RDFS and OWL

1st Semantic Web Services Winter Retreat
Semantic Web Stack
Overview

- RDF Schema
  - Principles
  - Vocabulary
- RDF(S) Semantics
- Expressivity limitations of RDFS
- Web Ontology Language OWL
  - OWL Layering
  - OWL and Description Logics
  - OWL Syntaxes
How to represent the semantics of data models

THE RDF SCHEMA (RDFS)
• Types in RDF:
  `<#john, rdf:type, #Student>`
• What is a “#Student”?

• A language for defining RDF types:
  – Define classes:
    • “#Student is a class”
  – Relationships between classes:
    • “#Student is a sub-class of #Person”
  – Properties of classes:
    • “#Person has a property hasName”

• RDF Schema is such a language
• Classes:
  `<#Student, rdf:type, #rdfs:Class>`
• Class hierarchies:
  `<#Student, rdfs:subClassOf, #Person>`

• Properties:
  `<#hasName, rdf:type, rdf:Property>`
• Property hierarchies:
  `<#hasMother, rdfs:subPropertyOf, #hasParent>`

• Associating properties with classes (a):
  – “The property #hasName only applies to #Person”
    `<#hasName, rdfs:domain, #Person>`

• Associating properties with classes (b):
  – “The type of the property #hasName is #xsd:string”
    `<#hasName, rdfs:range, xsd:string>`
RDF Vocabulary Revisited

- **Classes:**
  - `rdf:Property`, `rdf:Statement`, `rdf:XMLLiteral`
  - `rdf:Seq`, `rdf:Bag`, `rdf:Alt`, `rdf:List`

- **Properties:**
  - `rdf:type`, `rdf:subject`, `rdf:predicate`, `rdf:object`,
  - `rdf:first`, `rdf:rest`, `rdf:_n`
  - `rdf:value`

- **Resources:**
  - `rdf:nil`
RDFS Vocabulary

- RDFS Extends the RDF Vocabulary
- RDFS vocabulary is defined in the namespace:

  http://www.w3.org/2000/01/rdf-schema#

### RDFS Classes
- rdfs:Resource
- rdfs:Class
- rdfs:Literal
- rdfs:Datatype
- rdfs:Container
- rdfs:ContainerMembershipProperty

### RDFS Properties
- rdfs:domain
- rdfs:range
- rdfs:subPropertyOf
- rdfs:subClassOf
- rdfs:member
- rdfs:seeAlso
- rdfs:isDefinedBy
- rdfs:comment
- rdfs:label
RDFS Principles

• **Resource**
  - All resources are implicitly instances of `rdfs:Resource`

• **Class**
  - Describe sets of resources
  - Classes are resources themselves - e.g. Webpages, people, document types
    • Class hierarchy can be defined through `rdfs:subClassOf`
    • Every class is a member of `rdfs:Class`

• **Property**
  - subset of RDFS Resources that are properties
    • **Domain**: class associated with property: `rdfs:domain`
    • **Range**: type of the property values: `rdfs:range`
    • Property hierarchy defined through: `rdfs:subPropertyOf`
RDFS Vocabulary Example

- rdfs:Resource
- rdfs:Class
  - rdfs:subclassOf
    - rdfs:subclassOf
      - rdfs:subclassOf
        - rdfs:subclassOf
          - rdfs:subclassOf
            - rdfs:subclassOf
              - rdfs:subclassOf
                - rdfs:subclassOf
                  - rdfs:subclassOf
                    - rdfs:subclassOf
                      - rdfs:subclassOf
• Metadata is “data about data”
• Any meta-data can be attached to a resource, using:
  – `rdfs:comment`
    • Human-readable description of the resource, e.g.
      `<ex:Person>, rdfs:comment, “A person is any human being”>`
  – `rdfs:label`
    • Human-readable version of the resource name, e.g.
      `<ex:Person>, rdfs:label, “Human being”>`
  – `rdfs:seeAlso`
    • Indicate additional information about the resource, e.g.
      `<ex:Person>, rdfs:seeAlso, <http://xmlns.com/wordnet/1.6/Human>>`
  – `rdfs:isDefinedBy`
    • A special kind of `rdfs:seeAlso`, e.g.
      `<ex:Person>, rdfs:isDefinedBy, <http://xmlns.com/wordnet/1.6/Human>>`
RDF Literals Revisited

- **Plain literals**
  - E.g. “any string”
  - Optional language tag, e.g. “Hello, how are you?”@en-GB

- **Typed literals**
  - E.g. "hello"^^xsd:string, "1"^^xsd:integer
  - Recommended datatypes:
    - XML Schema datatypes

- Only as *object* of a triple
• Each literal is an `rdfs:Literal`
• Say, we have: `<#john, #hasName, “John”>`
• Does this mean:
  `<“John”, rdf:type, rdfs:Literal>`
  – No! Literals may not occur as subject
• Add:
  – `<#john, #hasName, _:X>`
  – `<_:X, rdf:type, rdfs:Literal>`
Semantics

- RDF(S) vocabulary has built-in “meaning”
- RDF(S) Semantics
  - Makes meaning explicit
  - Defines what follows from an RDF graph
- Semantic notions
  - Subgraph
  - Instance
  - Entailment
Subgraph

- $E$ is a subgraph of $S$ if and only if $E$ predicates are a subset of $S$ predicates

\[
\langle \#\text{john}, \#\text{hasName}, _:\text{johnsname} \rangle \\
\langle _:\text{johnsname}, \#\text{firstName}, "John"^\text{xsd:string} \rangle \\
\langle _:\text{johnsname}, \#\text{lastName}, "Smith"^\text{xsd:string} \rangle
\]

- Subgraphs:

\[
\langle \#\text{john}, \#\text{hasName}, _:\text{johnsname} \rangle \\
\langle _:\text{johnsname}, \#\text{firstName}, "John"^\text{xsd:string} \rangle \\
\langle _:\text{johnsname}, \#\text{firstName}, "John"^\text{xsd:string} \rangle \\
\langle _:\text{johnsname}, \#\text{lastName}, "Smith"^\text{xsd:string} \rangle \\
\langle \#\text{john}, \#\text{hasName}, _:\text{johnsname} \rangle
\]
• *S*’ is an instance of *S* if and only if some blank nodes in *S* are replaced with blank nodes, literals or URIs

\[
\langle \#john, \#hasName, _\#johnsname \rangle \\
\langle _\#johnsname, \#firstName, "John"^xsd:string \rangle \\
\langle _\#johnsname, \#lastName, "Smith"^xsd:string \rangle
\]

• Instances:

\[
\langle \#john, \#hasName, \#abc \rangle \\
\langle \#abc, \#firstName, "John"^xsd:string \rangle \\
\langle \#abc, \#lastName, "Smith"^xsd:string \rangle
\]

\[
\langle \#john, \#hasName, _\#X \rangle \\
\langle _\#X, \#firstName, "John"^xsd:string \rangle \\
\langle _\#X, \#lastName, "Smith"^xsd:string \rangle
\]

\[
\langle \#john, \#hasName, _\#johnsname \rangle \\
\langle _\#johnsname, \#firstName, "John"^xsd:string \rangle \\
\langle _\#johnsname, \#lastName, "Smith"^xsd:string \rangle
\]

• Every graph is an instance of itself!
**Entailment**

- S entails E if E logically follows from S
  - Written: $S \models E$

- A graph entails all its subgraphs
  - If S’ is a subgraph of S: $S \models S'$

- All instances of a graph S entail S
  - If S” is an instance of S: $S'' \models S$
RDFS Entailment

\[
\begin{align*}
\text{entails} \\
\langle \text{http://example.org/#john} \rangle \text{ rdf:type } \langle \text{http://example.org/#Student} \rangle \\
\langle \text{http://example.org/#Student} \rangle \text{ rdfs:subClassOf } \langle \text{http://example.org/#Person} \rangle
\end{align*}
\]

\[
\begin{align*}
\text{entails} \\
\langle \text{http://example.org/#john} \rangle \text{ rdf:type } \langle \text{http://example.org/#Person} \rangle
\end{align*}
\]

\[
\begin{align*}
\langle \text{http://example.org/#hasName} \rangle \text{ rdfs:domain } \langle \text{http://example.org/#Student} \rangle \\
\langle \text{http://example.org/#mary} \rangle \text{ < http://example.org/#hasName> “Mary”}
\end{align*}
\]

\[
\begin{align*}
\text{entails} \\
\langle \text{http://example.org/#mary} \rangle \text{ rdf:type } \langle \text{http://example.org/#Student} \rangle
\end{align*}
\]

\[
\begin{align*}
\langle \text{http://example.org/#john} \rangle \text{ < http://example.org/#hasMother> } \langle \text{http://example.org/#mary} \rangle \\
\langle \text{http://example.org/#hasMother} \rangle \text{ rdfs:subPropertyOf } \langle \text{http://example.org/#hasParent} \rangle
\end{align*}
\]

\[
\begin{align*}
\text{entails} \\
\langle \text{http://example.org/#john} \rangle \text{ < http://example.org/#hasParent> } \langle \text{http://example.org/#mary} \rangle
\end{align*}
\]
RDFS Entailment

```
<http://example.org/#john> rdf:type <http://example.org/#Student>
<http://example.org/#Student> rdfs:subClassOf <http://example.org/#Person>
```

entails

```
<http://example.org/#john> rdf:type <http://example.org/#Person>
```

entails

```
<http://example.org/#hasName> rdfs:domain <http://example.org/#Student>
<http://example.org/#mary> <http://example.org/#hasName> “Mary”
```

entails

```
<http://example.org/#mary> rdf:type <http://example.org/#Student>
```

entails

```
<http://example.org/#john> <http://example.org/#hasMother> <http://example.org/#mary>
<http://example.org/#hasMother> rdfs:subPropertyOf <http://example.org/#hasParent>
```

entails

```
<http://example.org/#john> <http://example.org/#hasParent> <http://example.org/#mary>
```
RDFS Entailment

entails

<http://example.org/#john> rdf:type <http://example.org/#Student>
<http://example.org/#Student> rdfs:subClassOf <http://example.org/#Person>

entails

<http://example.org/#john> rdf:type <http://example.org/#Person>

entails

<http://example.org/#hasName> rdfs:domain <http://example.org/#Student>
<http://example.org/#mary> <http://example.org/#hasName> “Mary”

entails

<http://example.org/#john> <http://example.org/#hasMother> <http://example.org/#mary>
<http://example.org/#hasMother> rdfs:subPropertyOf <http://example.org/#hasParent>

entails

<http://example.org/#john> <http://example.org/#hasParent> <http://example.org/#mary>
Entailment Rules

- Semantics defined through *entailment rules*
- Rule:
  - If S contains \(<\text{triple pattern}>\) then add \(<\text{triple}>\)

- Executing all entailment rules yields *realization* of S

- S entails E if E is a subgraph of the realization of S

- Axiomatic triple are always added
RDF Axiomatic Triples

\[
\text{\texttt{<rdf:type, rdf:type, rdf:Property>}}
\]
\[
\text{\texttt{<rdf:subject, rdf:type, rdf:Property>}}
\]
\[
\text{\texttt{<rdf:predicate, rdf:type, rdf:Property>}}
\]
\[
\text{\texttt{<rdf:object, rdf:type, rdf:Property>}}
\]
\[
\text{\texttt{<rdf:first, rdf:type, rdf:Property>}}
\]
\[
\text{\texttt{<rdf:rest, rdf:type, rdf:Property>}}
\]
\[
\text{\texttt{<rdf:value, rdf:type, rdf:Property>}}
\]
\[
\text{\texttt{<rdf:_1, rdf:type, rdf:Property>}}
\]
\[
\text{\texttt{<rdf:_2, rdf:type, rdf:Property>}}
\]
\[
\text{\texttt{...}}
\]
\[
\text{\texttt{<rdf:nil, rdf:type, rdf:List>}}
\]
<rdf:type, rdfs:domain, rdfs:Resource>
<rdfs:domain, rdfs:domain, rdf:Property>
<rdfs:range, rdfs:domain, rdf:Property>
<rdfs:subPropertyOf, rdfs:domain, rdf:Property>
<rdfs:subClassOf, rdfs:domain, rdfs:Class>
<rdf:subject, rdfs:domain, rdf:Statement>
<rdf:predicate, rdfs:domain, rdf:Statement>
<rdf:object, rdfs:domain, rdf:Statement>
<rdfs:member, rdfs:domain, rdfs:Resource>
<rdf:first, rdfs:domain, rdf:List>
<rdf:rest, rdfs:domain, rdf:List>
<rdfs:seeAlso, rdfs:domain, rdfs:Resource>
<rdfs:isDefinedBy, rdfs:domain, rdfs:Resource>
<rdfs:comment, rdfs:domain, rdfs:Resource>
<rdfs:label, rdfs:domain, rdfs:Resource>
<rdfs:value, rdfs:domain, rdfs:Resource>
<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>
<rdfs:domain, rdfs:range, rdfs:Resource>
<rdfs:range, rdfs:range, rdfs:Resource>
<rdfs:member, rdfs:range, rdfs:Resource>
<rdf:first, rdfs:range, rdfs:Resource>
<rdf:rest, rdfs:range, rdf:List>
<rdfs:seeAlso, rdfs:range, rdfs:Resource>
<rdfs:isDefinedBy, rdfs:range, rdfs:Resource>
<rdfs:comment, rdfs:range, rdfs:Literal>
<rdfs:label, rdfs:range, rdfs:Literal>
<rdf:value, rdfs:range, rdfs:Resource>

<rdf:Alt, rdfs:subClassOf, rdfs:Container>
<rdf:Bag, rdfs:subClassOf, rdfs:Container>
<rdf:Seq, rdfs:subClassOf, rdfs:Container>
<rdfs:ContainerMembershipProperty, rdfs:subClassOf, rdf:Property>

<rdfs:isDefinedBy, rdfs:subPropertyOf, rdfs:seeAlso>

<rdf:XMLLiteral, rdf:type, rdfs:Datatype>
<rdf:XMLLiteral, rdfs:subClassOf, rdfs:Literal>
<rdfs:Datatype, rdfs:subClassOf, rdfs:Class>

<rdf:_1, rdf:type, rdfs:ContainerMembershipProperty>
<rdf:_1, rdfs:domain, rdfs:Resource>
<rdf:_1, rdfs:range, rdfs:Resource>
<rdf:_2, rdf:type, rdfs:ContainerMembershipProperty>
<rdf:_2, rdfs:domain, rdfs:Resource>
<rdf:_2, rdfs:range, rdfs:Resource>
...

RDF Entailment

- if E contains \(<A, B, C>\) then add \(<B, \text{rdf:}\text{type}, \text{rdf:}\text{Property}>\)

- if E contains \(<A, B, l>\) (l is a valid XML literal) then add \(<_:X, \text{rdf:}\text{type}, \text{rdf:}\text{XMLLiteral}>\)

where \(_:X\) identifies to blank node allocated to 1
everything in the subject is a resource:
if $E$ contains $<A, B, C>$
    then add $<A, \text{rdf:}\text{type}, \text{rdfs:Resource}>$

every non-literal in the object is a resource:
if $E$ contains $<A, B, C>$ (C is not a literal)
    then add $<C, \text{rdf:}\text{type}, \text{rdfs:Resource}>$

every class is subclass of rdfs:Resource:
if $E$ contains $<A, \text{rdf:}\text{type}, \text{rdfs:Class}>$
    then add $<A, \text{rdfs:subClassOf}, \text{rdfs:Resource}>$

inheritance:
if $E$ contains $<A, \text{rdf:}\text{type}, B>, <B, \text{rdfs:subClassOf}, C>$
    then add $<A, \text{rdf:}\text{type}, C>$

rdfs:subClassOf is transitive:
if $E$ contains $<A, \text{rdfs:subClassOf}, B>, <B, \text{rdfs:subClassOf}, C>$
    then add $<A, \text{rdfs:subClassOf}, C>$
**RDFS Entailment 2**

**rdfs:subClassOf is reflexive:**
if E contains $<A, \text{rdf:type}, \text{rdfs:Class}>$
then add $<A, \text{rdfs:subClassOf}, A>$

**rdfs:subPropertyOf is transitive:**
if E contains $<A, \text{rdfs:subPropertyOf}, B>, <B, \text{rdfs:subPropertyOf}, C>$
then add $<A, \text{rdfs:subPropertyOf}, C>$

**rdfs:subPropertyOf is reflexive:**
if E contains $<P, \text{rdf:type}, \text{rdf:Property}>$
then add $<P, \text{rdfs:subPropertyOf}, P>$

**domain of properties:**
if E contains $<P, \text{rdfs:domain}, C>, <A, P, B>$
then add $<A, \text{rdf:type}, C>$

**range of properties:**
if E contains $<P, \text{rdfs:range}, C>, <A, P, B>$
then add $<B, \text{rdf:type}, C>$
every literal is a member of rdfs:Literal:
if $E$ contains $<A, B, l>$ (l is a plain literal)
then add $<_:X, rdf:type, rdfs:Literal>$

every datatype is subclass of rdfs:Literal:
if $E$ contains $<A, rdf:type, rdfs:Datatype>$
then add $<A, rdfs:subClassOf, rdfs:Literal>$
More on literals

Recall:
if $E$ contains $<A, B, l>$ (l is a valid XML literal)
    then add $<_:X, \text{rdf:} \text{type}, \text{rdf:XMLLiteral}>$

every literal is a member of rdfs:Literal:
if $E$ contains $<A, B, l>$ (l is a plain literal)
    then add $<_:X, \text{rdf:} \text{type}, \text{rdfs:Literal}>$

---

allocating blank nodes to literals:
if $E$ contains $<A, B, l>$ (l is a literal)
    then add $<A, B, _:n>$
_:n is allocated to l

“dereferencing” blank nodes:
if $E$ contains $<A, B, _:n>$ (_:n is allocated to a literal l)
    then add $<A, B, l>$
Why do we need more expressive power?

LIMITATION OF RDFS
What we discussed so far...

- **RDF Schema**
  - RDFS Vocabulary
  - RDFS Metadata
  - Literals and Datatypes in RDFS

- **Semantics of RDF and RDF Schema**
  - Semantic notions
  - RDF(S) Entailment

- **SPARQL**
  - SPARQL Queries
  - Query Answer
What we discussed so far…

- RDF Schema
  - RDFS Vocabulary
  - RDFS Metadata
  - Literals and Datatypes in RDFS
- Semantics of RDF
  - Semantic notions
  - RDF(S) Entailment
- SPARQL
  - SPARQL Queries
  - Query Answer
What we discussed so far...

• RDF Schema
  – RDFS Vocabulary
  – RDFS Metadata
  – Literals and Datatypes in RDFS

• Semantics of RDF
  – Semantic notions
  – RDF(S) Entailment

• SPARQL
  – SPARQL Queries
  – Query Answer
What we discussed so far…

- RDF Schema
  - RDFS Vocabulary
  - RDFS Metadata
  - Literals and Datatypes in RDFS

- Semantics of RDF and RDF Schema
  - Semantic notions
  - RDF(S) Entailment

- SPARQL
  - SPARQL Queries
  - Query Answer
What we discussed so far...

- **RDF Schema**
  - RDFS Vocabulary
  - RDFS Metadata
  - Literals and Datatypes in RDF

- **Semantics of RDF and RDF**
  - Semantic notions
  - RDF(S) Entailment

- **SPARQL**
  - SPARQL Queries
  - Query Answer
What we discussed so far...

- **RDF Schema**
  - RDFS Vocabulary
  - RDFS Metadata
  - Literals and Datatypes in RDFS

- **Semantics of RDF and RDF Schema**
  - Semantic notions
  - RDF(S) Entailment

- **SPARQL**
  - SPARQL Queries
  - Query Answer
What we discussed so far…

• RDF Schema
  – RDFS Vocabulary
  – RDFS Metadata
  – Literals and Datatypes in RDFS

• Semantics of RDF and RDF Schema
  – Semantic notions
  – RDF(S) Entailment

• SPARQL
  – SPARQL Queries
  – Query Answer

PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?fullName
WHERE {?x vCard:FN ?fullName}
Requirements for Ontology Languages

• Well-defined syntax
• Convenience of expression
• Formal semantics
  – Needed in reasoning, e.g.:
    • Class membership
    • Equivalence of classes
    • Consistency
    • Classification
• Efficient reasoning support
• Sufficient expressive power
Limitations of the Expressive Power of RDF Schema

• Local scope of properties
  – rdfs:range defines the range of a property (e.g. eats) for all classes
  – In RDF Schema we cannot declare range restrictions that apply to some classes only
    • E.g. we cannot say that cows eat only plants, while other animals may eat meat, too
Limitations of the Expressive Power of RDF Schema (cont’)

• Disjointness of classes
  – Sometimes we wish to say that classes are disjoint (e.g. male and female)

• Boolean combinations of classes
  – Sometimes we wish to build new classes by combining other classes using union, intersection, and complement
    • E.g. person is the disjoint union of the classes male and female
Limitations of the Expressive Power of RDF Schema (cont’)

• Cardinality restrictions
  – E.g. a person has exactly two parents, a course is taught by at least one lecturer

• Special characteristics of properties
  – Transitive property (like “greater than”)
  – Unique property (like “is mother of”)
  – A property is the inverse of another property (like “eats” and “is eaten by”)

Other Limitations of RDFS

• No semantics for:
  – Containers
  – Collections
  – Reification

• Domain and range of property *infer* information rather than *check* data

• RDF/XML syntax very verbose
RDFS as an Ontology Language

• Classes
• Properties
• Class hierarchies
• Property hierarchies
• Domain and range restrictions
Expressiveness limitations of RDF(S)

- Only binary relations

- Characteristics of Properties
  - e.g. inverse, transitive, symmetric

- Local range restrictions
  - e.g. for class Person, the property hasName has range xsd:string

- Complex concept descriptions
  - e.g. Person is defined by Man and Woman

- Cardinality restrictions
  - e.g. a Person may have at most 1 name

- Disjointness axioms
  - e.g. nobody can be both a Man and a Woman
Stack of Languages

- XML
  - Surface syntax, no semantics
- XML Schema
  - Describes structure of XML documents
- RDF
  - Datamodel for “relations” between “things”
- RDF Schema
  - RDF Vocabulary Definition Language
- OWL
  - A more expressive Vocabulary Definition Language

This and following slides in part due to Frank van Harmelen
RDF Schema Recap

• RDFS provides
  – Classes
  – Class hierarchies
  – Properties
  – Property hierarchies
  – Domain and range restrictions

• RDFS does not provide
  – Property characteristics (inverse, transitive, ...)
  – Local range restrictions
  – Complex concept definitions
  – Cardinality restrictions
  – Disjointness axioms
Extending RDF Schema

• OWL extends RDF Schema to a full-fledged knowledge representation language for the Web
  – logical expressions (and, or, not)
  – (in)equality
  – local properties
  – required/optional properties
  – required values
  – enumerated classes
  – symmetry, inverse
Design Goals for OWL

• Shareable
• Changing over time
• Interoperability
• Inconsistency detection
• Balancing expressivity and complexity
• Ease of use
• Compatible with existing standards
• Internationalization
Requirements for OWL

• Ontologies are **object on the Web**
• with **their own meta-data**, versioning, etc...
• Ontologies are **extendable**

```xml
<owl:Ontology rdf:about="">
  <rdfs:comment>An example OWL ontology</rdfs:comment>
  <owl:priorVersion
    rdf:resource="http://www.mydomain.org/uni-ns-old"/>
  <owl:imports
    rdf:resource="http://www.mydomain.org/persons"/>
  <rdfs:label>University Ontology</rdfs:label>
</owl:Ontology>
```

*From Grigoris Antoniou and Frank van Harmelen: A Semantic Web Primer, MIT Press 2004*
Requirements for OWL

- Ontologies are **object on the Web**
- with **their own meta-data**, versioning, etc...
- Ontologies are **extendable**
- They contain **classes**, **properties**, **data-types**, **range/domain**, **individuals**
- **Equality** (for classes, for individuals)
- **Classes as instances**
- **Cardinality** constraints
- **XML** syntax
Language Layers of OWL

• **OWL Lite**
  – Classification hierarchy
  – Simple constraints

• **OWL DL**
  – Maximal expressiveness while maintaining tractability
  – Standard formalization in a DL

• **OWL Full**
  – Very high expressiveness
  – Losing tractability
  – All syntactic freedom of RDF (self-modifying)
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
• **No restriction on use of vocabulary** (as long as legal RDF)
  – Classes as instances (and much more)

• **RDF style model theory**
  – Reasoning using FOL engine
  – Semantics should correspond to OWL DL for restricted KBs
• Use of vocabulary restricted
  – No classes as instances
  – Defined by abstract syntax

• Standard DL-based model theory
  – Direct correspondence with a DL
  – Partial 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 (+ datatypes)

- Semantically, only small restriction on OWL DL
  - No nominals
  - No arbitrary cardinality
<table>
<thead>
<tr>
<th>OWL Construct</th>
<th>DL</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersectionOf</td>
<td>$C_1 \sqcap \ldots \sqcap C_n$</td>
<td>$Human \sqcap Male$</td>
</tr>
<tr>
<td>unionOf</td>
<td>$C_1 \sqcup \ldots \sqcup C_n$</td>
<td>$Doctor \sqcup Lawyer$</td>
</tr>
<tr>
<td>complementOf</td>
<td>$\neg C$</td>
<td>$\neg Male$</td>
</tr>
<tr>
<td>oneOf</td>
<td>${o_1, \ldots, o_n}$</td>
<td>${john, mary}$</td>
</tr>
<tr>
<td>allValuesFrom</td>
<td>$\forall P.C$</td>
<td>$\forall hasChild.Doctor$</td>
</tr>
<tr>
<td>someValuesFrom</td>
<td>$\exists P.C$</td>
<td>$\exists hasChild.Lawyer$</td>
</tr>
<tr>
<td>value</td>
<td>$\exists P.{o}$</td>
<td>$\exists citizenOf.USA$</td>
</tr>
<tr>
<td>minCardinality</td>
<td>$\geq nP.C$</td>
<td>$\geq 2 hasChild.Lawyer$</td>
</tr>
<tr>
<td>maxCardinality</td>
<td>$\leq nP.C$</td>
<td>$\leq 1 hasChild.Male$</td>
</tr>
<tr>
<td>cardinality</td>
<td>$= nP.C$</td>
<td>$= 1 hasParent.Female$</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>$C_1 \sqsubseteq C_2$</td>
<td>Human $\sqsubseteq$ Animal $\sqcap$ Biped</td>
</tr>
<tr>
<td>EquivalentClasses</td>
<td>$C_1 \equiv \ldots \equiv C_n$</td>
<td>Man $\equiv$ Human $\sqcap$ Male</td>
</tr>
<tr>
<td>SubPropertyOf</td>
<td>$P_1 \sqsubseteq P_2$</td>
<td>hasDaughter $\sqsubseteq$ hasChild</td>
</tr>
<tr>
<td>EquivalentProperties</td>
<td>$P_1 \equiv \ldots \equiv P_n$</td>
<td>cost $\equiv$ price</td>
</tr>
<tr>
<td>SameIndividual</td>
<td>$o_1 = \ldots = o_n$</td>
<td>President_Bush $=$ G_W_Bush</td>
</tr>
<tr>
<td>DisjointClasses</td>
<td>$C_i \sqsubseteq \neg C_j$</td>
<td>Male $\sqsubseteq \neg$ Female</td>
</tr>
<tr>
<td>DifferentIndividuals</td>
<td>$o_i \neq o_j$</td>
<td>john $\neq$ peter</td>
</tr>
<tr>
<td>inverseOf</td>
<td>$P_1 \equiv P_2^-$</td>
<td>hasChild $\equiv$ hasParent$^-$</td>
</tr>
<tr>
<td>Transitive</td>
<td>$P^+ \sqsubseteq P$</td>
<td>ancestor$^+$ $\sqsubseteq$ ancestor</td>
</tr>
<tr>
<td>Symmetric</td>
<td>$P \equiv P^-$</td>
<td>connectedTo$^+$ $\equiv$ connectedTo$^-$</td>
</tr>
</tbody>
</table>
Syntaxes of OWL

- **RDF**
  - Official exchange syntax
  - Hard for humans
  - RDF parsers are hard to write

- **UML**
  - Large user base

- **XML**
  - Not the RDF syntax
  - Better for humans
  - More XML than RDF tools available

- **Abstract syntax**
  - Not defined for OWL Full
  - Human readable
Example from [OwlGuide]:

```xml
<!ENTITY vin "http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#" >
<!ENTITY food "http://www.w3.org/TR/2004/REC-owl-guide-20040210/food#" >
...  
         xmlns:food="http://www.w3.org/TR/2004/REC-owl-guide-20040210/food#"  
         ... >
    <owl:Class rdf:ID="Wine">  
      <rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>  
      <rdfs:label xml:lang="en">wine</rdfs:label>  
      <rdfs:label xml:lang="fr">vin</rdfs:label>  
      ...  
    </owl:Class>  

    <owl:Class rdf:ID="Pasta">  
      <rdfs:subClassOf rdf:resource="#EdibleThing" />  
      ...  
    </owl:Class>  
</rdf:RDF>
```
Classes

- Classes are defined using **owl:Class**
  - **owl:Class** is a subclass of **rdfs:Class**
- Disjointness is defined using **owl:disjointWith**

```xml
<owl:Class rdf:about="#associateProfessor">
  <owl:disjointWith rdf:resource="#professor"/>
  <owl:disjointWith rdf:resource="#assistantProfessor"/>
</owl:Class>
```
Classes (2)

- **owl:equivalentClass** defines equivalence of classes

  <owl:Class rdf:ID="faculty">
    <owl:equivalentClass rdf:resource="#academicStaffMember"/>
  </owl:Class>

  - **owl:Thing** is the most general class, which contains everything
  - **owl:Nothing** is the empty class
Properties

• In OWL there are two kinds of properties
  – **Object properties**, which relate objects to other objects
    • E.g. is-TaughtBy, supervises
  – **Data type properties**, which relate objects to datatype values
    • E.g. phone, title, age, etc.
Datatype Properties

• OWL makes use of XML Schema data types, using the layered architecture of the SW

  <owl:DatatypeProperty rdf:ID="age">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
#nonNegativeInteger"/>
  </owl:DatatypeProperty>
• User-defined data types

<owl:ObjectProperty rdf:ID="isTaughtBy">
  <owl:domain rdf:resource="#course"/>
  <owl:range rdf:resource="#academicStaffMember"/>
  <rdfs:subPropertyOf rdf:resource="#involves"/>
</owl:ObjectProperty>
Inverse and Equivalent Properties

<owl:ObjectProperty rdf:ID="teaches">
  <rdfs:range rdf:resource="#course"/>
  <rdfs:domain rdf:resource="#academicStaffMember"/>
  <owl:inverseOf rdf:resource="#isTaughtBy"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="lecturesIn">
  <owl:equivalentProperty rdf:resource="#teaches"/>
</owl:ObjectProperty>
Property Restrictions

- In OWL we can declare that the class C satisfies certain conditions
  - All instances of C satisfy the conditions
- This is equivalent to saying that C is subclass of a class C', where C' collects all objects that satisfy the conditions
  - C' can remain anonymous
Property Restrictions (2)

- A (restriction) class is achieved through an `owl:Restriction` element.
- This element contains an `owl:onProperty` element and one or more `restriction declarations`.
- One type defines `cardinality restrictions` (at least one, at most 3,...)
The other type defines restrictions on the kinds of values the property may take

- `owl:allValuesFrom` specifies universal quantification
- `owl:hasValue` specifies a specific value
- `owl:someValuesFrom` specifies existential quantification
<owl:Class rdf:about="#firstYearCourse">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isTaughtBy"/>
      <owl:allValuesFrom rdf:resource="#Professor"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#mathCourse">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isTaughtBy"/>
      <owl:hasValue rdf:resource="#949352"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#academicStaffMember">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#teaches"/>
      <owl:someValuesFrom rdf:resource="#undergraduateCourse"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
Cardinality Restrictions

- We can specify minimum and maximum number using `owl:minCardinality` and `owl:maxCardinality`.
- It is possible to specify a precise number by using the same minimum and maximum number.
- For convenience, OWL offers also `owl:cardinality`.
Cardinality Restrictions (2)

<owl:Class rdf:about="#course">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isTaughtBy"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
Special Properties

- **owl:TransitiveProperty** (transitive property)
  - E.g. “has better grade than”, “is ancestor of”
- **owl:SymmetricProperty** (symmetry)
  - E.g. “has same grade as”, “is sibling of”
- **owl:FunctionalProperty** defines a property that has at most one value for each object
  - E.g. “age”, “height”, “directSupervisor”
- **owl:InverseFunctionalProperty** defines a property for which two different objects cannot have the same value
<owl:ObjectProperty rdf:ID="hasSameGradeAs">  
  <rdf:type rdf:resource="&owl;TransitiveProperty"/>  
  <rdf:type rdf:resource="&owl;SymmetricProperty"/>  
  <rdfs:domain rdf:resource="#student"/>  
  <rdfs:range rdf:resource="#student"/>  
</owl:ObjectProperty>
Boolean Combinations

- We can combine classes using Boolean operations (union, intersection, complement)

```xml
<owl:Class rdf:about="#course">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:complementOf rdf:resource="#staffMember"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```
<owl:Class rdf:ID="peopleAtUni">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#staffMember"/>
    <owl:Class rdf:about="#student"/>
  </owl:unionOf>
</owl:Class>
<owl:Class rdf:ID="facultyInCS">
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#faculty"/>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#belongsTo"/>
            <owl:hasValue rdf:resource="#CSDepartment"/>
        </owl:Restriction>
    </owl:intersectionOf>
</owl:Class>
Nesting of Boolean Operators

<owl:Class rdf:ID="adminStaff">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#staffMember"/>
    <owl:complementOf>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#faculty"/>
        <owl:Class rdf:about="#techSupportStaff"/>
      </owl:unionOf>
    </owl:complementOf>
  </owl:intersectionOf>
</owl:Class>
Enumerations with owl:oneOf

<owl:oneOf rdf:parseType="Collection">
  <owl:Thing rdf:about="#Monday"/>
  <owl:Thing rdf:about="#Tuesday"/>
  <owl:Thing rdf:about="#Wednesday"/>
  <owl:Thing rdf:about="#Thursday"/>
  <owl:Thing rdf:about="#Friday"/>
  <owl:Thing rdf:about="#Saturday"/>
  <owl:Thing rdf:about="#Sunday"/>
</owl:oneOf>
Declaring Instances

- Instances of classes are declared as in RDF:

```xml
<rdf:Description rdf:ID="949352">
    <rdf:type rdf:resource= "#academicStaffMember"/>
</rdf:Description>
<academicStaffMember rdf:ID="949352">
    <uni:age rdf:datatype="&xsd;integer"> 39<uni:age>
</academicStaffMember>
```
No Unique-Names Assumption

• OWL does not adopt the unique-names assumption of database systems
  – If two instances have a different name or ID does not imply that they are different individuals

• Suppose we state that each course is taught by at most one staff member, and that a given course is taught by two staff members
  – An OWL reasoner does not flag an error
  – Instead it infers that the two resources are equal
Distinct Objects

- To ensure that different individuals are indeed recognized as such, we must explicitly assert their inequality:

  <lecturer rdf:about="949318">
    <owl:differentFrom rdf:resource="949352"/>
  </lecturer>
Distinct Objects (2)

- OWL provides a shorthand notation to assert the pairwise inequality of all individuals in a given list

```xml
<owl:allDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
    <lecturer rdf:about="949318"/>
    <lecturer rdf:about="949352"/>
    <lecturer rdf:about="949111"/>
  </owl:distinctMembers>
</owl:allDifferent>
```
Versioning Information

- **owl:priorVersion** indicates earlier versions of the current ontology
  - No formal meaning, can be exploited for ontology management
- **owl:versionInfo** generally contains a string giving information about the current version, e.g. keywords
OWL Abstract syntax

Class(professor partial)
Class(associateProfessor partial academicStaffMember)

DisjointClasses(associateProfessor assistantProfessor)
DisjointClasses(professor associateProfessor)

Class(faculty complete academicStaffMember)
In DL syntax:

\[
\begin{align*}
\text{associateProfessor} & \sqsubseteq \text{academicStaffMember} \\
\text{associateProfessor} & \sqsubseteq \neg \text{assistantProfessor} \\
\text{professor} & \sqsubseteq \neg \text{associateProfessor} \\
\text{faculty} & \equiv \text{academicStaffMember}
\end{align*}
\]
More examples

DatatypeProperty(age range(xsd:nonNegativeInteger))
ObjectProperty(lecturesIn)

ObjectProperty(isTaughtBy domain(course)
range(academicStaffMember))
SubPropertyOf(isTaughtBy involves)

ObjectProperty(teaches inverseOf(isTaughtBy)
domain(academicStaffMember)
range(course))

EquivalentProperties(lecturesIn teaches)

ObjectProperty(hasSameGradeAs Transitive Symmetric
domain(student)
range(student))
More Examples

`Individual(949318  type(lecturer))`

`Individual(949352  type(academicStaffMember)
value(age "39"^^&xsd;integer))`

`ObjectProperty(isTaughtBy Functional)`

`Individual(CIT1111  type(course)
value(isTaughtBy 949352)
value(isTaughtBy 949318))`

`DifferentIndividuals(949318 949352)`
`DifferentIndividuals(949352 949111 949318)`
More Examples

Class(firstYearCourse partial
      restriction(isTaughtBy allValuesFrom (Professor)))

Class(mathCourse partial
      restriction(isTaughtBy hasValue (949352)))

Class(academicStaffMember partial
      restriction (teaches someValuesFrom (undergraduateCourse)))

Class(course partial
      restriction (isTaughtBy minCardinality (1)))

Class(department partial
      restriction (hasMember minCardinality(10))
      restriction (hasMember maxCardinality(30)))
More Examples

Class(course partial complementOf(staffMember))

Class(peopleAtUni complete unionOf(staffMember student))

Class(facultyInCS complete intersectionOf(faculty
restriction (belongsTo hasValue (CSDepartment))))

Class(adminStaff complete intersectionOf(staffMember
complementOf(unionOf(faculty techSupportStaff)))))
That’s almost all for day…

WRAP-UP
Things to keep in mind
(or summary)

• RDF
  – Reuse existing standards/tools
  – Standard format
  – Verbose
Things to keep in mind (or summary)

• RDF Schema
  – Advantages
    • A primitive ontology language
    • Offers certain modeling primitives with fixed meaning
    • Key concepts of RDF Schema
      – subclass relations, property, subproperty relations, domain and range restrictions
    • There exist query languages for RDF and RDFS
    • Allows metamodeling
  – Disadvantages
    • A quite primitive as a modeling language for the Web
    • Many desirable modeling primitives are missing
      – An ontology layer on top of RDF/RDFS is needed
Things to keep in mind
(or summary)

- Web Ontology Language OWL
  - OWL Layering
  - OWL and Description Logics
  - OWL Syntaxes
New languages underway

- **RDFa**
  - Integration of HTML world and Semantic Web
    - Means for "embedding" RDF-based annotation on traditional Web pages
    - Means for generating RDF triple stores from (annotated) Web pages

- **RIF**
  - Rules interchange format
    - Representing rules on the Web
    - Linking rule-based systems together

- **And more**
  - Multimedia annotation, Web-page Metadata annotation, Health Care and Life Science, Privacy
Semantic Web Tools

- **Browsers**
  - mSpace, Longwell, OINK, BrownSauce, Piggy Bank, Tabulator, etc

- **Annotators**
  - Annotea, Clipmarks, PhotoStuff, M-OntoMat-Annotizer, KIM, WSMT

- **Storages**
  - Oracle Spatial 10g, Kowari, Jena, Yars, 3Store, AllegroGraph, Joseki, ARC RDF Store

- **Ontology Mappers**
  - OntoMerge, HMARFA, CMS

- **Reasoners**
  - BOR, Bossam, FaCT++, Jess, OWLJessKB, RacerPro

- **Composite Applications/Frameworks**
  - Cerbera, Corse, IODT, Jena, TopBraid Composer, KAON
Bibliography

• Mandatory reading
  – Semantic Web Primer
    • Chapter 3 (only Sections 3.1 to 3.6)

• Further reading
  – RDF Primer
    • http://www.w3.org/TR/REC-rdf-syntax/
  – RDF Vocabulary Description Language 1.0: RDF Schema
    • http://www.w3.org/TR/rdf-schema/
Further Reading

• Mandatory reading
  – Semantic Web Primer
    • Chapters 4
  – [OWL Guide]
    • [http://www.w3.org/TR/owlguide/](http://www.w3.org/TR/owlguide/)
    • [http://www.cs.vu.nl/%7Efrankh/abstracts/JWS03.html](http://www.cs.vu.nl/%7Efrankh/abstracts/JWS03.html)

• Further reading
  – [OWL Reference]
    • [http://www.w3.org/TR/owl-ref/](http://www.w3.org/TR/owl-ref/)
  – [OWL Abstract syntax and Semantics]
    • [http://www.w3.org/TR/owl-semantics](http://www.w3.org/TR/owl-semantics)