Introduction to IRI, XML, RDF, RDFS, and OWL

Part1 (IRI, XML) *Part2* (RDF, RDFS) *Part3* (OWL)

Semantic Web (CSC751)

Ubbo Visser

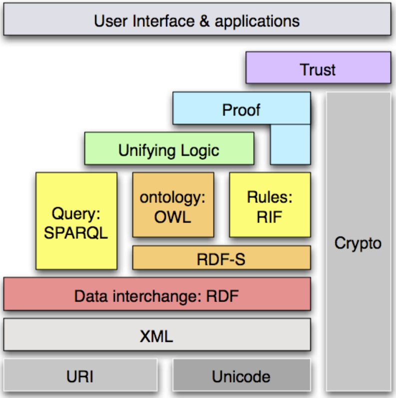
Department of Computer Science
University of Miami

September 19, 2023

UNIVERSITY
OF MIAMI

Outline

1. Announcements
2. Semantic Web stack
3. Identification of resource
4. Essentials of the eXtensible Markup Language (XML)
5. Resource Description Framework (RDF)
6. Resource Description Framework with Schema (RDFS)
7. Web Ontology Language (OWL)



Announcements

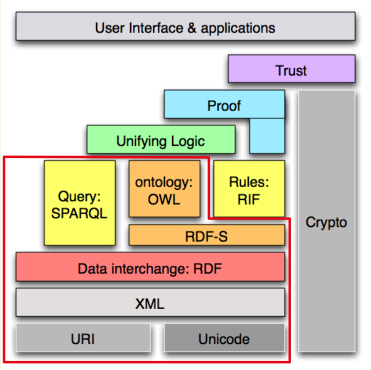
Assignment - Reading

- (Mandatory) Appendix A, Ch. 2 - 2.5 [HKR09].

Semantic Web Stack

Tim Berners-Lee version, 2006 [BL06]

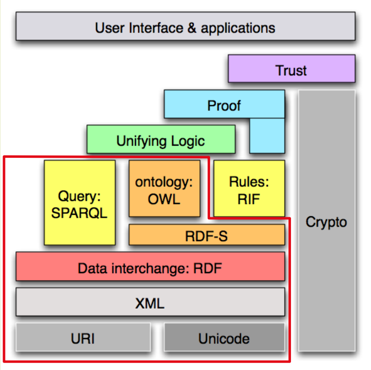
- **Semantic Web Stack/Cake/Layer Cake** provides the architecture of the Semantic Web.
- It is a realization of hierarchy of languages, where each layer below provides capabilities to immediate layer above.
- Each layer is associated with standards and specifications.
- The technologies inside the red boundary are standardized and accepted to build SW applications.
- The other layers are not clearly standardized yet. Combinations of all the layers realizes the SW vision.



Semantic Web Stack: layers in a nutshell

Standardized Semantic Web technologies

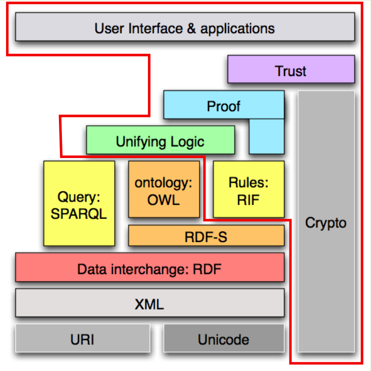
- IRI uniquely identifies resources in the domain, and Unicode allows to manipulate texts in different language settings.
- XML creates structured data, and QNames resolves ambiguities.
- RDF creates statements on resources.
- RDFS provides a lightweight ontology language.
- OWL provides an expressive ontology language.
- SPARQL queries RDF graphs.

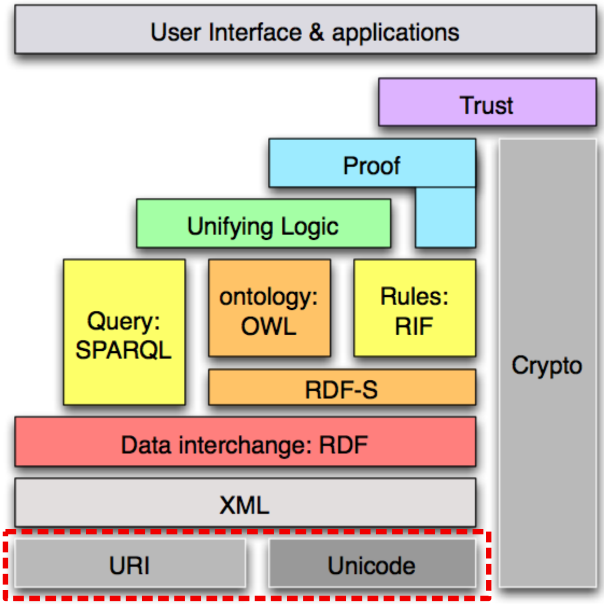


Semantic Web Stack : layers in a nutshell

Unrealized Semantic Web technologies

- RIF/SWRL allows to describe relations that can not be described using OWL. It is a rule language.
- Cryptography verifies SW statements are coming from trusted sources using appropriate digital signatures.
- Trust entails statements verifying that premises are coming from trusted sources and relying on unifying logic and proofs.
- User interfaces provides a visualization layer to humans to use SW applications.




User Interface & applications
Trust
Proof
Unifying Logic
Query: SPARQL
ontology: OWL
Rules: RIF
RDF-S
Crypto
Data interchange: RDF
XML
URI
Unicode

Identification of resource

Reason

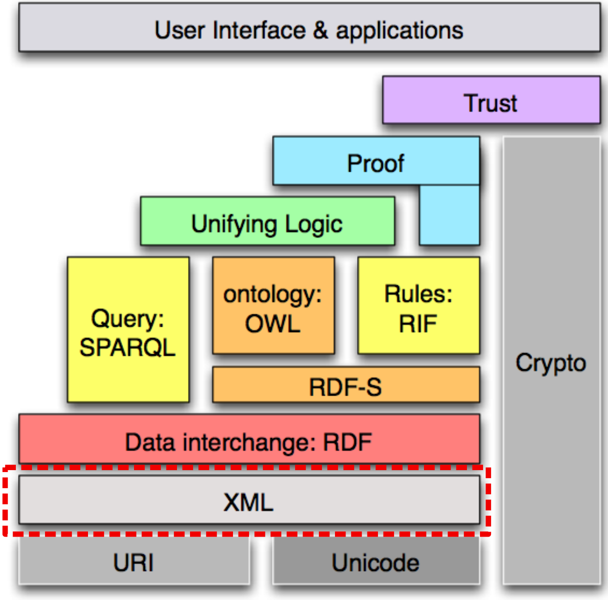
- We need a unambiguous way to identify things and concepts, because machines need to process and compose information automatically.
- We borrow the Web resource identification idea from the Web.
- **Uniform Resource Identifier (URI)**: theoretically distinguishes resources in the Web.
- **Uniform Resource Locators (URL)**: these are Web addresses that are used to access online documents.
- **Internationalized Resource Identifier (IRI)**: provides the way to encode Web addresses with Unicode.
- Therefore, **URLs** $\subseteq$ **URIs** $\subseteq$ **IRIs**
- Content negotiation.

URIs

Format

scheme:[//authority]path[?query][#fragment]

- scheme: type of URI, e.g., http, ftp, irc etc.
- [//authority]: domain name.
- path: some relative path.
- [?query]: this is optional and provides non-hierarchical information such as parameters for a Web service.
- [#fragment]: this is optional and it is commonly used for addressing parts of the document relative to the base URI.
- Not all characters are allowed in URIs.


User Interface & applications
Trust
Proof
Unifying Logic
Query: SPARQL
ontology: OWL
Rules: RIF
RDF-S
Crypto
Data interchange: RDF
XML
URI
Unicode

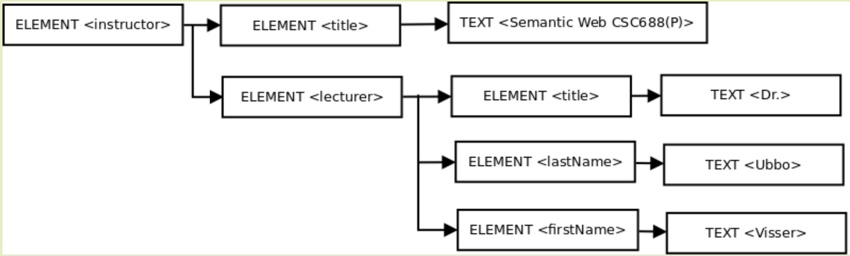
Essentials of the eXtensible Markup Language (XML)

XML

- XML is a markup language recommended by the World Wide Web Consortium (W3C) for data exchange and electronic publishing.
- It provides structure to unstructured text and annotated texts.
  - text is data, and
  - additional information about data is metadata (i.e., data about data).
- HTML is a popular markup language to visualize Web pages. It has tags such as <h2>Sam</h2> with predefined semantics for the visualization (e.g., **Sam**).
- XML tags can be chosen freely and their general meaning is not predefined. Hence, its whole purpose is to structure the documents.
- Database view: XML as a data model for semi-structured data.
- Every XML document is a text document with a declaration for which XML version and the character encoding is used. e.g., <?xml version="1.0" encoding="utf-8"?>

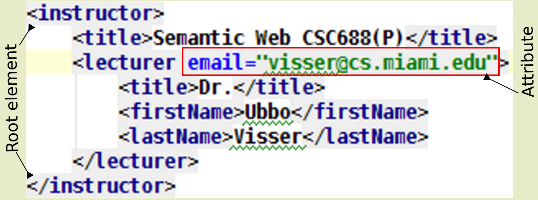
Tree structure

```xml
<instructor>
    <title>Semantic Web CSC688(P)</title>
    <lecturer>
        <title>Dr.</title>
        <firstName>Ubbo</firstName>
        <lastName>Visser</lastName>
    </lecturer>
</instructor>
```



XML elements and attributes

- XML elements:
    - There is one-and-only outermost element called root element.
    - XML elements are enclosed with matching tag-pairs.
    - Empty elements can abbreviated.
    - Element names are QNames.
- XML attributes:
    - Name value pairs inside of XML elements.
    - It is an alternative means to sub-elements describing data.



- Syntactically correct XML documents are said to be well-formed.
- HTML uses fixed vocabulary with fixed meaning and used for displaying information.
- XML uses arbitrary tags and whose meaning is not fixed.

Namespaces

- Disambiguate elements or attribute names using namespaces.

```
<instructor xmlns:lec="http://cs.miami.edu/lecture/"
            xmlns:person="http://cs.miami.edu/person/"
            xmlns="http://cs.miami.edu/">
    <lec:title>Semantic Web CSC688(P)</lec:title>
    <lecturer email="visser@cs.miami.edu">
        <person:title>Dr.</person:title>
        <person:firstName>Ubbo</person:firstName>
        <person:lastName>Visser</person:lastName>
    </lecturer>
</instructor>
```

- Declaration: **xmlns:namespace**="<URI>"
- Namespace affects from the declaration and below of the sub-tree.
- Multiple declarations are possible.
- If we need declaration that affects the whole document, we use a mechanism so called Document Type Definitions (DTD). We discuss this when we talk about RDF.
- We are interested in XML Schema.

XML Schema

- XML allows a lot of degree of freedom in encoding information.

```
<lecturer>Ubbo Visser</lecturer>
<lecturer name="Ubbo Visser"/>
<lecturer>
    <fullName>Ubbo Visser</fullName>
</lecturer>
<lecturer>
    <firstName>Ubbo</firstName>
    <secondName>Visser</secondName>
</lecturer>
<lecturer givenName="Ubbo" surname="Visser"/>
```

- The causes problems when exchanging XML documents among applications.
- So we need an agreement about the structure of the information, including the names of tags and attributes, and whether certain subelements are required or not.
- W3C XML Schema provides the vocabulary for this task.

XML Schema

- XML Schema itself is written in XML!
- XML documents are valid if it corresponds to a XML Schema.
- An XML Schema is a well-formed XML document that contains XML schema definitions.
- It has the root element, <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">, and contains element types, which can contain attribute types, which themselves refer to predefined or user-defined datatypes.
- Datatypes are, e.g., xsd:integer, xsd:string, xsd:time, xsd:date, xsd:anyURI, xsd:ID (a specific kind of string used as an identifier of XML elements)

XML Schema

```
<!DOCTYPE xsd:schema
        [ <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
                ]>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="lecturer" type="&xsd;string" minOccurs="1" maxOccurs="unbounded">
        <xsd:attribute name="email" type="&xsd;string" use="required"/>
        <xsd:attribute name="homepage" type="&xsd;anyURI" use="optional"/>
    </xsd:element>
</xsd:schema>
```

```
<lecturer email="visser@cs.miami.edu" homepage="http://www.cs.miami.edu/~visser/">
    Ubbo Visser
</lecturer>
<lecturer email="saminda@cs.miami.edu">
    Saminda Abeyruwan
</lecturer>
```

XML Schema: user defined types

- Simple types: this is obtained by restricting other types. We are not allowed to embed elements or attribute types.
- Complex types: this is obtained by grouping elements and attributes.
- Type inheritance: we define new complex types from existing complex types.

```xml
<xsd:simpleType name="humanAge">
    <xsd:restriction base="&xsd;integer">
        <xsd:minInclusive value="0"/>
        <xsd:maxInclusive value="200"/>
    </xsd:restriction>
</xsd:simpleType>
```

XML Schema: user defined types

```
<xsd:complexType name="bookType">
    <xsd:sequence>
        <xsd:element name="author" type="&xsd;string"
                     minOccurs="1" maxOccurs="unbounded"/>
        <xsd:element name="title" type="&xsd;string"
                     minOccurs="1" maxOccurs="1"/>
        <xsd:element name="publisher" type="&xsd;string"
                     minOccurs="1" maxOccurs="1"/>
        <xsd:element name="year" type="&xsd;gYear"
                     minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="ISBNnumber" type="&xsd;nonNegativeInteger"
                   use="optional"/>
</xsd:complexType>


<xsd:complexType name="researchBookType">
    <xsd:extension base="bookType"/>
    <xsd:sequence>
        <xsd:element name="field" type="&xsd;string"/>
    </xsd:sequence>
    <xsd:attribute name="price" type="&xsd;nonNegativeInteger"
                   use="optional"/>
</xsd:complexType>
```

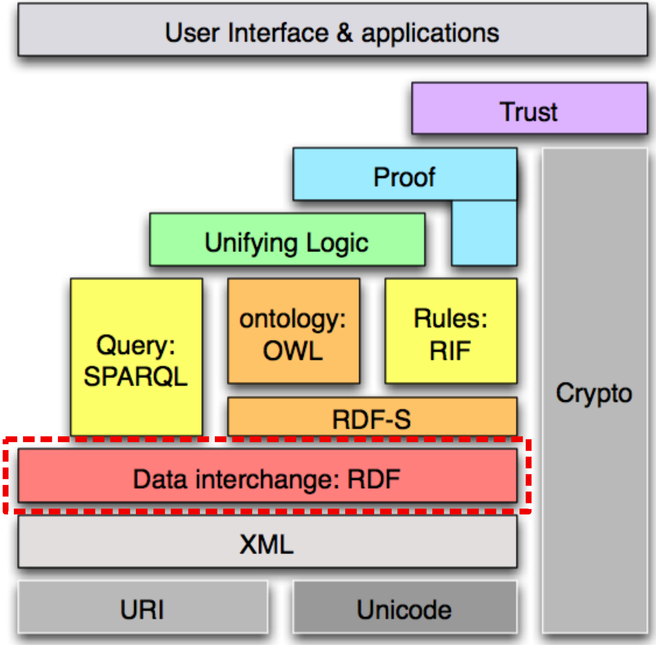
Random thoughts

" Any damn fool could produce a better data format than XML"
- James Clark[1]

e.g., SOA

Service Oriented Architecture, Web Services, WSDLs, and RESTFul services.

1 http://blog.jclark.com/2007/04/do-we-need-new-kind-of-schema-language.html


User Interface & applications
Trust
Proof
Unifying Logic
Query: SPARQL
ontology: OWL
Rules: RIF
RDF-S
Crypto
Data interchange: RDF
XML
URI
Unicode

Resource Description Framework (RDF)

Motivation

- Let say we have to encode these sentences,

$$TextBook(FOST) \tag{1}$$

$$isPublishedBy(FOST, CRC\_Press). \tag{2}$$

- How do we uniquely represent the concept *TextBook*, the relation *isPublishedBy* and the individuals *FOST* and *CRC_Press*?
- If 1 and 2 are available from two different sources, how do we merge them?
- RDF provides the solution to these two problems.

Resource identification

- We use URIs to represent concepts, relations, and individuals. E.g.,
  FOST → http://semantic-web-book.org/uri
  isPublishedBy → http://example.org/publishedBy
  CRC_Press → http://crcpress.com/uri

RDF: W3C Recommendation 2004[2]

- RDF uses a directed graph as a data model.
- The implementation uses labeled **Node-Edge-Node** triples.

- RDF is a data model for
  - describing metadata for web pages,
  - structured information, and
  - universal, machine-readable data exchange format.
- The most popular serialization mechanism is XML.

[2] http://www.w3.org/RDF/

RDF triple

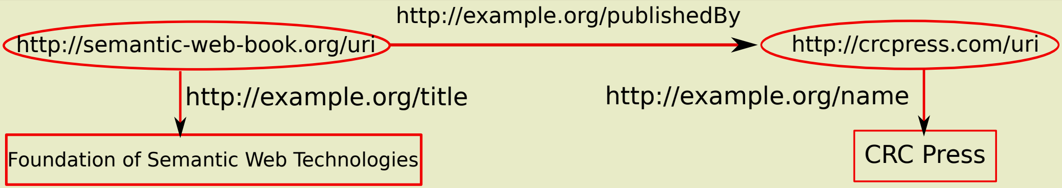
http://example.org/publishedBy

http://semantic-web-book.org/uri →	→ http://crcpress.com/uri	
Subject	Predicate	Object
URIs and blank nodes	URIs	URIs, blank nodes and literals

RDF components

- URIs uniquely represent resources.
- Literals are for data values.
  - Encoded as strings.
  - Meaning is interpreted by the associated datatype.
  - Untyped literals are treated as strings.
- Blank nodes for anonymously connecting sets of triples.

Graph: sets of triples



Turtle: Terse RDF Triple language

- How do we serialize a RDF graph ?

```
<http://semantic-web-book.org/uri>
    <http://example.org/publishedBy>  <http://crcpress.com/uri> .
<http://semantic-web-book.org/uri>
    <http://example.org/title>
        "Foundations of Semantic Web Technologies" .
<http://crcpress.com/uri>
    <http://example.org/name>        "CRC Press" .
```

- URIs in angle brackets, literals enclosed in quotes, and triples end with a period. All white spaces: blank lines, line feeds are skipped.

Turtle shortcut: prefixes

```
@prefix book: <http://semantic-web-book.org/> .
@prefix ex: <http://example.org/> .
@prefix crc: <http://crcpress.com/> .

book:uri  ex:publishedBy  crc:uri .
book:uri  ex:title        "Foundations of Semantic Web Technologies" .
crc:uri   ex:name         "CRC Press" .
```

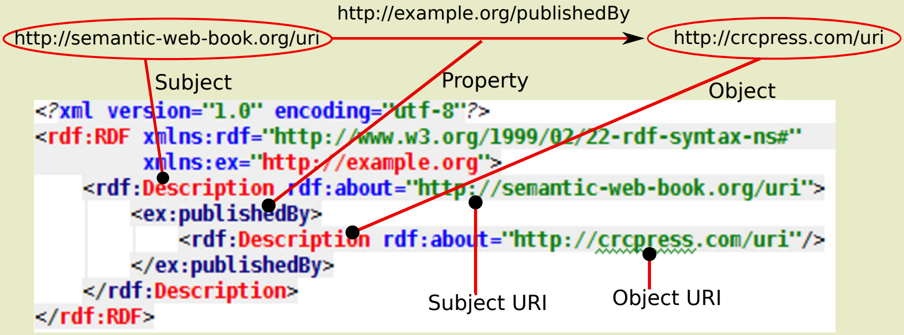
Turtle shortcut: grouping triples with same subject, and same subject and predicate

```
@prefix book: <http://semantic-web-book.org/> .
@prefix ex: <http://example.org/> .
@prefix crc: <http://crcpress.com/> .

book:uri  ex:publishedBy  crc:uri ;
          ex:title        "Foundations of Semantic Web Technologies" .
crc:uri   ex:name         "CRC Press", "CRC" .
```

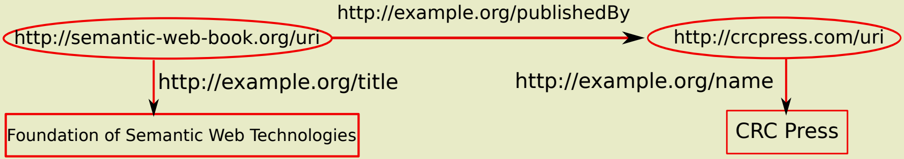
W3C recommendation: XML

- XML is extensively used as the message format between heterogeneous systems.
- Many programming languages provide full XML parsing libraries.
- The normative syntax for RDF is based on XML syntax.



- RDF language has its own namespace.
- Uses tags that belong to different namespaces.

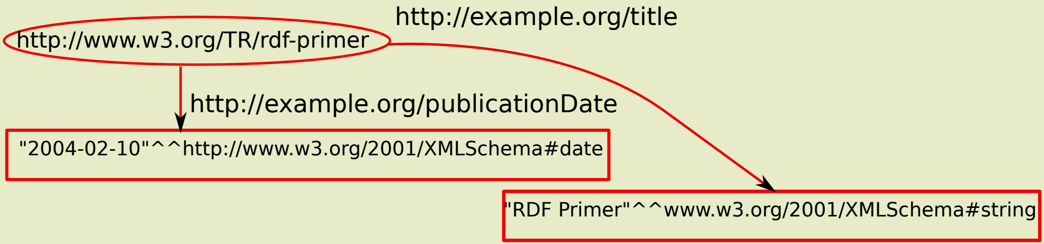
Untyped




```
<rdf:Description rdf:about="http://semantic-web-book.org/uri">
  <ex:title>Foundations of Semantic Web Technologies</ex:title>
  <ex:publishedBy>
    <rdf:Description rdf:about="http://crcpress.com/uri">
      <ex:name>CRC Press</ex:name>
    </rdf:Description>
  </ex:publishedBy>
</rdf:Description>
```

- Untyped text is taken as free text, and it is bound by XML version and character encoding.
- Subject could contain multiple properties, and
- Object can be used as subject for further triples.
- Datatypes can contain types from XML Schema.

Datatypes




```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
<http://www.w3.org/TR/rdf-primer>
       <http://example.org/title>  "RDF Primer"^^xsd:string ;
       <http://example.org/publicationDate> "2004-02-10"^^xsd:date .
```

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-primer">
  <ex:title rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    RDF Primer
  </ex:title>
  <ex:publicationDate
            rdf:datatype="http://www.w3.org/2001/XMLSchema#date">
    2004-02-10
  </ex:publicationDate>
</rdf:Description>
```

rdf:XMLLiteral for arbitrary XML fragments

```
<rdf:Description rdf:about="http://semantic-web-book/uri">
  <ex:title rdf:parseType="Literal">
    Foundations of
    <br />
    <b>Semantic Web Technologies</b>
  </ex:title>
</rdf:Description>
```

Alternative representation #1

```
<rdf:Description rdf:about="http://semantic-web-book/uri"
          ex:title= "Foundations of Semantic Web Technologies">
  <ex:publishedBy rdf:resource="http://crcpress.com/uri" />
</rdf:Description>
<rdf:Description rdf:about="http://crcpress.com/uri"
                                   ex:Name="CRC Press" />
```

XML ENTITY to prefix URLs

```
<?xml version="1.0" encoding="utf-8"?> <!DOCTYPE rdf:RDF[
        <!ENTITY book 'http://semantic-web-book.org/'>
]>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex ="http://example.org/">

  <rdf:Description rdf:about="&book;uri">
    <ex:title>Foundations of Semantic Web Technologies</ex:title>
  </rdf:Description>

</rdf:RDF>
```

Base namespace

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex ="http://example.org/"
         xml:base ="http://semantic-web-book.org/" >

  <rdf:Description rdf:about="uri">
    <ex:publishedBy rdf:resource="http://crcpress.com/uri" />
  </rdf:Description>

</rdf:RDF>
```

Many-valued relationships: n-ary

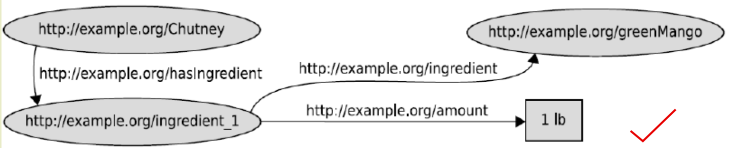
$$hasIngredient(Chutney, 1lb, GreenMango) \quad (3)$$

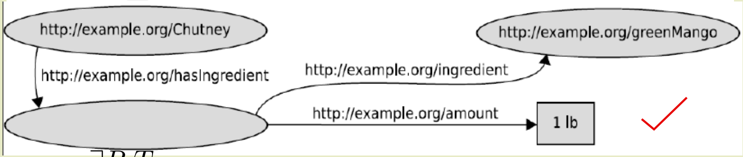
```
@prefix ex: <http://example.org/> .
ex:Chutney  ex:hasIngredient  "1lb green mango",
                              "1tsp. Cayenne pepper" .
```

?

```
@prefix ex: <http://example.org/> .
ex:Chutney  ex:ingredient  ex:greenMango;     ex:amount "1lb" ;
            ex:ingredient  ex:CayennePepper;  ex:amount "1tsp." .
```

?





$\exists R.T$

Using blank nodes: XML

```
<rdf:Description rdf:about="http://example.org/Chutney">
   <ex:hasIngredient rdf:nodeID="id1" />
</rdf:Description>
<rdf:Description rdf:nodeID="id1">
   <ex:ingredient rdf:resource="http://example.org/greenMango" />
   <ex:amount>1lb</ex:amount>
</rdf:Description>
```

Alternate

```
<rdf:Description rdf:about="http://example.org/Chutney">
   <ex:hasIngredient rdf:parseType="Resource">
     <ex:ingredient rdf:resource="http://example.org/greenMango" />
     <ex:amount>1lb</ex:amount>
   </ex:hasIngredient>
</rdf:Description>
```

Using blank nodes: Turtle

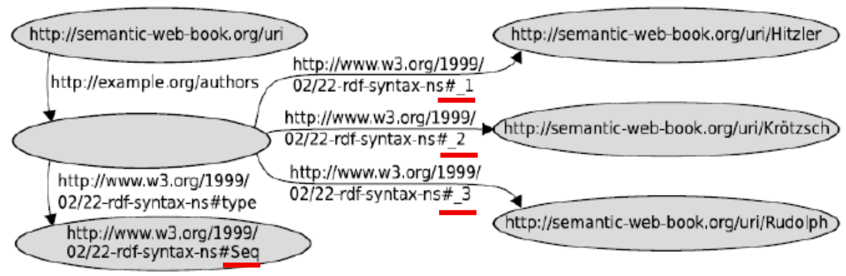
```
@prefix ex:  <http://example.org/> .
ex:Chutney    ex:hasIngredient  _:id1 .
_:id1         ex:ingredient  ex:greenMango;   ex:amount  "1lb" .
```

Alternate

```
@prefix ex:  <http://example.org/> .
ex:Chutney    ex:hasIngredient
             [ ex:ingredient  ex:greenMango;   ex:amount  "1lb" ] .
```

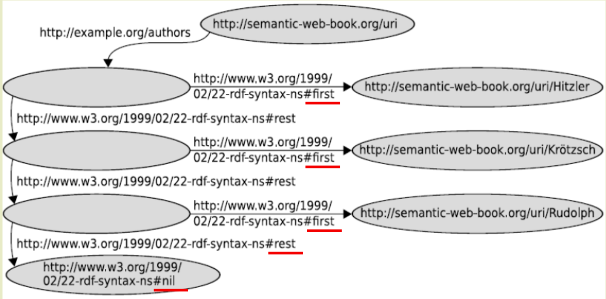
Open lists: containers

- This has provision to add new elements.
- rdf:Seq for ordered lists, rdf:Bag for unordered list, and rdf:Alt for set of alternatives.



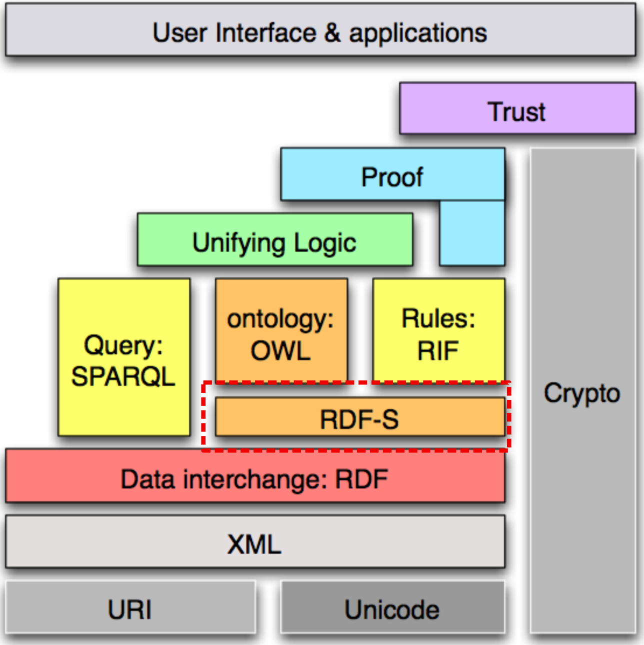

```
<rdf:Description rdf:about="http://semantic-web-book/uri">
 <ex:authors>
  <rdf:Seq>
   <rdf:li rdf:resource="http://semantic-web-book.org/uri/Hitzler" />
   <rdf:li rdf:resource="http://semantic-web-book.org/uri/Krötzsch" />
   <rdf:li rdf:resource="http://semantic-web-book.org/uri/Rudolph" />
  </rdf:Seq>
 </ex:authors>
</rdf:Description>
```

Close lists: collections




```
<rdf:Description rdf:about="http://semantic-web-book/uri">
  <ex:authors rdf:parseType="Collection">
     <rdf:Description
           rdf:about="http://semantic-web-book.org/uri/Hitzler" />
     <rdf:Description
           rdf:about="http://semantic-web-book.org/uri/Krötzsch" />
     <rdf:Description
           rdf:about="http://semantic-web-book.org/uri/Rudolph" />
  </ex:authors>
</rdf:Description>
```

```
@prefix book:  <http://semantic-web-book.org/> .
book:uri    <http://example.org/authors>
       ( book:uri/Hitzler  book:uri/Krötzsch  book:uri/Rudolph ) .
```


User Interface & applications
Trust
Proof
Unifying Logic
Query: SPARQL
ontology: OWL
Rules: RIF
RDF-S
Crypto
Data interchange: RDF
XML
URI
Unicode

Motivation

- We use RDF to represent facts:

$$isPublishedBy(FOST, CRC\_Press). \tag{4}$$

- How do we represent:

$$\exists hasDaughter.Daughter \sqsubseteq Parent \tag{5}$$

$$Women \sqsubseteq Person \tag{6}$$

$$hasWife \sqsubseteq hasSpouse \tag{7}$$

- These are known as terminological axioms (T-Box) or schema knowledge, and RDFS provides a **weaker** schema language for modeling.

RDFS common facts

- RDFS is a W3C recommendation.
- Every RDFS document is a valid RDF document.
- We use rdfs:http://www.w3.org/2000/01/rdf-schema# QName.
- RDFS is a knowledge representation language or **ontology language**.
- An ontology is a description of knowledge about a domain of interest, the core of which is a machine-processable specification with a formally defined meaning.
- RDFS is a lightweight ontology language (*"A little semantics goes a long way" - James Hendler*).

Representing things

- A concept (a.k.a. class) represents a set of things. We use URIs to represent classes.

Vocabulary

- Class membership:
  book:uri **rdf:type** ex:TextBook.
- An URI could have multiple memberships:
  book:uri **rdf:type** ex:TextBook
  book:uri **rdf:type** ex:MustRead
- Classes have hierarchies (a.k.a. **taxonomy**): *each text book is a book*
  ex:TextBook **rdfs:subClassOf** ex:Book
- Evey class URI is a member of:
  ex:TextBook **rdf:type** **rdfs:Class**
- and,
  rdfs:Class **rdf:type** **rdfs:Class**

Vocabulary

- rdfs:Resource : class of all resources
- rdf:Property : class of all properties
- rdf:XMLLiteral : we know this
- rdfs:Literal : class of all literal values
- rdfs:Datatype : class of all datatypes
- rdf:Bag, rdf:Alt, rdf:Seq, rdf:List, rdf:nil, and rdfs:Container : for containers; open and close
- rdfs:ContainerMembershipProperty : class of constrained properties
- rdfs:Statement : class of reified triples

Logical consequences : deduced, or inferred or implicit knowledge

$$x \quad rdf:type \quad ex:TextBook \tag{8}$$

$$ex:TextBook \quad rdfs:subClassOf \quad ex:Book \tag{9}$$

$$\models$$

$$x \quad rdf:type \quad ex:Book. \tag{10}$$

Logical consequences : **rdfs:subClassOf is transitive**

$$ex:TextBook \quad rdfs:subClassOf \quad ex:Book \tag{11}$$

$$ex:Book \quad rdfs:subClassOf \quad ex:PrintMedia \tag{12}$$

$$\models$$

$$ex:TextBook \quad rdfs:subClassOf \quad ex:PrintMedia. \tag{13}$$

Logical consequences : **rdfs:subClassOf is reflexive**

$$ex:MorningStar \quad rdfs:subClassOf \quad ex:EveningStar \tag{14}$$
$$ex:EveningStar \quad rdfs:subClassOf \quad ex:MorningStar \tag{15}$$

$$ex:Book \quad rdfs:subClassOf \quad ex:Book \tag{16}$$

Property hierarchies

$$ex{:}isHappilyMarriedTo \quad rdfs{:}subPropertyOf \quad ex{:}isMarriedTo \tag{17}$$
$$ex:Mary \quad ex:isHappilyMarriedTo \quad ex:Tom \tag{18}$$
$$\models$$
$$ex:Mary \quad ex:isMarriedTo \quad ex:Tom \tag{19}$$

Property restrictions

- We use restrictions to provide a certain restriction to URIs of subject and object.
- This is done via **rdfs:domain** and **rdfs:range**.

$$ex:isMarriedTo \quad rdfs:domain \quad ex:Person \tag{20}$$

$$ex:isMarriedTo \quad rdfs:range \quad ex:Person \tag{21}$$

- Property restrictions can be applied to datatypes:

$$ex:hasBirthdate \qquad rdfs:range \qquad xsd:date \tag{22}$$

Property restriction causes problems that might be hard to debug

$$:authorOf \quad rdfs:range \quad :TextBook \tag{23}$$

$$:authorOf \quad rdfs:range \quad :StoryBook \tag{24}$$

This implies that the type of $:authorOf$ both $:TextBook \wedge :StoryBook$

Restriction problems

$$:isMarriedTo \quad rdfs:domain \quad :Person \tag{25}$$

$$:isMarriedTo \quad rdfs:range \quad :Person \tag{26}$$

$$:UofM \quad rdf:type \quad :Institute \tag{27}$$

$$:Visser \quad :isMarriedTo \quad :UofM \tag{28}$$

$$\models$$

$$:UofM \quad rdf:type \quad :Person \tag{29}$$

Reification

- We want to say *"The detective supposes that the butler killed the gardener"*.
- These are unsatisfactory:

$$
\begin{aligned}
&:detective \quad :supposes \quad \text{“}Thebutlerkilledthegardner\text{”} \\
&:detective \quad :supposes \quad :TheButlerKilledTheGardner
\end{aligned}
$$

- What we would like:

$$:butler \qquad :killed \qquad :gardener. \tag{30}$$

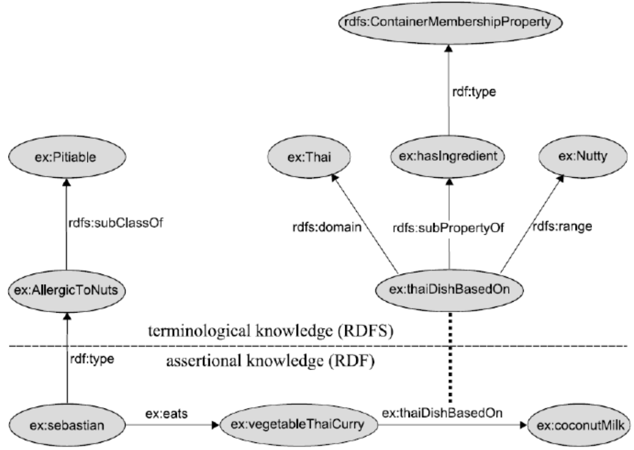
- We use reification:

$$
\begin{aligned}
:detective &\quad :supposes \quad \_:id \\
\_:id &\quad rdf:subject \quad :butler \\
\_:id &\quad rdf:predicate \quad :hasKilled \\
\_:id &\quad rdf:object \quad :gardener.
\end{aligned}
$$

Example

```
ex:vegetableThaiCurry  ex:thaiDishBasedOn    ex:coconutMilk .
ex:sebastian           rdf:type              ex:AllergicToNuts .
ex:sebastian           ex:eats               ex:vegetableThaiCurry .

ex:AllergicToNuts      rdfs:subClassOf       ex:Pitiable .
ex:thaiDishBasedOn     rdfs:domain           ex:Thai .
ex:thaiDishBasedOn     rdfs:range            ex:Nutty .
ex:thaiDishBasedOn     rdfs:subPropertyOf    ex:hasIngredient .
ex:hasIngredient       rdf:type     rdfs:ContainerMembershipProperty.
```



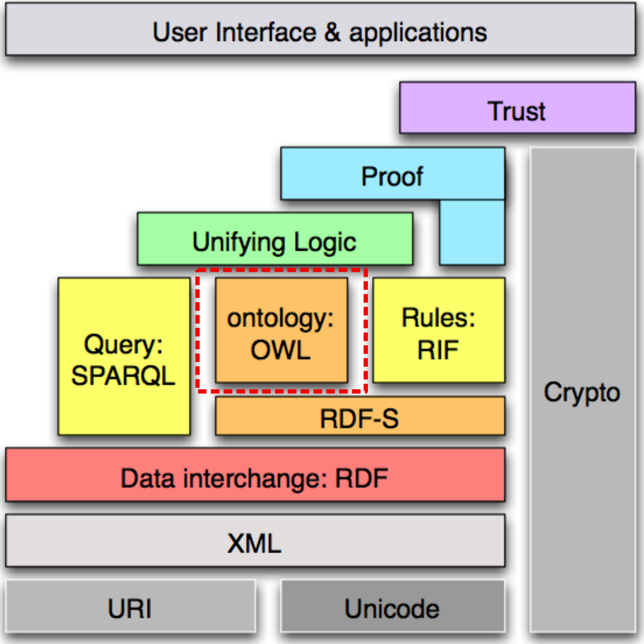
RDFS semantics is weaker

:Mary	rdf:type	:Person	$\rightsquigarrow$	$\text{Person}(\text{Mary})$
:Mother	rdf:subClassOf	:Woman	$\rightsquigarrow$	$\text{Mother} \sqsubseteq \text{Woman}$
:John	:hasWife	:Mary	$\rightsquigarrow$	$\text{hasWife}(\text{John}, \text{Mary})$
:hasWife	rdfs:subPropertyOf	:hasSpouse	$\rightsquigarrow$	$\text{hasWife} \sqsubseteq \text{hasSpouse}$
:hasWife	rdfs:range	:Woman	$\rightsquigarrow$	$\top \sqsubseteq \forall \text{hasWife}.\text{Woman}$
:hasWife	rdfs:domain	:Man	$\rightsquigarrow$	$\exists \text{hasWife}.\top \sqsubseteq \text{Man}$

Multiple views

Look at this statement $\mathit{Truck} \sqsubseteq \mathit{MotorVehicle}$. When this statement travels up the Semantic Stack, it will be subjected to three views:

1. XML structure
2. RDF graph (triple)
3. RDF Schema (semantic)


User Interface & applications
Trust
Proof
Unifying Logic
Query: SPARQL
ontology: OWL
Rules: RIF
Crypto
RDF-S
Data interchange: RDF
XML
URI
Unicode

Assignment

- Assignment #2 requires a substantial amount of reading, and modeling a simple ontology.
- Please start this assignment early!

Reading

- (Must read) Ch. 4 [HKR09].

Protégé

We use the Protégé ontology editor and knowledge acquisition system to demonstrate important aspects of ontology modeling.

Motivation

- How do we represent these sentences:
  - Every project has at least one participant.
  - Projects are always internal or external projects.
  - The superior of my superior is also my superior.
  - All examiners of an exam must be professors.
  - $Human \sqsubseteq \exists hasParent.Human$
  - $Orphan \sqsubseteq Human \sqcap \forall hasParent.\neg Alive$
  - $Orphan(HarryPotter)$
  - $hasParent(HarryPotter, JamesPotter)$
  - $\forall x, y(\exists(hasParent(x, z) \land hasBrother(z, y)) \Rightarrow hasUncle(x, y))$
  - $HappyFather \equiv \ \geq 2 hasChild.Female$
  - $Car \sqsubseteq =4 hasTyre.\top$
  - $PersonCommittingSuicide \equiv \exists hasKilled.Self$
  - $\neg hasColleague(UbboVisser, NadalRafael)$
- We use **OWL 2** Web Ontology Language.
- OWL 2 is a W3C recommendation for modeling ontologies.
- OWL Lite $\subseteq$ **OWL DL** $\subseteq$ OWL Full.

OWL sublanguages

OWL sublanguages

- OWL Full

OWL sublanguages

- OWL Full
  - contains OWL DL and OWL Lite

OWL sublanguages

- OWL Full
  - contains OWL DL and OWL Lite
  - only sublanguage containing all of RDFS

OWL sublanguages

- OWL Full
  - contains OWL DL and OWL Lite
  - only sublanguage containing all of RDFS
  - very expressive

OWL sublanguages

- OWL Full
  - contains OWL DL and OWL Lite
  - only sublanguage containing all of RDFS
  - very expressive
  - semantically difficult to understand and to work with

OWL sublanguages

- OWL Full
  - contains OWL DL and OWL Lite
  - only sublanguage containing all of RDFS
  - very expressive
  - semantically difficult to understand and to work with
  - undecidable

OWL sublanguages

- OWL Full
  - contains OWL DL and OWL Lite
  - only sublanguage containing all of RDFS
  - very expressive
  - semantically difficult to understand and to work with
  - undecidable
  - support by hardly any software tools

OWL sublanguages

- OWL Full
  - contains OWL DL and OWL Lite
  - only sublanguage containing all of RDFS
  - very expressive
  - semantically difficult to understand and to work with
  - undecidable
  - support by hardly any software tools
- OWL DL

OWL sublanguages

- OWL Full
  - contains OWL DL and OWL Lite
  - only sublanguage containing all of RDFS
  - very expressive
  - semantically difficult to understand and to work with
  - undecidable
  - support by hardly any software tools
- OWL DL
  - contains OWL Lite and is contained in OWL Full

OWL sublanguages

- OWL Full
  - contains OWL DL and OWL Lite
  - only sublanguage containing all of RDFS
  - very expressive
  - semantically difficult to understand and to work with
  - undecidable
  - support by hardly any software tools
- OWL DL
  - contains OWL Lite and is contained in OWL Full
  - decidable

OWL sublanguages

- OWL Full
  - contains OWL DL and OWL Lite
  - only sublanguage containing all of RDFS
  - very expressive
  - semantically difficult to understand and to work with
  - undecidable
  - support by hardly any software tools
- OWL DL
  - contains OWL Lite and is contained in OWL Full
  - decidable
  - fully supported by most software tools

OWL sublanguages

- OWL Full
  - contains OWL DL and OWL Lite
  - only sublanguage containing all of RDFS
  - very expressive
  - semantically difficult to understand and to work with
  - undecidable
  - support by hardly any software tools
- OWL DL
  - contains OWL Lite and is contained in OWL Full
  - decidable
  - fully supported by most software tools
  - worst-case computational complexity: NExpTime

OWL sublanguages

- OWL Full
  - contains OWL DL and OWL Lite
  - only sublanguage containing all of RDFS
  - very expressive
  - semantically difficult to understand and to work with
  - undecidable
  - support by hardly any software tools
- OWL DL
  - contains OWL Lite and is contained in OWL Full
  - decidable
  - fully supported by most software tools
  - worst-case computational complexity: NExpTime
- OWL Lite

OWL sublanguages

- OWL Full
  - contains OWL DL and OWL Lite
  - only sublanguage containing all of RDFS
  - very expressive
  - semantically difficult to understand and to work with
  - undecidable
  - support by hardly any software tools
- OWL DL
  - contains OWL Lite and is contained in OWL Full
  - decidable
  - fully supported by most software tools
  - worst-case computational complexity: NExpTime
- OWL Lite
  - contained in OWL Full and OWL DL

OWL sublanguages

- OWL Full
  - contains OWL DL and OWL Lite
  - only sublanguage containing all of RDFS
  - very expressive
  - semantically difficult to understand and to work with
  - undecidable
  - support by hardly any software tools
- OWL DL
  - contains OWL Lite and is contained in OWL Full
  - decidable
  - fully supported by most software tools
  - worst-case computational complexity: NExpTime
- OWL Lite
  - contained in OWL Full and OWL DL
  - decidable

OWL sublanguages

- OWL Full
  - contains OWL DL and OWL Lite
  - only sublanguage containing all of RDFS
  - very expressive
  - semantically difficult to understand and to work with
  - undecidable
  - support by hardly any software tools
- OWL DL
  - contains OWL Lite and is contained in OWL Full
  - decidable
  - fully supported by most software tools
  - worst-case computational complexity: NExpTime
- OWL Lite
  - contained in OWL Full and OWL DL
  - decidable
  - less expressive

OWL sublanguages

- OWL Full
  - contains OWL DL and OWL Lite
  - only sublanguage containing all of RDFS
  - very expressive
  - semantically difficult to understand and to work with
  - undecidable
  - support by hardly any software tools
- OWL DL
  - contains OWL Lite and is contained in OWL Full
  - decidable
  - fully supported by most software tools
  - worst-case computational complexity: NExpTime
- OWL Lite
  - contained in OWL Full and OWL DL
  - decidable
  - less expressive
  - worst-case computational complexity: ExpTime

OWL syntax and intuitive semantics

OWL syntax and intuitive semantics

- The header of an OWL ontology.

OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.

OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - owl:Thing ($\top$), owl:Nothing ($\bot$), owl:topProperty (U), and owl:bottomProperty

OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - owl:Thing ($\top$), owl:Nothing ($\bot$), owl:topProperty (U), and owl:bottomProperty
  - Classes (a.k.a. concepts): *Professor*

OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - owl:Thing ($\top$), owl:Nothing ($\bot$), owl:topProperty (U), and owl:bottomProperty
  - Classes (a.k.a. concepts): *Professor*
  - Individuals: $Professor(UbboVisser)$

OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - owl:Thing ($\top$), owl:Nothing ($\bot$), owl:topProperty (U), and owl:bottomProperty
  - Classes (a.k.a. concepts): *Professor*
  - Individuals: $Professor(UbboVisser)$
  - Abstract roles (a.k.a. object properties): *hasAffiliation*.

OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - owl:Thing ($\top$), owl:Nothing ($\bot$), owl:topProperty (U), and owl:bottomProperty
  - Classes (a.k.a. concepts): *Professor*
  - Individuals: $Professor(UbboVisser)$
  - Abstract roles (a.k.a. object properties): *hasAffiliation*.
  - Concrete roles (a.k.a. datatype properties): *firstName*.

OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - owl:Thing ($\top$), owl:Nothing ($\bot$), owl:topProperty (U), and owl:bottomProperty
  - Classes (a.k.a. concepts): *Professor*
  - Individuals: *Professor*(*UbboVisser*)
  - Abstract roles (a.k.a. object properties): *hasAffiliation*.
  - Concrete roles (a.k.a. datatype properties): *firstName*.
  - Domain and ranges. Use these as the last resort.

OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - owl:Thing ($\top$), owl:Nothing ($\bot$), owl:topProperty (U), and owl:bottomProperty
  - Classes (a.k.a. concepts): *Professor*
  - Individuals: $Professor(UbboVisser)$
  - Abstract roles (a.k.a. object properties): *hasAffiliation*.
  - Concrete roles (a.k.a. datatype properties): *firstName*.
  - Domain and ranges. Use these as the last resort.
  - Simple class relations.
    - $Professor \sqsubseteq FacultyMember$.
    - $Book \sqsubseteq Publication$.
    - $Professor \sqsubseteq \neg Publication \equiv Professor \sqcap Publication \sqsubseteq \bot$.
    - $Man \sqsubseteq Person$.
    - $Person \equiv Human$.

OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - owl:Thing ($\top$), owl:Nothing ($\bot$), owl:topProperty (U), and owl:bottomProperty
  - Classes (a.k.a. concepts): *Professor*
  - Individuals: $Professor(UbboVisser)$
  - Abstract roles (a.k.a. object properties): *hasAffiliation*.
  - Concrete roles (a.k.a. datatype properties): *firstName*.
  - Domain and ranges. Use these as the last resort.
  - Simple class relations.
    - $Professor \sqsubseteq FacultyMember$.
    - $Book \sqsubseteq Publication$.
    - $Professor \sqsubseteq \neg Publication \equiv Professor \sqcap Publication \sqsubseteq \bot$.
    - $Man \sqsubseteq Person$.
    - $Person \equiv Human$.
    - There is no Unique Name Assumption (UNA): owl:sameAs.

OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - owl:Thing ($\top$), owl:Nothing ($\bot$), owl:topProperty (U), and owl:bottomProperty
  - Classes (a.k.a. concepts): *Professor*
  - Individuals: $Professor(UbboVisser)$
  - Abstract roles (a.k.a. object properties): *hasAffiliation*.
  - Concrete roles (a.k.a. datatype properties): *firstName*.
  - Domain and ranges. Use these as the last resort.
  - Simple class relations.
    - $Professor \sqsubseteq FacultyMember$.
    - $Book \sqsubseteq Publication$.
    - $Professor \sqsubseteq \neg Publication \equiv Professor \sqcap Publication \sqsubseteq \bot$.
    - $Man \sqsubseteq Person$.
    - $Person \equiv Human$.
    - There is no Unique Name Assumption (UNA): owl:sameAs.
    - Close Classes: owl:oneOf.

OWL syntax and intuitive semantics

OWL syntax and intuitive semantics

- Simple class relations.

OWL syntax and intuitive semantics

- Simple class relations.
  - Conjunction of classes: $StaffOfCS \sqsubseteq Staff \sqcap MemberOfCS$.

OWL syntax and intuitive semantics

- Simple class relations.
  - Conjunction of classes: $StaffOfCS \sqsubseteq Staff \sqcap MemberOfCS$.
  - $Mother \equiv Woman \sqcap Parent$,
    $\forall x(Mother(X) \Leftrightarrow Woman(x) \wedge Parent(x))$

$$
\begin{array}{rcl}
:Mother & owl:equivalentClass & \_:x. \\
\_:x & rdf:type & owl:Class. \\
\_:x & owl:intersectionOf & (:Woman\ \ :Parent).
\end{array}
$$

OWL syntax and intuitive semantics

- Simple class relations.
  - Conjunction of classes: $StaffOfCS \sqsubseteq Staff \sqcap MemberOfCS$.
  - $Mother \equiv Woman \sqcap Parent$,
    $\forall x(Mother(X) \Leftrightarrow Woman(x) \wedge Parent(x))$

$$
\begin{array}{rcl}
:Mother & owl:equivalentClass & \_:x. \\
\_:x & rdf:type & owl:Class. \\
\_:x & owl:intersectionOf & (:Woman\ \ :Parent).
\end{array}
$$

  - Disjunction of classes: $Professor \sqsubseteq ActivelyTeaching \sqcup Retired$.

OWL syntax and intuitive semantics

- Simple class relations.
  - Conjunction of classes: $StaffOfCS \sqsubseteq Staff \sqcap MemberOfCS$.
  - $Mother \equiv Woman \sqcap Parent$,
    $\forall x(Mother(X) \Leftrightarrow Woman(x) \wedge Parent(x))$

$$\begin{array}{rcl} :Mother & owl:equivalentClass & \_:x. \\ \_:x & rdf:type & owl:Class. \\ \_:x & owl:intersectionOf & (:Woman\ \ :Parent). \end{array}$$

  - Disjunction of classes: $Professor \sqsubseteq ActivelyTeaching \sqcup Retired$.
  - $Parent \equiv Mother \sqcup Father$,
    $\forall x(Parent(X) \Leftrightarrow Mother(x) \vee Father(x))$

$$\begin{array}{rcl} :Parent & owl:equivalentClass & \_:x. \\ \_:x & rdf:type & owl:Class. \\ \_:x & owl:unionOf & (:Mother\ \ :Father). \end{array}$$

OWL syntax and intuitive semantics

OWL syntax and intuitive semantics

- Simple class relations.

OWL syntax and intuitive semantics

- Simple class relations.
  - Negation: $ChildlessPerson \equiv Person \sqcap \neg Parent$, $\forall x(ChildlessPerson(X) \Leftrightarrow Person(x) \wedge \neg Parent(x))$

$$
\begin{array}{rll}
:ChildlessPerson & owl:equivalentClass & \_:x. \\
\_:x & rdf:type & owl:Class. \\
\_:x & owl:intersectionOf & (:Person\ \_:y). \\
\_:y & owl:complementOf & :Parent.
\end{array}
$$

OWL syntax and intuitive semantics

- Role restrictions.

OWL syntax and intuitive semantics

- Role restrictions.
  - **All** examiners of an exam must be professors,
    $Exam \sqsubseteq \forall hasExaminer.Professor$

OWL syntax and intuitive semantics

- Role restrictions.
  - **All** examiners of an exam must be professors,
    $Exam \sqsubseteq \forall hasExaminer.Professor$
  - Any exam must have at least one examiner.
    $Exam \sqsubseteq \exists hasExaminer.Professor$

OWL syntax and intuitive semantics

- Role restrictions.
  - **All** examiners of an exam must be professors,
    $Exam \sqsubseteq \forall hasExaminer.Professor$
  - Any exam must have at least one examiner.
    $Exam \sqsubseteq \exists hasExaminer.Professor$
  - Universal quantification: only to be used with a role - a.k.a. property restrictions.
    $Person \sqcap Happy \equiv \forall hasChild.Parent$
    $\forall x(Person(x) \wedge Happy(x) \Leftrightarrow \forall y(hasChild(x,y) \Rightarrow Happy(y)))$

$$
\begin{array}{lcl}
\_:x & rdf:type & owl:Class. \\
\_:x & owl:intersectionOf & (:Person\ \ :Happy). \\
\_:x & owl:equivalentClass & \_:y. \\
\_:y & rdf:type & owl:Restriction. \\
\_:y & owl:onProperty & :hasChild. \\
\_:y & owl:allValuesFrom & :Parent.
\end{array}
$$

OWL syntax and intuitive semantics

OWL syntax and intuitive semantics

- Existential quantification: only to be used with a role - a.k.a. property restrictions
  $Parent \equiv \exists hasChild.Person$
  $\forall x(Parent(x) \Leftrightarrow \exists y(hasChild(x,y) \wedge Person(y)))$

$$
\begin{array}{rcl}
:Parent & owl:equivalentClass & \_:x. \\
\_:x & rdf:type & owl:Restriction. \\
\_:x & owl:onProperty & :hasChild. \\
\_:x & owl:someValuesFrom & :Person.
\end{array}
$$

OWL syntax and intuitive semantics

- Existential quantification: only to be used with a role - a.k.a. property restrictions
  $Parent \equiv \exists hasChild.Person$
  $\forall x(Parent(x) \Leftrightarrow \exists y(hasChild(x,y) \wedge Person(y)))$

$$
\begin{array}{rcl}
:Parent & owl:equivalentClass & \_:x. \\
\_:x & rdf:type & owl:Restriction. \\
\_:x & owl:onProperty & :hasChild. \\
\_:x & owl:someValuesFrom & :Person.
\end{array}
$$

- Cardinality restrictions: at most, at least and exactly. Lets understand these constructs using WorkingWithFemaleColleagues.owl
  $Exam \sqsubseteq \leq 2 hasExaminer.\top$
  $Exam \sqsubseteq \geq 3 hasTopics.\top$
  $Exam \sqsubseteq = 3 hasTopics.\top$

OWL syntax and intuitive semantics

OWL syntax and intuitive semantics

- Role relationships:
  - $hasExaminer \sqsubseteq hasParticipant$
  - $hasParticipant \equiv hasAttendee$
  - $hasAttendee^{-} \equiv participatesIn$
  - $hasExaminer^{-} \equiv examinerOf$

OWL syntax and intuitive semantics

- Role relationships:
  - $hasExaminer \sqsubseteq hasParticipant$
  - $hasParticipant \equiv hasAttendee$
  - $hasAttendee^{-} \equiv participatesIn$
  - $hasExaminer^{-} \equiv examinerOf$
- Properties can have following restrictions:

Role char.	DL	e.g.,	General presentation
Transitive	Tra(R)	hasAncestor	$R(a,b)$ *and* $R(b,c) \Rightarrow R(a,c)$
Symmetric	Sym(R)	hasSpouse	$R(a,b) \Rightarrow R(b,a)$
Asymmetric	Asy(R)	hasChild	$R(a,b) \Rightarrow$ *not* $R(b,a)$
Reflexive	Ref(R)	hasRelative	$R(a,a)$ *for all* a
Irreflexive	Irr(R)	parentOf	*not* $R(a,a)$ *for any* a
Functional	Fnc(R)	hasHusband	$R(a,b)$ *and* $R(a,c) \Rightarrow b = c$
InverseFunctional	Ifn(R)	hasHusband	$R(a,b)$ *and* $R(c,b) \Rightarrow a = c$

OWL syntax and intuitive semantics

- Role relationships:
  - $Sym(hasColleague)$
  - $Tra(hasColleague)$
  - $Fun(hasTeamLeader)$
  - $Ifn(isTeamLeaderFor)$
  - $hasColleague(UbboVisser, AndreasSeekircher)$
  - $hasColleague(AndreasSeekircher, JustinStoecker)$
  - $hasColleague(JustinStoecker, SamindaAbeyruwan)$
  - $isTeamLeaderFor(UbboVisser, RoboCanes)$

OWL syntax and intuitive semantics

- Role relationships:
  - $Sym(hasColleague)$
  - $Tra(hasColleague)$
  - $Fun(hasTeamLeader)$
  - $Ifn(isTeamLeaderFor)$
  - $hasColleague(UbboVisser, AndreasSeekircher)$
  - $hasColleague(AndreasSeekircher, JustinStoecker)$
  - $hasColleague(JustinStoecker, SamindaAbeyruwan)$
  - $isTeamLeaderFor(UbboVisser, RoboCanes)$
- Self: $PersonCommittingSuicide \equiv \exists kills.Self$

OWL syntax and intuitive semantics

- Role relationships:
  - $Sym(hasColleague)$
  - $Tra(hasColleague)$
  - $Fun(hasTeamLeader)$
  - $Ifn(isTeamLeaderFor)$
  - $hasColleague(UbboVisser, AndreasSeekircher)$
  - $hasColleague(AndreasSeekircher, JustinStoecker)$
  - $hasColleague(JustinStoecker, SamindaAbeyruwan)$
  - $isTeamLeaderFor(UbboVisser, RoboCanes)$
- Self: $PersonCommittingSuicide \equiv \exists kills.Self$
- Disjoint properties, Dis(S,R) : $Dis(hasParent, hasChild)$

OWL syntax and intuitive semantics

- Role relationships:
  - $Sym(hasColleague)$
  - $Tra(hasColleague)$
  - $Fun(hasTeamLeader)$
  - $Ifn(isTeamLeaderFor)$
  - $hasColleague(UbboVisser, AndreasSeekircher)$
  - $hasColleague(AndreasSeekircher, JustinStoecker)$
  - $hasColleague(JustinStoecker, SamindaAbeyruwan)$
  - $isTeamLeaderFor(UbboVisser, RoboCanes)$
- Self: $PersonCommittingSuicide \equiv \exists kills.Self$
- Disjoint properties, Dis(S,R) : $Dis(hasParent, hasChild)$
- Negated role assignment: $\neg hasColleague(UbboVisser, NadalRafael)$

OWL syntax and intuitive semantics

- Role relationships:
  - $Sym(hasColleague)$
  - $Tra(hasColleague)$
  - $Fun(hasTeamLeader)$
  - $Ifn(isTeamLeaderFor)$
  - $hasColleague(UbboVisser, AndreasSeekircher)$
  - $hasColleague(AndreasSeekircher, JustinStoecker)$
  - $hasColleague(JustinStoecker, SamindaAbeyruwan)$
  - $isTeamLeaderFor(UbboVisser, RoboCanes)$
- Self: $PersonCommittingSuicide \equiv \exists kills.Self$
- Disjoint properties, Dis(S,R) : $Dis(hasParent, hasChild)$
- Negated role assignment: $\neg hasColleague(UbboVisser, NadalRafael)$
- Role chains: $hasParent \circ hasBrother \sqsubseteq hasUncle$

OWL syntax and intuitive semantics

- Role relationships:
  - $Sym(hasColleague)$
  - $Tra(hasColleague)$
  - $Fun(hasTeamLeader)$
  - $Ifn(isTeamLeaderFor)$
  - $hasColleague(UbboVisser, AndreasSeekircher)$
  - $hasColleague(AndreasSeekircher, JustinStoecker)$
  - $hasColleague(JustinStoecker, SamindaAbeyruwan)$
  - $isTeamLeaderFor(UbboVisser, RoboCanes)$
- Self: $PersonCommittingSuicide \equiv \exists kills.Self$
- Disjoint properties, Dis(S,R) : $Dis(hasParent, hasChild)$
- Negated role assignment: $\neg hasColleague(UbboVisser, NadalRafael)$
- Role chains: $hasParent \circ hasBrother \sqsubseteq hasUncle$
- Datatypes (D) :
  - hasAge(Sam, "40"^^xsd:integer)
  - ¬hasHeight(Sam, "6.0"^^xsd:float)
  - $Teenager \equiv Person \sqcap \exists hasAge.(xsd : integer \geq 12 \; and \; \leq 19)$

Type separation, and punning, and declarations

- In OWL 2 DL, a **class name** may also occur as a **abstract role name**. But, they are treated as distinct. This is called **punning**.
- When a class name is used as a abstract role name, they are identified by the same URI. It is the same resource in the sense of RDF.
- In OWL 2 DL, they are considered as semantically distinct, i.e., two different interpretation of the same resource.
- e.g.,
  $Professor(UbboVisser)$
  $Professor(UbboVisser, UniversityOfMiami)$
- **owl:hasKey**: Given a class C, a set of abstract or concrete roles $r_1, \ldots, r_n$ is said to be a **key** for class C, if no two named instances of C coincide on all values of all the roles. This relates to inverse functionality, but inverse functionality only implied the existence.

OWL Species

- OWL Full:
  - Unrestricted OWL 2 DL plus all of RDF(S).
  - There is no reasoner that supports the semantics of OWL Full.
  - Type separation is not enforced. i.e., OWL Full individuals, classes, and roles can be mixed freely. e.g., individual in one statement becomes a role in next statement.
- OWL DL:
  - Description logic version of OWL.
  - Model-theoretic semantics of SROIQ(D) is used, called OWL 2 Direct Semantics.
  - Reasoner support exists.
- OWL Lite:
  - OWL Lite is essentially difficult to deal with as OWL DL. Therefore, this has minor role in practice.

OWL 2 Profiles

- There are sublanguages of OWL 2, which have polynomial inference algorithms.
- OWL 2 EL (OWL 2 EL++):
    - Polynomial time algorithms exist for satisfiability checking, classification, and instance checking.
    - e.g., SNOMED CT
    - Allowed : $\sqcap$ $\exists$ $\top$ $\bot$ $\sqsubseteq$ $\sqcap$ $\exists$ $\top$ $\bot$, closed classes must have only one member, and property chain axioms and range restrictions under certain conditions.
    - Disallowed : $\neg$ $\sqcup$, arbitrary universal quantification, and role inverses.
    - e.g.,
      $Human \sqsubseteq \exists hasParent.Person$,
      $\exists married.\top \sqcap CatholicPriest \sqsubseteq \bot$,
      $hasParent \circ hasParent \sqsubseteq hasGrandparent$.

OWL 2 Profiles

- OWL 2 QL (DL Lite_R):
  - Answer to: what fraction of OWL 2 DL can be captured by rational database systems?
  - Query answering in LOGSPACE w.r.t. data via translation into SQL.
  - Allowed:
  - Domain, range, and subproperties.
  - Subclass statements with:
    - Left hand side: class name or expression of type $\exists R.\top$
    - Right hand side: intersection of class names, expression of types $\exists R.C$, and negation of left hand expressions.
    - No closed classes.
  - e.g., $\exists married.\top \sqsubseteq \neg Free \sqcap \exists has.Sorrow$
- OWL RL (DLP) :
  - Answer to: what fraction of OWL 2 DL can be expressed naively by rules?
  - Read section 4.3.2.3 of [HKR09].

Ontology Vs. Database [Hor10, BHS03]

- Ontology:
  - Open World Assumption (OWA): missing information treated as unknown.
  - No Unique Name Assumption (NUNA): individual may have multiple synonyms.
  - Ontologies provide entailments.

Ontology Vs. Database [Hor10, BHS03]

- Ontology:
  - Open World Assumption (OWA): missing information treated as unknown.
  - No Unique Name Assumption (NUNA): individual may have multiple synonyms.
  - Ontologies provide entailments.
- Database:
  - Close World Assumption (CWA): missing information is false.
  - Unique Name Assumption (UNA): each individual is uniquely identifiable.
  - Database schema provides structure on data.

E.g.,

- T-Box:

$$
\begin{aligned}
HogwartsStudent &\equiv Student \sqcap \exists attendsSchool.Hogwarts \\
HogwartsStudent &\sqsubseteq \forall hasPet.(Owl \sqcup Cat \sqcup Toad) \\
hasPet &\equiv isPetOf^{-} \\
\exists hasPet.\top &\sqsubseteq Human \\
Phoenix &\sqsubseteq \forall isPetOf.Wizard \\
Muggle &\sqcap Wizard \sqsubseteq \bot
\end{aligned}
$$

E.g.,

- T-Box:

$$
\begin{aligned}
HogwartsStudent &\equiv Student \sqcap \exists attendsSchool.Hogwarts \\
HogwartsStudent &\sqsubseteq \forall hasPet.(Owl \sqcup Cat \sqcup Toad) \\
hasPet &\equiv isPetOf^{-} \\
\exists hasPet.\top &\sqsubseteq Human \\
Phoenix &\sqsubseteq \forall isPetOf.Wizard \\
Muggle &\sqcap Wizard \sqsubseteq \bot
\end{aligned}
$$

- A-Box:

$$
\begin{gathered}
Wizard(HarryPotter) \\
Wizard(DracoMalfoy) \\
hasFriend(HarryPotter, RonWeasley) \\
hasFriend(HarryPotter, HermioneGranger) \\
hasPet(HarryPotter, Hedwig)
\end{gathered}
$$

E.g.,

E.g.,

- Is *DracoMalfoy* a friend of *HarryPotter*?
  Ontology: Don't know (OWA), Database: No!

E.g.,

- Is *DracoMalfoy* a friend of *HarryPotter*?
  Ontology: Don't know (OWA), Database: No!
- How many friends does *HarryPotter* have?
  Ontology: At least 1 (NUNA), Database: 2!

E.g.,

- Is *DracoMalfoy* a friend of *HarryPotter*?
  Ontology: Don't know (OWA), Database: No!
- How many friends does *HarryPotter* have?
  Ontology: At least 1 (NUNA), Database: 2!
- A-Box: $Dis(RonWeasley, HermioneGranger)$

E.g.,

- Is *DracoMalfoy* a friend of *HarryPotter*?
  Ontology: Don't know (OWA), Database: No!
- How many friends does *HarryPotter* have?
  Ontology: At least 1 (NUNA), Database: 2!
- A-Box: $Dis(RonWeasley, HermioneGranger)$
- How many friends does *HarryPotter* have?
  Ontology: at least 2, Database: 2!

E.g.,

- Is *DracoMalfoy* a friend of *HarryPotter*?
  Ontology: Don't know (OWA), Database: No!
- How many friends does *HarryPotter* have?
  Ontology: At least 1 (NUNA), Database: 2!
- A-Box: $Dis(RonWeasley, HermioneGranger)$
- How many friends does *HarryPotter* have?
  Ontology: at least 2, Database: 2!
- T-Box :
  $HarryPottersFriends \equiv \forall hasFriend.\{RonWeasley \sqcup HermioneGranger\}$
  $Wizard \sqcap HarryPottersFriends(HarryPotter)$

E.g.,

- Is *DracoMalfoy* a friend of *HarryPotter*?
  Ontology: Don't know (OWA), Database: No!
- How many friends does *HarryPotter* have?
  Ontology: At least 1 (NUNA), Database: 2!
- A-Box: $Dis(RonWeasley, HermioneGranger)$
- How many friends does *HarryPotter* have?
  Ontology: at least 2, Database: 2!
- T-Box :
  $HarryPottersFriends \equiv \forall hasFriend.\{RonWeasley \sqcup HermioneGranger\}$
  $Wizard \sqcap HarryPottersFriends(HarryPotter)$
- How many friends does *HarryPotter* have?
  Ontology: 2!, Database: 2!

E.g.,

- If the domain or range axioms are used correctly, they provide powerful inferences:
- T-Box says: $\exists hasPet.\top \sqsubseteq Human$, $Phoenix \sqsubseteq \forall isPetOf.Wizard$
- A-Box:
  $Wizard(Dumbledore)$
  $Phoenix(Fawkes)$
  $isPetOf(Fawkes, Dumbledore)$
- Ontology infers that $Human \sqcap Wizard(Dumbledore)$
- Database rejects, because domain of *hasPet* is *Human*, and *Dumbledore* is not *Human* (CWA).
- Ontologies use theorem proving to answer questions.
- It involves both T-Box and A-Box, and has higher worst case complexities.

OWL 2 features

Feature	**DL**	**FOL**	**Vocabulary**
Top/bottom class	$\top/\bot$	Axiomatize	owl:Thing/ owl:Nothing
Class intersection	$\sqcap$	$\wedge$	owl:intersectionOf
Class union	$\sqcup$	$\vee$	owl:unionOf
Class complement	$\neg$	$\neg$	owl:complementOf
Enumerated classes	$\{a\}$	$\approx$	owl:oneOf
Property restriction	**DL (owl:onProperty)**	**FOL**	**Vocabulary**
Existential	$\exists R.C$	$\exists x \ldots$	owl:someValuesFrom
Universal	$\forall R.C$	$\forall x \ldots$	owl:allValuesFrom
Min. cardinality	$\geq_n R.C$	$\exists x_1, \ldots, x_n$	owl:minQualifiedCardinality/ owl:onClass
Max. cardinality	$\leq_n R.C$	$\forall x_1, \ldots, x_n (\ldots \Rightarrow \ldots)$	owl:maxQualifiedCardinality/ owl:onClass
Local reflexivity	$\exists R.Self$	$R(x, x)$	owl:hasSelf

OWL 2 features

Feature	**DL**	**FOL**	**Vocabulary**
Property chain	$\circ$	Axiomatize	owl:propertyChainAxiom
Inverse	R^{-}	Axiomatize	owl:inverseOf
Key	-	Axiomatize	owl:hasKey
Property disjointness	$Dis(R, S)$	Axiomatize	owl:propertyDisjointWith
Property characteristics	**DL**	**FOL**	**Vocabulary**
Symmetric	$Sym(R)$	Axiomatize	owl:SymmetricProperty
Asymmetric	$Asy(R)$	Axiomatize	owl:AsymmetricProperty
Reflexive	$Ref(R)$	Axiomatize	owl:ReflexiveProperty
Irreflexive	$Irr(R)$	Axiomatize	owl:IrreflexiveProperty
Transitive	$Tra(R)$	Axiomatize	owl:TransitiveProperty
Functional	$Fun(R)$	Axiomatize	owl:FunctionalProperty
Inverse functional	$Ifn(R)$	Axiomatize	owl:InverseFunctionalProperty
Subclass	$C \sqsubseteq D$	$\forall x.C(x) \Rightarrow D(x)$	rdfs:subClassOf
Subproperty	$R \sqsubseteq S$	$\forall x, y.R(x, y) \Rightarrow S(x, y)$	rdfs:subPropertyOf

Model these axioms in an ontology

- $\{John\} \sqcap \{Bill\} \sqsubseteq \bot$
- $\{John\} \equiv \{Jim\}$
- $\neg hasWife(Bill, Mary)$
- $Woman \sqcap Man \sqsubseteq \bot$
- $Parent \equiv Mother \sqcup Father$
- $ChildlessPerson \equiv Person \sqcap \neg Parent$
- $Person \sqcap \neg Parent(Jack)$
- $Parent \equiv \exists hasChild.Person$
- $Orphan \equiv \forall hasChild^{-}.Dead$
- $JohnsChildren \equiv \exists hasParent.\{John\}$
- $NarcissisticPerson \equiv \exists loves.Self$
- $MyBirthdayGuests \equiv \{Bill, John, Mary\}$
- $hasParent \circ hasParent \sqsubseteq hasGrandparent$

Acknowledgement

Acknowledgement

The majority of the slides for this course have been prepared by Saminda Abeyruwan.

Franz Baader, Ian Horrocks, and Ulrike Sattler.
Description logics as ontology languages for the semantic web.
In *Festschrift in honor of Jörg Siekmann, Lecture Notes in Artificial Intelligence*, pages 228–248. Springer-Verlag, 2003.

Tim Berners-Lee.
Artificial Intelligence and the Semantic Web.
http://www.w3.org/2006/Talks/0718-aaai-tbl/, 2006.

Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph.
Foundations of Semantic Web Technologies.
Chapman & Hall/CRC, 2009.

Ian Horrocks.
Description Logic: A formal foundation for languages and tools. Tutorial at the Semantic Technology Conference (SemTech). San Francisco, California, USA.
http://www.cs.ox.ac.uk/people/ian.horrocks/Seminars/seminars.html#tutorials, 2010.