Ontologies for Security and Privacy

Lecture 3 – Computational Ontologies

Alessandro Oltramari
CyLab, Carnegie Mellon University
Which semantics?

“A little semantics, goes a long way”  
James Hendler, 2001

“A Little Semantic Web Goes a Long Way in Biology”  
Wolstencroft et al., 2005

Source: Dr. Leo Obrst, Mitre; Mills Davis, Project10X

Carnegie Mellon
Ontology Life-Cycle

• DEVELOPMENT
  o Ontology design ➔ manual
  o Ontology implementation ➔ manual
  o Ontology learning ➔ (semi)automatic

• VALIDATION
  o Soundness/Consistency check ➔ (semi)automatic
  o Competency Questions (CQ)
    o Formalization of CQ ➔ manual/domain experts
    o Queries
    o Validation of answers
  o Performance in a variety of application-driven tasks
  o …

• MAINTANANCE
  o Ontology revision
  o Ontology evolution
Today’s focus

• DEVELOPMENT
  o Ontology design
  o **Ontology implementation**
  o Ontology learning
  ➔ manual
  ➔ manual
  ➔ (semi)automatic

• VALIDATION
  o **Soundness/Consistency check** ➔ (semi)automatic
  o Competency Questions (CQ)
    o Formalization of CQ
    o **Queries**
    o Validation of answers
  o Performance in a variety of application-driven tasks
  o …

• MAINTANANCE
  o Ontology revision
  o Ontology evolution
The realization of the Semantic Web strongly relies on the use of shared vocabularies for describing (Big) Data contents, which will be then used by human and artificial agents in interactive environments (interoperability).

In order to achieve this goal, it is necessary to develop standard languages (RDF, OWL,...) and technologies (reasoners, ontology development platforms,..) in support of data modeling.
Ontology Languages

- Ontology languages provide formal specifications for representing the elements of a given domain.
  - RDF, RDF(S), DAML-ONT, OIL, DAML+OIL
  - OWL (W3C’s standard)

- Core requirements of Ontology Languages:
  - Well-defined syntax
  - Well-defined semantics
  - Reasoning support
  - Expressive power
Syntax

• Well-defined syntax enables **machine-processing** of information
  – RDF syntax (XML-based) is hard to be considered as user-friendly!

  – Instead of directly writing the code, ontology engineers can make use of development tools, such as **Protégé**, Swoop, Top Braid Composer, ...
Formal Semantics

- Well-defined semantics allows for non-ambiguous reasoning:
  - **Class membership**: if $x$ is an instance of class $C$, and $C$ is a subclass of $D$, then we can infer that $x$ is an instance of $D$.
  - **Equivalence of classes**: If class $A$ is equivalent to class $B$, and class $B$ equivalent to class $C$, then $A$ is equivalent to $C$, too.
  - **Consistency**: given $x$ instance of class $A$ and suppose that:
    - $A$ is a subclass of $B \cap C$
    - $B$ and $D$ are disjoint
    - $A$ is subclass of $D$
    - **Inconsistency** [the ontology must be fixed]
  - **Classification**: if we have declared that certain property/value pairs are sufficient condition for membership of a class $A$, and if $x$ satisfies these conditions, we can conclude that $x$ must be instance of $A$. 

Carnegie Mellon
Automatic Reasoning

- Inferences can be made mechanically, instead of manually.

- Automatic Reasoning is a key-feature for:
  - Maintaining the consistency of an ontology
  - Checking for unintended relationships between classes
  - Automatically classify instances of classes
  - Designing large ontologies (eventually involving multiple authors)
  - Integrating different ontologies (“Ontology Matching”)
Ontology Web Language (OWL)

- OWL is partially mapped on Description Logics, and makes use of existing reasoners such as Pellet, HermiT, FaCT, RACER, etc.

- Description logics are a decidable fragment of FOL.

**REMINDER:** Logics are *decidable* if computations/algorithms based on the logic will terminate in a finite time.
OWL sub-languages

OWL-LITE
Syntactically simplest → class hierarchy and simple constraints

OWL-DL
Based on DL → supports automated reasoning

OWL-Full
Most expressive → when expressivity is more important than reasoning
In the following we introduce some examples of OWL core components; most are taken from the ontology pizza.owl (we’ll see how this ontology looks like in the Protégé-OWL platform).

```xml
<owl:Class rdf:resource="AnchoviesTopping">
  <rdfs:subClassOf rdf:resource="#FishTopping"/>
  <owl:disjointWith rdf:resource="#PrawnsTopping"/>
  <owl:disjointWith rdf:resource="#MixedSeafoodTopping"/>
</owl:Class>
```

OWL extends RDF: it adopts RDF meaning of classes and properties, extending the expressivity with its own primitives

- `owl:Thing` → the most general class (it subsumes every class);
- `owl:Nothing` → the empty class (every class subsumes it)

Reminder – layers of languages

**XML → XML(S) → RDF → RDF(S) → OWL**
Object Property

- Object property relates classes to other classes:

```xml
<owl:ObjectProperty rdf:about="#hasTopping">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#hasIngredient"/>
  </rdfs:subPropertyOf>
  <rdfs:domain rdf:resource="#Pizza"/>
  <rdfs:range rdf:resource="#PizzaTopping"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="isToppingOf"/>
  </owl:inverseOf>
</owl:ObjectProperty>
```

**InverseOf** interchange domain with range
Datatype Property

- Datatype property relates classes (objects) to datatype values:

```xml
<owl:DataProperty rdf:about="#Table_number">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#nonNegativeInteger"/>
</owl:DataProperty>
```

OWL does not provide predefined datatypes: it uses XML Schema datatypes
Property restrictions (1/2)

- `<rdfs:subClassOf>` relates a class C with an higher class D.
- OWL allows to declare that if C satisfies some conditions, which namely represent all the conditions a sub-class of D must have, then C is subclass of D.

```
<owl:Class rdf:resource="#AmericanHotPizza">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasTopping"/>
      <owl:someValuesFrom rdf:resource="#JalapenoPepperTopping"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

- existential quantification (∃) = `owl:someValuesFrom`
- universal quantification (∀) = `owl:AllValuesFrom`
Property restrictions (2/2)

- Other types of restrictions are
  - `owl:hasValue` (states the specific value that a property must have)
  - `owl:minCardinality` (states the minimum value that a property must have)
  - `owl:maxCardinality` (states the maximum value that a property must have)

- `owl:Restriction` defines an anonymous class, which has no id, no `owl:Class` definitions, and local scope; it can only be used in the one place where the restriction appears.

```
<owl:Class rdf:ID="PizzaTopping">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasTopping"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```
Special properties

- **owl:TransitiveProperty**
  - e.g. “is taller than”

- **owl:SymmetricProperty**
  - e.g., “is sibling of”

- **owl:FunctionalProperty**: defines a property that has at most one unique value for each object
  - e.g. “age”; “height”, ...

- **owl:InverseFunctionalProperty**: defines a property for which two different objects cannot have the same value
  - e.g. “is_SSN_of”
Boolean combinations

- Set theory $\text{owl:complement \ owl:unionOf \ owl:intersectionOf}$
- CheesyPizza is equivalent to the intersection between Pizza class and the class of all the pizzas that have some cheese topping

```xml
<owl:Class rdf:ID="CheeseyPizza">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:someValuesFrom rdf:resource="#CheeseTopping"/>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#hasTopping"/>
          </owl:onProperty>
        </owl:Restriction>
        <owl:Class rdf:about="#Pizza"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```
Enumerations

<owl:oneOf rdf:parseType="Collection">
  <Country rdf:ID="America"/>
  <Country rdf:ID="England"/>
  <Country rdf:ID="France"/>
  <Country rdf:ID="Germany"/>
  <Country rdf:ID="Italy"/>
</owl:oneOf>
Disjoint Union and Disjoint Classes

- **DisjointUnion** defines a class as the union of other classes, all of which are pairwise disjoint.

  \[
  \text{DisjointUnion}(:\text{BrainHemisphere} :\text{LeftHemisphere} :\text{RightHemisphere})
  \]

- **DisjointClasses** states that all classes from the set are pairwise disjoint.

  \[
  \text{DisjointClasses}(:\text{LeftLung} :\text{RightLung})
  \]