Ontology

Description Logic

- is a formal language at the basis of ontologies description
- It is decidable
  - is a subset of C2 a decidable subset of FOL (FOL is semi-decidable)
- Many description logic languages with increasing expressivity and increasing complexity
A descriptive language \( L \) consists of three finite sets: \((NC; NR; Ob)\).

- Elements in \( NC \) are indicated with letters \( A, B, \) ecc. and are called **atomic concepts** of \( L \);
- Elements in \( NR \) are indicated with letters \( r, s, \) ecc. and are called **roles** of \( L \);
- Elements in \( Ob \) are indicated with letters \( a, b, \) ecc. are called **object (names)** of \( L \).

**ALC**

- **ALC = Attribute concept Language with Complements**
- A Basic Description Logic
- **Example Axioms**
  - \( \text{HappyMan} \equiv \text{Human} \land \neg \text{Female} \land \exists \text{married.} \text{Doctor} \land \forall \text{hasChild.(Doctor} \lor \text{Professor}) \)
  - \( \exists \text{hasChild.} \text{Human} \subseteq \text{Human} \)
  - ...  
  - \( \text{NC} = \{\text{Human, Female, Doctor, Professor,...}\} \)
  - \( \text{NR} = \{\text{married, hasChild, ...}\} \)
**ALC syntax**

- Let NC be a **set of concept names** and NR be a **set of role names**.
- The set of **ALC-concept descriptions** is the smallest set such that:
  - T, ⊥, and every concept name A ∈ NC is an **ALC-concept description**,
  - if C and D are **ALC-concept descriptions** and r ∈ NR, then C ∩ D, C ∪ D, ¬C, ∀r.C, and ∃r.C are **ALC-concept descriptions**.

**ALC semantics**

- $I = (Δ^I, ·^I)$ **interpretation**
- $A^I ⊆ Δ^I$ for $A ∈ NC$
- $r^I ⊆ Δ^I × Δ^I$ for $r ∈ NR$
- $T^I = Δ^I$
- $⊥^I = ∅$
- $(¬C)^I = Δ^I \setminus (C)^I$
- $(C ∩ D)^I = (C)^I ∩ (D)^I$
- $(C ∪ D)^I = (C)^I ∪ (D)^I$
- $(∀r.C)^I = \{ x ∈ Δ^I \mid ∀y ∈ Δ^I. (x,y) ∈ r^I ⇒ y ∈ C^I \}$
- $(∃r.C)^I = \{ x ∈ Δ^I \mid ∃y ∈ Δ^I. (x,y) ∈ r^I ∧ y ∈ C^I \}$
General concept inclusion (GCI)
- $C \sqsubseteq D$ (C and D are ALC concepts)
- $(C \sqsubseteq D)^I \iff (C)^I \subseteq (D)^I$
- $C \equiv D \iff C \sqsubseteq D$ and $D \sqsubseteq C$

T-Box (Terminological Box) a set of axioms describing the concepts relations, a set of GCI

Example TBox
- Human $\sqsubseteq$ Animal
- Bird $\sqsubseteq$ Animal $\sqcap$ CanFly
- Helicopter $\sqsubseteq$ InanimatedObject $\sqcap$ CanFly
- Animal $\sqcap$ InanimatedObject $\equiv \bot$

The ABox (Assertional Box) contains axioms telling:
- that an object with name $x$ belongs to an ALC concept expression:
  - $x : C$
  - $(x : C)^I$ if $x^I \in (C)^I$
- that two objects are in a role
  - $<x,y> : r$
  - $(<x,y> : r)^I$ if $(x^I,y^I) \in (r)^I$

Examples
- $b_1 : Bird$
- $h_1 : Helicopter$
- $<john, jane> : married$
An ALC KB is made of a TBox and an ABox

The TBox states the containment relations among the concepts that typically form a hierarchy of concepts

ALC to FOL

ALC can be translated to FOL inductively

- \([A]_x = A(x)\)
- \([C \cap D]_x = [C]_x \land [D]_x\)
- \([C \cup D]_x = [C]_x \lor [D]_x\)
- \([-C]_x = \neg [C]_x\)
- \([\exists r. C]_x = \exists y. r(x, y) \land [C]_y\)
- \([\forall r. C]_x = \forall y. r(x, y) \Rightarrow [C]_y\)
- \([T]_x = T, [\bot]_x = \bot\)
- \([C \sqsubseteq D]_x = \forall x. [C]_x \Rightarrow [D]_x\)
- \([C \equiv D]_x = \forall x. [C]_x \Leftrightarrow [D]_x\)
- \([a : C] = [C]_a\)
- \([<a, b> : r] = r(a, b)\)
Example

- $\exists$ hasChild. Human $\subseteq$ Human

- $[\exists$ hasChild. Human $\subseteq$ Human] =
  $\forall x. [\exists$ hasChild. Human]$_x$ $\Rightarrow$ [Human]$_x$ =
  $\forall x. \exists y. \text{hasChild}(x, y) \land [\text{Human}]_y$ $\Rightarrow$ [Human]$_x$ =
  $\forall x. \exists y. \text{hasChild}(x, y) \land \text{Human}(y) \Rightarrow \text{Human}(x)$

Many DLs

Many **Descriptive Logics (DL) derived from ALC**

- $\mathcal{S} \rightarrow$ ALC + transitive roles (ex. $\text{Tr(isPartOf)}$)
- $\mathcal{H} \rightarrow$ role hierarchy (ex., $\text{hasDaughter} \subseteq \text{hasChild}$)
- $\mathcal{F} \rightarrow$ functional roles
- $\mathcal{O} \rightarrow$ define concept as enumeration $\{a_1, \ldots, a_n\}$
- $\mathcal{N} \rightarrow$ cardinality restrictions (ex., $\leq 2\text{hasChild}$)
- $\mathcal{Q} \rightarrow$ qualified cardinality restrictions (ex., $\geq 3\text{hasChild.Female}$)
- $\mathcal{I} \rightarrow$ inverse roles (ex., $\text{isChildOf} \equiv \text{hasChild}^{-1}$)

Every DL characterized by the use of particular logical operators

- OWL-Lite = $\mathcal{SHIF}$
- OWL-DL = $\mathcal{SHOIN}$
**Concepts**

**Atomic** *(often indicated with letters A and B)*

- **WOMAN**
  - intuitively means "WOMAN"

**Complex** *(often indicated with letters C and D)*

- **PERSON ⊓ FEMALE**
  - reads "PERSON and FEMALE" or "PERSON intersection FEMALE"
  - intuitively means "person of female genre"

**Often called**

- **terms or classes** *(because represent sets of objects)*

**Equivalences**

**Concepts Equivalence**

- **WOMAN ⊑ PERSON ⊓ FEMALE**
  - Intuitively tells that "WOMAN equals to PERSON and FEMALE"

**In general**

- **C ⊑ D**
  - reads "C equals to D"
  - Express the equivalence of "C" and "D"

**Concepts definition**

- **WOMAN ⊑ PERSON ⊓ FEMALE**
  - Definition of term "WOMAN" from terms "PERSON" and "FEMALE"
**Subsumption (sussunzione)**

**Concepts Subsumption**

GIRL ⊑ WOMAN

Reads “GIRL is subsumed from WOMAN” or “WOMAN subsumes GIRL”

Intuitively means “a girl is a woman”

In general

C ⊑ D

Every individual in “C” is also described by “D”

**Equivalence as double subsumption**

C ≡ D is the same as C ⊑ D and D ⊑ C

**DL vs. Boolean Algebra**

operators ¬, ∨, ⊓, ⊔, ⊤ form a boolean algebra:

¬ T equals to ⊥

¬ ¬ C equals to C

¬ (C ∩ D) equals to ¬ C ⊔ ¬ D

¬ (C ⊔ D) equals to ¬ C ∩ ¬ D
Existential quantification

Example:

\[ \text{MOTHER} \sqsubseteq \exists \text{hasChild} \]

Intuitively means "every mother has as child at least an individual"
reads "MOTHER is subsumed from the set of individuals that have at least one child"

Translated to FOL

\[ \text{MOTHER} \sqsubseteq \exists \text{hasChild} \text{ becomes} \]
\[ \forall x (\text{MADRE}(x) \rightarrow \exists \text{y hasChild}(x, y)) \]

Qualified Existential Quantification

Example:

\[ \exists \text{hasChild.FEMALE} \]

denotes the set of all individuals that have a female as child.

Translated to FOL:

\[ \exists \text{hasChild.FEMALE} \text{ becomes} \]
\[ \exists y (\text{hasChild}(x, y) \land \text{FEMALE}(y)) \]
**Universal quantification**

Example:

\[ \forall \text{hasChild}. \text{FEMALE} \]

reads "the set of all individuals that have only female children"

Translated to FOL:

\[ \forall \text{hasChild}. \text{FEMALE} \]

becomes

\[ \exists y \forall x (\text{hasChild}(x, y) \land [\text{FEMALE}]y) \]

\[ \forall y (\text{hasChild}(x, y) \rightarrow \text{FEMALE}(y)) \]

---

**Inverse role**

We define “sonOf” starting from “hasChild” using notation “hasChild –”

\[ \text{sonOf} \equiv \text{hasChild} – \]

Example

\[ \exists \text{hasChild}. \text{FEMALE} \]

becomes

\[ \exists y (\text{hasChild}(y, x) \land [\text{FEMALE}]y) \]

\[ \forall y (\text{hasChild}(y, x) \rightarrow [\text{FEMALE}]y) \]

In general

\[ R \text{ express the relation } R(x, y) \]

\[ R^- \text{ express relation } R(y, x) \]

\[ \exists R^- . C \text{ becomes } [\exists R^- . C] = \exists y (R(y, x) \land [C]y) \]

\[ \forall R^- . C \text{ becomes } [\forall R^- . C] = \forall y (R(y, x) \rightarrow [C]y) \]
Domain and range (1)

• Roles, in general, have **sense** only for certain subsets of the universe.

Example

`hasChild` relates in general two people, while it does not have sense for other inanimate objects.

• We can associate to a role `R` two sets, called **domain** and **range** of the role, that represents the sets of individuals used for the variables `x` and `y` in expression `R(x,y)`

\[
\forall R - D \subseteq \text{domain of } R \\
\forall R - C \subseteq \text{range of } R
\]

Example

`ownerOf`:

\[
\forall \text{ownerOf}. \text{PERSON} \\
\text{domain of ownerOf is the set of Person}
\]

\[
\forall \text{ownerOf}. \text{GOOD} \\
\text{range of ownerOf is the set of Goods}
\]

In short

`R : D \rightarrow C`

`ownerOf : \text{PERSON} \rightarrow \text{GOOD}`

Domain and range (2)
It is possible to express cardinality constraints on roles:

\[ \leq n_R \quad \geq n_R \]

or qualified cardinality constraints:

\[ \leq n_R.C \quad \geq n_R.C \]

where \( n \) is an integer \( \geq 0 \)

Example:

\[ \text{PARENT}_3F \equiv \geq 3 \text{hasChild}.\text{FEMALE} \]

“PARENT\(_3\)F is the set of individuals that are parents of at least 3 daughters”

Translated to FOL

\[ \leq n_R \text{ becomes } \exists \leq n_y R(x, y) \]

\[ \geq n_R \text{ becomes } \exists \geq n_y R(x, y) \]

\[ \leq n_R.C \text{ becomes } \exists \leq n_y (R(x, y) \land [C]y) \]

\[ \geq n_R.C \text{ becomes } \exists \geq n_y (R(x, y) \land [C]y) \]
Cardinality constraints(3)

The last example:

\[ \text{PARENT}3\text{F} \equiv \geq \text{3} \text{hasChild.FEMALE} \]
becomes:

\[
\forall x \ (\text{PARENT}3\text{F}(x) \leftrightarrow \exists y_1, \exists y_2, \exists y_3, \\\n( \text{hasChild}(x, y_1) \land \text{FEMALE}(y_1) \land y_1 \neq y_2 \land y_1 \neq y_3 \land y_2 \neq y_3 ),
\]

Definitions:

\[
\begin{align*}
= nR & \triangleq \leq nR \land \geq nR \\
= nR.C & \triangleq \leq nR.C \land \geq nR.C
\end{align*}
\]

Observations:

\[
\begin{align*}
\geq 1R.C & \quad \text{same as} \quad \exists R.C \\
\leq 0R.C & \quad \text{same as} \quad \neg \exists R.C \\
\geq 0R.C & \quad \text{same as} \quad \top
\end{align*}
\]
Functional roles (1)

A **functional role** is a binary relation where every element in the domain is in relation with at most an element in the range.

Example:

wifeOf : WOMAN → MAN

domain and range:

\[ \top \subseteq \forall \text{wifeOf.MAN} \]
\[ \top \subseteq \forall \text{wifeOf.WOMAN} \]

**role wifeOf is functional**
because every wife can have at most a husband

\[ \text{WOMAN} \sqsubseteq \leq \text{wifeOf} \]

Role subsumption

In many DL it is possible to express subsumption and equivalence between roles with expressions like:

\[ R \sqsubseteq S \text{ becomes in FOL } \forall x \forall y (R(x, y) \rightarrow S(x, y)) \]
\[ R \equiv S \text{ becomes in FOL } \forall x \forall y (R(x, y) \leftrightarrow S(x, y)) \]

Example

parentOf \sqsubseteq \text{relativeOf}

*parent* is a kind of relationship

sonOf \equiv \text{parentOf-}

*sonOf* is the inverse of *parentOf*

Symmetric property

\[ R \sqsubseteq R- \text{ becomes in FOL } \forall x \forall y (R(x, y) \rightarrow R(y, x)) \]
\[ \text{SiblingOf} \sqsubseteq \text{SiblingOf-} \]
Roles composition

in some DL it is possible to build complex roles using composition operator $\circ$

Given two roles $R$ and $S$:

$$R \circ S \text{ becomes in FOL } \exists z (R(x, z) \land S(z, y))$$

Role composition is useful but often it is not admitted for problems with decidibility.

Transitive property

Definition

$$(R \circ R) \sqsubseteq R \text{ becomes in FOL } \forall x \forall y (\exists z (R(x, z) \land R(z, y)) \rightarrow R(x, y))$$

Many DL (like SHOIN$^+$OWL) do not allow role composition but provide an operator to declare a role as transitive.

$$Tr(R)$$

Equivalence

$$\forall (R \circ S).C \text{ same as } \forall (R.\forall S.C)$$
$$\exists (R \circ S).C \text{ same as } \exists (R.\exists S.C)$$
Example

TBox

\[\text{Faculty} \sqsubseteq \text{Organization}\]
\[\text{FullProfessor} \sqsubseteq \text{Teacher}\]
\[\text{FullProfessor} \sqsubseteq \exists \text{hasSDS.ScientificDisciplinarySector}\]
\[\text{Course} \sqsubseteq \exists \text{courseTakenAtTime.Interval}\]
\[\text{Person} \sqsubseteq \exists \text{hasCompetence.ConceptSkill}\]
\[\text{Course} \sqsubseteq \exists \text{subject.ConceptSkill}\]

ABox

Paolo Rossi: FullProfessor,
Knowledge Management and Protection Systems : Course,
Distributed Systems: ConceptSkill,
Cloud Computing: ConceptSkill,
(Paolo Rossi, Knowledge Management and Protection Systems):courseTaken,
(Paolo Rossi, Cloud Computing):hasCompetence,
(Knowledge Management and Protection Systems, Distributed Systems):subject

OWL (1)

- **OWL** (Web Ontology Language) is the standard proposed by **W3C** for the definitions of ontologies
  - OWL Lite: \text{SHIF(D)};
  - OWL DL: corresponds to \text{SHION(D)};
  - OWL 2 DL: corresponds to \text{SROIQ(D)} and is the “normal”
    - OWL 2 (sublanguage): “maximum” expressivity while keeping reasoning problems decidable—but still very expensive;
  - (Other) profiles are tailored for specific ends, e.g.,
    - OWL 2 QL: is specifically designed for efficient database integration;
    - OWL 2 EL: is a lightweight language with polynomial time reasoning;
    - OWL 2 RL: is designed for compatibility with rule-based inference tools.

[Http://www.w3.org/TR/owl-ref/](http://www.w3.org/TR/owl-ref/)
An OWL ontology is made of a TBox and an ABox both represented as RDF graphs (set of triples)

Some RDFS constructs are adopted from OWL

OWL introduces many constructs not present in RDFS, represented as RDF triples

In OWL

- Terms or concepts are classes
- The Operators to define terms are called class constructors
- Roles are called properties
- The definitions in the TBox are called class axioms
- ABox assertions are facts
**Identifier**

- All class descriptions describe a resource of type: `owl:Class`
- In the simplest case the description is made of a class identifier (Unique Resource Identifier = URI), corresponding to an atomic term of the DL
- **RDF Syntax:**
  
  ```
  <owl:Class rdf:ID="ClassName"/>
  <owl:Class rdf:about="ClassName"/>
  ```

Two class identifiers are predefined in OWL

- `owl:Thing` = the set of all individuals (\(T\) for universal class)
  - All OWL classes are subclass of `owl:Thing`
- `owl:Nothing` = empty set (\(\bot\) for empty class)
  - The `owl:Nothing` class is a subclass of all classes

**Enumeration**

- A class can be described from enumeration of a finite set of nominals \(a_1, \ldots, a_n\) using operator `owl:oneOf`
  - **DL syntax:**
    \[
    [a_1, \ldots, a_n]
    \]

  - **RDF syntax:**

    ```
    <owl:oneOf rdf:parseType="Collection">
      <owl:Thing rdf:about="#a1" />
      ... 
      <owl:Thing rdf:about="#an" />
    </owl:oneOf>
    ```
Intersection

- A class $A$ can be described as the *intersection* of a finite number of classes $C_1, \ldots, C_n$ using operator `owl:intersectionOf`

- DL syntax:
  $A \equiv C_1 \cap \ldots \cap C_n$

- RDF syntax:
  ```
  <owl:Class rdf:ID="A">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#C1" />
      \ldots
      <owl:Class rdf:about="#Cn" />
    </owl:intersectionOf>
  </owl:Class>
  ```

Union

- A class $A$ can be described as *union* of a finite number of classes $C_1, \ldots, C_n$ using operator `owl:unionOf`

- DL syntax:
  $A \equiv C_1 \cup \ldots \cup C_n$

- RDF syntax:
  ```
  <owl:Class rdf:ID="A">
    <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#C1" />
      \ldots
      <owl:Class rdf:about="#Cn" />
    </owl:unionOf>
  </owl:Class>
  ```
Complement

- A class $A$ can be described as complement of class $B$ using operator `owl:complementOf`
- **DL syntax:**
  $A \equiv \neg B$
- **RDF syntax:**
  ```xml
  <owl:Class rdf:ID="A">
    <owl:complementOf>
      <owl:Class rdf:about="#B" />
    </owl:complementOf>
  </owl:Class>
  ```

Class axioms: Subclasses

- Between two class descriptions $C$ e $D$ can be defined the subclass relation using operator `rdfs:subClassOf`
- **DL syntax:**
  $C \sqsubseteq D$
- **RDF syntax:**
  ```xml
  <owl:Class rdf:about="#C">
    <rdfs:subClassOf rdf:resource="#D" />
  </owl:Class>
  ```
Class axioms: *Equivalence*

- Between two call descriptions $C$ and $D$ can be defined the equality relation using operator `owl:equivalentClass`

- **DL syntax:**
  $$ C \equiv D $$

- **RDF syntax:**
  ```xml
  <owl:Class rdf:about="#C">
    <owl:equivalentClass rdf:resource="#D" />
  </owl:Class>
  ```

Property restrictions (1)

- A class can be described as a restriction on a property, the restriction can be the existential qualified role using operator `owl:someValuesFrom`

- **DL syntax:**
  $$ \exists R . C $$

- **RDF syntax:**
  ```xml
  <owl:Restriction>
    <owl:onProperty rdf:resource="#R" />
    <owl:someValuesFrom rdf:resource="#C" />
  </owl:Restriction>
  ```
Property restriction (2)

- A class can be described as a restriction on a property, the restriction can be the universal qualified role using operator `owl:allValuesFrom`
- **DL syntax:**
  \[ \forall R.C \]
- **RDF syntax:**

  ```xml
  <owl:Restriction>
    <owl:onProperty rdf:resource="#R"/>
    <owl:allValuesFrom rdf:resource="#C"/>
  </owl:Restriction>
  ```

Examples (W3C OWL Guide: [http://www.w3.org/TR/owl-ref/](http://www.w3.org/TR/owl-ref/))

```xml
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent"/>
  <owl:someValuesFrom rdf:resource="#Doctor"/>
</owl:Restriction>

<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent"/>
  <owl:allValuesFrom rdf:resource="#Human"/>
</owl:Restriction>
```
Property restrictions (4)

- A class can be described as a restriction of all individuals that associate through a role R to a single individual a using operator owl:hasValue

- **DL syntax**
  \( \forall R.\{a\} \)

- **RDF syntax**

  ```
  <owl:Restriction>
    <owl:onProperty rdf:resource="#R" />
    <owl:hasValue rdf:resource="#a" />
  </owl:Restriction>
  ```


```xml
<owl:Class rdf:ID="Burgundy">
  ...
  <rdfs:subClassOf rdf:resource="Wine" />  
  <owl:Restriction>
    <owl:onProperty rdf:resource="#hasTaste" />  
    <owl:hasValue rdf:resource="#Dry" />  
  </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```
Property restrictions (6)

- A class can be described as a restriction on a set of individuals with role R with a maximum number n of associated individuals using operator `owl:maxCardinality`
- DL syntax: \[ \leq n_R \]
- RDF syntax:

  ```xml
  <owl:Restriction>
    <owl:onProperty rdf:resource="#R"/>
    <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">n</owl:maxCardinality>
  </owl:Restriction>
  ```

Property restrictions (7)

- A class can be described as a restriction on a set of individuals with role R with at least n associated individuals, using operator `owl:minCardinality`
- DL syntax: \[ \geq n_R \]
- RDF syntax:

  ```xml
  <owl:Restriction>
    <owl:onProperty rdf:resource="#R"/>
    <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">n</owl:minCardinality>
  </owl:Restriction>
  ```
Property restriction (8)

- A class can be described as a restriction on a set of individuals with role R associated exactly to n individuals, using operator \texttt{owl:cardinality}

- \textit{DL syntax:}
  \[=nR\]

- \textit{RDF syntax:}

  \begin{verbatim}
  <owl:Restriction>
  <owl:onProperty rdf:resource="#R" />
  <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">n</owl:cardinality>
  </owl:Restriction>
  \end{verbatim}

Examples (W3C OWL Guide: http://www.w3.org/TR/owl-ref)

\begin{verbatim}
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">2</owl:maxCardinality>
</owl:Restriction>

<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">2</owl:minCardinality>
</owl:Restriction>

<owl:Restriction>
  <owl:onProperty rdf:resource="#hasIDFiscalCode" />
  <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
</owl:Restriction>
\end{verbatim}
Class axioms: *Disjunction*

- Between two classes $C$ and $D$ can be declared a disjoint relation using operator `owl:disjointWith`

- **DL syntax:**
  
  $$C \cap D \equiv \bot$$

- **RDF syntax:**

  ```
  <owl:Class rdf:about="#C">
    <owl:disjointWith rdf:resource="#D" />
  </owl:Class>
  ```

Properties

- Coherently with *RDFS*, in *OWL* also properties *(as roles in DL)* can be seen as *particular classes*

- Can have sub properties and can be combined using some constructors

- As every class is an `owl:Class` resource, all properties are resources of type `rdf:Property`

- In OWL properties can be resources of three types
  - `owl:ObjectProperty` (property between two objects)
  - `owl:DatatypeProperty` (property between an object and a datatype)
  - `owl:AnnotationProperty` (property used to associate metadata)
Subproperties

A property $R$ can be defined as a subproperty of property $S$ using operator `rdfs:subPropertyOf`:

- **DL syntax:**
  \[ R \sqsubseteq S \]
- **RDF syntax:**
  ```xml
  <owl:ObjectProperty rdf:ID="R">
    <rdfs:subPropertyOf rdf:resource="#S" />
  </owl:ObjectProperty>
  ```

Domain

- Of property $R$ can be specified the domain $D$ using operator `rdfs:domain`:
- **DL syntax:**
  \[ T \sqsubseteq \forall R . D \]
- **RDF syntax:**
  ```xml
  <owl:ObjectProperty rdf:ID="R">
    <rdfs:domain>
      <owl:Class rdf:about="#D" />
    </rdfs:domain>
  </owl:ObjectProperty>
  ```
Range

- Of property $R$ can be specified the range $C$ using operator `rdfs:range`
- **DL syntax:**
  $$\top \sqsubseteq \forall R.C$$
- **RDF syntax:**
  ```xml
  <owl:ObjectProperty rdf:ID="R">
    <rdfs:range>
      <owl:Class rdf:about="#C" />
    </rdfs:range>
  </owl:ObjectProperty>
  ```

Equivalent property

- A property $R$ can be defined as equivalent to another property $S$ using operator `owl:equivalentProperty`
- **DL syntax:**
  $$R \equiv S$$
- **RDF syntax:**
  ```xml
  <owl:ObjectProperty rdf:ID="R">
    <owl:equivalentProperty rdf:resource="#S" />
  </owl:ObjectProperty>
  ```
### Inverse property

- Given property $R$ the inverse property $S$ can be defined using operator `owl:inverseOf`
- **DL syntax:**
  $$S \equiv R^-$$
- **RDF syntax:**
  ```
  <owl:ObjectProperty rdf:ID="S">
  <owl:inverseOf rdf:resource="#R" />
  </owl:ObjectProperty>
  ```

### Functional property (1)

- A property $R$ is functional if it associate at max 1 individual for each element in the domain, expressed from operator `owl:FunctionalProperty`
- **DL syntax:**
  $$\top \sqsubseteq \leq_1 R$$
- **RDF syntax:**
  ```
  <owl:FunctionalProperty rdf:about="#R" />
  ...
  <owl:ObjectProperty rdf:ID="R">
  <rdfs:domain rdf:resource="#D" />
  <rdfs:range rdf:resource="#C" />
  </owl:ObjectProperty>
  ```
Functional property (2)


```
<owl:ObjectProperty rdf:ID="hasHusband">
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  <rdfs:domain rdf:resource="#Woman"/>
  <rdfs:range rdf:resource="#Man"/>
</owl:ObjectProperty>
```

• A property $R$ is inverse functional if the inverse property is functional, thus each element in the range is associated from max 1 individual in the domain, expressed from operator $\text{owl:InverseFunctionalProperty}$

• **DL Syntax:**
  $$\top \sqsubseteq 1R$$

• **RDF Syntax:**
  ```
  <owl:InverseFunctionalProperty rdf:ID="R">
    <rdfs:domain rdf:resource="#D"/>
    <rdfs:range rdf:resource="#C"/>
  </owl:InverseFunctionalProperty>
  ```
Transitive property (1)

- In OWL it is possible to declare that a property is transitive using operator `owl:TransitiveProperty`.

- **DL syntax:**
  \[ Tr(R) \]

- **RDF syntax:**
  ```xml
  <owl:TransitiveProperty rdf:ID="R">
    <rdfs:domain rdf:resource="#D" />
    <rdfs:range rdf:resource="#D" />
  </owl:TransitiveProperty>
  ```


```xml
<owl:ObjectProperty rdf:ID="subRegionOf">
  <rdf:type rdf:resource="&owl:TransitiveProperty"/>
  <rdfs:domain rdf:resource="#Region"/>
  <rdfs:range rdf:resource="#Region"/>
</owl:ObjectProperty>
```
Symmetric property (1)

- In OWL it is possible to declare a symmetric property
  `owl:SymmetricProperty`

- **DL syntax:**
  \[ R \subseteq R^- \]

- **RDF syntax:**
  ```
  <owl:SymmetricProperty rdf:ID="R">
    <rdfs:domain rdf:resource="#D" />
    <rdfs:range rdf:resource="#D" />
  </owl:SymmetricProperty>
  ```


```
<owl:SymmetricProperty rdf:ID="friendOf">
  <rdfs:domain rdf:resource="#Human"/>
  <rdfs:range rdf:resource="#Human"/>
</owl:SymmetricProperty>
```
Example

\[
<\text{owl:ObjectProperty rdf:about}="&UniFI\#isCoordinatorOf">
  <\text{rdfs:subPropertyOf} rdf:resource="&UniFI\#isWorkingFor"/>
  <\text{rdfs:domain} rdf:resource="&UniFI\#Coordinator"/>
  <\text{owl:inverseOf} rdf:resource="&UniFI\#hasCoordinator"/>
  <\text{rdfs:range}>
    <\text{owl:Class}>
      <\text{owl:unionOf} rdf:parseType="Collection">
        <\text{rdf:Description} rdf:about="&UniFI\#Center"/>
        <\text{rdf:Description} rdf:about="&UniFI\#ResearchProject"/>
      </\text{owl:unionOf}>
    </\text{owl:Class}>
  </\text{rdfs:range}>
</\text{owl:ObjectProperty}>
\]

Class member

- In OWL it is possible to declare that an individual a belong to class C

- **DL syntax:**
  \[
  C(a)
  \]

- **RDF syntax:**
  \[
  <C rdf:ID="a"> \ldots \ldots </C>
  \]
  \[
  <C rdf:about="a"> \ldots \ldots </C>
  \]
Property instances

• In OWL it is possible to specify that property $R$ of an individual $a$ has value $b$

• *DL syntax:* $R(a,b)$

• *RDF syntax:*
  
  ```
  <C rdf:ID="a">
    <R rdf:resource="#b" />
  ...
  </C>
  ```

Unique names?

- Two distinct names are necessary referring to two different objects?
- Depends... We have two possibilities:
  - We use **Unique Name Assumption** (UNA), every object has a unique name
  - or **Not Unique Name Assumption** (NUNA), the same object can be identified using two or more different names
- OWL and semantic web adopt NUNA, this implies that it should be needed a way to express that two objects are the same or that are different
Object identity (1)

- **OWL** allows to assert that two names refers to the same individual using operator `owl:sameAs`

- **DL syntax:**
  \[ a = b \]

- **RDF syntax:**
  ```xml
  <rdf:Description rdf:about="#a">
    <owl:sameAs rdf:resource="#b"/>
  </rdf:Description>
  ```

Object identity (2)

- It is also possible to assert that two names refer to different individuals using operators `owl:differentFrom`

- **RDF syntax:**
  ```xml
  <C rdf:ID="a">
    <owl:differentFrom rdf:resource="#b"/>
    ...
  </C>
  ```
Object identity (3)

- It is also possible to express that \( n \) names refer to all different individuals using operator `owl:AllDifferent`

- **RDF syntax:**

  ```xml
  <owl:AllDifferent>
    <owl:distinctMembers rdf:parseType="Collection">
      <C rdf:about="#a1"/>
      ...
      <C rdf:about="#an"/>
    </owl:distinctMembers>
  </owl:AllDifferent>
  ```

Open or closed world?

- A term that is not explicitly asserted and that cannot be derived as true, is it false?
  - **Closed World Assumption (CWA)** → yes, is false
  - **Open World Assumption (OWA)** → we don’t know it could be true or false

- **OWL and semantic web adopt the OWA** that make it difficult to check for consistency:
  - **TBox:** \( A \equiv A \lor R \)
  - **ABox:** \( A(a1), R(a1,x1) \)
OWL2 (1)

- OWL2 introduces some new features:
  - **DisjointUnion** of classes
  - **DisjointClasses** (all classes are pair-wise disjoint)
  - Negative object and data properties assertions
  - Self restriction on a property
    - $\exists R.\text{Self} = \{x : R(x, x)\}$
    - Narcissist $\equiv$ Person $\sqcap \exists \text{loves.Self}$
  - Reflexive and irreflexive properties
    - $T \sqsubseteq \exists R.\text{Self}$
    - $T \sqsubseteq \neg \exists R.\text{Self}$
  - Disjoint properties
    - Disjoint($R_1, R_2, \ldots$)
  - Asymmetric properties
    - Disjoint($R, R^\dashv$)

OWL2 (2)

- OWL2 introduces some new features:
  - Property chains inclusion
    - role composition can be used to define a new property
    - $R_1 \circ R_2 \circ \ldots \circ R_n \sqsubseteq \text{CR}$
    - Example:
      - $\text{parentOf} \circ \text{parentOf} \sqsubseteq \text{grandParentOf}$
  - Qualified cardinality restrictions
    - $\leq nR.C$
    - $= nR.C$
    - $\geq nR.C$
  - Keys of a class
    - A set of properties uniquely identifies an individual of a class
Protégé

- Protégé ([http://protege.stanford.edu/](http://protege.stanford.edu/)), is an open-source editor for KBs and Ontologies (RDF, OWL, NT...) developed by Stanford University
- Allows the visualization, creation, editing of:
  - Entities
  - Classes
  - Properties (Object Property & Data Property)
  - Instances (Individuals).
- SPARQL Query Interface to query the KB.
- Graphic visualizations of classes, properties and instances.

Protégé – Classes
Protégé – **Object Properties**

Protégé – **Instances**
Protégé – OntoGraph

Manchester syntax

- Used in Protégé to express restrictions on a class
  - C and D \( C \sqcap D \)
  - C or D \( C \sqcup D \)
  - not C \( \neg C \)
  - p some C \( \exists p . C \)
  - p only C \( \forall p . C \)
  - p exactly n C \( = n . C \)
  - p min n C \( \geq n p . C \)
  - p max n C \( \leq n p . C \)
  - p value v \( \forall p . \{ v \} \)
  - { v1, v2, ... vn } 
  - inverse p \( p^- \)
Examples

- \(\text{PERSON and (parentOf some PERSON)}\)
- Human and not Female and (married some Doctor) and (hasChild only (Doctor or Professor))
- Person and (sonOf exactly 2 Person)

LOV

- Linked Open Vocabulary
- Vocabulary descriptions should be available via Linked Data
- If you open the URL of a property or class with a browser it forwards to the HTML documentation of the vocabulary
- If you request (Accept) rdfxml or turtle formats it provides a machine readable description
lov.linkeddata.es

- a repository where are available links to vocabularies (ontologies) available as linked open data.
- contains a RDF description of vocabularies

SKOS

- Used for Taxonomy representation
- Taxonomies are used for classification, are hierarchies of concepts
- SKOS = Simple Knowledge Organization System
- Describes a set of skos:Concepts
- Related via properties:
  - broader (e.g. <Ontology> skos:broader <KnowledgeRepresentation>),
    - read "has broader"
  - narrower (e.g. <Animal> skos:narrower <Mammifer>),
    - read "has narrower"
  - related (e.g. <Human> skos:related <Philosophy>),
  - ...

**SKOS**

- broader & narrower build hierarchies, related used to associate concepts from different hierarchies
- broader & narrower are not transitive
  - broader $\subseteq$ broaderTransitive $\subseteq$ semanticRelation
  - narrower $\subseteq$ narrowerTransitive $\subseteq$ semanticRelation
  - related $\subseteq$ semanticRelation
- broaderTransitive disjoint with related
- related is symmetric
- semanticRelation: Concept $\rightarrow$ Concept
- narrower inverse of broader
- narrowerTransitive inverse of broaderTransitive

**SKOS**

- Can represent different ConceptSchemes
  - cs rdf:type ConceptScheme
  - c1 skos:inScheme cs
  - cs skos:hasTopConcept cx
  - cx skos:topConceptOf cs
  - skos:topConceptOf $\subseteq$ skos:inScheme
  - skos:topConceptOf inverse of skos:hasTopConcept
SKOS

- concepts from different concept schemes can be associated using specific properties:
  - `exactMatch`
  - `closeMatch`
  - `broadMatch`
  - `narrowMatch`
  - `relatedMatch`

SKOS

- properties:
  - `exactMatch` $\sqsubseteq$ `closeMatch` $\sqsubseteq$ `mappingRelation`
  - `mappingRelation` $\sqsubseteq$ `semanticRelation`
  - `broadMatch` $\sqsubseteq$ `mappingRelation`
  - `narrowMatch` $\sqsubseteq$ `mappingRelation`
  - `relatedMatch` $\sqsubseteq$ `mappingRelation`
  - `broadMatch` $\sqsubseteq$ `broader`
  - `narrowMatch` $\sqsubseteq$ `narrower`
  - `relatedMatch` $\sqsubseteq$ `related`
  - `exactMatch`, `closeMatch`, `relatedMatch` are symmetric
  - `exactMatch` is transitive
  - `narrowMatch` inverse of `broadMatch`
SKOS labels

- SKOS introduces some annotationProperties for labelling resources
  - skos:prefLabel – one for each language
  - skos:altLabel – other alternative labels
  - skos:hiddenLabel – labels used for search (e.g. mispelled names)

SKOS example

- US library of congress publish as linked data the Library of Congress Subject Headings
  - [http://id.loc.gov/authorities/subjects.html](http://id.loc.gov/authorities/subjects.html)
  - Example Concept "Security measures"
    - [http://id.loc.gov/authorities/subjects/sh99005297](http://id.loc.gov/authorities/subjects/sh99005297)
Dublin Core

- 15 properties used to associate content with the usual bibliographic descriptions
  - title,
  - creator,
  - contributor,
  - subject,
  - language,
  - publisher,
  - identifier,
  - date,
  - description,
  - rights,
  - coverage,
  - format,
  - relation,
  - source,
  - type

Two prefixes
- dc: <http://purl.org/dc/elements/1.1/>
- dcterms: <http://purl.org/dc/terms/>

The first is the legacy one
- "dc" properties can be used as both data properties and object properties

Example:
- <divinacommedia> dc:creator "Dante Alighieri"
- <divinacommedia> dc:creator
  <http://dbpedia.org/resource/DanteAlighieri>
"dcterms" properties introduce many sub-properties:

- abstract ⊑ description
- license ⊑ rights
- spatial ⊑ coverage
- available ⊑ date
- bibliographicCitation ⊑ identifier
- conformsTo ⊑ relation
- created ⊑ date
- dateAccepted ⊑ date
- dateCopyrighted ⊑ date
- dateSubmitted ⊑ date
- extent ⊑ format
- hasPart ⊑ relation
- isPartOf ⊑ relation
- ...

**FOAF**

- FOAF = Friend Of a Friend
- allows to describe people and their relations
Supported by search engines (google, yahoo)
Used to describe content to be indexed
Many classes and properties

Given two vocabularies, we can build a mapping between:
- **classes** using:
  - subClassOf
  - equivalentClass
- **properties** using:
  - subPropertyOf
  - equivalentProperty

Aka Ontology Alignment
Objects can be linked using:

- **owl:sameAs**
  - to state that two URL refers exactly to the same entity
- **rdfs:seeAlso**
  - to state that two URL refers to related entities

Use **sameAs** relations with care, it is a transitive, symmetric, reflexive property.