Semantic Web
5. RDF-S and OWL

Gerd Gröner, Matthias Thimm
{groener,thimm}@uni-koblenz.de

Institute for Web Science and Technologies (WeST)
University of Koblenz-Landau

July 12, 2013
Recap

- The RDF Model
  - URIs, literals, blank nodes
  - statements, graphs
- Complex statements
  - Typed literals
  - Containers: sequences, bags, alternatives
  - Linked lists
  - Reification
- RDF encodings: Turtle, RDF/XML

Home assignment:
Represent the following statements in RDF (Home assignment):

Jack Smith studies at Koblenz University
Koblenz University has the web site http://www.uni-koblenz.de
Jack is a friend of Anna
Anna studies at Boston University
Boston University has the website http://www.bu.edu
Anna has a dog called “Snoopy”
Recap RDF vocabulary:

- `rdf:XMLLiteral`: XML literal values
- `rdf:Property`: class of properties
- `rdf:Statement`: class of RDF statements
- `rdf:Alt`, `rdf:Bag`, `rdf:Seq`: containers
- `rdf:List`: class of RDF Lists
- `rdf:nil`: the empty list
- `rdf:type`: type of an instance
- `rdf:first`: first item in a list
- `rdf:rest`: rest of a list
- `rdf:value`: for structured values
- `rdf:subject`: subject of a statement
- `rdf:predicate`: predicate of a statement
- `rdf:object`: object of a statement

What is missing?
RDF Schema

- RDFS: RDF Schema

- Extends the basic RDF vocabulary with meta-modeling capabilities

- RDFS defines a basic set of classes and properties, together with their semantics (interpretation) and logic

- Some elements:
  - Class, subClassOf, DataType
  - Property, subPropertyOf
  - Domain, range
RDF Schema

- RDFS: RDF Schema
- Extends the basic RDF vocabulary with *meta-modeling* capabilities
RDF Schema

- RDFS: RDF Schema
- Extends the basic RDF vocabulary with *meta-modeling* capabilities
- RDFS defines a basic set of classes and properties, together with their semantics (interpretation) and logic
RDFS: RDF Schema

- Extends the basic RDF vocabulary with *meta-modeling* capabilities
- RDFS defines a basic set of classes and properties, together with their semantics (interpretation) and logic

Some elements:
- Class, subClassOf, DataType
- Property, subPropertyOf
- Domain, range
OWL: Web Ontology Language
OWL: Web Ontology Language
Extends RDFS even further
OWL: Web Ontology Language
Extends RDFS even further
Incorporates aspects of description logics
OWL

- OWL: Web Ontology Language
- Extends RDFS even further
- Incorporates aspects of description logics
- Defines formal semantics for RDF
Outline

1. RDF Schema
2. OWL
3. Summary and Exercises
RDFS namespace:
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
RDFS Vocabulary 1/2

- RDFS namespace:
  \[\text{xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"}\]

- RDFS classes:
  - rdfs:Resource: everything is a resource
  - rdfs:Literal: the class of literal values
  - rdfs:Class: the class of classes
  - rdfs:Datatype: the class of datatypes
  - rdfs:Container: the class of containers
  - rdfs:ContainerMembershipProperty: rdf:_1, rdf:_2, ...
RDFS properties:

- `rdfs:subClassOf`: subclass of a class
- `rdfs:subPropertyOf`: subproperty of a property
- `rdfs:domain`: domain of a property
- `rdfs:range`: range of a property
- `rdfs:label`: the name
- `rdfs:comment`: a description of resource
- `rdfs:member`: a member of a resource
- `rdfs:seeAlso`: further information about a resource
- `rdfs:isDefinedBy`: the definition of a resource
RDFS Principles

- rdfs:Resource: all resources are instances of rdfs:Resource (like $\top$ in description logics):
  
  (ex:John, rdf:type, rdfs:Resource)
  (rdfs:Resource, rdf:type, rdfs:Resource)
RDFS Principles

- **rdfs:Resource**: all resources are instances of rdfs:Resource (like $\top$ in description logics):
  - (ex:John, rdf:type, rdfs:Resource)
  - (rdfs:Resource, rdf:type, rdfs:Resource)

- **rdfs:Class**: sets of resources, can be hierarchically structured (like concepts in description logics):
  - (ex:Bird, rdf:type, rdfs:Class)
  - (ex:Bird, rdfs:subClassOf, ex:Animal)
RDFS Principles

- **rdfs:Resource**: all resources are instances of rdfs:Resource (like $\top$ in description logics):
  
  (ex:John, rdf:type, rdfs:Resource)
  (rdfs:Resource, rdf:type, rdfs:Resource)

- **rdfs:Class**: sets of resources, can be hierarchically structured (like concepts in description logics):
  
  (ex:Bird, rdf:type, rdfs:Class)
  (ex:Bird, rdfs:subClassOf, ex:Animal)

- **rdfs:Property**: resources that link resources, can appear in the middle-position of triples (like relations in description logics):
  
  (ex:Color, rdf:type, rdfs:Class)
  (ex:hasColor, rdf:type, rdfs:Property)
  (ex:hasColor, rdfs:range, ex:Color)
  (ex:hasColor, rdfs:domain, rdfs:Resource)
Example

Image taken from Jos de Bruijn, Semantic Web Technologies course, 2008
RDF-S terms come with a well-defined meaning that allows limited *reasoning* on RDF.
RDF-S terms come with a well-defined meaning that allows limited *reasoning* on RDF.

Rules (informally):
- If $(X, \text{rdfs:domain}, Y)$ and $(U, X, V)$ then $(U, \text{rdf:type}, Y)$
RDF-S terms come with a well-defined meaning that allows limited *reasoning* on RDF.

Rules (informally):

- If \((X, \text{rdfs:domain}, Y)\) and \((U, X, V)\) then \((U, \text{rdf:type}, Y)\)
- If \((\text{ex:motherOf}, \text{rdfs:domain}, \text{ex:Human})\) and \((\text{ex:Mia}, \text{ex:motherOf}, \text{ex:Ann})\) then \((\text{ex:Mia}, \text{rdf:type}, \text{ex:Human})\)
RDF-S terms come with a well-defined meaning that allows limited reasoning on RDF.

Rules (informally):

- If \((X, \text{rdfs:domain}, Y)\) and \((U, X, V)\) then \((U, \text{rdf:type}, Y)\)
- If \((\text{ex:motherOf}, \text{rdfs:domain}, \text{ex:Human})\) and \((\text{ex:Mia}, \text{ex:motherOf}, \text{ex:Ann})\) then \((\text{ex:Mia}, \text{rdf:type}, \text{ex:Human})\)
- If \((X, \text{rdfs:range}, Y)\) and \((U, X, V)\) then \((V, \text{rdf:type}, Y)\)
RDF-S terms come with a well-defined meaning that allows limited reasoning on RDF.

Rules (informally):

- If $(X, \text{rdfs:domain}, Y)$ and $(U, X, V)$ then $(U, \text{rdf:type}, Y)$
  - If $(\text{ex:motherOf}, \text{rdfs:domain}, \text{ex:Human})$ and $(\text{ex:Mia}, \text{ex:motherOf}, \text{ex:Ann})$ then $(\text{ex:Mia}, \text{rdf:type}, \text{ex:Human})$

- If $(X, \text{rdfs:range}, Y)$ and $(U, X, V)$ then $(V, \text{rdf:type}, Y)$
  - If $(\text{ex:hasColor}, \text{rdfs:range}, \text{ex:Color})$ and $(\text{ex:hammer}, \text{ex:hasColor}, \text{ex:green})$ then $(\text{ex:green}, \text{rdf:type}, \text{ex:Color})$
Rules (informally, cont’d):

- If \((X, \text{rdfs:subPropertyOf}, Y)\) and \((Y, \text{rdfs:subPropertyOf}, Z)\) then \((X, \text{rdfs:subPropertyOf}, Z)\)
Semantics 2/3

- Rules (informally, cont’d):
  - If \((X, \text{rdfs:subPropertyOf}, Y)\) and \((Y, \text{rdfs:subPropertyOf}, Z)\) then \((X, \text{rdfs:subPropertyOf}, Z)\)
  - If \((\text{ex:brotherOf}, \text{rdfs:subPropertyOf}, \text{ex:SiblingOf})\) and \((\text{ex:SiblingOf}, \text{rdfs:subPropertyOf}, \text{ex:RelativeOf})\) then \((\text{ex:brotherOf}, \text{rdfs:subPropertyOf}, \text{ex:RelativeOf})\)
Rules (informally, cont’d):

- If \((X, \text{rdfs:subPropertyOf}, Y)\) and \((Y, \text{rdfs:subPropertyOf}, Z)\) then \((X, \text{rdfs:subPropertyOf}, Z)\)
- If \((\text{ex:brotherOf}, \text{rdfs:subPropertyOf}, \text{ex:SiblingOf})\) and \((\text{ex:SiblingOf}, \text{rdfs:subPropertyOf}, \text{ex:RelativeOf})\) then \((\text{ex:brotherOf}, \text{rdfs:subPropertyOf}, \text{ex:RelativeOf})\)
- If \((X, \text{rdfs:subPropertyOf}, Y)\) and \((U, X, V)\) then \((U, Y, V)\)
Rules (informally, cont’d):

- If $(X, \text{rdfs:subPropertyOf}, Y)$ and $(Y, \text{rdfs:subPropertyOf}, Z)$ then $(X, \text{rdfs:subPropertyOf}, Z)$
  - If $(\text{ex:brotherOf}, \text{rdfs:subPropertyOf}, \text{ex:SiblingOf})$ and $(\text{ex:SiblingOf}, \text{rdfs:subPropertyOf}, \text{ex:RelativeOf})$ then $(\text{ex:brotherOf}, \text{rdfs:subPropertyOf}, \text{ex:RelativeOf})$

- If $(X, \text{rdfs:subPropertyOf}, Y)$ and $(U, X, V)$ then $(U, Y, V)$
  - If $(\text{ex:Carl}, \text{ex:brotherOf}, \text{ex:Dave})$ and $(\text{ex:brotherOf}, \text{rdfs:subPropertyOf}, \text{ex:SiblingOf})$ then $(\text{ex:Carl}, \text{ex:SiblingOf}, \text{ex:Dave})$
Rules (informally, cont’d):

- If \((X, \text{rdfs:subClassOf}, Y)\) and \((Y, \text{rdfs:subClassOf}, Z)\) then \((X, \text{rdfs:subClassOf}, Z)\)
Rules (informally, cont’d):

- If \((X, \text{rdfs:subClassOf}, Y)\) and \((Y, \text{rdfs:subClassOf}, Z)\) then 
  \((X, \text{rdfs:subClassOf}, Z)\)

- If \((\text{ex:Cat}, \text{rdfs:subClassOf}, \text{ex:Mammal})\) and 
  \((\text{ex:Mammal}, \text{rdfs:subClassOf}, \text{ex:Animal})\) then 
  \((\text{ex:Cat}, \text{rdfs:subClassOf}, \text{ex:Animal})\)
▶ Rules (informally, cont’d):
▶ If \((X, \text{rdfs:subClassOf}, Y)\) and \((Y, \text{rdfs:subClassOf}, Z)\) then \((X, \text{rdfs:subClassOf}, Z)\)
  ▶ If \((\text{ex:Cat}, \text{rdfs:subClassOf}, \text{ex:Mammal})\) and \((\text{ex:Mammal}, \text{rdfs:subClassOf}, \text{ex:Animal})\) then \((\text{ex:Cat}, \text{rdfs:subClassOf}, \text{ex:Animal})\)
▶ If \((X, \text{rdfs:subClassOf}, Y)\) and \((U, \text{rdf:type}, X)\) then \((U, \text{rdf:type}, Y)\)
- Rules (informally, cont’d):
  - If \((X, \text{rdfs:subClassOf}, Y)\) and \((Y, \text{rdfs:subClassOf}, Z)\) then \((X, \text{rdfs:subClassOf}, Z)\)
    - If \((\text{ex:Cat}, \text{rdfs:subClassOf}, \text{ex:Mammal})\) and \((\text{ex:Mammal}, \text{rdfs:subClassOf}, \text{ex:Animal})\) then \((\text{ex:Cat}, \text{rdfs:subClassOf}, \text{ex:Animal})\)
  - If \((X, \text{rdfs:subClassOf}, Y)\) and \((U, \text{rdf:type}, X)\) then \((U, \text{rdf:type}, Y)\)
    - If \((\text{ex:Cat}, \text{rdfs:subClassOf}, \text{ex:Mammal})\) and \((\text{ex:Kitty}, \text{rdf:type}, \text{ex:Cat})\) then \((\text{ex:Kitty}, \text{rdf:type}, \text{ex:Mammal})\)
Axiomatic Triples

The following triples are assumed to be always true:

(rdf:type, rdfs:domain, rdfs:Resource)
(rdfs:domain, rdfs:domain, rdf:Property)
(rdfs:range, rdfs:domain, rdf:Property)
(rdfs:subPropertyOf, rdfs:domain, rdf:Property)
...
(rdf:type, rdfs:range, rdfs:Class)
(rdfs:domain, rdfs:range, rdfs:Class)
(rdfs:range, rdfs:range, rdfs:Class)
(rdfs:subPropertyOf, rdfs:range, rdf:Property)
(rdfs:subClassOf, rdfs:range, rdfs:Class)
...
(rdf:Alt, rdfs:subClassOf, rdfs:Container)
(rdf:Bag, rdfs:subClassOf, rdfs:Container)
...
Entailment

- If a triplestore (=data base for RDF triples) implements the RDF/RDFS entailment regime the previous rules are applied when answering queries.

Small example anyway: Querying 
(ex:Cat, rdfs:subClassOf, ex:Mammal) 
(ex:Kitty, rdf:type, ex:Cat) 
(ex:Dumbo, rdf:type, ex:Mammal) 

with "Get all mammals" returns Dumbo and Kitty.
Entailment

- If a triplestore (=data base for RDF triples) implements the RDF/RDFS entailment regime the previous rules are applied when answering queries
- more on queries later (→ SPARQL)
If a triplestore (=data base for RDF triples) implements the RDF/RDFS entailment regime the previous rules are applied when answering queries

more on queries later (→ SPARQL)

Small example anyway: Querying

(ex:Cat,rdfs:subClassOf,ex:Mammal)
(ex:Kitty, rdf:type, ex:Cat)
(ex:Dumbo, rdf:type, ex:Mammal)

with “Get all mammals” returns Dumbo and Kitty.
Outline

1. RDF Schema
2. OWL
3. Summary and Exercises
Limitations of RDFS

- RDFS has limited expressive power for several applications
  - No conditional range and domain restrictions
    - The range of hasChild should be Human when applied to humans and Elephant when applied to elephants
Limitations of RDFS

- RDFS has limited expressive power for several applications
  - No conditional range and domain restrictions
    - The range of hasChild should be Human when applied to humans and Elephant when applied to elephants
  - No existence/cardinality constraints
    - All instances of Person must have a mother
    - All instances of Person must have two parents
Limitations of RDFS

- RDFS has limited expressive power for several applications
  - No conditional range and domain restrictions
    - The range of `hasChild` should be `Human` when applied to humans and `Elephant` when applied to elephants
  - No existence/cardinality constraints
    - All instances of `Person` must have a mother
    - All instances of `Person` must have two parents
  - No constraints on properties
    - `Knows` is symmetric
    - `isAncestor` is transitive
The OWL family comprises actually three different languages:

- OWL Lite
  - Classification hierarchy
  - Simple constraints

- OWL DL
  - Maximal expressiveness
  - Still tractable

- OWL Full
  - Even higher expressiveness
  - Not tractable
The OWL family comprises actually three different languages:

- OWL Lite
  - Classification hierarchy
  - Simple constraints
- OWL DL
  - Maximal expressiveness
  - Still tractable
The OWL family comprises actually three different languages:

- **OWL Lite**
  - Classification hierarchy
  - Simple constraints
- **OWL DL**
  - Maximal expressiveness
  - Still tractable
- **OWL Full**
  - Even higher expressiveness
  - Not tractable
Features of OWL Language Layers

- OWL Lite
  - (sub)classes, individuals
  - (sub)properties, domain, range
  - conjunction
  - (in)equality
  - cardinality 0/1
  - datatypes
  - inverse, transitive, symmetric properties
  - someValuesFrom
  - allValuesFrom

- OWL DL
  - Negation
  - Disjunction
  - Full cardinality
  - Enumerated types
  - hasValue

- OWL Full
  - Meta-classes
  - More compatible with RDFS
OWL Full

- No restriction on vocabulary
  - use classes as instances
OWL Full

- No restriction on vocabulary
  - use classes as instances
- \( \text{owl:Class} \equiv \text{rdfs:Class} \)
OWL Full

- No restriction on vocabulary
  - use classes as instances
- owl:Class ≡ rdfs:Class
- RDF-style model theory
  - Reasoning using FOL engine
  - Semantics should correspond to OWL DL for restricted KBs
- based on $SROIQ$
Use of vocabulary restricted
- no classes as instances
- defined by abstract syntax
Use of vocabulary restricted
  ▶ no classes as instances
  ▶ defined by abstract syntax

owl:Class ⊏ rdfs:Class
OWL DL

- Use of vocabulary restricted
  - no classes as instances
  - defined by abstract syntax
- `owl:Class ⊏ rdfs:Class`
- Standard DL-based model theory
  - Direct correspondence with a DL
  - Reasoning via DL engines
- based on $SHOIN$
OWL Lite

- No explicit negation or union
- Restricted cardinality (0/1)
- No nominals (oneOf)
- DL-based semantics
  - Reasoning via DL engines
OWL Lite

- No explicit negation or union
- Restricted cardinality (0/1)
- No nominals (oneOf)
- DL-based semantics
  - Reasoning via DL engines
- `owl:Class ⊏ rdfs:Class`
OWL Lite

- No explicit negation or union
- Restricted cardinality (0/1)
- No nominals (oneOf)
- DL-based semantics
  - Reasoning via DL engines
- `owl:Class ⊏ rdfs:Class`
- Semantically, only small restriction on OWL DL
  - No nominals
  - no arbitrary cardinality
- based on $SHIF$
## OWL concepts

<table>
<thead>
<tr>
<th>OWL concept</th>
<th>DL</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersectionOf</td>
<td>$C_1 \sqcap \ldots \sqcap C_n$</td>
<td>$Actor \sqcap Politician$</td>
</tr>
<tr>
<td>unionOf</td>
<td>$C_1 \sqcup \ldots \sqcup C_n$</td>
<td>$Male \sqcup Female$</td>
</tr>
<tr>
<td>complementOf</td>
<td>$\neg C$</td>
<td>$\neg Student$</td>
</tr>
<tr>
<td>oneOf</td>
<td>${o_1, \ldots, o_n}$</td>
<td>${carl, dave}$</td>
</tr>
<tr>
<td>allValuesFrom</td>
<td>$\forall P.C$</td>
<td>$\forall hasChild.Female$</td>
</tr>
<tr>
<td>someValuesFrom</td>
<td>$\exists P.C$</td>
<td>$\exists hasChild.Female$</td>
</tr>
<tr>
<td>value</td>
<td>$\exists P.{v}$</td>
<td>$\exists hasColor.green$</td>
</tr>
<tr>
<td>minCardinality</td>
<td>$\geq_n P.C$</td>
<td>$\geq_3 hasChild.Male$</td>
</tr>
<tr>
<td>maxCardinality</td>
<td>$\leq_n P.C$</td>
<td>$\leq_2 hasChild.Male$</td>
</tr>
<tr>
<td>cardinality</td>
<td>$=_n P.C$</td>
<td>$=_3 hasChild.Male$</td>
</tr>
</tbody>
</table>
## OWL Axioms

<table>
<thead>
<tr>
<th>OWL Axiom</th>
<th>DL</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SubClassOf</td>
<td>$C_1 \sqsubseteq C_n$</td>
<td>$\text{Platypus} \sqsubseteq \text{Mammal} \cap \text{Oviparous}$</td>
</tr>
<tr>
<td>EquivalentClasses</td>
<td>$C_1 \equiv C_2$</td>
<td>$\text{Man} \equiv \text{Human} \cap \text{Male}$</td>
</tr>
<tr>
<td>SubPropertyOf</td>
<td>$P_1 \sqsubseteq P_2$</td>
<td>$\text{hasParent} \sqsubseteq \text{hasAncestor}$</td>
</tr>
<tr>
<td>EquivalentProperties</td>
<td>$P_1 \equiv P_2$</td>
<td>$\text{hasCost} \equiv \text{hasPrice}$</td>
</tr>
<tr>
<td>SameIndividual</td>
<td>$o_1 \equiv o_2$</td>
<td>$\text{jack} \equiv \text{jacksmith}$</td>
</tr>
<tr>
<td>DisjointClasses</td>
<td>$C_1 \sqsubseteq \neg C_2$</td>
<td>$\text{Male} \sqsubseteq \neg \text{Female}$</td>
</tr>
<tr>
<td>inverseOf</td>
<td>$P_1 \equiv P_2^-$</td>
<td>$\text{hasChild} \equiv \text{hasParent}^-$</td>
</tr>
<tr>
<td>Transitive</td>
<td>$P^+ \sqsubseteq P$</td>
<td>$\text{hasAncestor}^+ \sqsubseteq \text{hasAncestor}$</td>
</tr>
<tr>
<td>Symmetric</td>
<td>$P \equiv P^-$</td>
<td>$\text{knows} \equiv \text{knows}^-$</td>
</tr>
</tbody>
</table>
In OWL:

Class( associateProfessor partial academicStaffMember )
DisjointClasses( associateProfessor assistantProfessor )
DisjointClasses( professor associateProfessor )
Class( faculty complete academicStaffMember )

In DL:

associateProfessor ⊑ academicStaffMember
associateProfessor ⊑ ¬assistantProfessor
professor ⊑ ¬associateProfessor
faculty ≡ academicStaffMember
In OWL:

DatatypeProperty(age range(xsd:nonNegativeInteger))
SubPropertyOf(isTaughtBy involves)
ObjectProperty(teaches inverseOf(isTaughtBy)
  domain(academicStaffMember) range(course))

In DL:

\[ \top \sqsubseteq \forall \text{age . xsd:nonNegativeInteger} \]
\[ \text{isTaughtBy} \sqsubseteq \text{involves} \]
\[ \text{teaches} \equiv \text{isTaughtBy}^- \]
\[ \top \sqsubseteq \forall \text{teaches}^- . \text{academicStaffMember} \]
\[ \top \sqsubseteq \forall \text{teaches} . \text{course} \]
OWL in RDF

- OWL namespace:
  \[
  \text{xmlns:owl} = \text{http://www.w3.org/2002/07/owl}
  \]

- Class \(C\):
  \[
  (C, \text{rdf:type}, \text{owl:class})
  \]

- DisjointClasses \(C_1, C_2\):
  \[
  (C_1, \text{owl:disjointOf}, C_2)
  \]

- SymmetricObjectProperty \(P\):
  \[
  (P, \text{rdf:type}, \text{owl:SymmetricProperty})
  \]

- SameIndividual \(o_1, o_2\):
  \[
  (o_1, \text{owl:sameAs}, o_2)
  \]

- ObjectAllValuesFrom \(P, C\):
  \[
  (x, \text{rdf:type}, \text{owl:Restriction})
  \]
  \[
  (x, \text{owl:onProperty}, P)
  \]
  \[
  (x, \text{owl:allValuesFrom}, C)
  \]

- Example:
  \[
  (\text{ex:ParentOfOnlyDaughters}, \text{owl:equivalentClass}, x)
  \]
  \[
  (x, \text{rdf:type}, \text{owl:Restriction})
  \]
  \[
  (x, \text{owl:onProperty}, \text{ex:hasChild})
  \]
  \[
  (x, \text{owl:allValuesFrom}, \text{ex:Female})
  \]
OWL in RDF

- **OWL namespace:**
  xmlns:owl="http://www.w3.org/2002/07/owl"

- **OWL expressions can be represented in RDF**
  - Class(C) $\rightarrow$ (C, rdf:type, owl:class)
  - DisjointClasses(C1 C2) $\rightarrow$ (C1, owl:disjointOf, C2)
  - SymmetricObjectProperty(P) $\rightarrow$ (P, rdf:type, owl:SymmetricProperty)
  - SameIndividual(o1 o2) $\rightarrow$ (o1, owl:sameAs, o2)
  - ObjectAllValuesFrom(P C) $\rightarrow$
    - (x, rdf:type, owl:Restriction)
    - (x, owl:onProperty, P)
    - (x, owl:allValuesFrom, C)

  **Example:**
  (ex:ParentOfOnlyDaughters, owl:equivalentClass, x)
  (x, rdf:type, owl:Restriction)
  (x, owl:onProperty, ex:hasChild)
  (x, owl:allValuesFrom, ex:Female)
OWL in RDF

- OWL namespace:
  ```xml
  xmlns:owl="http://www.w3.org/2002/07/owl"
  ```

- OWL expressions can be represented in RDF:
  - Class(C) → (C, rdf:type, owl:class)
  - DisjointClasses(C1 C2) → (C1, owl:disjointOf, C2)
OWL in RDF

- OWL namespace:
  ```xml
  xmlns:owl="http://www.w3.org/2002/07/owl"
  ```

- OWL expressions can be represented in RDF:
  - Class(C) → (C, rdf:type, owl:class)
  - DisjointClasses(C1 C2) → (C1, owl:disjointOf, C2)
  - SymmetricObjectProperty(P) → (P, rdf:type, owl:SymmetricProperty)
    - Example:
      ```xml
      (ex:ParentOfOnlyDaughters, owl:equivalentClass, x)
      (x, rdf:type, owl:Restriction)
      (x, owl:onProperty, ex:hasChild)
      (x, owl:allValuesFrom, ex:Female)
      ```
OWL in RDF

- **OWL namespace:**
  xmlns:owl="http://www.w3.org/2002/07/owl"

- **OWL expressions can be represented in RDF**
  - Class(C) → (C, rdf:type, owl:class)
  - DisjointClasses(C1 C2) → (C1, owl:disjointOf, C2)
  - SymmetricObjectProperty(P) → (P, rdf:type, owl:SymmetricProperty)
  - SameIndividual(o1 o2) → (o1, owl:sameAs, o2)
OWL in RDF

- OWL namespace:
  \[\text{xmlns:owl} = "http://www.w3.org/2002/07/owl"\]

- OWL expressions can be represented in RDF
  - Class(C) \(\rightarrow\) (C, rdf:type, owl:class)
  - DisjointClasses(C1 C2) \(\rightarrow\) (C1, owl:disjointOf, C2)
  - SymmetricObjectProperty(P) \(\rightarrow\) (P, rdf:type, owl:SymmetricProperty)
  - SameIndividual(o1 o2) \(\rightarrow\) (o1, owl:sameAs, o2)
  - ObjectAllValuesFrom(P C) \(\rightarrow\)
    - (\(\_x\), rdf:type, owl:Restriction)
    - (\(\_x\), owl:onProperty, P)
    - (\(\_x\), owl:allValuesFrom, C)
OWL in RDF

- OWL namespace:
  \[\text{xmlns:owl} = \text{http://www.w3.org/2002/07/owl}\]

- OWL expressions can be represented in RDF

  - Class(C) → (C, rdf:type, owl:class)
  - DisjointClasses(C1 C2) → (C1, owl:disjointOf, C2)
  - SymmetricObjectProperty(P) → (P, rdf:type, owl:SymmetricProperty)
  - SameIndividual(o1 o2) → (o1, owl:sameAs, o2)
  - ObjectAllValuesFrom(P C) →
    - (_x, rdf:type, owl:Restriction)
    - (_x, owl:onProperty, P)
    - (_x, owl:allValuesFrom, C)

  Example:
  (ex:ParentOfOnlyDaughters, owl:equivalentClass, _x)
  (_x, rdf:type, owl:Restriction)
  (_x, owl:onProperty, ex:hasChild)
  (_x, owl:allValuesFrom, ex:Female)
OWL in RDF

- **OWL namespace:**
  xmlns:owl="http://www.w3.org/2002/07/owl"

- **OWL expressions can be represented in RDF**
  - Class(C) → (C, rdf:type, owl:class)
  - DisjointClasses(C1 C2) → (C1, owl:disjointOf, C2)
  - SymmetricObjectProperty(P) → (P, rdf:type, owl:SymmetricProperty)
  - SameIndividual(o1 o2) → (o1, owl:sameAs, o2)
  - ObjectAllValuesFrom(P C) →
    - (_x, rdf:type, owl:Restriction)
    - (_x, owl:onProperty, P)
    - (_x, owl:allValuesFrom, C)
  - Example:
    (ex:ParentOfOnlyDaughters, owl:equivalentClass, _x)
    (_x, rdf:type, owl:Restriction)
    (_x, owl:onProperty, ex:hasChild)
    (_x, owl:allValuesFrom, ex:Female)

...
Outline

1. RDF Schema
2. OWL
3. Summary and Exercises
Summary

- RDF Schema
  - Resources, classes, properties, ...
  - Sub-classes, domain, range, ...
  - Semantics and entailment

- OWL
  - OWL Lite, OWL DL, OWL Full
  - OWL in RDF


What can be entailed about “ex:tweety”?

(ex:tweety,rdf:type,ex:penguin)
(ex:penguin,rdfs:subClassOf,ex:bird)
(ex:bird,owl:disjointOf,ex:fish)
Exercises

▶ What can be entailed about “ex:tweety”?

(ex:tweety,rdf:type,ex:penguin)
(ex:penguin,rdfs:subClassOf,ex:bird)
(ex:bird,owl:disjointOf,ex:fish)

▶ Represent the following statement in RDF using RDFS and OWL:

Dave belongs to a group of people who have only sons and no sisters
Exercises

▶ What can be entailed about “ex:tweety”?
(ex:tweety,rdf:type,ex:penguin)
(ex:penguin,rdfs:subClassOf,ex:bird)
(ex:bird,owl:disjointOf,ex:fish)

▶ Represent the following statement in RDF using RDFS and OWL:

Dave belongs to a group of people who have only sons and no sisters

▶ Represent the following statement in RDF using RDFS and OWL:

Carl belongs to a group of people who like everyone who likes them
Exercises

➤ What can be entailed about “ex:tweety”?
   (ex:tweety,rdf:type,ex:penguin)
   (ex:penguin,rdfs:subClassOf,ex:bird)
   (ex:bird,owl:disjointOf,ex:fish)

➤ Represent the following statement in RDF using RDFS and OWL:
   *Dave belongs to a group of people who have only sons and no sisters*

➤ Represent the following statement in RDF using RDFS and OWL:
   *Carl belongs to a group of people who like everyone who likes them*

Not possible?