Kap. 11: Semantic Web


**Semantic Web**

In diesem Kapitel betrachten wir, wie Wissensrepräsentation aus dem World Wide Web ein Semantic Web machen kann.

Das Semantic Web ist eine Vision von Tim Berners-Lee, dem Erfinder des WWW.

Die Kernidee ist die explizite Repräsentation von Wissen im WWW, so dass es von Such- und anderen Maschinen verwendet werden kann.


Unter http://www.semanticwebprimer.org sind Folien, Beispiele etc. online zu finden. Auf diesem Foliensatz basieren auch die nachfolgenden Abschnitte.

**XML**

- eXtensible Markup Language
- Web standard (W3C) for data exchange:
  - Description of in- and output data of applications
  - Reduces degrees of freedom for industrial data description standards
- Complementary to HTML:
  - HTML describes presentation
  - XML describes content
- Database perspective: XML as data model for semi-structured data.

**The semantic web layer cake**

Universal Resource Identifiers

e.g. http://www.somedomain.org/thispage#thistag

XML + NS + xsmlschema

Unicode

URI

Digital Signature

Trust

Proof

Logic

Ontology vocabulary

RDF + rdfschema

Data

Rules

Self-desc. doc.
XML syntax (1) - XML element

XML element:
- Description of an object, which is embraced by matching tags like `<author>` and `</author>.
- Content of an element: Text and/or other (sub)elements.
- Elements can be nested
- Elements can be empty: `<year></year>` (short: `<year/>`)

Start tag

subelements

- `<firstname>` Bernhard `</firstname>`
- `<lastname>` Ganter `</lastname>`
- `<email>` ganter@tu-dresden.de `</email>`

email address may be wrong!

End tag

Element author

The Tree Model of XML Documents: An Example

The tree representation of an XML document is an ordered labeled tree:
- There is exactly one root
- There are no cycles
- Each non-root node has exactly one parent
- Each node has a label.
- The order of elements is important
- ... but the order of attributes is not important

The tree model:

```
<email>
  <from name="Michael Maher"
       address="michaelmaher@cs.gu.edu.au"/>
  <to name="Grigoris Antoniou"
       address="grigoris@cs.unibremen.de"/>
  <subject>Where is your draft?</subject>
</head>
<body>
  Grigoris, where is the draft of the paper you promised me last week?
</body>
</email>
```
Path Expressions in XPath

XPath is core for XML query languages.
Language for addressing parts of an XML document.
- It operates on the tree data model of XML.
- It has a non-XML syntax.

Examples
- Address all books with title “Artificial Intelligence”
  /book[@title="Artificial Intelligence"]
- Address the first author element node in the XML document
  //author[1]
- Address the last book element within the first author element node in the document
  //author[1]/book[last()]
- Address all book element nodes without a title attribute
  //book[not(@title)]

XML Schema

Complex language for data description:
- Many standardised base types, e.g. float, double, decimal, boolean
  in particular: string and integer
- Types and typed references
- Class hierarchy and inheritance
- Consistency constraints

Standard („W3C Recommendation“) as extension to XML

Namespaces

An XML document may use more than one DTD or schema
Prefixes are used to avoid name clashes.
Prefixes have URIs as values.
They usually point to a description of the namespace syntax.

Example:

```xml
<vu:instructors xmlns:vu="http://www.vu.com/empDTD"
                 xmlns:gu="http://www.gu.au/empDTD"
                 xmlns:unik="http://www.unik.de/empDTD">
  <unik:dozent unik:title="Dr."
                unik:name="Andreas Hotho"
                unik:department="Computer Science"/>
  <gu:academicStaff gu:title="lecturer"
                   gu:name="Mate Jones"
                   gu:school="Information Technology"/>
</vu:instructors>
```

Style Sheets

Move data and metadata from one XML representation to another,
e.g., when applications that use different DTDs or schemas need to communicate.

The *extensible stylesheet language* XSL includes
- a transformation language (XSLT)
- a formatting language

XSLT specifies rules with which an input XML document is transformed to
- another XML document
- an HTML document
- plain text

The output document may use the same DTD or schema, or a completely different vocabulary.
XSLT can be used independently of the formatting language.
The semantic web layer cake

RDF Recommendation consists of several parts
- RDF Primer (http://www.w3.org/TR/rdf-primer/)
- RDF Schema (http://www.w3.org/TR/rdf-schema/)
- ...

RDF – Resource Description Framework

- RDF provides metadata about web resources
- key component: Object -> Attribute -> Value triple
- Interconnected triples constitute a labelled graph
- RDF uses XML syntax

 http://www.aifb.uni-karlsruhe.de/WBS/ysu

W3C York 6086592

RDF Schema

- RDFS defines vocabulary for RDF
- Vocabulary is organised as type hierarchy
  - Class, subClassOf
  - type
  - Property, subPropertyOf
  - domain, range

RDF Schema syntax in XML

```
<rdf:Description ID="Person">
  <rdf:type resource="http://www.w3.org/...#Class"/>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/...#Resource"/>
</rdf:Description>

<rdf:Description ID="Teacher">
  <rdf:type resource="http://www.w3.org/...#Class"/>
  <rdfs:subClassOf rdf:resource="#Person"/>
</rdf:Description>

<rdf:Description ID="teaches">
  <rdf:type resource="http://www.w3.org/...#Property"/>
  <rdfs:domain rdf:resource="#Teacher"/>
  <rdfs:range rdf:resource="#Student"/>
</rdf:Description>

<rdf:Description ID="teaches well">
  <rdf:type resource="http://www.w3.org/...#Property"/>
  <rdfs:subPropertyOf rdf:resource="#teaches"/>
</rdf:Description>
```
RDFS - logical

**Classes:**
- **unary predicates**
- **subClassOf relation:** implication
  \[(\forall x) \ (\text{tutor}(x) \rightarrow \text{student}(x))\]

**Properties:**
- **binary predicates**
- **subPropertyOf relation:** implication
  \[(\forall x)(\forall y) \ (\text{supervises}(x,y) \rightarrow \text{responsibleFor}(x,y))\]

**RDF statements are triples** *(Object, Property, Object)*
- Objects can be:
  - **URLs**
  - **classes**
  - **properties**
  - **triples**
- **triples(!)** --- (reification, i.e. second-order)

RDF Schema as ontology language

RDFS(S) is useful for simple ontologies, but not for complex modelling
→ „Need for more expressivity!“

More expressive languages:
- **OWL** (based on description logics)
- **F-Logic** (based on logic programming)
- Hybrids and rules extensions for OWL

The Semantic Web layer cake

OWL - general info

- **W3C Recommendation since 2004**
- **Semantic fragment of FOL** (First-order predicate logic)
- **Three variants:** OWL Lite \(\subseteq\) OWL DL \(\subseteq\) OWL Full
- **RDFS is fragment of OWL Full.**
- **No reification in OWL DL.**
- **OWL DL is decidable**
- **OWL DL = SHOIN(D)**
Head of a document

Classes, roles and Individuals

Class relationships

Complex class definitions
  ▪ Boolean class constructors
  ▪ Role restrictions

Role properties

OWL documents

OWL documents are RDF Documents.

They consist of
  ▪ Head with general information
  ▪ Rest with the ontology

Head of an OWL document

Definition of namespaces in the root

```xml
<rdf:RDF
  xmlns="http://www.semanticweb-grundlagen.de/beispielontologie#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#">
  ...
</rdf:RDF>
```

General information

```xml
<owl:Ontology rdf:about=""/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">SWRC Ontology December 2005</rdfs:comment>
  <owl:versionInfo>v0.5</owl:versionInfo>
  <owl:imports rdf:resource="http://www.semanticweb-grundlagen.de/foo"/>
</owl:Ontology>
```
Classes, roles and individuals

Basic components of OWL ontologies:

Classes
■ like resources in RDFS
■ like classes in DL

Individuals
■ like resources in RDFS
■ like individuals in DL

Roles
■ like attributes in RDFS
■ like roles in DL

Classes

Definition
<owl:Class rdf:ID="Professor"/>

■ predefined:
  ▪ owl:Thing
  ▪ owl:Nothing

Individuals

Definition by class membership
<rdf:Description rdf:ID="GerdStumme">
  <rdf:type rdf:resource="#Professor"/>
</rdf:Description>

■ equivalent:
  <Professor rdf:ID="GerdStumme"/>

Abstract Roles

Abstract roles defined like classes
<owl:ObjectProperty
  rdf:ID="Affiliation"/>

Domain and Range of abstract Roles
<owl:ObjectProperty rdf:ID="Affiliation">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Organisation"/>
</owl:ObjectProperty>

Domain: ∃Affiliation. T ⊑ Person
Range: T ⊑ ∀Affiliation.Organisation
Concrete Roles

Concrete roles have datatypes in range
<owl:DatatypeProperty rdf:ID="firstname"/>

Domain and range of concrete roles
<owl:DatatypeProperty rdf:ID="firstname">
  <rdfs:domain rdf:resource="#Person" />  
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

One can use many XML Schema Datatypes.  
The standard requires at least integer and string.

Individuals and Roles

<Person rdf:ID="GerdStumme">
  <Affiliation rdf:resource="#KDE"/>
  <Affiliation rdf:resource="#L3S"/>
  <firstname rdf:datatype="&xsd;string">Gerd</firstname>
</Person>

Roles are in general not functional.

OWL - contents

Head of a document  
Classes, roles and Individuals

Class relationships

Complex class definitions
- Boolean class constructors
- Role restrictions

Role properties

Simple class relationships

<owl:Class rdf:ID="Professor">
  <rdfs:subClassOf rdf:resource="#Faculty"/>
</owl:Class>

<owl:Class rdf:ID="Faculty">
  <rdfs:subClassOf rdf:resource="#Person"/>
</owl:Class>

It can be inferred that Professor is a subclass of Person.
Simple class relationships

We infer that Professor and Book are also disjoint classes.

We infer that Book is a subclass of Publikation.

Individuals and class relationships

We infer that FcaMethodenAnwendungen is a Publication.

We infer that ProfessorStumme is a Professor.

Inequality of individuals expressed using owl:differentFrom.

Inequality of individuals expressed using owl:differentFrom.
Relationships between individuals

<owl:AllDifferent>
  <owl:distinctMembers
    rdf:parseType="Collection">
    <Person rdf:about="#GerdStumme"/>
    <Person rdf:about="#DominikBenz"/>
    <Person rdf:about="#AndreasHotho"/>
  </owl:distinctMembers>
</owl:AllDifferent>

Shortcut for multiple usage of owl:differentFrom.

Closed classes (nominals)

<owl:Class rdf:about="#KdeTechAdminStaff">
  <owl:oneOf rdf:parseType="Collection">
    <Person rdf:about="#MonikaVopicka"/>
    <Person rdf:about="#SvenStefani"/>
  </owl:oneOf>
</owl:Class>

There are exactly those two Individuals in the class KdeTechAdminStaff.

OWL – contents

Head of a document
Classes, roles and Individuals
Class relationships
Complex class definitions
  - Boolean class constructors
  - Role restrictions
Role properties

Boolean class constructors

Conjunction:
  owl:intersectionOf
Disjunction:
  owl:unionOf
Negation:
  owl:complementOf

Can be used to construct complex classes from class names.
**Conjunction**

\[ \text{KdeTechAdminStaff} \equiv \text{TechAdminStaff} \cap \text{MemberOfKDE} \]

\[
\text{<owl:Class rdf:about="#KdeTechAdminStaff">
  \text{<owl:equivalentClass>}
    \text{<owl:intersectionOf rdf:parseType="Collection">}
      \text{<owl:Class rdf:about="#TechAdminStaff"/>}
      \text{<owl:Class rdf:about="#MemberOfKDE"/>}
    \text{</owl:intersectionOf>}
  \text{</owl:equivalentClass>}
\text{</owl:Class>}
\]

We infer that all individuals in KdeTechAdminStaff are also in TechAdminStaff.

**Negation**

\[ \text{Faculty} \cap \text{Publication} \equiv \bot \]

\[
\text{<owl:Class rdf:about="#Faculty">
  \text{<owl:subClassOf>}
    \text{<owl:complementOf rdf:resource="#Publication"/>}
  \text{</owl:subClassOf>}
\text{</owl:Class>}
\]

This is a complicated way of saying the following:

\[
\text{<owl:Class rdf:about="#Faculty">
  \text{<owl:disjointWith rdf:resource="#Publication"/>}
\text{</owl:Class>}
\]

**Disjunction**

\[ \text{Professor} \subseteq \text{activeTeacher} \cup \text{retired} \]

\[
\text{<owl:Class rdf:about="#Professor">
  \text{<owl:subClassOf>}
    \text{<owl:unionOf rdf:parseType="Collection">}
      \text{<owl:Class rdf:about="#activeTeacher"/>}
      \text{<owl:Class rdf:about="#retired"/>}
    \text{</owl:unionOf>}
  \text{</owl:subClassOf>}
\text{</owl:Class>}
\]

**OWL – contents**

Head of a document
Classes, roles and Individuals
Class relationships
Complex class definitions
- Boolean class constructors
- Role restrictions
Role properties
Role restrictions (allValuesFrom)

Using roles for defining complex classes

<owl:Class rdf:ID="Exam">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasExaminer"/>
      <owl:allValuesFrom rdf:resource="#Professor"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

I.e. all examiners of an exam must be professors.

Role restrictions (someValuesFrom)

<owl:Class rdf:about="#Exam">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasExaminer"/>
      <owl:someValuesFrom rdf:resource="#Person"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

I.e. each exam must have at least one examiner.

Number restrictions (cardinalities)

<owl:Class rdf:about="#Exam">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasExaminer"/>
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">2</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

An exam must have at most two examiners.

<owl:Class rdf:about="#Exam">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasTopic"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">3</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

An exam must cover at least three topics.
Number restrictions (cardinalities)

```xml
<owl:Class rdf:about="#Exam">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasTopic"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">3</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

An exam must cover exactly three topics.

Role restrictions (hasValue)

```xml
<owl:Class rdf:ID="examProfStumme">
  <rdfs:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasExaminer"/>
      <owl:hasValue rdf:resource="#GerdStumme"/>
    </owl:Restriction>
  </rdfs:equivalentClass>
</owl:Class>
```

`owl:hasValue` always points to an individual. This is equivalent to the example on the next slide.

OWL - contents

Head of a document
Classes, roles and Individuals
Class relationships
Complex class definitions
- Boolean class constructors
- Role restrictions
Role properties
Role relationships

\[
\text{hasExaminer} \subseteq \text{hasParticipant}
\]

<owl:ObjectProperty rdf:ID="\text{hasExaminer}"
<rdfs:subPropertyOf rdf:resource="#\text{hasParticipant}"/>
</owl:ObjectProperty>

Similar: \text{owl:equivalentProperty}

Roles can be inverse to each other:
<owl:ObjectProperty rdf:ID="\text{hasExaminer}"
<owl:inverseOf rdf:resource="#\text{examinerOf}"/>
</owl:ObjectProperty>

\[
\text{hasExaminer} \equiv \text{examinerOf}
\]

Role properties

- **Domain**
- **Range**
- **Transitivity** i.e. (a, b) and r(b, c) implies r(a, c)
- **Symmetry** i.e. r(a, b) implies r(b, a)
- **Functionality** i.e. r(a, b) and r(a, c) implies b = c
- **Inverse functionality**

Domain and Range

<owl:ObjectProperty rdf:ID="\text{Affiliation}"
<rdfs:range rdf:resource="#\text{Organisation}"/>
</owl:ObjectProperty>

Is equivalent to the following:
<owl:Class rdf:about="\&\text{owl;Thing}"
<rdfs:subClassOf
<owl:Restriction
<owl:onProperty rdf:resource="#\text{Affiliation}"/>
<owl:allValuesFrom rdf:resource="#\text{Organisation}"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

Range: \text{T} \subseteq \forall \text{Affiliation.Organisation}

Domain and Range

<owl:ObjectProperty rdf:ID="\text{Affiliation}"
<rdfs:range rdf:resource="#\text{Organisation}"/>
</owl:ObjectProperty>

<Number rdf:ID="\text{Five}"
<Affiliation rdf:resource="#\text{PrimeNumber}"/>
</Number>

It follows that \text{PrimeNumber} is an Organisation!

\text{T} \subseteq \forall \text{Affiliation.Organisation}

\text{Number(Five), Affiliation(Five,PrimeNumber)}
Role properties

<owl:ObjectProperty rdf:ID="hasColleague">
  <rdf:type rdf:resource="&owl;TransitiveProperty"/>
  <rdf:type rdf:resource="&owl;SymmetricProperty"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasProjectLeader">
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="isProjectLeaderFor">
  <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
</owl:ObjectProperty>

<Person rdf:ID="AndreasHotho">
  <hasColleague rdf:resource="#DominikBenz"/>
  <hasColleague rdf:resource="#BeateKrause"/>
  <isProjectLeaderFor rdf:resource="#BibSonomy"/>
</Person>

<Projekt rdf:ID="BibSonomy">
  <hasProjectLeader rdf:resource="#AndreasHotho"/>
  <hasProjectLeader rdf:resource="#GerdStumme"/>
</Projekt>

Logical consequences from example

BeateKrause hasColleague AndreasHotho

BeateKrause hasColleague DominikBenz

AndreasHotho owl:sameAs GerdStumme

BeateKrause hasColleague GerdStumme

The Semantic Web layer cake

OWL variants

OWL Full
- Contains OWL DL and OWL Lite
- Contains all of RDFS
- Undecidable
- Limited support by existing software

OWL DL (= SHOIN(D))
- Contains OWL Lite and is contained in OWL Full
- Decidable
- Tools available
- Complexity NExpTime (worst-case)

OWL Lite (= SHIF(D))
- Is contained in OWL DL and OWL Full
- Decidable
- Less expressive
- Complexity ExpTime (worst-case)
Unlimited usage of all OWL and RDFS constructs (must be valid RDFS).

Difficult is e.g. the non-existent type separation (Classes, Roles, Individuals), hence:
- `owl:Thing` is the same as `rdfs:resource`
- `owl:Class` is the same as `rdfs:Class`
- `owl:DatatypeProperty` subclass of `owl:ObjectProperty`
- `owl:ObjectProperty` is the same as `rdf:Property`

Why types are not seperated in OWL Full

```xml
<owl:Class rdf:about="#Book">
  <germanName rdf:datatype="&xsd;string">
    Buch
  </germanName>
  <frenchName rdf:datatype="&xsd;string">
    livre
  </frenchName>
</owl:Class>
```

One often does not really need inferencing over such information.

OWL DL

- is SHOIN(D).
- Allowed are only certain RDFS constructs (like those in the examples.
  Not allowed: `rdfs:Class`, `rdfs:Property`
- Type separation. Classes and Roles must be declared explicitly.
- Concrete Roles must not be transitive, symmetric, inverse or inverse functional.
- Number restrictions must not be used with transitive roles, their subroles, or their inverses.

OWL Lite

- is SHIF(D).
- All restrictions for OWL DL apply.
- Not allowed: `oneOf`, `unionOf`, `complementOf`, `hasValue`, `disjointWith`
- Number restrictions only allowed with 0 and 1.
- Some restrictions on the occurrence of anonymous (complex) classes apply, e.g. they must not occur in the subject of `rdfs:subClassOf`.
Editors
- Protégé, http://protege.stanford.edu

Inference engines
- KAON2, http://kaon2.semanticweb.org
- FACT++, http://owl.man.ac.uk/factplusplus/

Appendix: OWL language components

Head
- rdfs:comment
- rdfs:label
- rdfs:seeAlso
- rdfs:isDefinedBy
- owl:versionInfo
- owl:priorVersion
- owl:backwardCompatibleWith
- owl:incompatibleWith
- owl:DeprecatedClass
- owl:DeprecatedProperty
- owl:imports

Class constructors and relationships
- owl:Class
- owl:Thing
- owl:Nothing
- rdfs:subClassOf
- owl:disjointWith
- owl:equivalentClass
- owl:intersectionOf
- owl:unionOf
- owl:complementOf

Role restrictions
- owl:allValuesFrom
- owl:someValuesFrom
- owl:hasValue
- owl:cardinality
- owl:minCardinality
- owl:maxCardinality
- owl:oneOf

Role constructors, relations and properties
- owl:ObjectProperty
- owl:DatatypeProperty
- rdfs:subPropertyOf
- owl:equivalentProperty
- owl:InverseOf
- rdfs:domain
- rdfs:range
- rdf:resource="&owl;TransitiveProperty"
- rdf:resource="&owl;SymmetricProperty"
- rdf:resource="&owl;FunctionalProperty"
- rdf:resource="&owl;InverseFunctionalProperty"

Relations between individuals
- owl:sameAs
- owl:differentFrom
- owl:AllDifferent
  (together with
  owl:distinctMembers)

Required datatypes
- xsd:strong
- xsd:integer
### OWL as DL: Class Constructors

<table>
<thead>
<tr>
<th>Constructor</th>
<th>DL Syntax</th>
<th>Example</th>
<th>FOL Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersectionOf</td>
<td>$C_1 \cap \ldots \cap C_n$</td>
<td>$\text{Human} \cap \text{Male}$</td>
<td>$C_1(x) \land \ldots \land C_n(x)$</td>
</tr>
<tr>
<td>unionOf</td>
<td>$C_1 \cup \ldots \cup C_n$</td>
<td>$\text{Doctor} \cup \text{Lawyer}$</td>
<td>$C_1(x) \lor \ldots \lor C_n(x)$</td>
</tr>
<tr>
<td>complementOf</td>
<td>$\sim C$</td>
<td>$\text{~Male}$</td>
<td>$\neg C(x)$</td>
</tr>
<tr>
<td>oneOf</td>
<td>${x_1 \cup \ldots \cup {x_n}$</td>
<td>${\text{john} \cup {\text{mary}}$</td>
<td>$x = x_1 \lor \ldots \lor x = x_n$</td>
</tr>
<tr>
<td>allValuesFrom</td>
<td>$\exists P.C$</td>
<td>$\forall P.C$</td>
<td>$\forall y. P(x, y) \rightarrow C(y)$</td>
</tr>
<tr>
<td>someValuesFrom</td>
<td>$\forall P.C$</td>
<td>$\exists P.C$</td>
<td>$\exists y. P(x, y) \land C(y)$</td>
</tr>
<tr>
<td>maxCardinality</td>
<td>$\leq n P$</td>
<td>$\leq 1 \text{hasChild}$</td>
<td>$\exists y. P(x, y)$</td>
</tr>
<tr>
<td>minCardinality</td>
<td>$\geq n P$</td>
<td>$\geq 2 \text{hasChild}$</td>
<td>$\exists \exists y. P(x, y)$</td>
</tr>
</tbody>
</table>

**XMLS datatypes as well as classes in $\forall P.C$ and $\exists P.C$**
- E.g., `hasAge:nonNegativeInteger`

**Arbitrarily complex nesting of constructors**
- E.g., `Person \cap \exists \text{hasChild}\cdot\text{Doctor} \cup \exists \text{hasChild}\cdot\text{Doctor}`

### RDFS Syntax

E.g., `Person \cap \exists \text{hasChild}\cdot\text{Doctor} \cup \exists \text{hasChild}\cdot\text{Doctor}`:

```xml
<owl:Class>
  <owl:intersectionOf rdf:parseType="collection">
    <owl:Class rdf:about="#Person"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasChild"/>
      <owl:toClass>
        <owl:unionOf rdf:parseType="collection">
          <owl:Class rdf:about="#Doctor"/>
          <owl:Restriction>
            <owl:onProperty rdf:resource="#hasChild"/>
            <owl:hasClass rdf:resource="#Doctor"/>
          </owl:Restriction>
        </owl:unionOf>
      </owl:toClass>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

### OWL as DL: Axioms

<table>
<thead>
<tr>
<th>Axiom</th>
<th>DL Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>subClassOf</td>
<td>$C_1 \subseteq C_2$</td>
<td>Human $\subseteq$ Animal $\cap$ Biped</td>
</tr>
<tr>
<td>equivalentClass</td>
<td>$C_1 = C_2$</td>
<td>Man = Human $\cap$ Male</td>
</tr>
<tr>
<td>disjointWith</td>
<td>$C_1 \cap \sim C_2$</td>
<td>Male $\subseteq$ ~Female</td>
</tr>
<tr>
<td>sameIndividualAs</td>
<td>${x_1} \equiv {x_2}$</td>
<td>President.Bush $\equiv$ {G.W.Bush}</td>
</tr>
<tr>
<td>differentFrom</td>
<td>${x_1} \subseteq {x_2}$</td>
<td>{john} $\subseteq$ {peter}</td>
</tr>
<tr>
<td>subPropertyOf</td>
<td>$P_1 \subseteq P_2$</td>
<td>hasDaughter $\subseteq$ hasChild</td>
</tr>
<tr>
<td>equivalentProperty</td>
<td>$P_1 \equiv P_2$</td>
<td>cost $\equiv$ price</td>
</tr>
<tr>
<td>inverseOf</td>
<td>$P_1 \equiv P_2$</td>
<td>hasChild $=$ hasParent$^{-}$</td>
</tr>
<tr>
<td>transitiveProperty</td>
<td>$P^+ \subseteq P$</td>
<td>ancestor$^+$ $=$ ancestor</td>
</tr>
<tr>
<td>functionalProperty</td>
<td>$T \subseteq 1P^-$</td>
<td>T $\subseteq$ 1hasMother</td>
</tr>
<tr>
<td>InverseFunctionalProperty</td>
<td>$T \subseteq \exists 1P$</td>
<td>T $\subseteq$ 1hasSSN</td>
</tr>
</tbody>
</table>

**Axioms (mostly) reducible to inclusion ($\subseteq$)**
- C $\equiv$ D iff both C $\subseteq$ D and D $\subseteq$ C

**Obvious FOL equivalences**
- E.g., C $\equiv$ D iff $\forall x. C(x) \leftrightarrow D(x)$
- C $\subseteq$ D iff $\forall x. C(x) \Rightarrow D(x)$