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:subPropertyOf; $\mathcal{H}$).
- OWL Lite: $\mathcal{SHIF}(D)$, Reasoning in ExpTime.
- OWL DL: $\mathcal{SHOIN}(D)$, decidable.
  Pellet (2007) 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`)
- progress in OWL 2.0 towards $\mathcal{SROIQ}(D)$ and more datatypes ...

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 238)

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)	$C_1 \sqcap \ldots \sqcap C_n$	$C_1(x) \land \ldots \land C_n(x)$
unionOf(C_1, C_2)	$C_1 \sqcup \ldots \sqcup C_n$	$C_1(x) \lor \ldots \lor C_n(x)$
complementOf(C_1)	$\neg C_1$	$\neg C_1(x)$
oneOf$(x_1, \ldots, x_n)$	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	$x = x_1 \lor \ldots \lor x = x_n$

OWL : $x \in C$, Restriction on P	DL Syntax	FOL
someValuesFrom(C')	$\exists P.C'$	$\exists y : P(x,y) \land C'(y)$
allValuesFrom(C')	$\forall P.C'$	$\forall y : P(x,y) \rightarrow C'(y)$
hasValue(y)	$\exists P.\{y\}$	$P(x,y)$
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	$P^+ \equiv P$	$\forall x,y,z : ((P(x,y) \wedge P(y,z)) \rightarrow P(x,z))$
		$\forall x,z : ((\exists y : P(x,y) \wedge P(y,z)) \rightarrow P(x,z))$
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 by (x_1 x_2 $\ldots$ x_n)
  - as RDF lists: rdf:List, rdf:first, rdf:rest

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]

- without reader:

  jena -t -if paradox.rdf

  Outputs the same RDF facts in N3 without checking consistency.

- with reasoner:

  jena -e -pellet -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; T-Box Reasoning

```
<?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]

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.

Discussion

- Two classes are *equivalent* (wrt. the knowledge base) if they have the same interpretation in every *model* of the KB.
- C_1 is characterized to be the intersection of classes A and B.
- for C_2, it is asserted that C_1 is a subset of A and that it is a subset of B.
- Thus there can be some c that is in A, B, C_1, but not in C_2.
- Thus, C_1 and C_2 are not equivalent.

Discussion: Formal Notation

The DL equivalent to the knowledge base (TBox) is

$$\mathcal{T} = \{C_1 \equiv (A \sqcap B)\ , \quad C_2 \sqsubseteq A\ , \quad C_2 \sqsubseteq B\}$$

The First-Order Logic equivalent is

$$\mathcal{KB} = \{\forall x : A(x) \wedge B(x) \leftrightarrow C_1(x)\ , \quad \forall x : C_2(x) \rightarrow A(x) \wedge B(x)\}$$

Thus, $\mathcal{KB} \models \forall x : C_2(x) \rightarrow A(x) \wedge B(x)$.

Or, in DL: $\mathcal{T} \models C_2 \sqsubseteq C_1$.

On the other hand, $\mathcal{M} = (\mathcal{D},\mathcal{I})$ with $\mathcal{D} = \{c\}$ and

$$\mathcal{I}(A) = \{c\},\ \ \mathcal{I}(B) = \{c\},\ \ \mathcal{I}(C_1) = \{c\},\ \ \mathcal{I}(C_2) = \emptyset$$

is a model of $\mathcal{KB}$ (wrt. first-order logic) and $\mathcal{T}$ (wrt. DL) that shows that C_1 and C_2 are not equivalent.

OWL:RESTRICTION – EXAMPLE

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

[Filename: RDF/restriction.sparql]

RESTRICTIONS ONLY AS BLANK NODES

Consider the following (bad) specification:

```
:badIdea a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 1.
```

This is not allowed in OWL-DL.

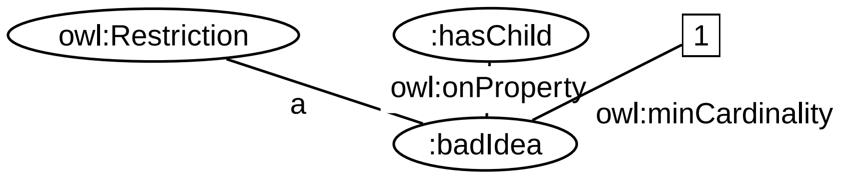
Correct specification:

```
:badIdea owl:equivalentClass
 [a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 1].
```

Why? ... there are many reasons, for one of them see next slide.

Restrictions Only as Blank Nodes (Cont'd)

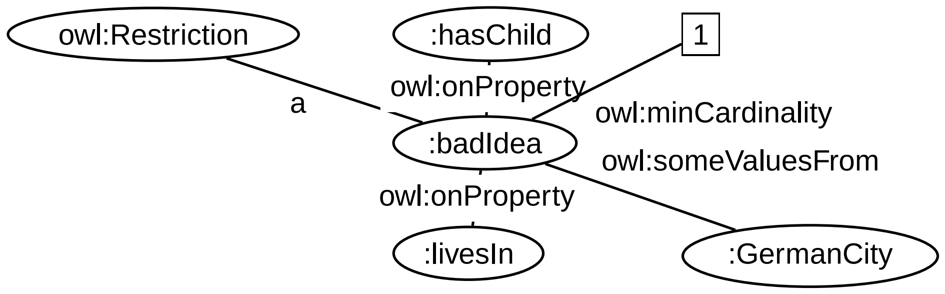
A class with two such specifications:



```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/>.
:badIdea a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 1 .
:badIdea a owl:Restriction; owl:onProperty :livesIn; owl:someValuesFrom :GermanCity.
```

[Filename: RDF/badIdea.n3]

- call jena -t -pellet -if badIdea.n3:



The two restriction specifications are messed up.

Restrictions Only as Blank Nodes (Cont'd)

- Thus specify each Restriction specification with a separate blank node:

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/>.
:TwoRestrictions owl:intersectionOf
 ( [ a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 1]
   [ a owl:Restriction; owl:onProperty :livesIn; owl:someValuesFrom :GermanCity] ) .
```

[Filename: RDF/twoRestrictions.n3]

The DL equivalent: $\text{TwoRestrictions} \equiv (\exists\ \text{hasChild}.\top) \sqcap (\exists\ \text{livesIn.GermanCity})$

Another reason:

```
:AnotherBadDesignExample a owl:Restriction;
   owl:onProperty :hasChild; owl:minCardinality 1;
   rdfs:subClassOf :Person.
```

... mixes the *definition* of the Restriction with an axiom; the meaning is unclear (and the outcome is up to the strategy of the Reasoner). Obviously, the designer intended to specify an intersection, $\text{ABDE} \equiv \exists \geq 1\ \text{hasChild}.\top \sqcap \text{Person}$, but the DL translation actually specifies a definition and an assertive axiom: $\text{ABDE} \equiv \exists \geq 1\ \text{hasChild}.\top \wedge \text{ABDE} \sqsubseteq \text{Person}$

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 Sue).
- 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 358.

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.
:Union1 owl:unionOf (:A :B).
:CompA owl:complementOf :A.    :CompB owl:complementOf :B.
:IntersectComps owl:intersectionOf (:CompA :CompB).
:Union2 owl:complementOf :IntersectComps.
:x rdf:type :A.                 :x rdf:type :B.
:y rdf:type :CompA. # a negative assertion y not in A would be better -> OWL 2
: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 ?C ?D
from <file:union.n3>
where {{?X rdf:type ?C} UNION {:Union1 owl:equivalentClass ?D}}
```

[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]).
```

[Filename: RDF/childless.n3]

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

[Filename: RDF/childless.sparql]

- 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.org/mondial/10/meta#>.
@prefix it: <foo://italian/>.
it:Italy owl:sameAs <http://www.semwebtech.org/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 109.

```
@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 110.

```
@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.org/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.org/mondial/10/meta#>.
<bla:MontanunionMembers> owl:intersectionOf
 (mon:Country
  [owl:oneOf
   (<http://www.semwebtech.org/mondial/10/countries/NL/>
    <http://www.semwebtech.org/mondial/10/countries/B/>
    <http://www.semwebtech.org/mondial/10/countries/L/>
    <http://www.semwebtech.org/mondial/10/countries/F/>
    <http://www.semwebtech.org/mondial/10/countries/I/>
    <http://www.semwebtech.org/mondial/10/countries/D/>)]).
<bla:Result> owl:intersectionOf (mon:Organization
  [a owl:Restriction; owl:onProperty mon:hasMember;
   owl:someValuesFrom <bla:MontanunionMembers>]).
```

[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.

ONEOF (EXAMPLE CONT'D)

- previous example: "all organizations that share a member with the Montanunion."
  (DL: $x \in \exists$hasMember.MontanunionMembers)
- "all organizations where *all* members are also members of the Montanunion."
  (DL: $x \in \forall$hasMember.MontanunionMembers)
  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.
  Only if the membership is "closed", results can be proven:

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix mon: <http://www.semwebtech.org/mondial/10/meta#>.
<http://www.semwebtech.org/mondial/10/organizations/Benelux/>
 a [a owl:Restriction;
     owl:onProperty mon:hasMember; owl:cardinality 3].
<bla:SubsetOfMU> owl:intersectionOf (mon:Organization
  [a owl:Restriction; owl:onProperty mon:hasMember;
     owl:allValuesFrom <bla:MontanunionMembers>]).
mon:name a owl:FunctionalProperty. # not yet given in th
```

[Filename: RDF/montanunion2.n3]

```
select ?X
from <file:montanunion.n3>
from <file:montanunion2.n3>
from <file:mondial-europe.n3>
from <file:mondial-meta.n3>
where {?X a <bla:SubsetOfMU>}
```

[RDF/montanunion2.sparql]

ONEOF (EXAMPLE CONT'D)

- "all organizations that cover *all* members of the Montanunion."
  (DL: $x \in \forall$hasMember.MontanunionMembers)
  owl:oneOf is closed, so there is no problem with the universal quantifier.

```
@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#>.
<bla:EUMembers> owl:equivalentClass  [a owl:Restriction;
    owl:onProperty mon:isMember; owl:hasValue
    <http://www.semwebtech.org/mondial/10/organizations/EU/>].
```

[Filename: RDF/montanunion3.n3]

```
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select ?X # ?Y ?Z
from <file:montanunion.n3>
from <file:montanunion3.n3>
from <file:mondial-europe.n3>
from <file:mondial-meta.n3>
where {#{?Y a <bla:EUMembers>} UNION {?Z a <bla:MontanunionMembers>} UNION
       {<bla:MontanunionMembers> rdfs:subClassOf ?X}}
```

[RDF/montanunion3.sparql]

ONEOF (EXAMPLE CONT'D)

Previous example:

- only for one organization
- defined a class that contains all members of the organization
- not possible to define a *family of classes* – one class for each organization.
- this would require a *parameterized constructor*:
  "c_{org} is the set of all members of org"
  Second-Order Logic: each organization can be seen as a unary predicate (=set):
  $\forall Org : Org(c) \leftrightarrow$ hasMember(Org, c)
  or in F-Logic syntax: `C isa Org :- Org:organization[hasMember->C]`
  yields e.g.
  $I(eu) = \{germany, france, \ldots\}$,
  $I(nato) = \{usa, canada, germany, \ldots\}$
  Recall that "organization" itself is a predicate:
  $I(organization) = \{eu, nato, \ldots, \}$
  So we have again reification: organizations are both first-order-individuals and classes.

CONVENIENCE CONSTRUCT: OWL:ALLDIFFERENT

- owl:oneOf defines a class as a closed set;
- in owl:oneOf (x$_1$, . . . , x$_n$), two items may be the same (open world),

owl:AllDifferent

- Triples of the form :a owl:differentFrom :b state that two individuals are different.
  For a database with n elements, one needs
  $(n-1) + (n-2) + \ldots + 2 + 1 = \sum_{i=1..n} i = n \cdot (n+1)/2 = O(n^2)$ such statements.
- The –purely syntactical– convenience construct

  [a owl:AllDifferent; owl:members (r_1 r_2 . . . r_n)]

  provides a shorthand notation.
  - it is *immediately* translated into the set of all statements
    $\{r_i$ owl:differentFrom $r_j \mid i \neq j \in 1..n\}$
  - [a owl:AllDifferent; owl:members (...)]
    is to be understood as a (blank node) that acts as a *specification* that the listed things are different that does not actually exist in the model.

[Syntax] owl:AllDifferent in RDF/XML

```
<?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="Foo">
  <owl:equivalentClass> <owl:Class>
    <owl:oneOf rdf:parseType="Collection">
     <owl:Thing rdf:about="a"/>  <owl:Thing rdf:about="b"/>
     <owl:Thing rdf:about="c"/>  <owl:Thing rdf:about="d"/>
    </owl:oneOf>
   </owl:Class> </owl:equivalentClass>
 </owl:Class>
 <owl:AllDifferent>  <!-- use like a class, but is only a shorthand -->
    <owl:members rdf:parseType="Collection">
     <owl:Thing rdf:about="a"/>  <owl:Thing rdf:about="b"/>
     <owl:Thing rdf:about="c"/>  <owl:Thing rdf:about="d"/>
    </owl:members>
 </owl:AllDifferent>
 <owl:Thing rdf:about="a"> <owl:sameAs rdf:resource="b"/> </owl:Thing>
</rdf:RDF>
```

[Filename: RDF/alldiff.rdf]

```
prefix : <foo://bla/>
prefix owl:
  <http://www.w3.org/2002/07/owl#>
select ?X ?P ?P2 ?V
from <file:alldiff.rdf>
where {?X a owl:AllDifferent ;
       ?P [?P2 ?V]}
```

[Filename: RDF/alldiffxml.sparql]

- AllDifferent is only intended as a kind of command to the application to add all pairwise “different-from” statements, it does not actually introduce itself as triples:
- querying {?X a owl:AllDifferent} is actually not intended.

[Syntax] owl:AllDifferent in N3

Example:

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/>.
:Foo owl:equivalentClass [ owl:oneOf (:a :b :c :d) ].
# both the following syntaxes are equivalent and  correct:
[ a owl:AllDifferent; owl:members (:a :b)].
[] a owl:AllDifferent; owl:members (:c :d).
:a owl:sameAs :b.
# :b owl:sameAs :d.
```

[Filename: RDF/alldiff.n3]

```
prefix : <foo://bla/>
select ?X ?Y
from <file:alldiff.n3>
where {?X a owl:AllDifferent ; ?P [?P2 ?V]}
```

[Filename: RDF/alldiff.sparql]

ONEOF: A TEST

- owl:oneOf defines a “closed set” (use with anonymous class; see below):
- note that in owl:oneOf (x_1, . . . , x_n), two items may be the same (open world),
- optional owl:AllDifferent to guarantee that (x_1, . . . , x_n) are pairwise distinct.

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/>.
:Person owl:equivalentClass [ owl:oneOf (:john :alice :bob) ].
# :john owl:sameAs :alice.  # to show that it is consistent that they are the same
[] a owl:AllDifferent; owl:members (:john :alice :bob).  # to guarantee distinctness
# :name a owl:FunctionalProperty.  # this also guarantees distinctness ;)
: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}
```

[Filename: 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.org/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:someValuesFrom :CountryCapital]).            [Filename: RDF/organizations-query.n3]
```

```
prefix : <http://www.semwebtech.org/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 (⇒ 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 owl:equivalentClass [a owl:Re
  owl:onProperty :child; owl:cardinality 1].
:TwoChildrenParent owl:equivalentClass [a owl
  owl:onProperty :child; owl:cardinality 2].
:ThreeChildrenParent owl:equivalentClass [a owl:Restriction;
  owl:onProperty :child; owl:cardinality 3].
:OnlyFemaleChildrenParent owl:equivalentClass [a owl:Restriction;
  owl:onProperty :child; owl:allValuesFrom :Female].
```

[Filename: RDF/allvaluesfrom.n3]

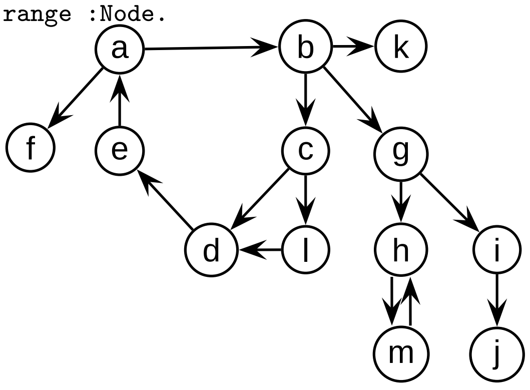
```
prefix : <foo://bla/names#>
select ?N
from <file:allvaluesfrom.n3>
where {?X :name ?N .
  ?X a :OnlyFemaleChildrenParent}
```

[Filename: RDF/allvaluesfrom.sparql]

Example: Win-Move-Game in OWL

```
@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/>.

:Node a owl:Class; owl:equivalentClass
  [ a owl:Class; owl:oneOf (:a :b :c :d :e :f :g :h :i :j :k :l :m)].
:edge a owl:ObjectProperty; rdfs:domain :Node; rdfs:range :Node.
:out a owl:DatatypeProperty.
:a a :Node; :out 2; :edge :b, :f.
:b a :Node; :out 3; :edge :c, :g, :k.
:c a :Node; :out 2; :edge :d, :l.
:d a :Node; :out 1; :edge :e.
:e a :Node; :out 1; :edge :a.
:f a :Node; :out 0 .
:g a :Node; :out 2; :edge :i, :h.
:h a :Node; :out 1; :edge :m.
:i a :Node; :out 1; :edge :j.
:j a :Node; :out 0 .
:k a :Node; :out 0 .
:l a :Node; :out 1; :edge :d.
:m a :Node; :out 1; :edge :h.
```



[Filename: RDF/winmove-graph.n3]

Win-Move-Game in OWL – the Game Axioms

“If a player cannot move, he loses.”

Which nodes are WinNodes, which one are LoseNodes (i.e., the player who has to move wins/loses)?

- if a player can move to some LoseNode (for the other), he will win.
- if a player can move only to WinNodes (for the other), he will lose.
- recall that there can be nodes that are neither WinNodes nor LoseNodes.

```
@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/>.

:WinNode a owl:Class; owl:intersectionOf ( :Node
  [a owl:Restriction; owl:onProperty :edge; owl:someValuesFrom :LoseNode]).
:LoseNode a owl:Class; owl:intersectionOf ( :Node
  [a owl:Restriction; owl:onProperty :edge; owl:allValuesFrom :WinNode]).
```

[Filename: RDF/winmove-axioms.n3]

Win-Move-Game in OWL – Closure

```
@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/>.
:DeadEndNode a owl:Class;  rdfs:subClassOf :Node;
  owl:equivalentClass [ a owl:Restriction; owl:onProperty :out; owl:hasValue 0],
                      [ a owl:Restriction; owl:onProperty :edge; owl:cardinality 0].
:OneExitNode a owl:Class; rdfs:subClassOf :Node;
  owl:equivalentClass [ a owl:Restriction; owl:onProperty :out; owl:hasValue 1],
                      [ a owl:Restriction; owl:onProperty :edge; owl:cardinality 1].
:TwoExitsNode a owl:Class; rdfs:subClassOf :Node;
  owl:equivalentClass [ a owl:Restriction; owl:onProperty :out; owl:hasValue 2],
                      [ a owl:Restriction; owl:onProperty :edge; owl:cardinality 2].
:ThreeExitsNode a owl:Class; rdfs:subClassOf :Node;
  owl:equivalentClass [ a owl:Restriction; owl:onProperty :out; owl:hasValue 3],
                      [ a owl:Restriction; owl:onProperty :edge; owl:cardinality 3].
```

[Filename: RDF/winmove-closure.n3]

Win-Move-Game in OWL: DeadEndNodes

Prove that DeadEndNodes are LoseNodes:

- obvious: Player cannot move from there
- exercise: give a formal (Tableau) proof
- The OWL Reasoner does it:

```
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix : <foo://bla/>
select ?X
from <file:winmove-axioms.n3>
from <file:winmove-closure.n3>
where {:DeadEndNode rdfs:subClassOf :LoseNode}
```

[Filename: RDF/deadendnodes.sparql]

The answer contains an (empty) tuple which means “yes”.

Win-Move-Game in OWL

```
prefix : <foo://bla/>
select ?W ?L ?DE
from <file:winmove-graph.n3>
from <file:winmove-axioms.n3>
from <file:winmove-closure.n3>
where {{?W a :WinNode} UNION
       {?L a :LoseNode} UNION
       {?DE a :DeadEndNode}}
```

[Filename: RDF/winmove.sparql]

lose: f, k, j, e, l
win: c, a, i, b, d

Exercise

- Is it possible to characterize DrawNodes in OWL?