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

Part 1
Part 2
Part 3
Semantic Web (CSC688 P)

Ubbo Visser (Instructor)    Saminda Abeyruwan (Research Assistant)

Department of Computer Science
University of Miami

September 27, 2011
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

- Assignment #1 is due today.

Reading

- (Mandatory) Appendix A, Ch. 2 - 2.5 [HKR09].
**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.
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.
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, $\text{URLs} \subseteq \text{URIs} \subseteq \text{IRIs}$
- Content negotiation.
URLs

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 URLS.
User Interface & applications

Proof
Unifying Logic
Query: SPARQL
ontology: OWL

Trust

Rules: RIF

Data interchange: RDF

Crypto

XML

URI

Unicode
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 annotate 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 visualize (e.g., Sam).
- XML tags can be chosen freely and their 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 used. E.g.,

```xml
<?xml version="1.0" encoding="utf-8"?>
```
Tree structure

```
<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 be 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.

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 allows a lot of degree of freedom in encoding information.

```xml
<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 is corresponding 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 identifier of XML elements)
XML Schema

```xml
<!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

```xml
<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

e.g., SOA

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

Resource Description Framework (RDF)

Motivation

Let say we have to encode these sentences,

\[ TextBook(FOST) \] (1)

\[ isPublishedBy(FOST, CRC\_Press). \] (2)

How do we uniquely represent the concept \textit{TextBook}, the relation \textit{isPublishedBy} and the individuals \textit{FOST} and \textit{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. 

\textit{e.g., FOST} \rightarrow \text{http://semantic-web-book.org/uri}

\textit{isPublishedBy} \rightarrow \text{http://example.org/publishedBy}

\textit{CRC\_Press} \rightarrow \text{http://crcpress.com/uri}
RDF: W3C Recommendation 2004\(^2\)

- RDF uses a directed graph as 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.
- Most popular serialization mechanism is XML.

\(^2\)http://www.w3.org/RDF/
### RDF triple

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
</table>

- **Subject**: URIs and blank nodes
- **Predicate**: URIs
- **Object**: 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

http://semantic-web-book.org/uri

http://example.org/publishedBy

http://example.org/title

http://example.org/name

Foundation of Semantic Web Technologies

CRC Press

Turtle: Terse RDF Triple language

How do we serialize a RDF graph?

```
<http://semantic-web-book.org/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

```turtle
@prefix ex: <http://example.org/> .
@prefix crc: <http://crcpress.com/> .

crc:uri ex:name "CRC Press" .
```

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

```turtle
@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 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

http://www.w3.org/TR/rdf-primer
http://example.org/title
http://example.org/publicationDate
"2004-02-10"^^http://www.w3.org/2001/XMLSchema#date
"RDF Primer"^^www.w3.org/2001/XMLSchema#string

@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

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

Alternative representation #1

```xml
<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
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE rdf:RDF[
]

<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

```xml
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex = "http://example.org/"
    <rdf:Description rdf:about="uri">
        <ex:publishedBy rdf:resource="http://crcpress.com/uri" />
    </rdf:Description>
</rdf:RDF>
```
Many-valued relationships: n-ary

\[
\text{hasIngredient}(\text{Chutney}, 1\text{lb}, \text{GreenMango})
\]

\[
\text{ex:Chutney ex:hasIngredient "1lb green mango", } \text{ex:ingredient ex:greenMango; ex:amount "1lb"; ex:ingredient ex:CayennePepper; ex:amount "1tsp.".}
\]
Using blank nodes: XML

```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>

<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

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

@prefix ex: <http://example.org/> .
ex:Chutney a ex:hasIngredient
  [ ex:ingredient <http://example.org/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.org/uri">
  <ex:authors>
    <rdf:Seq>
    </rdf:Seq>
  </ex:authors>
</rdf:Description>
```
Close lists: collections

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

book:uri  <http://example.org/authors>  
User Interface & applications

Proof

Unifying Logic

Query: SPARQL

ontology: OWL

Rules: RIF

RDF-S

Data interchange: RDF

XML

URI

Unicode

Trust

Crypto
Motivation

- We use RDF to represent facts:

  \[ \text{isPublishedBy}(\text{FOST}, \text{CRC\_Press}). \] \hspace{1cm} (4)

- How do we represent:

  \[ \exists \text{hasDaughter} \cdot \text{Daughter} \sqsubseteq \text{Parent} \] \hspace{1cm} (5)

  \[ \text{Woment} \sqsubseteq \text{Person} \] \hspace{1cm} (6)

  \[ \text{hasWife} \sqsubseteq \text{hasSpouse} \] \hspace{1cm} (7)

- These are known as terminological axioms (T-Box) or schema knowledge, and RDFS provides a \textit{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:**
  
  \[
  \text{book:uri} \quad \text{rdf:type} \quad \text{ex:TextBook}.
  \]

- **An URI could have multiple memberships:**
  
  \[
  \text{book:uri} \quad \text{rdf:type} \quad \text{ex:TextBook}
  \]
  
  \[
  \text{book:uri} \quad \text{rdf:type} \quad \text{ex:MustRead}
  \]

- **Classes have hierarchies (a.k.a. taxonomy):** *each text book is a book*
  
  \[
  \text{ex:TextBook} \quad \text{rdfs:subClassOf} \quad \text{ex:Book}
  \]

- **Every class URI is a member of:**
  
  \[
  \text{ex:TextBook} \quad \text{rdf:type} \quad \text{rdfs:Class}
  \]
  
  and,

  \[
  \text{rdfs:Class} \quad \text{rdf:type} \quad \text{rdfs:Class}
  \]
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

\[
x \text{ rdf : type } \text{ ex : TextBook} \quad (8)
\]

\[
\text{ex : TextBook} \quad \text{rdfs : subClassOf} \quad \text{ex : Book} \quad (9)
\]

\[
\models
\]

\[
x \text{ rdf : type } \text{ ex : Book}. \quad (10)
\]

Logical consequences: \text{rdfs:subClassOf is transitive}

\[
\text{ex : TextBook} \quad \text{rdfs : subClassOf} \quad \text{ex : Book} \quad (11)
\]

\[
\text{ex : Book} \quad \text{rdfs : subClassOf} \quad \text{ex : PrintMedia} \quad (12)
\]

\[
\models
\]

\[
\text{ex : TextBook} \quad \text{rdfs : subClassOf} \quad \text{ex : PrintMedia}. \quad (13)
\]
Logical consequences: \texttt{rdfs:subClassOf} is reflexive

\begin{align*}
\text{ex} & : \text{MorningStar} & \text{rdfs} : \text{subClassOf} & \text{ex} : \text{EveningStar} & (14) \\
\text{ex} & : \text{EveningStar} & \text{rdfs} : \text{subClassOf} & \text{ex} : \text{MorningStar} & (15) \\
\text{ex} & : \text{Book} & \text{rdfs} : \text{subClassOf} & \text{ex} : \text{Book} & (16)
\end{align*}

Property hierarchies

\begin{align*}
\text{ex:isHappilyMarriedTo} & \quad \text{rdfs:subPropertyOf} & \quad \text{ex:isMarriedTo} & (17) \\
\text{ex} & : \text{Mary} & \text{ex} : \text{isHappilyMarriedTo} & \text{ex} : \text{Tom} & (18) \\
\models & \quad \text{ex} & : \text{Mary} & \text{ex} : \text{isMarriedTo} & \text{ex} : \text{Tom} & (19)
\end{align*}
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`.

  \[
  \text{ex : isMarriedTo} \quad \text{rdfs : domain} \quad \text{ex : Person} \quad (20)
  \]

  \[
  \text{ex : isMarriedTo} \quad \text{rdfs : range} \quad \text{ex : Person} \quad (21)
  \]

- Property restrictions can be applied to datatypes:

  \[
  \text{ex : hasBirthdate} \quad \text{rdfs : range} \quad \text{xsd : data} \quad (22)
  \]
Property restriction causes problems that might be hard to debug

\[ : \text{authorOf} \ rdfs : \text{range} \ : \text{TextBook} \]  
\[ : \text{authorOf} \ rdfs : \text{range} \ : \text{StoryBook} \]

This implies that the type of \( : \text{authorOf} \) both \( : \text{TextBook} \land : \text{StoryBook} \)

Restriction problems

\[ : \text{isMarriedTo} \ rdfs : \text{domain} \ : \text{Person} \]  
\[ : \text{isMarriedTo} \ rdfs : \text{range} \ : \text{Person} \]  
\[ : \text{UofM} \ rdf : \text{type} \ : \text{Institute} \]  
\[ : \text{Visser} \ : \text{isMarriedTo} \ : \text{UofM} \]  
\[ \models \]  
\[ : \text{UofM} \ rdf : \text{type} \ : \text{Person} \]
We want to say "The detective supposes that the butler killed the gardener".

These are unsatisfactory:

\[ \text{detective} : \text{supposes} : \text{The butler killed the gardener} \]

What we would like:

\[ \text{butler} : \text{killed} : \text{gardener}. \]  (30)

We use reification:

\[ \text{detective} : \text{supposes} : \_ : \text{id} \]
\[ \_ : \text{id} \quad \text{rdf} : \text{subject} : \text{butler} \]
\[ \_ : \text{id} \quad \text{rdf} : \text{predicate} : \text{hasKilled} \]
\[ \_ : \text{id} \quad \text{rdf} : \text{object} : \text{gardener}. \]
ex:vegetableThaiCurry  ex:thaiDishBasedOn  ex:coconutMilk.
ex:sebastian  rdf:type  ex:peanutAllergy.
ex:sebastian  ex:eats  ex:vegetableThaiCurry.
ex:AllergicToNuts  rdfs:subClassOf  ex:Pitable.
ex:thaiDishBasedOn  rdfs:domain  ex:Thai.
ex:thaiDishBasedOn  rdfs:range  ex:Nutty.
ex:hasIngredient  rdfs:subPropertyOf  ex:hasIngredient.
ex:hasIngredient  rdf:type  rdfs:ContainerMembershipProperty.

terminological knowledge (RDFS)
assertional knowledge (RDF)
RDFS semantics is weaker

:Mary rdf:type :Person ⇝ Person(Mary)
:Mother rdf:subClassOf :Woman ⇝ Mother ⊑ Woman
:John :hasWife :Mary ⇝ hasWife(John, Mary)
:hasWife rdfs:subPropertyOf :hasSpouse ⇝ hasWife ⊑ hasSpouse
:hasWife rdfs:range :Woman ⇝ T ⊑ ∀hasWife.Woman
:hasWife rdfs:domain :Man ⇝ ∃hasWife. T ⊑ Man

Multiple views

Look at this statement Truck ⊑ 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

RDF-S

Data interchange: RDF

XML

URI

Unicode

Crypto

Outline
  Annon.
  SW stack
  IRI
  XML
  RDF
  RDFS
  OWL
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.
- \( \text{Human} \sqsubseteq \exists \text{hasParent}. \text{Human} \)
- \( \text{Orphan} \sqsubseteq \text{Human} \cap \forall \text{hasParent}. \neg \text{Alive} \)
- \( \text{Orphan}(\text{HarryPotter}) \)
- \( \text{hasParent}(\text{HarryPotter}, \text{JamesPotter}) \)
- \( \forall x, y (\exists (\text{hasParent}(x, z) \land \text{hasBrother}(z, y)) \Rightarrow \text{hasUncle}(x, y)) \)
- \( \text{HappyFather} \equiv \geq 2 \text{hasChild}. \text{Female} \)
- \( \text{Car} \sqsubseteq 4 \text{hasTyre}. \top \)
- \( \text{PersonCommittingSuicide} \equiv \exists \text{hasKilled}. \text{Self} \)
- \( \neg \text{hasColleague}(\text{UbboVisser}, \text{NadalRafael}) \)

- We use **OWL 2** Web Ontology Language.
- OWL 2 is a W3C recommendation for modeling ontologies.
- OWL Lite \( \subseteq \text{OWL DL} \subseteq \text{OWL Full} \).
OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - owl:Thing (⊤), owl:Nothing (⊥), 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 ⊑ FacultyMember.
    - Book ⊑ Publication.
    - Professor ⊑ ¬Publication ≡ Professor ⊓ Publication ⊑ ⊥.
    - Man ⊑ Person.
    - Person ≡ Human.
    - There is no Unique Name Assumption (UNA): owl:sameAs.
    - Close Classes: owl:oneOf.
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($\text{UbboVisser}$)
  - 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 ⊑ FacultyMember.
    - Book ⊑ Publication.
    - Professor ⊑ ¬Publication ≡ Professor ⊓ Publication ⊑ ⊥.
    - Man ⊑ Person.
    - Person ≡ Human.
    - There is no Unique Name Assumption (UNA): owl:sameAs.
    - Close Classes: owl:oneOf.
OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - owl:Thing (⊤), owl:Nothing (⊥), 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 ⊑ FacultyMember.
    - Book ⊑ Publication.
    - Professor ⊑ ¬Publication ⊑ Professor ∩ Publication ⊑ ⊥.
    - Man ⊑ Person.
    - Person ⊑ Human.
    - There is no Unique Name Assumption (UNA): owl:sameAs.
    - Close Classes: owl:oneOf.
OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - owl:Thing (⊤), owl:Nothing (⊥), 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 ⊑ FacultyMember.
    - Book ⊑ Publication.
    - Professor ⊑ ¬Publication ≡ Professor ⊓ Publication ⊑ ⊥.
    - Man ⊑ Person.
    - Person ≡ Human.
    - There is no Unique Name Assumption (UNA): owl:sameAs.
    - Close Classes: owl:oneOf.
OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - `owl:Thing (⊤), owl:Nothing (⊥), 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 ⊑ FacultyMember`
      - `Book ⊑ Publication`
      - `Professor ⊑ ¬Publication ≡ Professor ⊓ Publication ⊑ ⊥`
      - `Man ⊑ Person`
      - `Person ≡ Human`
      - There is no Unique Name Assumption (UNA): `owl:sameAs`
      - Close Classes: `owl:oneOf`
OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - owl:Thing (⊤), owl:Nothing (⊥), 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 ⊑ FacultyMember.
    - Book ⊑ Publication.
    - Professor ⊑ ¬Publication ≡ Professor ⊓ Publication ⊑ ⊥.
    - Man ⊑ Person.
    - Person ≡ Human.
    - There is no Unique Name Assumption (UNA): owl:sameAs.
    - Close Classes: owl:oneOf.
OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - owl:Thing (⊤), owl:Nothing (⊤), 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 ⊑ FacultyMember.
    - Book ⊑ Publication.
    - Professor ⊑ ¬Publication ≡ Professor ⊓ Publication ⊑ ⊥.
    - Man ⊑ Person.
    - Person ≡ Human.
    - There is no Unique Name Assumption (UNA): owl:sameAs.
    - Close Classes: owl:oneOf.
OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - `owl:Thing` (⊤), `owl:Nothing` (⊥), `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 ⊑ FacultyMember`
  - `Book ⊑ Publication`
  - `Professor ⊑ ¬Publication ≡ Professor ∩ Publication ⊑ ⊥`
  - `Man ⊑ Person`
  - `Person ≡ Human`
  - There is no Unique Name Assumption (UNA): `owl:sameAs`
  - Close Classes: `owl:oneOf`
OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - owl:Thing (⊤), owl:Nothing (⊥), 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 ⊑ FacultyMember.
    - Book ⊑ Publication.
    - Professor ⊑ ¬ Publication ≡ Professor ∩ Publication ⊑ ⊥.
    - Man ⊑ Person.
    - Person ≡ Human.
    - There is no Unique Name Assumption (UNA): owl:sameAs.
    - Close Classes: owl:oneOf.
OWL syntax and intuitive semantics

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - owl:Thing (⊤), owl:Nothing (⊥), 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 ⊑ FacultyMember.
    - Book ⊑ Publication.
    - Professor ⊑ ¬Publication ≡ Professor ∩ Publication ⊑ ⊥.
    - Man ⊑ Person.
    - Person ≡ Human.
    - There is no Unique Name Assumption (UNA): owl:sameAs.
    - Close Classes: owl:oneOf.
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 \(\sqsubseteq\) ¬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

- The header of an OWL ontology.
- Classes, roles, and individuals.
  - `owl:Thing (⊤), owl:Nothing (⊥), 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 ⊑ FacultyMember`
    - `Book ⊑ Publication`
    - `Professor ⊑ ¬Publication ≡ Professor ∩ Publication ⊑ ⊥`
    - `Man ⊑ Person`
    - `Person ≡ Human`
    - There is no Unique Name Assumption (UNA): `owl:sameAs`
    - Close Classes: `owl:oneOf`
OWL syntax and intuitive semantics

- Simple class relations.
  - Conjunction of classes: \textit{StaffOfCS} \sqsubseteq \textit{Staff} \sqcap \textit{MemberOfCS}.
  - \textit{Mother} \equiv \textit{Woman} \sqcap \textit{Parent},
    \[ \forall x (\text{Mother}(x) \iff \text{Woman}(x) \land \text{Parent}(x)) \]
    
    \[
    : \text{Mother} \quad \text{owl:equivalentClass} \quad _:\_ : x. \\
    _ : x \quad \text{rdf:type} \quad \text{owl:Class}. \\
    _ : x \quad \text{owl:intersectionOf} \quad (:\text{Woman} : \text{Parent}).
    \]

- Disjunction of classes: \textit{Professor} \sqsubseteq \textit{ActivelyTeaching} \sqcup \textit{Retired}.
  - \textit{Parent} \equiv \textit{Mother} \sqcup \textit{Father},
    \[ \forall x (\text{Parent}(x) \iff \text{Mother}(x) \lor \text{Father}(x)) \]
    
    \[
    : \text{Parent} \quad \text{owl:equivalentClass} \quad _ : x. \\
    _ : x \quad \text{rdf:type} \quad \text{owl:Class}. \\
    _ : x \quad \text{owl:unionOf} \quad (:\text{Mother} : \text{Father}).
    \]
OWL syntax and intuitive semantics

- **Simple class relations.**
  - Conjunction of classes: \( \text{StaffOfCS} \sqsubseteq \text{Staff} \sqcap \text{MemberOfCS} \).
  - \( \text{Mother} \equiv \text{Woman} \sqcap \text{Parent} \),
    \( \forall x (\text{Mother}(x) \iff \text{Woman}(x) \land \text{Parent}(x)) \)
    
    : \text{Mother} \ owl : \text{equivalentClass} \ _ : x.
    
    _ : x \ rdf : \text{type} \ owl : \text{Class}.
    
    _ : x \ owl : \text{intersectionOf} \ (: \text{Woman} : \text{Parent}).

  - Disjunction of classes: \( \text{Professor} \sqsubseteq \text{ActivelyTeaching} \sqcup \text{Retired} \).
  - \( \text{Parent} \equiv \text{Mother} \sqcup \text{Father} \),
    \( \forall x (\text{Parent}(x) \iff \text{Mother}(x) \lor \text{Father}(x)) \)
    
    : \text{Parent} \ owl : \text{equivalentClass} \ _ : x.
    
    _ : x \ rdf : \text{type} \ owl : \text{Class}.
    
    _ : x \ owl : \text{unionOf} \ (: \text{Mother} : \text{Father}).
OWL syntax and intuitive semantics

Simple class relations.

Conjunction of classes: \( StaffOfCS \sqsubseteq Staff \sqcap MemberOfCS \).

\( Mother \equiv Woman \sqcap Parent, \)
\[ \forall x (Mother(x) \iff Woman(x) \land Parent(x)) \]

: Mother owl : equivalentClass _ : x.
_ : x rdf : type owl : Class.
_ : x owl : intersectionOf (: Woman : Parent).

Disjunction of classes: \( Professor \sqsubseteq ActivelyTeaching \sqcup Retired \).

\( Parent \equiv Mother \sqcup Father, \)
\[ \forall x (Parent(x) \iff Mother(x) \lor Father(x)) \]

: Parent owl : equivalentClass _ : x.
_ : x rdf : type owl : Class.
_ : x owl : unionOf (: Mother : Father).
**OWL syntax and intuitive semantics**

- **Simple class relations.**
  - Conjunction of classes: $\text{StaffOfCS} \sqsubseteq \text{Staff} \sqcap \text{MemberOfCS}$.
  - $\text{Mother} \equiv \text{Woman} \sqcap \text{Parent}$,
    $\forall x (\text{Mother}(X) \iff \text{Woman}(x) \land \text{Parent}(x))$

    
    $$
    : \text{Mother} \quad \text{owl} : \text{equivalentClass} \quad _ : x.
    $$

    $$
    _ : x \quad \text{rdf} : \text{type} \quad \text{owl} : \text{Class}.
    $$

    $$
    _ : x \quad \text{owl} : \text{intersectionOf} \quad (: \text{Woman} : \text{Parent}).
    $$

  - Disjunction of classes: $\text{Professor} \sqsubseteq \text{ActivelyTeaching} \sqcup \text{Retired}$.
  - $\text{Parent} \equiv \text{Mother} \sqcup \text{Father}$,
    $\forall x (\text{Parent}(X) \iff \text{Mother}(x) \lor \text{Father}(x))$

    $$
    : \text{Parent} \quad \text{owl} : \text{equivalentClass} \quad _ : x.
    $$

    $$
    _ : x \quad \text{rdf} : \text{type} \quad \text{owl} : \text{Class}.
    $$

    $$
    _ : x \quad \text{owl} : \text{unionOf} \quad (: \text{Mother} : \text{Father}).
    $$
Simple class relations.

- Conjunction of classes: \( \text{StaffOfCS} \sqsubseteq \text{Staff} \sqcap \text{MemberOfCS} \).
- \( \text{Mother} \equiv \text{Woman} \sqcap \text{Parent}, \)
  \( \forall x (\text{Mother}(x) \Leftrightarrow \text{Woman}(x) \land \text{Parent}(x)) \)
  
  \[
  : \text{Mother} \quad \text{owl} : \text{equivalentClass} \quad _: \ x. \\
  _: \ x \quad \text{rdf} : \text{type} \quad \text{owl} : \text{Class}. \\
  _: \ x \quad \text{owl} : \text{intersectionOf} \quad (:\text{Woman} : \text{Parent}).
  
- Disjunction of classes: \( \text{Professor} \sqsubseteq \text{ActivelyTeaching} \sqcup \text{Retired} \).
- \( \text{Parent} \equiv \text{Mother} \sqcup \text{Father}, \)
  \( \forall x (\text{Parent}(x) \Leftrightarrow \text{Mother}(x) \lor \text{Father}(x)) \)
  
  \[
  : \text{Parent} \quad \text{owl} : \text{equivalentClass} \quad _: \ x. \\
  _: \ x \quad \text{rdf} : \text{type} \quad \text{owl} : \text{Class}. \\
  _: \ x \quad \text{owl} : \text{unionOf} \quad (:\text{Mother} : \text{Father}).
  

OWL syntax and intuitive semantics

Simple class relations.

- Conjunction of classes: \( \text{StaffOfCS} \sqsubseteq \text{Staff} \sqcap \text{MemberOfCS} \).
- \( \text{Mother} \equiv \text{Woman} \sqcap \text{Parent} \),
  \( \forall x (\text{Mother}(x) \Leftrightarrow \text{Woman}(x) \land \text{Parent}(x)) \)

  \[ : \text{Mother} \quad \text{owl : equivalentClass} \quad _\_ : x. \]
  \[ _\_ : x \quad \text{rdf : type} \quad \text{owl : Class}. \]
  \[ _\_ : x \quad \text{owl : intersectionOf} \quad ( : \text{Woman} : \text{Parent}) . \]

- Disjunction of classes: \( \text{Professor} \sqsubseteq \text{ActivelyTeaching} \sqcup \text{Retired} \).
- \( \text{Parent} \equiv \text{Mother} \sqcup \text{Father} \),
  \( \forall x (\text{Parent}(x) \Leftrightarrow \text{Mother}(x) \lor \text{Father}(x)) \)

  \[ : \text{Parent} \quad \text{owl : equivalentClass} \quad _\_ : x. \]
  \[ _\_ : x \quad \text{rdf : type} \quad \text{owl : Class}. \]
  \[ _\_ : x \quad \text{owl : unionOf} \quad ( : \text{Mother} : \text{Father}) . \]
Simple class relations.

- Negation: \( \text{ChildlessPerson} \equiv \text{Person} \cap \neg \text{Parent} \),
  \( \forall x (\text{ChildlessPerson}(x) \iff \text{Person}(x) \land \neg \text{Parent}(x)) \)

```
: ChildlessPerson owl:equivalentClass _ : x.
_ : x rdf:type owl:Class.
_ : x owl:intersectionOf (: Person _ : y).
```
**OWL syntax and intuitive semantics**

- **Simple class relations.**
  - Negation: \( \text{ChildlessPerson} \equiv \text{Person} \cap \neg \text{Parent} \),
  - \( \forall x (\text{ChildlessPerson}(x) \Leftrightarrow \text{Person}(x) \land \neg \text{Parent}(x)) \)

\[
:_x \text{owl:equivalentClass} \_ : x. \\
:_x \text{rdf:type} \_ : \text{Class}. \\
:_x \text{owl:intersectionOf} (: \text{Person} : y). \\
:_y \text{owl:complementOf} : \text{Parent}.
\]
**OWL syntax and intuitive semantics**

**Simple class relations.**

- Negation: \( \text{ChildlessPerson} \equiv \text{Person} \cap \neg \text{Parent} \),
- \( \forall x (\text{ChildlessPerson}(x) \iff \text{Person}(x) \land \neg \text{Parent}(x)) \)

```plaintext
: ChildlessPerson owl:equivalentClass _ : x.
_ : x rdf:type owl:Class.
_ : x owl:intersectionOf (: Person _ : y).
```
OWL syntax and intuitive semantics

- **Role restrictions.**
  - **All** examiners of an exam must be professors,
    \[ \text{Exam} \sqsubseteq \forall \text{hasExaminer} \cdot \text{Professor} \]
  - That any exam must have at least one examiner.
    \[ \text{Exam} \sqsubseteq \exists \text{hasExaminer} \cdot \text{Professor} \]
  - **Universal quantification:** only to be used with a role - a.k.a. property restrictions.
    \[ \text{Person} \sqcap \text{Happy} \equiv \forall \text{hasChild} \cdot \text{Parent} \]
    \[ \forall x (\text{Person}(x) \land \text{Happy}(x) \iff \forall y (\text{hasChild}(x, y) \Rightarrow \text{Happy}(y))) \]

```
_ : 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.
```
OWL syntax and intuitive semantics

- Role restrictions.

  - **All** examiners of an exam must be professors,
    \[ \text{Exam} \sqsubseteq \forall \text{hasExaminer}.\text{Professor} \]
  
  - That any exam must have at least one examiner.
    \[ \text{Exam} \sqsubseteq \exists \text{hasExaminer}.\text{Professor} \]
  
  - **Universal quantification**: only to be used with a role - a.k.a. property restrictions.
    \[ \text{Person} \sqcap \text{Happy} \equiv \forall \text{hasChild}.\text{Parent} \]
    \[ \forall x (\text{Person}(x) \land \text{Happy}(x) \leftrightarrow \forall y (\text{hasChild}(x, y) \Rightarrow \text{Happy}(y))) \]

  - \_ : x \quad \text{rdf} : \text{type} \quad \text{owl} : \text{Class}.
  
  - \_ : x \quad \text{owl} : \text{intersectionOf} \quad (: \text{Person} : \text{Happy}).
  
  - \_ : x \quad \text{owl} : \text{equivalentClass} \quad \_ : y.
  
  - \_ : y \quad \text{rdf} : \text{type} \quad \text{owl} : \text{Restriction}.
  
  - \_ : y \quad \text{owl} : \text{onProperty} \quad : \text{hasChild}.
  
  - \_ : y \quad \text{owl} : \text{allValuesFrom} \quad : \text{Parent}.
OWL syntax and intuitive semantics

- **Role restrictions.**
  - **All** examiners of an exam must be professors,
    \[ \text{Exam} \sqsubseteq \forall \text{hasExaminer} . \text{Professor} \]
  - That any exam must have at least one examiner.
    \[ \text{Exam} \sqsubseteq \exists \text{hasExaminer} . \text{Professor} \]
  - Universal quantification: only to be used with a role - a.k.a. property restrictions.
    \[ \text{Person} \sqcap \text{Happy} \equiv \forall \text{hasChild} . \text{Parent} \]
    \[ \forall x (\text{Person}(x) \land \text{Happy}(x) \iff \forall y (\text{hasChild}(x, y) \Rightarrow \text{Happy}(y))) \]

\[
\begin{align*}
_ : x & \quad \text{rdf} : \text{type} \quad \text{owl} : \text{Class}. \\
_ : x & \quad \text{owl} : \text{intersectionOf} \quad (: \text{Person} : \text{Happy}). \\
_ : x & \quad \text{owl} : \text{equivalentClass} \quad _ : y. \\
_ : y & \quad \text{rdf} : \text{type} \quad \text{owl} : \text{Restriction}. \\
_ : y & \quad \text{owl} : \text{onProperty} \quad : \text{hasChild}. \\
_ : y & \quad \text{owl} : \text{allValuesFrom} \quad : \text{Parent}.
\end{align*}
\]
OWL syntax and intuitive semantics

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

| _ : 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. |
OWL syntax and intuitive semantics

- **Existential quantification**: only to be used with a role - a.k.a. property restrictions

  \[
  \text{Parent} \equiv \exists \text{hasChild} . \text{Person} \\
  \forall x(\text{Parent}(x) \leftrightarrow \exists y(\text{hasChild}(x, y) \land \text{Person}(y)))
  \]

  
  : Parent owl : equivalentClass _ : x. 

  _ : x rdf : type owl : Restriction. 

  _ : x owl : onProperty : hasChild. 

  _ : x owl : someValuesFrom : Person.

- **Cardinality restrictions**: at most, at least and exactly. Lets understand these constructs using WorkingWithFemaleColleagues.owl

  \[
  \text{Exam } \sqsubseteq\ 2 \text{hasExaminer} . \top \\
  \text{Exam } \sqsupseteq\ 3 \text{hasTopics} . \top \\
  \text{Exam } \sqsubseteq\ = 3 \text{hasTopics} . \top
  \]
OWL syntax and intuitive semantics

**Existential quantification**: only to be used with a role - a.k.a. property restrictions

\[
\text{Parent} \equiv \exists \text{hasChild}. \text{Person}
\]

\[
\forall x (\text{Parent}(x) \iff \exists y (\text{hasChild}(x, y) \land \text{Person}(y)))
\]

: Parent owl : equivalentClass _ : x.

_ : x rdf : type owl : Restriction.

_ : x owl : onProperty : hasChild.

_ : x owl : someValuesFrom : Person.

**Cardinality restrictions**: at most, at least and exactly. Let's understand these constructs using WorkingWithFemaleColleagues.owl

\[
\text{Exam} \sqsubseteq \leq 2 \text{hasExaminer}. \top
\]

\[
\text{Exam} \sqsubseteq \geq 3 \text{hasTopics}. \top
\]

\[
\text{Exam} \sqsubseteq = 3 \text{hasTopics}. \top
\]
OWL syntax and intuitive semantics

- **Existential quantification**: only to be used with a role - a.k.a. property restrictions

  \[ Parent \equiv \exists \text{hasChild}. \text{Person} \]

  \[ \forall x (Parent(x) \iff \exists y (\text{hasChild}(x, y) \land \text{Person}(y))) \]

  
  : Parent  owl : equivalentClass  _ : x.

  _ : x  rdf : type  owl : Restriction.

  _ : x  owl : onProperty  : hasChild.

  _ : x  owl : someValuesFrom  : Person.

- **Cardinality restrictions**: at most, at least and exactly. Lets understand these constructs using WorkingWithFemaleColleagues.owl

  \[ \text{Exam} \sqsubseteq \leq 2 \text{hasExaminer}. \top \]

  \[ \text{Exam} \sqsupseteq \geq 3 \text{hasTopics}. \top \]

  \[ \text{Exam} \sqsubseteq = 3 \text{hasTopics}. \top \]
OWL syntax and intuitive semantics

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

- Properties can have following restrictions:

<table>
<thead>
<tr>
<th>Role char.</th>
<th>DL</th>
<th>e.g.,</th>
<th>General presentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transitive</td>
<td>Tra(R)</td>
<td>hasAncestor</td>
<td>$R(a, b)$ and $R(b, c) \Rightarrow R(a, c)$</td>
</tr>
<tr>
<td>Symmetric</td>
<td>Sym(R)</td>
<td>hasSpouse</td>
<td>$R(a, b) \Rightarrow R(b, a)$</td>
</tr>
<tr>
<td>Asymmetric</td>
<td>Asy(R)</td>
<td>hasChild</td>
<td>$R(a, b) \Rightarrow \neg R(b, a)$</td>
</tr>
<tr>
<td>Reflexive</td>
<td>Ref(R)</td>
<td>hasRelative</td>
<td>$R(a, a)$ for all $a$</td>
</tr>
<tr>
<td>Irreflexive</td>
<td>Irr(R)</td>
<td>parentOf</td>
<td>$\neg R(a, a)$ for any $a$</td>
</tr>
<tr>
<td>Functional</td>
<td>Fnc(R)</td>
<td>hasHusband</td>
<td>$R(a, b)$ and $R(a, c) \Rightarrow b = c$</td>
</tr>
<tr>
<td>InverseFunctional</td>
<td>Ifn(R)</td>
<td>hasHusband</td>
<td>$R(a, b)$ and $R(c, b) \Rightarrow a = c$</td>
</tr>
</tbody>
</table>
OWL syntax and intuitive semantics

- Role relationships:
  - `hasExaminer ⊑ hasParticipant`
  - `hasParticipant ≡ hasAttendee`
  - `hasAttendee⁻ ≡ participatesIn`
  - `hasExaminer⁻ ≡ examinerOf`

- Properties can have following restrictions:

<table>
<thead>
<tr>
<th>Role char.</th>
<th>DL</th>
<th>e.g.,</th>
<th>General presentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transitive</td>
<td>Tra(R)</td>
<td>hasAncestor</td>
<td><code>R(a, b) and R(b, c) ⇒ R(a, c)</code></td>
</tr>
<tr>
<td>Symmetric</td>
<td>Sym(R)</td>
<td>hasSpouse</td>
<td><code>R(a, b) ⇒ R(b, a)</code></td>
</tr>
<tr>
<td>Asymmetric</td>
<td>Asy(R)</td>
<td>hasChild</td>
<td><code>R(a, b) ⇒ not R(b, a)</code></td>
</tr>
<tr>
<td>Reflexive</td>
<td>Ref(R)</td>
<td>hasRelative</td>
<td><code>R(a, a) for all a</code></td>
</tr>
<tr>
<td>Irreflexive</td>
<td>Irr(R)</td>
<td>parentOf</td>
<td><code>not R(a, a) for any a</code></td>
</tr>
<tr>
<td>Functional</td>
<td>Fnc(R)</td>
<td>hasHusband</td>
<td><code>R(a, b) and R(a, c) ⇒ b = c</code></td>
</tr>
<tr>
<td>InverseFunctional</td>
<td>Ifn(R)</td>
<td>hasHusband</td>
<td><code>R(a, b) and R(c, b) ⇒ a = c</code></td>
</tr>
</tbody>
</table>
OWL syntax and intuitive semantics

- Role relationships:
  - `hasExaminer ⊑ hasParticipant`
  - `hasParticipant ≡ hasAttendee`
  - `hasAttendee − ≡ participatesIn`
  - `hasExaminer − ≡ examinerOf`

- Properties can have following restrictions:

<table>
<thead>
<tr>
<th>Role char.</th>
<th>DL</th>
<th>e.g.,</th>
<th>General presentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transitive</td>
<td>Tra(R)</td>
<td>hasAncestor</td>
<td>( R(a, b) ) and ( R(b, c) ) (⇒) ( R(a, c) )</td>
</tr>
<tr>
<td>Symmetric</td>
<td>Sym(R)</td>
<td>hasSpouse</td>
<td>( R(a, b) ) (⇒) ( R(b, a) )</td>
</tr>
<tr>
<td>Asymmetric</td>
<td>Asy(R)</td>
<td>hasChild</td>
<td>( R(a, b) ) (⇒) ( \neg R(b, a) )</td>
</tr>
<tr>
<td>Reflexive</td>
<td>Ref(R)</td>
<td>hasRelative</td>
<td>( R(a, a) ) for all ( a )</td>
</tr>
<tr>
<td>Irreflexive</td>
<td>Irr(R)</td>
<td>parentOf</td>
<td>( \neg R(a, a) ) for any ( a )</td>
</tr>
<tr>
<td>Functional</td>
<td>Fnc(R)</td>
<td>hasHusband</td>
<td>( R(a, b) ) and ( R(a, c) ) (⇒) ( b = c )</td>
</tr>
<tr>
<td>InverseFunctional</td>
<td>Ifn(R)</td>
<td>hasHusband</td>
<td>( R(a, b) ) and ( R(c, b) ) (⇒) ( a = c )</td>
</tr>
</tbody>
</table>
OWL syntax and intuitive semantics

- **Role relationships:**
  - \( Sym(hasColleague) \)
  - \( Tra(hasColleague) \)
  - \( Fun(hasTeamLeader) \)
  - \( Ifn(isTeamLeaderFor) \)
  - \( hasColleague(UabboVisser, AndreasSeekircher) \)
  - \( hasColleague(AndreasSeekircher, JustinStoecker) \)
  - \( hasColleague(JustinStoecker, SamindaAbeyruwan) \)
  - \( isTeamLeaderFor(UabboVisser, RoboCanes) \)

- **Self:** \( PersonCommittingSuicide \equiv \exists kills. Self \)
- **Disjoint properties:** \( Dis(S, R) : Dis(hasParent, hasChild) \)
- **Negated role assignment:** \( \neg hasColleague(UabboVisser, NadalRafael) \)
- **Role chains:** \( hasParent \circ hasBrother \sqsubseteq hasUncle \)
- **Datatypes (D):**
  - \( hasAge(Sam, "30"^\text{xsd:integer}) \)
  - \( \neg hasHeight(Sam, "6.0"^\text{xsd:float}) \)
  - \( Teenager \equiv Person \sqcap \exists hasAge.(\text{xsd:integer} \geq 12 \text{ and } \leq 19) \)
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 ≡ ∃kills.Self

- Disjoint properties, Dis(S,R) : Dis(hasParent, hasChild)

- Negated role assignment: ¬hasColleague(UbboVisser, NadalRafael)

- Role chains: hasParent ◦ hasBrother ⊑ hasUncle

- Datatypes (D):
  - hasAge(Sam, "30"^^xsd:integer)
  - ¬hasHeight(Sam, "6.0"^^xsd:float)
  - Teenager ≡ Person ⊓ ∃hasAge.(xsd : integer ≥ 12 and ≤ 19)
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: \( \text{PersonCommittingSuicide} \equiv \exists \text{kills}.\text{Self} \)

- Disjoint properties, Dis(S,R): \( \text{Dis(hasParent, hasChild)} \)
  - Negated role assignment: \( \neg \text{hasColleague(UbboVisser, NadalRafael)} \)
  - Role chains: \( \text{hasParent} \circ \text{hasBrother} \sqsubseteq \text{hasUncle} \)
  - Datatypes (D):
    - hasAge(Sam, “30”^xsd:integer)
    - \( \neg \text{hasHeight(Sam, “6.0”^xsd:float)} \)
    - Teenager \( \equiv \text{Person} \sqcap \exists \text{hasAge}.(\text{xsd : integer } \geq 12 \text{ and } \leq 19) \)
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 ≡ ∃kills.Self
- Disjoint properties, Dis(S,R) : Dis(hasParent, hasChild)
- Negated role assignment: ¬hasColleague(UbboVisser, NadalRafael)
- Role chains: hasParent ◦ hasBrother ⊑ hasUncle
- Datatypes (D):
  - hasAge(Sam, “30”ˆxsd:integer)
  - ¬hasHeight(Sam, “6.0”ˆxsd:float)
  - Teenager ≡ Person ∩ ∃hasAge.(xsd : integer ≥ 12 and ≤ 19)
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 ≡ ∃kills.Self

- Disjoint properties, Dis(S,R) : Dis(hasParent, hasChild)

- Negated role assignment: ¬hasColleague(UbboVisser, NadalRafael)

- Role chains: hasParent o hasBrother ⊑ hasUncle

- Datatypes (D):
  - hasAge(Sam, "30"^xsd:integer)
  - ¬hasHeight(Sam, "6.0"^xsd:float)
  - Teenager ≡ Person ⊓ ∃hasAge.(xsd : integer ≥ 12 and ≤ 19)
OWL syntax and intuitive semantics

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

- Self: \( PersonCommittingSuicide \equiv \exists \text{kills.} \text{Self} \)

- Disjoint properties, \( Dis(S,R) : Dis(hasParent, hasChild) \)

- Negated role assignment: \( \neg hasColleague(UabboVisser, NadalRafael) \)

- Role chains: \( hasParent \circ hasBrother \sqsubseteq hasUncle \)

- Datatypes (D):
  - \( hasAge(Sam, "30"^\text{xsd:integer}) \)
  - \( \neg hasHeight(Sam, "6.0"^\text{xsd:float}) \)
  - \( Teenager \equiv Person \sqcap \exists hasAge.(\text{xsd:integer} \geq 12 \text{ and } \leq 19) \)
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: $\sqcap \exists \top \sqsubseteq \sqcap \exists \top \sqsubseteq$, 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.,
    
    $\text{Human} \sqsubseteq \exists \text{hasParent}.\text{Person}$,
    
    $\exists \text{married}.\top \sqcap \text{CatholicPriest} \sqsubseteq \bot$,
    
    $\text{hasParent} \circ \text{hasParent} \sqsubseteq \text{hasGrandparent}$. 

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., $\exists \text{married.} \top \sqsubseteq \neg \text{Free} \sqcap \exists \text{has}. \text{Sorrow}$.

- **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 provides 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.
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 provides entailments.

**Database:**
- Close World Assumption (CWA): missing information is false.
- Unique Name Assumption (UNA): each individual has is uniquely identifiable.
- Database schema provides structure on data.
E.g.,

- **T-Box:**

  \[
  \text{HogwartsStudent} \equiv \text{Student} \sqcap \exists \text{attendsSchool} . \text{Hogwarts} \\
  \text{HogwartsStudent} \sqsubseteq \forall \text{hasPet} . (\text{Owl} \sqcup \text{Cat} \sqcup \text{Toad}) \\
  \text{hasPet} \equiv \text{isPetOf}^\sim \\
  \exists \text{hasPet} . \top \sqsubseteq \text{Human} \\
  \text{Phoenix} \sqsubseteq \forall \text{isPetOf} . \text{Wizard} \\
  \text{Muggle} \sqcap \text{Wizard} \sqsubseteq \bot
  \]

- **A-Box:**

  \[
  \text{Wizard}(\text{HarryPotter}) \\
  \text{Wizard}(\text{DracoMalfoy}) \\
  \text{hasFriend}(\text{HarryPotter}, \text{RonWeasley}) \\
  \text{hasFriend}(\text{HarryPotter}, \text{HermioneGranger}) \\
  \text{hasPet}(\text{HarryPotter}, \text{Hedwig})
  \]
E.g.,

- **T-Box:**

  \[
  \text{HogwartsStudent} \equiv \text{Student} \sqcap \exists \text{attendsSchool}.\text{Hogwarts} \\
  \text{HogwartsStudent} \sqsubseteq \forall \text{hasPet}.(\text{Owl} \sqcup \text{Cat} \sqcup \text{Toad}) \\
  \text{hasPet} \equiv \text{isPetOf}^- \\
  \exists \text{hasPet}.\top \sqsubseteq \text{Human} \\
  \text{Phoenix} \sqsubseteq \forall \text{isPetOf} . \text{Wizard} \\
  \text{Muggle} \sqcap \text{Wizard} \sqsubseteq \bot
  \]

- **A-Box:**

  \[
  \text{Wizard}(\text{HarryPotter}) \\
  \text{Wizard}(\text{DracoMalfoy}) \\
  \text{hasFriend}(\text{HarryPotter}, \text{RonWeasley}) \\
  \text{hasFriend}(\text{HarryPotter}, \text{HermioneGranger}) \\
  \text{hasPet}(\text{HarryPotter}, \text{Hedwig})
  \]
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:**
  \[
  \text{HarryPottersFriends} \equiv \forall \text{hasFriend}. \{ \text{RonWeasley} \sqcap \text{HermioneGranger} \}
  \]
  \[
  \text{Wizard} \sqcap \text{HarryPottersFriends}(\text{HarryPotter})
  \]
- How many friends does *HarryPotter* have?
  Ontology: 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**: 
  
  \[
  \text{HarryPottersFriends} \equiv \forall \text{hasFriend.}\{\text{RonWeasley} \sqcup \text{HermioneGranger}\} \\
  \text{Wizard} \sqcap \text{HarryPottersFriends(HarryPotter)}
  \]

- How many friends does *HarryPotter* have?  
  Ontology: 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: \( \text{Dis} (\text{RonWeasley}, \text{HermioneGranger}) \)

- How many friends does *HarryPotter* have?
  Ontology: at least 2, Database: 2!

- T-Box:
  \[
  \text{HarryPottersFriends} \equiv \forall \text{hasFriend}. \{ \text{RonWeasley} \sqcap \text{HermioneGranger} \}
  \]
  \[
  \text{Wizard} \sqcap \text{HarryPottersFriends}(\text{HarryPotter})
  \]

- How many friends does *HarryPotter* have?
  Ontology: 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:**  \( \text{Dis}(\text{RonWeasley}, \text{HermioneGranger}) \)
  How many friends does *HarryPotter* have?
  Ontology: at least 2, Database: 2!

  **T-Box:**
  \[ \text{HarryPottersFriends} \equiv \forall \text{hasFriend}.\{\text{RonWeasley} \sqcup \text{HermioneGranger}\} \]
  \[ \text{Wizard} \sqcap \text{HarryPottersFriends}(\text{HarryPotter}) \]

- How many friends does *HarryPotter* have?
  Ontology: 2!, Database: 2!
E.g.,

- **Is Draco Malfoy a friend of Harry Potter?**
  - Ontology: Don’t know (OWA), Database: No!

- **How many friends does Harry Potter have?**
  - Ontology: At least 1 (NUNA), Database: 2!
  
  **A-Box**: \( \text{Dis}(\text{Ron Weasley}, \text{Hermione Granger}) \)

- **How many friends does Harry Potter have?**
  - Ontology: at least 2, Database: 2!

  **T-Box**:
  
  \[
  \text{Harry Potters Friends} \equiv \forall \text{hasFriend}.\{\text{Ron Weasley} \sqcup \text{Hermione Granger}\}
  \]
  
  \[
  \text{Wizard} \sqcap \text{Harry Potters Friends}(\text{Harry Potter})
  \]

- **How many friends does Harry Potter have?**
  - Ontology: 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**: $\text{Dis}(\text{RonWeasley}, \text{HermioneGranger})$

- How many friends does *HarryPotter* have?  
  Ontology: at least 2, Database: 2!

- **T-Box**:
  
  $\text{HarryPottersFriends} \equiv \forall \text{hasFriend}.\{\text{RonWeasley} \sqcap \text{HermioneGranger}\}$

  $\text{Wizard} \sqcap \text{HarryPottersFriends}(\text{HarryPotter})$

- How many friends does *HarryPotter* have?  
  Ontology: 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**:  
  \[ \text{Dis}(\text{RonWeasley}, \text{HermioneGranger}) \]

- How many friends does *HarryPotter* have?  
  Ontology: at least 2, Database: 2!

- **T-Box**:  
  \[ \text{HarryPottersFriends} \equiv \forall \text{hasFriend}. \{ \text{RonWeasley} \sqcap \text{HermioneGranger} \} \]
  \[ \text{Wizard} \sqcap \text{HarryPottersFriends}(\text{HarryPotter}) \]

- 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: \( \exists \text{hasPet}. \top \sqsubseteq \text{Human}, \text{Phoenix} \sqsubseteq \forall \text{isPetOf}. \text{Wizard} \)

- A-Box:
  
  \[
  \begin{align*}
  \text{Wizard}(\text{Dumbledore}) \\
  \text{Phoenix}(\text{Fawkes}) \\
  \text{isPetOf}(\text{Fawkes}, \text{Dumbledore})
  \end{align*}
  \]

- Ontology infers that \( \text{Human} \sqcap \text{Wizard}(\text{Dumbledore}) \)

- Database rejects, because domain of \( \text{hasPet} \) is \( \text{Human} \), and \( \text{Dumbledore} \) is not \( \text{Human} \) (CWA).

- Ontologies use theorem proving to answer questions.

- It involves both A-Box and T-Box, and has higher worst case complexities.
# OWL 2 features

<table>
<thead>
<tr>
<th>Feature</th>
<th>DL</th>
<th>FOL</th>
<th>Vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top/bottom class</td>
<td>$\top / \bot$</td>
<td>Axiomatize</td>
<td>$\text{owl:Thing} / \text{owl:Nothing}$</td>
</tr>
<tr>
<td>Class intersection</td>
<td>$\sqcap$</td>
<td>$\land$</td>
<td>$\text{owl:intersectionOf}$</td>
</tr>
<tr>
<td>Class union</td>
<td>$\sqcup$</td>
<td>$\lor$</td>
<td>$\text{owl:unionOf}$</td>
</tr>
<tr>
<td>Class complement</td>
<td>$\neg$</td>
<td>$\neg$</td>
<td>$\text{owl:complementOf}$</td>
</tr>
<tr>
<td>Enumerated classes</td>
<td>${a}$</td>
<td>$\approx$</td>
<td>$\text{owl:oneOf}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property restriction</th>
<th>DL</th>
<th>FOL</th>
<th>Vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existential</td>
<td>$\exists R.C$</td>
<td>$\exists x \ldots$</td>
<td>$\text{owl:someValuesFrom}$</td>
</tr>
<tr>
<td>Universal</td>
<td>$\forall R.C$</td>
<td>$\forall x \ldots$</td>
<td>$\text{owl:allValuesFrom}$</td>
</tr>
<tr>
<td>Min. cardinality</td>
<td>$\geq n R.C$</td>
<td>$\exists x_1, \ldots, x_n$</td>
<td>$\text{owl:minQualifiedCardinality} / \text{owl:onClass}$</td>
</tr>
<tr>
<td>Max. cardinality</td>
<td>$\leq n R.C$</td>
<td>$\forall x_1, \ldots, x_n$</td>
<td>$\text{owl:maxQualifiedCardinality} / \text{owl:onClass}$</td>
</tr>
<tr>
<td>Local reflexivity</td>
<td>$\exists R.Self$</td>
<td>$R(x, x)$</td>
<td>$\text{owl:hasSelf}$</td>
</tr>
</tbody>
</table>
## OWL 2 features

<table>
<thead>
<tr>
<th>Feature</th>
<th>DL</th>
<th>FOL</th>
<th>Vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property chain</td>
<td>○</td>
<td>Axiomatize</td>
<td>owl:propertyChainAxiom</td>
</tr>
<tr>
<td>Inverse</td>
<td>$R^-$</td>
<td>Axiomatize</td>
<td>owl:inverseOf</td>
</tr>
<tr>
<td>Key</td>
<td>-</td>
<td>Axiomatize</td>
<td>owl:hasKey</td>
</tr>
<tr>
<td>Property disjointness</td>
<td>$\text{Dis}(R, S)$</td>
<td>Axiomatize</td>
<td>owl:propertyDisjointWith</td>
</tr>
<tr>
<td><strong>Property characteristics</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symmetric</td>
<td>$\text{Sym}(R)$</td>
<td>Axiomatize</td>
<td>owl:SymmetricProperty</td>
</tr>
<tr>
<td>Asymmetric</td>
<td>$\text{Asy}(R)$</td>
<td>Axiomatize</td>
<td>owl:AsymmetricProperty</td>
</tr>
<tr>
<td>Reflexive</td>
<td>$\text{Ref}(R)$</td>
<td>Axiomatize</td>
<td>owl:ReflexiveProperty</td>
</tr>
<tr>
<td>Irreflexive</td>
<td>$\text{Irr}(R)$</td>
<td>Axiomatize</td>
<td>owl:IrreflexiveProperty</td>
</tr>
<tr>
<td>Transitive</td>
<td>$\text{Tra}(R)$</td>
<td>Axiomatize</td>
<td>owl:TransitiveProperty</td>
</tr>
<tr>
<td>Functional</td>
<td>$\text{Fun}(R)$</td>
<td>Axiomatize</td>
<td>owl:FunctionalProperty</td>
</tr>
<tr>
<td>Inverse functional</td>
<td>$\text{Ifn}(R)$</td>
<td>Axiomatize</td>
<td>owl:InverseFunctionalProperty</td>
</tr>
<tr>
<td><strong>Subclass</strong></td>
<td>$C \sqsubseteq D$</td>
<td>$\forall x. C(x) \Rightarrow D(x)$</td>
<td>rdfs:subClassOf</td>
</tr>
<tr>
<td><strong>Subproperty</strong></td>
<td>$R \sqsubseteq S$</td>
<td>$\forall x, y. R(x, y) \Rightarrow S(x, y)$</td>
<td>rdfs:subPropertyOf</td>
</tr>
</tbody>
</table>
Model these axioms in an ontology

- \{John\} \cap \{Bill\} \sqsubseteq \bot
- \{John\} \equiv \{Jim\}
- \neg hasWife(Bill, Mary)
- Woman \cap Man \sqsubseteq \bot
- Parent \equiv Mother \sqcup Father
- ChildlessPerson \equiv Person \cap \neg Parent
- Person \cap \neg Parent(Jack)
- Parent \equiv \exists hasChild . Person
- Orphan \equiv \forall hasChild \neg . Dead
- JohnsChildren \equiv \exists hasParent . \{John\}
- NarcisticPerson \equiv \exists loves . Self
- MyBirthdayGuests \equiv \{Bill, John, Mary\}
- hasParent \circ hasParent \sqsubseteq hasGrandparent
Franz Baader, Ian Horrocks, and Ulrike Sattler.
Description logics as ontology languages for the semantic web.

Tim Berners-Lee.
Artificial Intelligence and the Semantic Web.

Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph.
Foundations of Semantic Web Technologies.

Ian Horrocks.
Description Logic: A formal foundation for languages and tools. Tutorial at the Semantic Technology Conference (SemTech).
San Francisco, California, USA.