mon:longitude rdf:datatype="&Coords;#longitudeT">-9.1</mon:longitude>
  <mon:latitude rdf:datatype="&Coords;#latitudeT">38.7</mon:latitude> </mon:City>
</rdf:RDF>
```

[Filename: RDF/coordinates2.rdf]

... and now to the query:

```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
select ?N
from <file:coordinates2.rdf>
where {?X :name ?N . ?X a :EasternHemispherePlace}
```

[Filename: RDF/coordinates2.sparql]

Comments

- 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!

Qualified Role Restrictions: Example

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

- define "more than a million" as a rdfs:Datatype
- 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 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 rdfs:Datatype; owl:onDatatype xsd:int; owl:withRestrictions (_:m1).
_:m1 xsd: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
  [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:onClass :BigCity; owl:minCardinality 2]).
[owl:intersectionOf (mon:Country ## with 2 big cities, no provinces ## europe: empty
  [a owl:Restriction; owl:onProperty mon:hasCity; owl: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; owl: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].
```

[Filename: RDF/bigcities.n3]

Example: Cont'd

```
@prefix mon: <http://www.semwebtech.org/mondial/10/meta#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/>.
:grmny a mon:Country; mon:hasCity :bln, :mch .
:bln a :BigCity; mon:population 3500000 .
:mch a :BigCity; mon:population 1500000 .
:frc a mon:Country; mon:hasProvince :ile, :prov .
:ile owl:differentFrom :prov.
:prs a mon:City; mon:cityIn :ile; mon:population 2000000 .
:mrs a mon:City; mon:cityIn :prov; mon:population 1500000
```

[Filename: RDF/dummy-cities.n3]

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

[Filename: RDF/bigcities.sparql]

7.4 [Aside] OWL vs. RDF Lists

- RDF provides structures for representing lists by triples (cf. Slide 244): rdf:List, rdf:first, rdf:rest.
  These are *distinguished* classes/properties.
- OWL/reasoners have a still unclear relationship with these:
  - use of lists for its internal representation of owl:unionOf, owl:oneOf etc. (that are actually based on collections),
  - do or do not allow the user to query this internal representation,
  - ignore user-defined lists over usual resources.

UnionOf (etc) as Triples: Lists

- owl:unionOf (x y z), owl:oneOf (x y z) is actually only syntactic sugar for RDF lists.
- The following are equivalent:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/>.

:Male a owl:Class.
:Female a owl:Class.

:Person a owl:Class; owl:unionOf (:Male :Female).
:EqToPerson a owl:Class;
  owl:unionOf
  [ a rdf:List; rdf:first :Male;
    rdf:rest [ a rdf:List; rdf:first :Female; rdf:rest rdf:nil]].
:x a :Person.
```

[Filename: RDF/union-list.n3]

- jena -t -if union-list.n3: both in usual N3 notation as owl:unionOf (:Male :Female).

UnionOf (etc) as Triples (Cont'd)

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <foo://bla/>
select ?C
from <file:union-list.n3>
where {:Person owl:equivalentClass ?C}
```

[Filename: RDF/union-list.sparql]

- jena -q -pellet -qf union-list.sparql: both are equivalent.

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <foo://bla/>
select ?P1 ?P2 ?X ?Q ?R ?S ?T
from <file:union-list.n3>
where {{:Person owl:equivalentClass :EqToPerson} UNION
  {:Person ?P1 ?X . ?X ?Q ?R . OPTIONAL {?R ?S ?T}} UNION
  {:EqToPerson ?P2 ?X . ?X ?Q ?R} . OPTIONAL {?R ?S ?T}}
```

[Filename: RDF/union-list2.sparql]

- both have actually the same list structure
  (pellet2/nov 2008: fails; pellet 2.3/sept 2009: fails)

Reasoning over Lists (Pitfalls!)

- rdf:first and rdf:rest are (partially) ignored for reasoning (at least by pellet?); they cannot be used for deriving other properties from it.
- they can even not be used in queries (since pellet2/nov 2008; before it just showed weird behavior)

```
prefix rdf:
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <foo://bla/>
select ?X ?Y ?Z
from <file:union-list.n3>
where {?X a rdf:List; rdf:first ?Y .
       OPTIONAL {?X rdf:rest ?Z}}
```

[Filename: RDF/union-list3.sparql]

- jena-tool with pellet2.3: OK.
- pellet2.3: NullPointerException.

Extension of a class defined by a list

Given an RDF list as below, define an owl:Class :Invited which contains exactly the elements in the list (i.e., in the above sample data, :alice, :bob, :carol, :dave).

```
@prefix : <foo:bla#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
# Problem: when the real rdf namespace is used, rdf:first/rest are ignored
@prefix rdfL: <http://www.w3.org/1999/02/22-rdf-syntax-nsL#>.    # <<<<<<<<<<<<<
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.

:Invited a owl:Class.
:InvitationList rdfs:subClassOf rdfL:List.
:list1 a :InvitationList; rdfL:first :alice;
  rdfL:rest [a rdfL:List; rdfL:first :bob;
    rdfL:rest [a rdfL:List; rdfL:first :carol;
      rdfL:rest [a rdfL:List; rdfL:first :dave; rdfL:rest rdf:nil]]].

# rest of an InvitationList is also an InvitationList
:InvitationList owl:equivalentClass
  [a owl:Restriction;
    owl:onProperty rdfL:rest; owl:allValuesFrom :InvitationList],
  [ a owl:Restriction;
    owl:onProperty rdfL:first; owl:allValuesFrom :Invited].
```

[Filename: RDF/invitation-list.n3]

```
prefix : <foo:bla#>
select ?I
from <file:invitation-list.n3>
where {?I a :Invited}
```

[Filename: RDF/invitation-list.sparql]

7.5 OWL 2: Properties

- $\mathcal{SHIQ}$/OWL-DL concentrate on *concept* definitions ($\mathcal{SQ}$ portion),
  - The $\mathcal{H}$ allows for a hierarchy of *properties* as already provided by RDFS, the $\mathcal{I}$ allows for inverse.
- $\mathcal{SHOIQ}/\mathcal{SHOIQ}(\mathcal{D})$ add nominals and datatypes (i.e., provide database-oriented functionality for handling *instances*),
- $\mathcal{SROIQ}$ provides more expressiveness around *properties*.

Transitive and Symmetric Properties

- transitive: descendants (cf. Slide 234), train connections etc.
- symmetric: married

```
@prefix : <foo://bla/names#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
  [ :name "John";  :married [ :name "Mary" ]] .
  :married rdf:type owl:SymmetricProperty.
```

[Filename: RDF/symmetric-married.n3]

```
prefix : <foo://bla/names#>
select ?X ?Y
from <file:symmetric-married.n3>
where { [ :name ?X ; :married [ :name ?Y]] }
```

[Filename: RDF/symmetric-married.sparql]

SYMMETRIC PROPERTIES

```
@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 2)

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/names#> .
:john a :Person; :knows :mary; :child :alice.
:knows a owl: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 : <foo://bla/names#> .
:john :hasAnimal :pluto, :garfield.
:pluto :bites :garfield.
# we exclude neurotic animals:
:bites a owl: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.

Asymmetry

- asymmetric(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 : <foo://bla/names#>.
[ a owl:AllDifferent; owl:members (:a :b)].
:rel a owl:AsymmetricProperty.
# :a :rel :b.
:b :rel :a.
```

[Filename: RDF/asymmetry.n3]

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

[Filename: RDF/asymmetry.sparql]

Irreflexive and Asymmetric 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 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 owl:AsymmetricProperty.
:higher a owl: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.

Disjoint Properties

- Syntax: (prop$_1$ owl:propertyDisjointWith prop$_2$)
- for more than 2 properties (similar to owl:AllDifferent):
  [a owl:AllDisjointProperties; owl:members (...)]

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

At the Decidability Border

Some combinations of advanced constructs in DL that are part of OWL 2 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 : <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;
    owl: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]

- use jena -e to export class/instance tree

Across the Decidability Border

- Cardinality restrictions on complex (e.g. transitive) properties are not allowed (undecidable) ⇒ rejected by the reasoner

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 : <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.    # << trans.Prop <<<

:City a owl:Class; owl:equivalentClass
 [a owl:Restriction; owl:onProperty :belongsTo;                       # << cardinality <<
    owl: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 2

- Role Chains/Property Chains: SubPropertyOf(PropertyChain(owns hasPart) owns) asserts that if x owns y and y has a part z, then x owns z.

  SubPropertyOf(PropertyChain(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/owl11-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

- uncle $\equiv$ brotherOf $\circ$ child
- if a list is a owl:subProperty of something, it is interpreted as a role chain definition.

```
@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/> .

[] rdfs:subPropertyOf :uncleOf;
  owl:propertyChain (:brotherOf :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".
```

[Filename: RDF/uncle.n3]

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

[Filename: RDF/uncle.sparql]

Syntax: Role Chains in XML/RDF

... as expected: a blank node that refers to 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">
 <rdf:Description>
   <rdfs:subPropertyOf rdf:resource="#uncleOf"/>
   <owl:propertyChain>
     <rdf:List>
       <rdf:rest rdf:parseType="Collection">
         <owl:ObjectProperty rdf:about="#child"/>
       </rdf:rest>
       <rdf:first rdf:resource="#brotherOf"/>
     </rdf:List>
   </owl:propertyChain>
 </rdf:Description>
 <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>
```

[Filename: RDF/uncle.rdf]

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

[Filename: RDF/uncle2.sparql]

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 !!!!!!!!!!
[ ] rdfs:subPropertyOf :uncleOf;
    owl:propertyChain (:brotherOf :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.
  WARNING - Unsupported axiom: Ignoring transitivity and/or complex subproperty axioms for uncleOf

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

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix : <foo://bla/>.
:Cyclic a owl:Class;
  owl:equivalentClass [ owl:intersectionOf
    (:Node [a owl:Restriction; owl:onProperty :to;
            owl:hasSelf "true"^^xsd:boolean ])].
:b a :Cyclic.
:a a :Node; :to :a, :b.
# :a a [ owl:complementOf :Cyclic ].
```

[Filename: RDF/cyclic.n3]

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

[Filename: RDF/cyclic.sparql]

Self restrictions (Cont'd)

... just another example:

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix : <foo://bla/>.
:NeuroticAnimal a owl:Class;
  owl:equivalentClass [ owl:intersectionOf
   ( :Animal
     [a owl:Restriction; owl:onProperty :bites; owl:hasSelf "true"^^xsd:boolean])].
:pluto a :Animal; :bites :pluto, :garfield.
:garfield a :NeuroticAnimal.
```

[Filename: RDF/neurotic.n3]

```
prefix : <foo://bla/>
select ?N ?N2
from <file:neurotic.n3>
where {{?N a :NeuroticAnimal} UNION
       {?N :bites ?N2}}
```

[Filename: RDF/neurotic.sparql]

7.6 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)
□{X_1 ← alice}
□{X_2 ← 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)) — Try □{X_1 ← alice} — Try □{X_2 ← bob}

¬name(X,N) — for X_1 and X_2:

X_1: ¬ name(alice,N) — N_1 ←"Alice"

X_2: ¬ name(bob,N) — N_2 ←"Bob"

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

DL Tableaux: Examples

Consider the "Only female children" example from Slide 369.

TwoChildrenParent(sue)
child(sue,ann)
Female(ann)
child(sue,barbara)
Female(barbara)
ann ≠ barbara
TwoChildrenParent ⊑ ∃2 child.⊤
OnlyFemaleChildrenParent ⊑ Person ⊓ ∀child.Female

Query: ?- OnlyFemaleChildrenParent(X).

[¬ OnlyFemaleChildrenParent(X)]
¬(Person ⊓ ∀child.Female(X))

¬ Person(X) — try □{X ← sue}

¬ ∀child.Female(sue)
∃ child.(¬ Female)(sue)
child(sue,y)
¬ Female(y)
count Sue's children=3: ann,barbara,y

ann=barbara — □
ann=y — □ ¬ Female(ann)
barbara=y — □ ¬ 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 412).

(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).

[¬ higher(X,Y)]

from (2) for a:

¬ Corner(a) | (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)

[¬ higher(X,Y)]

from (2) for a:

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

□

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

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

(10) higher(a,b)

from (2) for b:

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

□

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

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

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

¬$\top$(b) | (13) $\forall$ higher.Corner(b)

□

(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 | e=b | e=c

higher(b,a) | higher(b,b) | (15) higher(b,c)

□ (Antisymm.) | □ (Irrefl.) | Answer: (b,c)

continue with (2) for c

$\neg$ Corner(c) | (16) $\exists 1$ higher.$\top$(c)

□ | 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) | (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:inverseOf :parent.
:Person rdfs:subClassOf
 [a owl:Restriction;
    owl:onProperty :parent;
    owl:cardinality 2].
: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 owl: <http://www.w3.org/2002/07/owl#>
prefix : <foo://bla/names#>
select ?A ?B ?C ?X
from <file:infinite-parents.n3>
where {{?A a :Parent} UNION
       {?B a :Grandparent} UNION
       {:parent rdfs:range ?C} 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?

7.7 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 364).

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/names#>.
:sue :child :barbara; :sibling :john.
:john :name "John"; :child :alice, :bob;  :sibling :sue.
:x a swrl:Variable.
:y a swrl:Variable.
:z a swrl:Variable.
 :uncleAuntRule a swrl:Imp;
   swrl:head ([ a swrl:IndividualPropertyAtom;
                swrl:propertyPredicate :uncleAunt;
                swrl:argument1 :y ;   swrl:argument2 :z ]);
   swrl:body ([ a swrl:IndividualPropertyAtom;
                swrl:propertyPredicate :child;
                swrl:argument1 :x ;   swrl:argument2 :y ]
              [ a swrl:IndividualPropertyAtom;
                swrl:propertyPredicate :sibling;
                swrl:argument1 :x ;   swrl:argument2 :z ]).
```

[Filename: RDF/uncle-rule.n3]

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

[Filename: RDF/uncle-rule.sparql]

DL-Safe Rules consider only named Resources

- analogous to SPARQL queries and owl:hasKey (cf. Slide 361)

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

⇒ does not interfere with the blocking, and

⇒ does not break decidability.


```
@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/names#>.
:john :child :bob; :sibling :paul, [].
:sibling a owl:SymmetricProperty.
:paul a [a owl:Restriction; owl:onProperty :child; 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 :child;
                swrl:argument1 :x ; swrl:argument2 :y ]
              [ a swrl:IndividualPropertyAtom; swrl:propertyPredicate :sibling;
                swrl:argument1 :x ; swrl:argument2 :z ]).
   :x a swrl:Variable.    :y a swrl:Variable.   :z a swrl:Variable.
[] rdfs:subPropertyOf :uncle2;
   owl:propertyChain ([owl:inverseOf :child] :sibling).
:Uncle owl:equivalentClass [a owl:Restriction; owl:onProperty :sibling;
  owl:someValuesFrom [a owl:Restriction; owl:onProperty :child;
                      owl:minCardinality 1]].
```

[Filename: RDF/uncle-comparison.n3]

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

```
prefix : <foo://bla/names#>
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"; :child :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 :sibling;
                swrl:argument1 :y ; swrl:argument2 :z ]);
   swrl:body ([ a swrl:IndividualPropertyAtom; swrl:propertyPredicate :child;
                swrl:argument1 :x ; swrl:argument2 :y ]
              [ a swrl:IndividualPropertyAtom; swrl:propertyPredicate :child;
                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 :child ?CH} union {?X :sibling ?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.

⇒ 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 N3 fragment next slide)

- Rule: all persons believe in God,
- jack has a blank node child :b who is a parent,
- the child 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 makes :b a ParentOfBeliever.
- how to show that the grandchild is not considered by the rule: add a statement that :b is not parent of a believer.
- run “classify” for the n3:
  - the ontology is consistent,
  - :b is accepted to be a :NotParentOfBeliever.


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

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

[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).

:x a swrl:Variable.   :y a swrl:Variable.    :z a swrl:Variable.
:PWBigCityRule a swrl:Imp;
   swrl:head ([ a swrl:ClassAtom; swrl:classPredicate :ProvinceWithBigCity;
                swrl:argument1 :x]);
```


[Filename: RDF/bigcities-rule.n3]

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

[Filename: RDF/bigcities-rule.sparql]

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: 2004-2008 REWERSE Eu NoE Working Group I5,
  continued with the MARS (Modular Active Rules in the Semantic Web) and SWAN (Semantic Web Application Node) projects
- policies
- verification

Application Areas

- Bioinformatics