7.2 OWL

- the OWL versions use certain DL semantics:
- Base: $\mathcal{ALC}_{\mathcal{R}^+}$: (i.e., with transitive roles). This logic is called $\mathcal{S}$ (reminiscent to its similarity to the modal logic S).
- roles can be ordered hierarchically (rdfs:subProperty; $\mathcal{H}$).
- OWL Lite: $\mathcal{SHIF}(D)$, Reasoning in ExpTime.
- OWL DL: $\mathcal{SHOIN}(D)$, decidable.
  Pellet implements $\mathcal{SHOIQ}(D)$. Decidability is in NExpTime (combined complexity wrt. TBox+ABox), but the actual complexity of a given task is constrained by the maximal used cardinality and use of nominals and inverses and behaves like the simpler classes.

  (Ian Horrocks and Ulrike Sattler: A Tableau Decision Procedure for SHOIQ(D); In IJCAI, 2005, pp. 448-453; available via `http://dblp.uni-trier.de`)

OWL NOTIONS

- Classes and Properties are nodes in an RDF model, their characteristics are specified by OWL properties.

OWL Class Definitions and Axioms (Overview)

- owl:Class
- The properties of an owl:Class (including owl:Restriction) node describe the properties of that class.
  An owl:Class is required to satisfy the conjunction of all constraints (implicit: intersection) stated by its subelements.
  These characterizations are roughly the same as discussed for DL class definitions:
  - Constructors: owl:unionOf, owl:intersectionOf, owl:complementOf ($\mathcal{ALC}$)
  - Enumeration Constructor: owl:oneOf (enumeration of elements; $\mathcal{O}$)
  - Axioms rdfs:subClassOf, owl:equivalentClass,
  - Axiom owl:disjointWith (also expressible in $\mathcal{ALC}$: C disjoint with D is equivalent to $C \sqsubseteq \neg D$)

OWL NOTIONS (CONT'D)

OWL Restriction Classes (Overview)

- owl:Restriction is a subclass of owl:Class, allowing for specification of a constraint on *one* property.
- one property is restricted by an owl:onProperty specifier and a constraint on this property:
  - ($\mathcal{N}$, $\mathcal{Q}$, $\mathcal{F}$) owl:cardinality, owl:minCardinality or owl:maxCardinality,
  - owl:allValuesFrom ($\forall R.C$), owl:someValuesFrom ($\exists R.C$),
  - owl:hasValue ($\mathcal{O}$),
  - including datatype restrictions for the range (D)
- by defining an owl:Restriction as subclass of another owl:Restriction, multiple such constraints can be defined.

OWL NOTIONS (CONT'D)

OWL Property Axioms (Overview)

- atomic constructor: rdf:Property
- from RDFS: rdfs:domain/rdfs:range assertions, rdfs:subPropertyOf
- Axiom owl:equivalentProperty
- Axioms: subclasses of rdf:Property:
  owl:TransitiveProperty, owl:SymmetricProperty, owl:FunctionalProperty,
  owl:InverseFunctionalProperty (see Slide 230)

OWL Individual Axioms (Overview)

- Individuals are modeled by unary classes
- owl:sameAs, owl:differentFrom, owl:AllDifferent(o$_1$,. . . ,o$_n$).

First-Order Logic Equivalents

OWL : $x \in C$	DL Syntax	FOL
C	C	$C(x)$
intersectionOf(C_1, C_2)	$C_1 \sqcap \ldots \sqcap C_n$	$C_1(x) \wedge \ldots \wedge C_n(x)$
unionOf(C_1, C_2)	$C_1 \sqcup \ldots \sqcup C_n$	$C_1(x) \vee \ldots \vee C_n(x)$
complementOf(C_1)	$\neg C_1$	$\neg C_1(x)$
oneOf$(x_1, \ldots, x_n)$	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	$x = x_1 \vee \ldots \vee x = x_n$

OWL : $x \in C$, Restriction on P	DL Syntax	FOL
someValuesFrom(C')	$\exists P.C'$	$\exists y : P(x,y) \wedge C'(y)$
allValuesFrom(C')	$\forall P.C'$	$\forall y : P(x,y) \rightarrow C'(y)$
hasValue(y)	$\exists P.\{y\} \sqcap \forall P.\{y\}$	$\exists^{=1} z P(x,z) \wedge P(x,y)$
maxCardinality(n)	$\leq n.P$	$\exists^{\leq n} y.P(x,y)$
minCardinality(n)	$\geq n.P$	$\exists^{\geq n} y.P(x,y)$
cardinality(n)	$n.P$	$\exists^{=n} y.P(x,y)$

First-Order Logic Equivalents (Cont'd)

OWL Class Axioms for C	DL Syntax	FOL
rdfs:subClassOf(C_1)	$C \sqsubseteq C_1$	$\forall x : C(x) \rightarrow C_1(x)$
equivalentClass(C_1)	$C \equiv C_1$	$\forall x : C(x) \leftrightarrow C_1(x)$
disjointWith(C_1)	$C \sqsubseteq \neg C_1$	$\forall x : C(x) \rightarrow \neg C_1(x)$

OWL Individual Axioms	DL Syntax	FOL
x_1 sameAs x_2	$\{x_1\} \equiv \{x_2\}$	$x_1 = x_2$
x_1 differentFrom x_2	$\{x_1\} \sqsubseteq \neg\{x_2\}$	$x_1 \neq x_2$
AllDifferent$(x_1, \ldots, x_n)$	$\bigwedge_{i \neq j} \{x_i\} \sqsubseteq \neg\{x_j\}$	$\bigwedge_{i \neq j} x_i \neq x_j$

First-Order Logic Equivalents (Cont'd)

OWL Properties	DL Syntax	FOL
P	P	$P(x,y)$

OWL Property Axioms for P	DL Syntax	FOL
rdfs:range(C)	$\top \sqsubseteq \forall P.C$	$\forall x,y : P(x,y) \rightarrow C(y)$
rdfs:domain(C)	$C \sqsupseteq \exists P.\top$	$\forall x,y : P(x,y) \rightarrow C(x)$
subPropertyOf(P_2)	$P \sqsubseteq P_2$	$\forall x,y : P(x,y) \rightarrow P_2(x,y)$
equivalentProperty(P_2)	$P \equiv P_2$	$\forall x,y : P(x,y) \leftrightarrow P_2(x,y)$
inverseOf(P_2)	$P \equiv P_2^-$	$\forall x,y : P(x,y) \leftrightarrow P_2(y,x)$
TransitiveProperty	$P^+ \equiv P$	$\forall x,z : \exists y : P(x,y) \wedge P(y,z) \rightarrow P_2(x,z)$
FunctionalProperty	$\top \sqsubseteq \leq 1 P.\top$	$\forall x,y_1,y_2 : P(x,y_1) \wedge P(x,y_2) \rightarrow y_1 = y_2$
InverseFunctionalProperty	$\top \sqsubseteq \leq 1 P^-.\top$	$\forall x,y_1,y_2 : P(y_1,x) \wedge P(y_2,x) \rightarrow y_1 = y_2$

Representation

- most OWL constructs have a straightforward representation in RDF/XML and N3.
- OWL in RDF/XML format: usage of class, property, and individual names:
  - as @rdf:about when used as identifier of a subject (owl:Class, rdf:Property and their subclasses),
  - as @rdf:resource as the object of a property.
- some constructs need auxiliary structures (collections):
  owl:unionOf, owl:intersectionOf, and owl:oneOf are based on Collections
  - representation in RDF/XML by rdf:parseType=”Collection”.
  - representation in N3 for the Jena tool by (x_1 x_2 $\ldots x_n$)
  - another proposal for N3 uses RDF lists: rdf:List, rdf:first, rdf:rest

Use of the Jena Tool

- option “-t”: transform

  jena -t -inf -r pellet [-il RDF/XML] -if *rdf-file* .

- option “-q”: query

  jena -q -inf -r pellet [-il RDF/XML] -qf *query-file* .

- option “-e”: export the class tree (available only when the pellet reasoner is activated). Input is an RDF or OWL file:

  jena -e -inf -r pellet [-il RDF/XML] -if *rdf-file*.

- option “-pellet” short for “-inf -r pellet”.

EXAMPLE: PARADOX

```
<?xml version="1.0"?>
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xml:base="foo://bla/">
 <owl:Class rdf:about="Paradox">
  <owl:complementOf rdf:resource="Paradox"/>
 </owl:Class>
</rdf:RDF>
```

[Filename: RDF/paradox.rdf]

```
prefix : <foo://bla/>
select ?X
from <file:paradox.rdf>
where {?X :name 3}
```

[Filename: RDF/paradox.sparql]

- without reasoner:

  jena -t -il RDF/XML -if paradox.rdf

  Outputs the same RDF facts in N3 without checking consistency.

- with reasoner:

  jena -t -inf -r pellet -il RDF/XML -if paradox.rdf

  reads the RDF file, creates a model (and checks consistency) and in this case reports that it is not consistent.

EXAMPLE: UNION AND SUBCLASS

```
<?xml version="1.0"?>
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:f="foo://bla/"
    xml:base="foo://bla/">
 <owl:Class rdf:about="Person">
  <owl:unionOf rdf:parseType="Collection">
   <owl:Class rdf:about="Male"/>
   <owl:Class rdf:about="Female"/>
  </owl:unionOf>
 </owl:Class>
 <owl:Class rdf:about="EqToPerson">
  <owl:unionOf rdf:parseType="Collection">
   <owl:Class rdf:about="Female"/>
   <owl:Class rdf:about="Male"/>
  </owl:unionOf>
 </owl:Class>
 <f:Person rdf:about="unknownPerson"/>
</rdf:RDF>
```
[Filename: RDF/union-subclass.rdf]

Example (Cont'd)

- print class tree (with jena -e -pellet):

```
owl:Thing
   bla:Person = bla:EqToPerson - (bla:unknownPerson)
      bla:Female
      bla:Male
```

- Male and Female are derived to be subclasses of Person.
- Person and EqToPerson are equivalent classes.
- unknownPerson is a member of Person and EqToPerson.

```
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/>
select ?SC ?C ?T ?CC ?CD
from <file:union-subclass.rdf>
where {{?SC rdfs:subClassOf ?C} UNION
       {:unknownPerson rdf:type ?T} UNION
       {?CC owl:equivalentClass ?CD}}
```

[Filename: RDF/union-subclass.sparql]

EXAMPLE: OWL:RESTRICTION

```
<?xml version="1.0"?>
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:f="foo://bla/"
    xml:base="foo://bla/">
 <owl:Class rdf:about="Parent">
  <owl:intersectionOf rdf:parseType="Collection">
   <owl:Class rdf:about="Person"/>
   <owl:Restriction>
    <owl:onProperty rdf:resource="hasChild"/>
    <owl:minCardinality>1</owl:minCardinality>
   </owl:Restriction>
  </owl:intersectionOf>
 </owl:Class>
 <f:Person rdf:about="john">
   <f:hasChild><f:Person rdf:about="alice"/></f:hasChild>
 </f:Person>
</rdf:RDF>
```

[Filename: RDF/restriction.rdf]

```
prefix rdf:
prefix : <foo://bla/>
select ?C
from <file:restriction.rdf>
where {:john a ?C}
```

[Filename: RDF/restriction.sparql]

EXERCISE

Consider

```
<owl:Class rdf:about="C1">
 <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="A"/>
  <owl:Class rdf:about="B"/>
 </owl:intersectionOf>
</owl:Class>
```

and

```
<owl:Class rdf:about="C2">
 <rdfs:subClassOf rdf:resource="A"/>
 <rdfs:subClassOf rdf:resource="B"/>
</owl:Class>
```

- give mathematical characterizations of both cases.
- discuss whether both fragments are equivalent or not.

EXERCISE

```
<?xml version="1.0"?>
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xml:base="foo://bla/">
 <owl:Class rdf:about="ParentA">
  <owl:intersectionOf rdf:parseType="Collection">
   <owl:Class rdf:about="Person"/>
   <owl:Restriction>
    <owl:onProperty rdf:resource="hasChild"/>
    <owl:minCardinality>1</owl:minCardinality>
   </owl:Restriction>
  </owl:intersectionOf>
 </owl:Class>
 <owl:Restriction rdf:about="ParentB">
  <rdfs:subClassOf rdf:resource="Person"/>
  <owl:onProperty rdf:resource="hasChild"/>
  <owl:minCardinality>1</owl:minCardinality>
 </owl:Restriction>
</rdf:RDF>
```

Exercise: Discuss how far both specifications should be regarded to be equivalent or not. Check if they are equivalent using the Jena/Pellet combination.

[Filename: RDF/equivalent-parent.rdf]

DISCUSSION

- A characterizes the class as the intersection of Person and the class “≥ 1hasChild.$\top$”. Note: adding (Parent rdfs:subClassOf X) would not *define* the class as intersection of X, Person, and ≥ 1hasChild.$\top$, but would state that every element of the class is also an instance of X.
- B states that the class is *defined* as ≥ 1hasChild.$\top$. It also states that this class is a subset of Person (this does not belong to the definition of the class, but is a separate axiom).

For checking equivalence, it is important to go the right way and to understand the semantics of DL knowledge bases.

Equivalence?

Ask the Jena/Pellet combination:

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <foo://bla/>
select ?X
from <file:equivalent-parent.rdf>
where {:ParentA owl:equivalentClass :ParentB}
```

[Filename: RDF/equivalent-parent.sparql]

- the answer is “yes”.
  Is it correct to conclude that both classes are equivalent?
- Note: the query “where :ParentA owl:equivalentClass ?X” does not yield ParentB.
- Note: the class tree also does not contain ParentB.

Check of Set-Theoretic Equivalence

- show that assuming an instance in $A \setminus B$ or in $A \setminus B$ makes the ontology inconsistent.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:f="foo://bla/"
    xml:base="foo://bla/">
 <owl:Class rdf:about="SymmetricDifference">
  <owl:unionOf rdf:parseType="Collection">
   <owl:Class rdf:about="AwithoutB">
    <owl:intersectionOf rdf:parseType="Collection">
     <owl:Class rdf:about="ParentA"/>
     <owl:Class rdf:about="ComplementB">
      <owl:complementOf rdf:resource="ParentB"/>
     </owl:Class>
    </owl:intersectionOf>
   </owl:Class>
   <owl:Class rdf:about="BwithoutA">
    <owl:intersectionOf rdf:parseType="Collection">
     <owl:Class rdf:about="ParentB"/>
     <owl:Class rdf:about="ComplementA">
      <owl:complementOf rdf:resource="ParentA"/>
     </owl:Class>
    </owl:intersectionOf>
   </owl:Class>
  </owl:unionOf>
 </owl:Class>
 <f:SymmetricDifference rdf:about="dummy1"/>
 <!-- <f:AwithoutB rdf:about="dummy2"/>-->
 <!-- <f:BwithoutA rdf:about="dummy3"/>-->
</rdf:RDF>
```

[Filename: RDF/equivalent-parent2.rdf]

```
prefix : <foo://bla/>
select ?X
from <file:equivalent-parent.rdf>
from <file:equivalent-parent2.rdf>
where {:x :hasChild ?X}
```

[Filename: RDF/equivalent-parent2.sparql]

They do not mean the same thing

Add a class Cat that is disjoint from Person, and add a cat that has one child.

- It is not a ParentA:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:f="foo://bla/"
    xml:base="foo://bla/">
 <owl:Class rdf:about="ParentA">
  <owl:intersectionOf rdf:parseType="Collection">
   <owl:Class rdf:about="Person"/>
   <owl:Restriction>
    <owl:onProperty rdf:resource="hasChild"/>
    <owl:minCardinality>1</owl:minCardinality>
   </owl:Restriction>
  </owl:intersectionOf>
 </owl:Class>
 <owl:Class rdf:about="Cat">
  <owl:disjointWith rdf:resource="Person"/>
 </owl:Class>
 <f:Cat rdf:about="garfield" f:name="Garfield">
  <f:hasChild rdf:resource="nermal"/>
 </f:Cat>
</rdf:RDF>
```

[Filename: RDF/equivalent-parentA.rdf]

Class tree:

```
owl:Thing - (bla:nermal)
   bla:Person
      bla:ParentA
   bla:Cat - (bla:garfield)
```

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix : <foo://bla/>
select ?X
from <file:equivalent-parentA.rdf>
where {:garfield rdf:type :ParentA}
```

[Filename: RDF/equivalent-parentA.sparql]

They do not mean the same thing (Cont'd)

- Do the same with the fragment for ParentB
- The cat is in ≥ 1hasChild.$\top$, thus it is derived (subclass!) that it is a Person, which is inconsistent.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:f="foo://bla/"
    xml:base="foo://bla/">
 <owl:Restriction rdf:about="ParentB">
  <rdfs:subClassOf rdf:resource="Person"/>
  <owl:onProperty rdf:resource="hasChild"/>
  <owl:minCardinality>1</owl:minCardinality>
 </owl:Restriction>
 <owl:Class rdf:about="Cat">
  <owl:disjointWith rdf:resource="Person"/>
 </owl:Class>
 <f:Cat rdf:about="garfield" f:name="Garfield">
  <f:hasChild rdf:resource="nermal"/>
 </f:Cat>
</rdf:RDF>
```

[Filename: RDF/equivalent-parentB.rdf]

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix : <foo://bla/>
select ?X
from <file:equivalent-parentB.rdf>
where {:garfield rdf:type :ParentB}
```

[Filename: RDF/equivalent-parentB.sparql]

Discussion (Cont'd)

Semantics

- Two classes are *equivalent* (wrt. the knowledge base) if they have the same interpretation in every *model* of the KB.
- The example with the cat is not a model.
  - so forget about the cat, and the classes are equivalent?
  - but the cat is not inconsistent with A.
  - can a cat (or anything else) make a (consistent) *definition* inconsistent?

Discussion (Cont'd)

The original knowledge base is more than two class definitions:

- definition of class ParentA:

  $\text{ParentA} \equiv \text{Person} \sqcap \exists \text{hasChild}.\top$

- definition of class ParentB:

  $\text{ParentB} \equiv \exists \text{hasChild}.\top$

- and an *axiom* (subclass axiom)

  $\text{ParentB} \sqsubseteq \text{Person}$

- and only under the presence of the axiom, ParentA and ParentB are equivalent.
- the axiom excludes cats that have children from any consideration.

Discussion (Cont'd)

- if the axiom is removed from the knowledge base, the classes are not equivalent:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/>.
:ParentA owl:intersectionOf (:Person
   [ a owl:Restriction;
     owl:onProperty :hasChild;
     owl:minCardinality 1]).
:ParentB a owl:Restriction;
     owl:onProperty :hasChild;
     owl:minCardinality 1.
# :ParentB rdfs:subClassOf :Person.
```

[Filename: RDF/equivalent-parent-defs.n3]

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <foo://bla/>
select ?X
from <file:equivalent-parent-defs.n3>
where {:ParentA owl:equivalentClass :ParentB}
```

[Filename: RDF/equivalent-parent-defs.sparql]

MULTIPLE RESTRICTIONS ON A PROPERTY

- “All persons that have at least two children, and one of them is male”

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix : <foo://bla/>.
### Test: multiple restrictions: the cardinality condition is then ignored
:HasTwoChildrenOneMale owl:intersectionOf (:Person
  [ a owl:Restriction; owl:onProperty :hasChild;
    owl:someValuesFrom :Male; owl:minCardinality 2]).
:name a owl:FunctionalProperty.
:Male rdfs:subClassOf :Person; owl:disjointWith :Female.
:Female rdfs:subClassOf :Person.
:kate a :Female; :name "Kate"; :hasChild :john.
:john a :Male; :name "John";
  :hasChild [a :Female; :name "Alice"], [a :Male; :name "Bob"].
:sue a :Female; :name "Sue";
  :hasChild [a :Female; :name "Anne"], [a :Female; :name "Barbara"].
```

[Filename: RDF/restriction-double.n3]

```
prefix : <foo://bla/>
select ?X
from <file:restriction-double.n3>
where {?X a :HasTwoChildrenOneMale}
```

[Filename: RDF/restriction-double.sparql]

- The cardinality condition is ignored in this case (Result: John and Kate).
- Solution: intersection of restrictions

Multiple Restrictions on a Property

- “All persons that have at least two children, and one of them is male”

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix : <foo://bla/>.
:HasTwoChildrenOneMale owl:intersectionOf (:Person
  [ a owl:Restriction; owl:onProperty :hasChild; owl:someValuesFrom :Male]
  [ a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 2]).
:name a owl:FunctionalProperty.
:Male rdfs:subClassOf :Person; owl:disjointWith :Female.
:Female rdfs:subClassOf :Person.
:kate a :Female; :name "Kate"; :hasChild :john.
:john a :Male; :name "John";
  :hasChild [a :Female; :name "Alice"], [a :Male; :name "Bob"].
:sue a :Female; :name "Sue";
  :hasChild [a :Female; :name "Anne"], [a :Female; :name "Barbara"].
```

```
prefix : <foo://bla/>
select ?X
from <file:intersect-restrictions.n3>
where {?X a :HasTwoChildrenOneMale}
```

[Filename: RDF/intersect-restrictions.sparql]

[Filename: RDF/intersect-restrictions.n3]

- Note: this is different from Qualified Range Restrictions such as “All persons that have at least two male children” – see Slide 338.

Use of a Derived Class

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix : <foo://bla/names#>.
:kate :name "Kate"; :child :john.
:john :name "John"; :child :alice.
:alice :name "Alice".
:Parent a owl:Class; owl:equivalentClass
 [ a owl:Restriction; owl:onProperty :child; owl:minCardinality 1].
:Grandparent owl:equivalentClass
 [a owl:Restriction; owl:onProperty :child; owl:someValuesFrom :Parent].
```

[Filename: RDF/grandparent.n3]

```
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix : <foo://bla/names#>
select ?A ?B
from <file:grandparent.n3>
where {{?A a :Parent} UNION
       {?B a :Grandparent} UNION
       {:Grandparent rdfs:subClassOf :Parent}}
```

[Filename: RDF/grandparent.sparql]

UNION AS $A \sqcup B \equiv \neg((\neg A) \sqcap (\neg B))$

```
@prefix : <foo://bla/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
:A rdf:type owl:Class.         :B rdf:type owl:Class.
:x rdf:type :A.                :x rdf:type :B.
:CompA owl:complementOf :A.    :CompB owl:complementOf :B.
:Union1 owl:unionOf (:A :B).
:Union2 owl:complementOf :IntersectComps.
:IntersectComps owl:intersectionOf (:CompA :CompB).
:y rdf:type :CompA. # a negative assertion y not in A would be better -> OWL 1.1
:y rdf:type :CompB.
```

[Filename: RDF/union.n3]

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix : <foo://bla/>
select ?X ?Y
from <file:union.n3>
where {{?X rdf:type ?Y} UNION {:Union1 owl:equivalentClass ?Y}}
```

[Filename: RDF/union.sparql]

Non-Existence of a Property

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/names#>.
:kate a :Person; :name "Kate"; :hasChild :john.
:john a :Person; :name "John"; :hasChild :alice, :bob.
:alice a :Person; :name "Alice".
:bob a :Person; :name "Bob".
:name a owl:FunctionalProperty.
:ChildlessA owl:intersectionOf (:Person
  [ a owl:Restriction; owl:onProperty :hasChild; owl:maxCardinality 0]).
:ChildlessB owl:intersectionOf (:Person
  [ a owl:Restriction; owl:onProperty :hasChild; owl:allValuesFrom owl:Nothing]).
:ParentA owl:intersectionOf (:Person [owl:complementOf :ChildlessA]).
:ParentB owl:intersectionOf (:Person
  [ a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 1]).
```

```
prefix : <foo://bla/names#>
select ?X ?Y
from <file:childless.n3>
where {{?X a :ChildlessA}
       union {?Y a :ParentA}}
```

[Filename: RDF/childless.sparql]

[Filename: RDF/childless.n3]

- export class tree: ChildlessA and ChildlessB are equivalent,
- note: due to the Open World Assumption, both classes are empty.
- Persons where no children are known are neither in ChildlessA or in Parent!

TBox vs. ABox

DL makes a clean separation between TBox and ABox vocabulary:

- TBox: RDFS/OWL vocabulary for information about classes and properties (further partitioned into definitions and axioms),
- ABox: Domain vocabulary and rdf:type.

RDFS/OWL allows to mix everything in a set of triples.

Nominals

- use individuals (that usually occur only in the ABox) in the TBox:
- as individuals :Italy (that are often implemented in the reasoner as unary classes) with *property* owl:hasValue o
  (the class of all things such that $\{$?x *property* o$\}$ holds).
- in enumerated classes *class* owl:oneOf (o$_1$,. . . ,o$_n$)
  (*class* is defined to be the set $\{$o$_1$,. . . ,o$_n$$\}$).

Difference to Reification

- Reification treats a class (e.g. :Penguin) or a property as an individual (:Penguin a :Species)
  - without reification, only specific RDFS and OWL properties are allowed for classes and properties only
  - reification assigns properties from an application domain to classes and properties.
- useful when talking about metadata notions,
- risk: allows for paradoxes

USING NOMINALS: ITALIAN CITIES

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix mon: <http://www.semwebtech.de/mondial/10/meta#>.
@prefix it: <foo://italian/>.
it:Italy owl:sameAs <http://www.semwebtech.de/mondial/10/countries/I/>.
it:ItalianProvince owl:intersectionOf
  (mon:Province
   [a owl:Restriction; owl:onProperty mon:isProvinceOf;
    owl:hasValue it:Italy]).        # Nominal: an individual in a TBox axiom
it:ItalianCity owl:intersectionOf
  (mon:City
   [a owl:Restriction;
    owl:onProperty mon:belongsTo;
    owl:someValuesFrom it:ItalianProvince]).
```

[Filename: RDF/italiancities.n3]

```
prefix it: <foo://italian/>
select ?X
from <file:mondial-meta.n3>
from <file:mondial-europe.n3>
from <file:italiancities.n3>
where {?X a it:ItalianCity}
```

[Filename: RDF/italiancities.sparql]

An Ontology in OWL

Consider the Italian-English-Ontology from Slide 104.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix f: <foo://bla/>.
f:Italian rdfs:subClassOf f:Person;
  owl:disjointWith f:English;
  owl:unionOf (f:Lazy f:LatinLover).
f:Lazy owl:disjointWith f:LatinLover.
f:English rdfs:subClassOf f:Person.
f:Gentleman rdfs:subClassOf f:English.
f:Hooligan rdfs:subClassOf f:English.
f:LatinLover rdfs:subClassOf f:Gentleman.
```

Class tree with jena -e:

```
owl:Thing
   bla:Person
      bla:English
         bla:Hooligan
         bla:Gentleman
      bla:Italian = bla:Lazy
owl:Nothing = bla:LatinLover
```

- LatinLover is empty, thus Italian ≡ Lazy.

[Filename: RDF/italian-english.n3]

Italians and Englishmen (Cont'd)

- the conclusions apply to the instance level:

```
@prefix : <foo://bla/>.
:mario a :Italian.
```

[Filename: RDF/mario.n3]

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix : <foo://bla/>
select ?C
from <file:italian-english.n3>
from <file:mario.n3>
where {:mario rdf:type ?C}
```

[Filename: RDF/italian-english.sparql]

An Ontology in OWL

Consider the Italian-Ontology from Slide 105.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix it: <foo://italian/>.
it:Bolzano owl:sameAs
<http://www.semwebtech.de/mondial/10/countries/I/provinces/TrentinoAltoAdige/cities/Bolzano/>.
it:Italian owl:intersectionOf
  (it:Person
   [a owl:Restriction; owl:onProperty it:livesIn;
    owl:someValuesFrom it:ItalianCity]);
  owl:unionOf (it:Lazy it:Mafioso it:LatinLover).
it:Professor rdfs:subClassOf it:Person.
it:Lazy owl:disjointWith it:ItalianProf;
   owl:disjointWith it:Mafioso;
   owl:disjointWith it:LatinLover.
it:Mafioso owl:disjointWith it:ItalianProf;
   owl:disjointWith it:LatinLover.
it:ItalianProf owl:intersectionOf (it:Italian it:Professor).
it:enrico a it:Professor; it:livesIn it:Bolzano.
```

[Filename: RDF/italian-prof.n3]

```
prefix : <foo://italian/>
select ?C
from <file:italian-prof.n3>
from <file:mondial-meta.n3>
from <file:mondial-europe.n3>
from <file:italiancities.n3>
where {:enrico a ?C}
```

[Filename: RDF/italian-prof.sparql]

ENUMERATED CLASSES: ONEOF

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix mon: <http://www.semwebtech.de/mondial/10/meta#>.
mon:Montanunion owl:intersectionOf
 (mon:Country
  [owl:oneOf
   (<http://www.semwebtech.de/mondial/10/countries/NL/>
    <http://www.semwebtech.de/mondial/10/countries/B/>
    <http://www.semwebtech.de/mondial/10/countries/L/>
    <http://www.semwebtech.de/mondial/10/countries/F/>
    <http://www.semwebtech.de/mondial/10/countries/I/>
    <http://www.semwebtech.de/mondial/10/countries/D/>)]).
<bla:Result> owl:intersectionOf (mon:Organization
  [a owl:Restriction; owl:onProperty mon:hasMember;
   owl:someValuesFrom mon:Montanunion]).
```

[Filename: RDF/montanunion.n3]

```
select ?X
from <file:montanunion.n3>
from <file:mondial-europe.n3>
from <file:mondial-meta.n3>
where {?X a <bla:Result>}
```

[RDF/montanunion.sparql]

- Query: all organizations that share a member with the Montanunion.
- Note: with owl:allValuesFrom (all organizations that are subsets of the Montanunion) the result is empty (although there is e.g. BeNeLux) due to open world: it is not known whether there may exist additional members of e.g. BeNeLux.

ONEOF: A TEST

- oneOf defines a “closed set”:
- note that in owl:oneOf($x_1, \ldots, x_n$), two items may be the same (open world),
- add owl:AllDifferent if needed.

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/>.
:Person owl:oneOf (:john :alice :bob).
### optionally choose e.g. *one* of the following (both together are inconsistent)
# owl:AllDifferent owl:distinctMembers (:john :alice :bob).
# :john owl:sameAs :alice.
:john :name "John".
:alice :name "Alice".
:bob :name "Bob".
:d a :Person.
:d owl:differentFrom :john; owl:differentFrom :alice.
# :d owl:differentFrom :bob.   ### adding this makes the ontology inconsistent
```

[Filename: RDF/three.n3]

- Who is :d?

oneOf: a Test (cont'd)

Who is :d?

- check the class tree:
  bla:Person - (bla:bob, bla:alice, bla:d, bla:john)
- and ask it:

```
prefix : <foo://bla/>
select ?N
from <file:three.n3>
where {:d :name ?N}
```

[RDF/three.sparql]

The answer is ?N/“Bob”.

ANSWER SETS TO QUERIES AS AD-HOC CONCEPTS

- all organizations whose headquarter city is a capital:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <http://www.semwebtech.de/mondial/10/meta#> .
:CountryCapital owl:intersectionOf
  (:City [a owl:Restriction; owl:onProperty :isCapitalOf;
          owl:someValuesFrom :Country]).
<bla:Result> owl:intersectionOf
  (:Organization [a owl:Restriction; owl:onProperty :hasHeadq;
    owl:allValuesFrom :CountryCapital]).
```

[Filename: RDF/organizations-query.n3]

Note: allValuesFrom is in this case equivalent with someValuesFrom since :hasHeadq is a owl:FunctionalProperty.

```
prefix : <http://www.semwebtech.de/mondial/10/meta#>
select ?A ?N
from <file:organizations-query.n3>
from <file:mondial-europe.n3>
from <file:mondial-meta.n3>
where {?X a <bla:Result> . ?X :abbrev ?A . ?X :hasHeadq ?C . ?C :name ?N}
```

[Filename:RDF/organizations-query.sparql]

HOW TO DEAL WITH OWL:ALLVALUESFROM IN AN OPEN WORLD?

- “forall items” is only applicable if additional items can be excluded ($\Rightarrow$ locally closed predicate/property),
- often, RDF data is generated from a database,
- certain predicates can be closed by defining restriction classes with maxCardinality.

OWL:ALLVALUESFROM

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/names#>.
[ a :Male; a :threeChildrenParent; :name "John";
   :child [a :Female; :name "Alice"], [a :Male; :name "Bob"],
          [a :Female; :name "Carol"]].
[ a :Female; a :twoChildrenParent; :name "Sue";
   :child [a :Female; :name "Anne";], [a :Female; :name "Barbara"]].
:name a owl:FunctionalProperty.
:oneChildParent a owl:Restriction;
  owl:onProperty :child; owl:cardinality 1.
:twoChildrenParent a owl:Restriction;
  owl:onProperty :child; owl:cardinality 2.
:threeChildrenParent a owl:Restriction;
  owl:onProperty :child; owl:cardinality 3.
:onlyFemaleChildrenParent a owl:Restriction;
  owl:onProperty :child; owl:allValuesFrom :Female.
```

[Filename: RDF/allvaluesfrom.n3]

```
prefix : <foo://bla/names#>
select ?N
from <file:allvaluesfrom.n3>
where {?X :name ?N .
  ?X a :onlyFemaleChildrenParent}
```

[RDF/allvaluesfrom.sparql]

DATATYPES: HASVALUE WITH LITERAL 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 xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix mon: <http://www.semwebtech.de/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 a owl:DataRange (unary or enumeration) and give it a url.