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`)

- OWL 2.0 towards $\mathcal{SROIQ}(D)$ and more datatypes ...

OWL NOTIONS; OWL-DL VS. RDF/RDFS; MODEL VS. GRAPH

- OWL is defined based on (Description Logics) model theory,

- OWL ontologies can be represented by RDF graphs,

- Only *certain* RDF graphs are allowed OWL-DL ontologies: those, where class names, property names, individuals etc. occur in a well-organized way.

- Reasoning works on the (Description Logic) model, the RDF graph is only a means to represent it.
  (recall: RDF/RDFS "reasoning" works on the graph level)

OWL VOCABULARIES

- An OWL-DL vocabulary $\mathcal{V}$ is a 7-tuple (= a sorted vocabulary)
  $\mathcal{V} = (\mathcal{V}_{cls}, \mathcal{V}_{objprop}, \mathcal{V}_{dtprop}, \mathcal{V}_{annprop}, \mathcal{V}_{indiv}, \mathcal{V}_{DT}, \mathcal{V}_{lit})$:

- $\mathcal{V}_{cls}$ is the set of URIs denoting class names,
  <http://.../mondial/10/meta#Country>

- $\mathcal{V}_{objprop}$ is the set of URIs denoting object property names,
  <http://.../mondial/10/meta#capital>

- $\mathcal{V}_{dtprop}$ is the set of URIs denoting datatype property names,
  <http://.../mondial/10/meta#population>

- ($\mathcal{V}_{annprop}$ is the set of URIs denoting annotation property names,)

- $\mathcal{V}_{indiv}$ is the set of URIs denoting individuals, <http://.../mondial/10/countries/D>

- $\mathcal{V}_{DT}$ is the set of URIs denoting datatype names,
  <http://www.w3.org/2001/XMLSchema#int>

- $\mathcal{V}_{lit}$ is the set of literals;

- the builtin notions (=URIs) from RDF, RDFS, OWL namespaces do not belong to the vocabulary of the ontology (they are only used for describing the ontology in RDF).

OWL INTERPRETATIONS

Since DL is a subset of FOL, the interpretation of an OWL-DL vocabulary can be given as a FOL interpretation

$$\mathcal{I} = (I_{indiv} \cup I_{cls} \cup I_{objprop} \cup I_{dtprop} \cup I_{annprop} \cup I_{DT} , \mathcal{U}_{obj} \cup \mathcal{U}_{DT})$$

where I interprets the vocabulary as

- I_{indiv} constant symbols (individuals),

- I_{cls}, I_{DT} unary predicates (classes and datatypes),

- $I_{objprop}, I_{dtprop}, I_{annprop}$ binary predicates (properties),

and the universe $\mathcal{U}$ is partitioned into

- an *object domain* $\mathcal{U}_{obj}$

- and a data domain $\mathcal{U}_{DT}$ (of all values of datatypes).

OWL INTERPRETATIONS

The interpretation I is as follows:

I_{indiv}:   each individual $a \in \mathcal{V}_{indiv}$ to an object $I(a) \in \mathcal{U}_{obj}$,

(e.g., $I(\texttt{<http://.../mondial/10/countries/D>}) = \textit{germany}$)

I_{cls}:   each class $C \in \mathcal{V}_{cls}$ to a set $I(C) \subseteq \mathcal{U}_{obj}$,

(e.g., $\textit{germany} \in I(\texttt{<http://.../mondial/10/meta#Country>})$)

I_{DT}:   each datatype $D \in \mathcal{V}_{DT}$ to a set $I(D) \subseteq \mathcal{U}_{DT}$,

(e.g., $I(\texttt{<http://www.w3.org/2001/XMLSchema#int>}) = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$)

$I_{objprop}$:   each object property $p \in \mathcal{V}_{objprop}$ to a binary relation $I(p) \subseteq \mathcal{U}_{obj} \times \mathcal{U}_{obj}$,

(e.g., $(\textit{germany}, \textit{berlin}) \in I(\texttt{<http://.../mondial/10/meta#capital>})$)

I_{dtprop}:   each datatype property $p \in \mathcal{V}_{dtprop}$ to a binary relation $I(p) \subseteq \mathcal{U}_{obj} \times \mathcal{U}_{D}$,

(e.g., $(\textit{germany}, 83536115) \in I(\texttt{<http://.../mondial/10/meta#population>})$)

$I_{annprop}$:   each annotation property $p \in \mathcal{V}_{annprop}$ to a binary relation $I(p) \subseteq \mathcal{U} \times \mathcal{U}$.

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 about it.
  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 intersections of owl:Restrictions, classes having multiple such constraints can be specified.

OWL NOTIONS (CONT'D)

OWL Property Axioms (Overview)

- Distinction between owl:ObjectProperty and owl:DatatypeProperty

- 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 302)

OWL Individual Axioms (Overview)

- Individuals are modeled by unary classes

- owl:sameAs, owl:differentFrom, owl:AllDifferent($o_1, \ldots, o_n$).

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)	$\exists P.\{y\}$	$P(x, y)$
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	$\top \sqsubseteq \leq 1P.\top$	$\forall x, y_1, y_2 : P(x, y_1) \wedge P(x, y_2) \rightarrow y_1 = y_2$
InverseFunctionalProperty	$\top \sqsubseteq \leq 1P^-.\top$	$\forall x, y_1, y_2 : P(y_1, x) \wedge P(y_2, x) \rightarrow y_1 = y_2$

SYNTACTICAL REPRESENTATION

- OWL specifications can be represented by graphs: OWL constructs have a straightforward representation as triples in RDF/XML and N3.

- there are several logic-based representations (e.g. *Manchester OWL Syntax*); TERP (which can be used with pellet) is a combination of Turtle and Manchester syntax.

- 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 ... x_n)
  - as RDF lists: rdf:List, rdf:first, rdf:rest

REQUIREMENT

- every entity in an OWL ontology must be explicitly typed (i.e., as a class, an object property, a datatype property, ..., or an instance of some class).
(for reasons of space this is not always done in the examples; in general, it may lead to incomplete results)

QUERYING OWL DATA

- queries are atomic and conjunctive DL queries against the underlying OWL-DL model.

- this model can still be seen as a graph:
  - many of the edges are those known from the basic RDF graph
  - some edges (and collections) are only there for encoding OWL stuff (describing owl:unionOf, owl:propertyChain etc.) – these should not be queried

- SPARQL-DL is a subset of SPARQL: not every SPARQL query pattern is allowed for use on an OWL ontology
(but the reasonable ones are, so in practice this is not a problem.)

- the query language SPARQL-DL allows exactly such well-sorted patterns using the notions of OWL.

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 reasoner:

        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.

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]

EXAMPLE: UNION AND SUBCLASS

```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="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]
```

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_2 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.

SUBCLASSES OF PROPERTIES

Triple syntax: *some property* rdf:type *a specific type of property*

According to their ranges

- owl:ObjectProperty – subclass of rdf:Property; object-valued (i.e. rdfs:range must be an Object class)

- owl:DatatypeProperty – subclass of rdf:Property; datatype-valued (i.e. its rdfs:range must be an rdfs:Datatype)

$\Rightarrow$ OWL ontologies require each property to be typed in such a way!
(for reasons of space sometimes omitted in examples)

According to their Cardinality

- specifying n:1 or 1:n cardinality:
  owl:FunctionalProperty, owl:InverseFunctionalProperty

$\Rightarrow$ useful for deriving that objects must be different from each other.

According to their Properties

- owl:TransitiveProperty, owl:SymmetricProperty see later ...

FUNCTIONAL CARDINALITY SPECIFICATION

property rdf:type owl:FunctionalProperty

- not a constraint, but

- if such a property results in two things ... these things are inferred to be the same.

```
@prefix : <foo://bla/names#>.
@prefix family: <foo://bla/persons/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
  :world :has_pope family:josephratzinger .
  :world :has_pope [ :name "Benedikt XVI" ] .
  :has_pope rdf:type owl:FunctionalProperty.
```
[Filename: RDF/popes.n3]

```
prefix : <foo://bla/names#>
prefix family: <foo://bla/persons/>
select ?N  from <file:popes.n3>
where { family:josephratzinger :name ?N }
```
[Filename: RDF/pope.sparql]

OWL:RESTRICTION – EXAMPLE

- owl:Restriction for $\exists p.C$ and $\forall p.C$.   (cf. earlier examples)

```
<?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>
```

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

[Filename: RDF/restriction.rdf]

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:
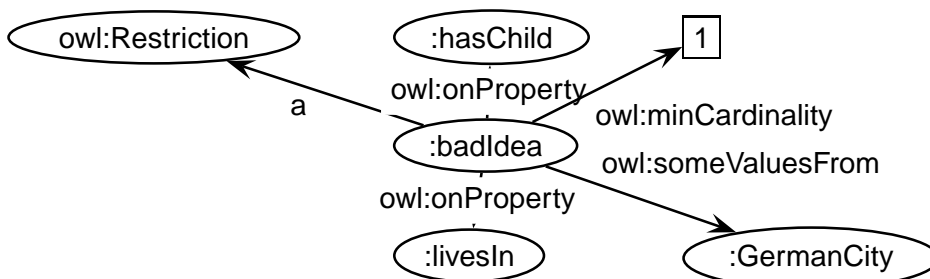
The two restriction specifications are messed up.

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:equivalentClass
 [ 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: TwoRestrictions $\equiv$ ($\exists$ hasChild.$\top$) $\sqcap$ ($\exists$ 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 assertive axiom: ABDE $\equiv \exists \geq 1$ hasChild.$\top \wedge$ ABDE $\sqsubseteq$ 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 someValuesFrom-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";                          [Filename: RDF/restriction-double.n3]
  :hasChild [a :Female; :name "Anne"], [a :Female; :name "Barbara"].
```

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

- The the someValuesFrom-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:equivalentClass
 [ 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";                    [Filename: RDF/intersect-restrictions.n3]
  :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]

- Note: this is different from Qualified Range Restrictions such as "All persons that have at least two male children" – see Slide 362.

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]

NON-EXISTENCE OF PROPERTY FILLERS

```
@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!

INVERSE PROPERTIES

- *owl:ObjectProperty* owl:inverseOf *owl:ObjectProperty*

- owl:DatatypeProperties cannot have an inverse
  (this would define properties of objects, cf. next slide)

```
@prefix : <foo://bla/names#> .
@prefix family: <foo://bla/persons/> .
@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#>.
  family:john :child family:alice, family:bob.
  family:john :parent family:kate .
  :descendant rdf:type owl:TransitiveProperty.
  :child rdfs:subPropertyOf :descendant.
  :child owl:inverseOf :parent.
```

```
prefix : <foo://bla/names#>
select ?X ?Y
from <file:inverse.n3>
where {?X :descendant ?Y}
```
[Filename: RDF/inverse.sparql]

[Filename: RDF/inverse.n3]

- an owl:DatatypeProperty must not have an inverse:

- ":john :age 35" would imply "35 :ageOf :john" which would mean that a literal has a property, which is not allowed.

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix : <foo://bla/names#> .
# :john :name "John"; :age 35; :child [:name "Alice"], [:name "Bob"; :age 8].
:age a owl:DatatypeProperty.
:child a rdf:Property.
:childOf owl:inverseOf :child.
:ageOf owl:inverseOf :age.
```
[Filename: RDF/inverseDTProp.n3]

```
jena -e -pellet -if inverseDTProp.n3
WARN [main] (OWLLoader.java:352) - Unsupported axiom:
Ignoring inverseOf axiom between foo://bla/names#ageOf (ObjectProperty)
  and foo://bla/names#age (DatatypeProperty)
```

SPECIFICATION OF INVERSE FUNCTIONAL PROPERTIES

- Mathematics: a mapping m is inverse-functional if the inverse of m is functional:
  $x \ p \ y$ is inverse-functional, if for every y, there is at most one x such that xpy holds.

- Example:
  - hasCarCode is functional: every country has one car code,
  - hasCarCode is also inverse functional: every car code uniquely identifies a country.

- OWL:
  ```
  :m-inverse owl:inverseOf :m .
  :m-inverse a owl:FunctionalProperty .
  ```
  not allowed for e.g. `mon:carCode a owl:DatatypeProperty`:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo:bla#>.
:carCode a owl:DatatypeProperty; rdfs:domain :Country;
  owl:inverseOf :isCarCodeOf.
# :Germany :carCode "D".                              [Filename: RDF/noinverse.n3]
```

- the statement is rejected.

OWL:INVERSEFUNCTIONALPROPERTY

- such cases are described with owl:InverseFunctionalProperty

- a property P is an owl:InverseFunctionalProperty if
  $\forall x, y_1, y_2 : P(y_1, x) \wedge P(y_2, x) \rightarrow y_1 = y_2$ holds

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo:bla#>.
:carCode rdfs:domain :Country; a owl:DatatypeProperty;
  a owl:FunctionalProperty; a owl:InverseFunctionalProperty.
:name a owl:DatatypeProperty; a owl:FunctionalProperty.
:Germany :carCode "D"; :name "Germany".
:DominicanRepublic :carCode "D"; :name "Dominican Republic".
```
[Filename: RDF/invfunctional.n3]

- the fragment is detected to be inconsistent.

OWL:HASKEY (OWL 2)

- description of key attributes $(k_1, \ldots, k_n)$ is a relevant issue in data modeling.
  Note that InverseFunctionalProperty covers the simple case that $n = 1$.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo:bla#>.
:name a owl:DatatypeProperty; a owl:FunctionalProperty.
:Country owl:hasKey (:carCode).
:Germany a :Country; :carCode "D"; :name "Germany".
:DominicanRepublic a :Country; :carCode "D"; :name "Dominican Republic".
:Duesseldorf  a :City; :carCode "D"; :name "Duesseldorf".
```
[Filename: RDF/haskey.n3]

- the fragment is inconsistent.

OWL:HASKEY (OWL 2)

- keys can also be used to detect that two resources (e.g. described by different Web sources) are actually the same:

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo:bla#>.
:Person owl:hasKey (:givenName :familyName).
 _:b1 a :Person; :firstName "John"; :familyName "Doe";  :age 32 .
 _:b2 a :Person; :firstName "John"; :familyName "Doe";  :address "Main Street 1" .
```
[Filename: RDF/haskey2.n3]

```
prefix : <foo:bla#>
select ?X ?P ?Y
from <file:haskey2.n3>
where {?X a :Person ; ?P ?Y}
```
[Filename: RDF/haskey2.sparql]

NAMED AND UNNAMED RESOURCES

(from the DL reasoner's perspective)

Named Resources

- resources with explicit global URIs
  <http://www.semwebtech.org/mondial/10/country/D>
  <foo://bla/bob>

- resources with local IDs/named blank nodes

- unnamed blank nodes

Unnamed (implicit) Resources

- things that exist only implicitly:
  John's child in

```
:Parent a owl:Class; owl:equivalentClass
   [ a owl:Restriction; owl:onProperty :child; owl:minCardinality 1].
:john a Parent.
```

- such resources can even have properties (see next slides).

- "every person has a father who is a person" and "john is a person".

- the *standard model* is *infinite*:
  john, john's father, john's father's father, ...

- pure RDF graphs are always finite,

- only with OWL axioms, one can specify such infinite models,

$\Rightarrow$ they have only finitely many *locally to path length n* different nodes,

- the reasoner can detect the necessary n ("blocking", cf. Slides 410 ff) and create "typical" different structures.

Aside: "standard model" vs "nonstandard model"

- the term "standard model" is not only "what we understand (in this case)", but is a notion of mathematical theory which –roughly– means "the simplest model of a specification"

- nonstandard models of the above are those where there is a cycle in the ancestors relation.
  (as the length of the cycle is arbitrary, this would not make it easier for the reasoner - there is only the possibility to have an owl:sameAs somewhere)

Implicit Resources

```
@prefix : <foo://bla/names#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
:Person owl:equivalentClass [a owl:Restriction;
  owl:onProperty :father; owl:someValuesFrom :Person].
:bob :name "Bob";  a :Person; :father :john.
:john :name "John";  a :Person.
```
[Filename: RDF/fathers-and-forefathers.n3]

```
prefix : <foo://bla/names#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?X ?F ?C
from <file:fathers-and-forefathers.n3>
where {{ ?X :father ?F } UNION { ?C a :Person }}
```
[Filename: RDF/fathers-and-forefathers.sparql]

- Reasoner: works on the model, including blocking, i.e. *modulo equivalence up to paths of length n*.

- SPARQL (and SWRL) rules: works on the graph – without the unnamed/implicit resorces.

7.3 RDF Graph vs. OWL Model; SPARQL vs. Reasoning

- SPARQL is an RDF (graph) query language

- OWL talks about models.

Consequences (Overview)

$\Rightarrow$ SPARQL queries are answered against the graph of triples

- Some OWL notions are directly represented by triples, such as c a owl:Class.

- Some others are directly supported by special handling in the reasoners,
  e.g., c rdfs:subClassOf d and c owl:equivalentClass d.

- some others are only "answered" when given explicitly in the RDF input! The results then
  do not incorporate further results that could be found by reasoning!

- OWL notions in the input are often not contained as triples, but are only translated into DL
  atoms for the reasoner. (e.g. owl:Restriction definitions)

- Most OWL notions in queries are not "understood" as OWL, but only matched.

- SPARQL answers are only concerned with the graph, not with implicit things that are only
  known in the model.

NOT REASONED: OWL:FUNCTIONALPROPERTY

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo:bla#>.
:p a owl:ObjectProperty; rdfs:domain :D.
:D owl:equivalentClass [ a owl:Restriction; owl:onProperty :p;
                                           owl:maxCardinality 1 ].
# :x :p :a, :b.    :a owl:differentFrom :b.
```
[Filename:RDF/functional.n3]

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <foo:bla#>
select ?P
from <file:functional.n3>
where {{ ?X :p ?Y } UNION {?P a owl:FunctionalProperty }}
```
[Filename:RDF/functional.sparql]

- SPARQL-DL (Sirin, Parsia OWLED 2007) is a proposal that allows certain OWL built-ins
  to be queried.

NOT ALLOWED: COMPLEX TERMS IN SPARQL QUERIES

- example: all cities that are a capital

- runs both with pellet and jena (Feb. 2013):

```
pellet query -query-file countrycaps.sparql \
        mondial-europe.n3 mondial-meta.n3 countrycaps.n3
```

```
@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]).        [Filename: RDF/countrycaps.n3]
```

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <http://www.semwebtech.org/mondial/10/meta#>
select ?N1 ?N2
where {{?X a :CountryCapital; :name ?N1} union
       {?Y a [a owl:Restriction; owl:onProperty :isCapitalOf;
              owl:someValuesFrom :Country]; :name ?N2}} [Filename:RDF/countrycaps.sparql]
```

- 53 answers, column ?N1 is filled, ?N2 is null.

NOT ALLOWED: COMPLEX TERMS IN SPARQL QUERIES (CONT'D)

- all organizations whose headquarter city is a capital:

- use pellet! jena does not support this (Feb. 2013):

```
pellet query -query-file organizations-query2.sparql \
   mondial-europe.n3 mondial-meta.n3
```

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <http://www.semwebtech.org/mondial/10/meta#>
select ?A ?H
where {?X a [ owl:intersectionOf
           (:Organization [a owl:Restriction; owl:onProperty :hasHeadq;
                        owl:someValuesFrom
                           [ a owl:Restriction; owl:onProperty :isCapitalOf;
                             owl:someValuesFrom :Country ] ] ) ];
      :abbrev ?A; :hasHeadq ?C . ?C :name ?H . }
```
[Filename:RDF/organizations-query2.sparql]

- 35 answers.

ANSWER SETS TO QUERIES AS AD-HOC CONCEPTS

- all organizations whose headquarter city is a capital:

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <http://www.semwebtech.org/mondial/10/meta#> .
:CountryCapital owl:equivalentClass
  [ owl:intersectionOf
    (:City [a owl:Restriction; owl:onProperty :isCapitalOf;
          owl:someValuesFrom :Country])].
<bla:Result>  owl:equivalentClass [ 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]

SPARQL ON THE GRAPH

- SPARQL does not return any answer related with nodes (=resources) that are only implicitly known (=non-named resources)

```
@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#>
:ParentOf12YOChild  owl:equivalentClass [a owl:Restriction;
  owl:onProperty :child; owl:someValuesFrom :12YOPerson].
:12YOPerson owl:equivalentClass [a owl:Restriction;
  owl:onProperty :age; owl:hasValue 12].
[ :name "John"; :age 35;  a :ParentOf12YOChild;
    :child [:name "Alice"; :age 10], [:name "Bob"; :age 8]].
:age rdf:type owl:FunctionalProperty.
# :12YOPerson owl:equivalentClass owl:Nothing.

:TwoChildrenParent  owl:equivalentClass [a owl:Restriction;
  owl:onProperty :child; owl:cardinality 2].
:ThreeChildrenParent  owl:equivalentClass [a owl:Restriction;
  owl:onProperty :child; owl:minCardinality 3].
```
[Filename: RDF/john-three-children-impl.n3]

- implicit resources exist only on the reasoning level,

- not considered by SPARQL queries:

```
prefix : <foo://bla/names#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?X ?C ?A ?T
from <file:john-three-children-impl.n3>
where {{ ?X :name "John" . ?X a ?C }
       UNION {?X :age ?A} UNION {?T a :12YOPerson}}
```

[Filename: RDF/john-three-children-impl.sparql]

- John is a ThreeChildrenParent,

- no person known who is 12 years old

- adding `:12YOPerson owl:equivalentClass owl:Nothing` makes it inconsistent.

- same applies to owl:hasKey (cf. Slides 316 and 328) and SWRL rules (cf. Slides 413 ff).

OWL:HASKEY AND NON-NAMED RESOURCES

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix : <foo:bla#>.
:XYThing owl:hasKey (:x :y).
:pointTo rdfs:range :XYThing.
:xy10 a :XYThing; :x 10; :y 10; :text "free".
:XYTen owl:intersectionOf
  ([ a owl:Restriction; owl:onProperty :x; owl:hasValue 10]
   [ a owl:Restriction; owl:onProperty :y; owl:hasValue 10]
   [ a owl:Restriction; owl:onProperty :text; owl:hasValue "pointedTo"]).
:foo a [ a owl:Restriction;
         owl:onProperty :pointTo; owl:onClass :XYTen; owl:qualifiedCardinality 1].
  # forces implicit existence of a node (10,10,"pointedTo").
# Make this implicit node named by forcing "another" node pointed to
# from :tenTen without any properties: via hasKey, xyxy = xy10
# :foo :pointTo :xyxy.                        [Filename: RDF/easykeys-impl.n3]
```

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <foo:bla#>
SELECT ?CT ?Y ?T ?SameAsxyxy
FROM <easykeys-impl.n3>
WHERE {{ :foo :pointTo [ :text ?CT ] }
        UNION { ?Y :text ?T }
        UNION { [:text ?T] }
        UNION { :xyxy owl:sameAs ?SameAsxyxy }}
```
[Filename: RDF/easykeys-impl.sparql]

- as long as the relevant node is only implicit (although quite some information about it is known), it is not considered in the answers.

329

[ASIDE] OWL VS. RDF LISTS

- RDF provides structures for representing lists by triples (cf. Slide 218): 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.

330

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)

- 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.

333

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]

334

7.4 Nominals: The O in SHOIQ

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 52.

```
@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.
```
[Filename: RDF/italian-english.n3]

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.

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 53.

```
@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]

```
@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>]).
```

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

[Filename: RDF/montanunion.n3]

- 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 of Benelux 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
```

```
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>}
```

[Filename: RDF/montanunion2.n3]

[RDF/montanunion2.sparql]

- "all organizations that cover *all* members of the Montanunion."

```
@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):

  $\forall Org : Org(c) \leftrightarrow \mathsf{hasMember}(Org, c)$

  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 ($x_1, \ldots, x_n$), 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 ($r_1\ r_2 \ldots r_n$)]

  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 $(x_1, \ldots, x_n)$, two items may be the same (open world),

- optional owl:AllDifferent to guarantee that $(x_1, \ldots, x_n)$ 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, :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".