KDI
OWL

Fausto Giunchiglia and Mattia Fumagallli

University of Trento

Roadmap

- Introduction
- The OWL Full Language
- OWL DL and OWL lite
- Exercises

Introduction

Requirements for Ontology Languages

Ontology languages allow users to write explicit, formal conceptualizations of domain models (i.e. formal ontologies)

The main requirements are:

- A well-defined formal syntax
- Sufficient expressive power
- Convenience of expression
- Formal semantics
- Support for efficient reasoning
- A good tread-off between expressivity and efficiency

OWL (Web Ontology Language) has been designed to meet these requirements for the specification of ontologies and to reason about them and their instances

Reasoning capabilities required

Class membership

If x is an instance of a 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 is equivalent to class C, then A is equivalent to C

Disjointness and Consistency

Determine that if the classes A and B are disjoint there cannot be individuals x which are instances of both A and B. This is an indication of an error in the ontology.

Classification

Certain property-value pairs are a sufficient conditions for membership in a class A; if an individual x satisfies such conditions, we can conclude that x must be an instance of A.

Limitations in the expressive power of RDF schema

Range restrictions

We cannot declare range restrictions that apply to some classes only (e.g. cows eat only plants, while other animals may eat meat too).

Disjointness of classes

We cannot declare that two classes are disjoint (e.g. male and female).

Combinations of classes

We cannot define new classes as union, intersection, and complement of other classes (e.g. person is the disjoint union of the classes male and female).

Cardinality restrictions

We cannot express restrictions in the number of relations (e.g. a person has exactly two parents, a course is taught by at least one lecturer)

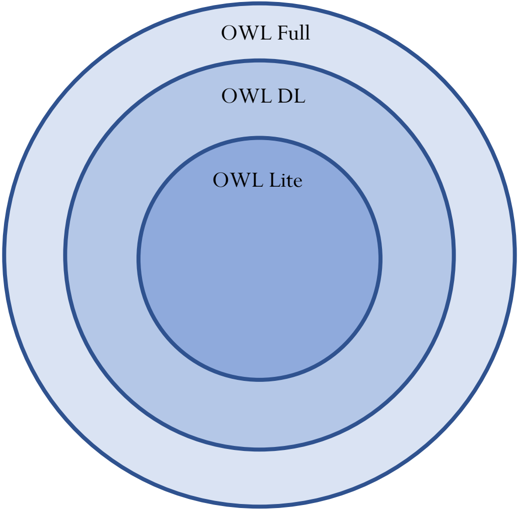
Meta-properties

Transitive property (e.g. “greater than”)

Unique property (e.g. “is mother of”)

Inverse property (e.g. “eats” and “is eaten by”)

OWL sub-languages



- Each OWL Lite representation belongs to OWL DL
- Each OWL DL representation belongs to OWL Full
- Each valid OWL Lite conclusion is also valid in OWL DL
- Each valid OWL DL conclusion is also valid in OWL Full

OWL sub-languages

OWL Lite trades expressivity for efficiency

- The less expressive of all languages (it cannot be used to express enumerated classes, disjointness, and arbitrary cardinality restrictions)
- It allows assigning simple cardinality constraints of kind 0 or 1
- It allows encoding simple classification hierarchies (e.g., taxonomies and thesauri)
- Partially compatible with RDF

OWL DL is a balance between expressivity and computational completeness

- More expressive than OWL Lite while guarantees decidability
- It allows expressing all DL constructs, some of them with certain restrictions (e.g. the restriction of not making a class an instance of another class)
- Partially compatible with RDF

OWL Full trades computational completeness for expressivity

- More expressive than OWL DL, maximum expressiveness (e.g., a class can be represented also as an individual)
- It is computationally very expensive and does not guarantee decidability
- Fully upward-compatible with RDF, both syntactically and semantically

The OWL Full language

OWL XML/RDF syntax

```
<rdf:RDF
```

HEADER

```
        xmlns:owl ="http://www.w3.org/2002/07/owl#"
        xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:xsd ="http://www.w3.org/2001/XLMSchema#">
```

ONTOLOGY

```
<owl:Ontology rdf:about="">
        <rdfs:comment>An example OWL ontology </rdfs:comment>
        <owl:priorVersion
                rdf:resource="http://www.mydomain.org/uni-ns-old"/>
        <owl:imports
                rdf:resource="http://www.mydomain.org/persons"/>
        <rdfs:label>University Ontology</rdfs:label>
</owl:Ontology>
```

```
</rdf:RDF>
```

Classes

Defined using owl:Class that is a subclass of rdfs:Class

owl:Thing is the most general class, which contains everything

owl:Nothing is the empty class

DISJOINT CLASSES *owl:disjointWith*

```
<owl:Class rdf:about="#associateProfessor">
    <owl:disjointWith rdf:resource="#professor"/>
    <owl:disjointWith rdf:resource="#assistantProfessor"/>
</owl:Class>
```

EQUIVALENT CLASSES *equivalentClass*

```
<owl:Class rdf:ID="faculty">
    <owl:equivalentClass rdf:resource= "#academicStaffMember"/>
</owl:Class>
```

Properties

Data type properties relate objects to datatype values (ATTRIBUTES)

```
<owl:DatatypeProperty rdf:ID="age">
    <rdfs:range rdf:resource= "http://www.w3.org/2001/XLMSchema
                #nonNegativeInteger"/>
</owl:DatatypeProperty>
```

Object properties relate objects to other objects (RELATIONS)

```
<owl:ObjectProperty rdf:ID="isTaughtBy">
    <owl:domain rdf:resource="#course"/>
    <owl:range rdf:resource= "#academicStaffMember"/>
    <rdfs:subPropertyOf rdf:resource="#involves"/>
</owl:ObjectProperty>
```

Property restrictions: a kind of class description (I)

VALUE CONSTRAINT *owl:allValuesFrom*

A value constraint puts constraints on the range of the property. It corresponds to universal quantification.

```
<owl:Class rdf:about="#firstYearCourse">
  <rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#isTaughtBy"/>
        <owl:allValuesFrom rdf:resource="#Professor"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Property restrictions: a kind of class description (I)

CARDINALITY CONSTRAINT *someValuesFrom / owl:hasValue*

A cardinality constraint puts constraints on the number of values. It corresponds to the existential quantification or can indicate a specific value.

```
<owl:Class rdf:about="#firstYearCourse">
  <rdfs:subClassOf>
    <owl:Restriction>
          <owl:onProperty rdf:resource="#teaches"/>
          <owl:someValuesFrom rdf:resource="#undergraduateCourse"/>
          (or)
          <owl:onProperty rdf:resource= "#isTaughtBy"/>
          <owl:hasValue rdf:resource= "#949352"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Cardinality restrictions (I)

owl:maxCardinality

It describes a class of all individuals that have at most N semantically distinct values (individuals or data values) for the property.

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

owl:minCardinality

It describes a class of all individuals that have at least N semantically distinct values (individuals or data values) for the property.

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

Cardinality restrictions (I)

owl:cardinality

It describes a class of all individuals that have exactly N semantically distinct values (individuals or data values) for the property concerned, where N is the value of the cardinality constraint.

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

This construct is redundant in that it can be replaced by a pair of matching owl:minCardinality and owl:maxCardinality constraints with the same value.

Meta-properties (I)

EQUIVALENCE ***owl:equivalentProperty***

x P y implies x Q y

```
<owl:equivalentProperty
    <owl:ObjectProperty rdf:ID="lecturesIn">
    <owl:equivalentProperty rdf:resource="#teaches"/>
</owl:ObjectProperty>
```

NOTE: in RDF we need P rdfs:subPropertyOf Q and Q rdfs:subPropertyOf P

INVERSE ***owl:inverseOf***

x P y implies y Q x

```
<owl:ObjectProperty rdf:ID="teaches">
    <rdfs:range rdf:resource="#course"/>
    <rdfs:domain rdf:resource= "#academicStaffMember"/>
    <owl:inverseOf rdf:resource="#isTaughtBy"/>
</owl:ObjectProperty>
```

Meta-properties (II)

SYMMETRIC ***owl:SymmetricProperty***

x P y implies y P x

```
<owl:ObjectProperty rdf:ID="married">
    <rdf:type rdf:resource="&owl;SymmetricProperty"/>
    <rdfs:range rdf:resource="#person"/>
    <rdfs:domain rdf:resource= "#person"/>
</owl:ObjectProperty>
```

TRANSITIVE ***owl:TransitiveProperty***

x P y and y P z implies x P z

```
<owl:ObjectProperty rdf:ID="ancestor">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#person"/>
    <rdfs:domain rdf:resource= "#person"/>
</owl:ObjectProperty>
```

Functional and inverse functional properties

FUNCTIONAL PROPERTY ***owl:FunctionalProperty***

A functional property is a property that can have only one value as range for any given individual (e.g., hasMother , hasPresident).

INVERSE FUNCTIONAL PROPERTY ***owl:InverseFunctionalProperty***

It defines a property that cannot have the same value as range for any given individual (e.g., MotherOf , PresidentOf).

Enumerations

It allows a class to be defined by exhaustively enumerating its instances. The class extension of a class described with *owl:oneOf* contains exactly the enumerated individuals, no more, no less.

```
<owl:Class rdf:ID="weekdays">
  <owl:oneOf rdf:parseType="Collection">
      <owl:Thing rdf:about="#Monday"/>
      <owl:Thing rdf:about="#Tuesday"/>
      <owl:Thing rdf:about="#Wednesday"/>
      <owl:Thing rdf:about="#Thursday"/>
      <owl:Thing rdf:about="#Friday"/>
      <owl:Thing rdf:about="#Saturday"/>
      <owl:Thing rdf:about="#Sunday"/>
  </owl:oneOf>
</owl:Class>
```

Intersection

```
<owl:Class>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Tosca" />
        <owl:Thing rdf:about="#Salome" />
      </owl:oneOf>
    </owl:Class>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Turandot" />
        <owl:Thing rdf:about="#Tosca" />
      </owl:oneOf>
    </owl:Class>
  </owl:intersectionOf>
</owl:Class>
```

Union

```
<owl:Class>
 <owl:unionOf rdf:parseType="Collection">
  <owl:Class>
   <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Tosca" />
    <owl:Thing rdf:about="#Salome" />
   </owl:oneOf>
  </owl:Class>
  <owl:Class>
   <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Turandot" />
    <owl:Thing rdf:about="#Tosca" />
   </owl:oneOf>
  </owl:Class>
 </owl:unionOf>
</owl:Class>
```

Complement

```
<owl:Class>
 <owl:complementOf>
  <owl:Class rdf:about="#Meat"/>
 </owl:complementOf>
</owl:Class>
```

Instances

Instances of classes are declared as in RDF:

```
<rdf:Description rdf:ID="949352">
  <rdf:type rdf:resource= "#academicStaffMember"/>
</rdf:Description>
<academicStaffMember rdf:ID="949352">
  <uni:age rdf:datatype="&xsd;integer">39<uni:age>
</academicStaffMember>
```

Same instances:

```
<rdf:Description rdf:about="#William_Jefferson_Clinton">
 <owl:sameAs rdf:resource="#BillClinton"/>
</rdf:Description>
```

Different instances:

```
<Opera rdf:ID="Nozze_di_Figaro">
 <owl:differentFrom rdf:resource="#Don_Giovanni"/>
</Opera>
```

OWL DL and OWL lite

OWL DL

OWL DL is a sublanguage of OWL which places a number of constraints on the use of the OWL language constructs which ensure that computational complexity is the same as corresponding Description Logic.

- Each individual must be an instance of a class, and in particular of owl:Thing
- Pairwise separation between classes, datatypes, datatype properties, object properties, annotation properties, ontology properties (i.e., the import and versioning stuff), individuals, data values and the built-in vocabulary. This means that, for example, a class cannot be at the same time an individual.
- No cardinality constraints can be placed on transitive properties or their inverses or any of their super-properties.
- It is allowed to use the intersectionOf construct with any number of classes and of any non negative integer in the cardinality restrictions value fields

OWL lite

OWL lite is a sublanguage of OWL DL which places further constraints on the use of the OWL language constructs which ensure a lower computational complexity

- Users are allowed to use a subset of the OWL, RDF and RDFS vocabulary
- To define a class, one must use the OWL construct owl:Class
- OWL constructs complementOf, disjointWith, hasValue, oneOf and unionOf are not allowed
- All three cardinality constructs – cardinality, maxCardinality and minCardinality, can only have 0 or 1 in their value fields
- equivalentClass and intersectionOf cannot be used in a triple if the subject or object represents an anonymous class

Exercises

Exercise 1

Suppose that a family consists of a father (John), a mother (Maria), two sisters (Sara and Jenifer) and two brothers (David and Robert). In an OWL representation the two brothers and the two sisters are codified as follows:

:David	:hasFather	:John
:Sara	:hasFather	:John
:John	:spouseOf	:Maria

Later on another property :hasChild is codified.

(i) What will be the output of the following SPARQL Query when a reasoner is activated?

:John	:hasChild	?y

Exercise 1 (cont)

(ii) Expand the OWL representation in a way that supports returning non-empty result of the following query and this expansion is independent of the entity-entity triples.

:John	:hasChild	?y

(iii) Add also the following axioms to the dataset.

:Jenifer	:hasFather	:John
:Robert	:hasFather	:John

What results the following query will return?

:John	:hasChild	?y

(iv) How can we infer the spouse relation in the reverse direction?

Solution 1

(i)	The result of the query is empty.
(ii)	We can make the property :hasFather as an inverse property of :hasChild :hasFather owl:inverseOf :hasChild Query Result: :David :Sara
(iii)	:David :Sara :Jenifer :Robert
(iv)	We can make the relation :spouseOf its own inverse as follows: :spouseOf owl:inverseOf :spouseOf

Exercise 2

Within a family, the following relations are applicable in both directions (from subject to object, and vice versa):

:spouseOf

:marriedTo

:siblingOf

whereas the same those not always apply to the following:

:brotherOf

:sisterOf

(i) Which property holds in the relations that are applicable in both directions?

(ii) How can we represent these relations in OWL?

(iii) In which basic category this property belongs?

Solution 2

(i)	Symmetric property
(ii)	:spouseOf rdf:type owl:SymmetricProperty :marriedTo rdf:type owl:SymmetricProperty :siblingOf rdf:type owl:SymmetricProperty
(iii)	The symmetric property is an object property. Moreover, the domain and range of the symmetric property are the same (owl:Class)

Exercise 3

Consider that in the family of John and Maria, also John's father (James) and mother (Jerry) live. Relations such as :hasAncestor and :hasDescendent can be applied between different levels. For example:

:John	:hasAncestor	:James
:Sara	:hasAncestor	:John
:James	:hasDescendent	:John
:John	:hasDescendent	:Sara

(i) Which property holds in the relations that are applicable in different levels of the hierarchy?

(ii) How can we represent these relations in OWL?

(iii) In which basic category this property belongs?

(iv) Show the results of the following queries:

a) :James :hasDescendent ?y

b) :John :hasAncestor ?y

Solution 3

(i)	Transitive property
(ii)	:hasAncestor rdf:type owl:TransitiveProperty :hasDescendent rdf:type owl:TransitiveProperty
(iii)	The transitive property is an object property.
(iv)	a) :John :Sara b) :James

Exercise 4

(i) In RDFS we can represent that two classes :Test and :Experiment are equivalent.

:Test	rdfs:subClassOf	:Experiment
:Experiment	rdfs:subClassOf	:Test

Convert this representation in OWL.

(ii) In RDFS we can represent that two properties :hasChild and :hasKid are equivalent.

:hasChild	rdfs:subPropertyOf	:hasKid
:hasKid	rdfs:subPropertyOf	:hasChild

Convert this representation in OWL.

(iii) Is there any way to represent the fact that two entities (or individuals) :Italia and :Il_Bel_Paese are the same?

Solution 4

(i)	:Test	owl:equivalentClass	:Experiment
(ii)	:hasChild	owl:equivalentProperty	:hasKid
(iii)	:Italia	owl:sameAs	:Il_Bel_Paese

Exercise 5

(i) Which OWL property allows to have exactly one value for a particular individual?

(ii) The following relations can be defined using the OWL property above.

:hasFather

:hasMother

Represent them in OWL and demonstrate their use with necessary entity-entity axioms.

Solution 5

(i)	OWL Functional property

(ii)

:hasFather	rdf:type	owl:FunctionalProperty
:hasMother	rdf:type	owl:FunctionalProperty

Two entity-entity axioms are provided below:

:John	:hasFather	:James
:John	:hasFather	:Handler

The objects :James and :Handler are the values of the same subject and property. We already have defined that :hasFather property is functional. Therefore, it can be concluded that :James and :Handler refer to the same person.

Exercise 6

(i) Which OWL property allows to have exactly one value for a particular object?

(ii) Demonstrate the use of such a property in developing applications such as the detection of possible duplicates.

Solution 6

(i)	OWL Inverse Functional property

(ii)	We can encode the property :SSN (social security number) as follows: :SSN rdf:type owl:InverseFunctionalProperty Two entity-entity axioms are provided below: mo:James :SSN N123812834 ps:Handler :SSN N123812834 The subjects :James and :Handler are attached to the same social security number, which cannot be shared by two different persons. Therefore, we can conclude that mo:James and ps:Handler are the same entity.

References

- OWL Web Ontology Language(W3C): http://www.w3.org/TR/2004/REC-owl-ref-20040210/
- G. Antoniou & F. van Harmelen (2004). A Semantic Web Primer (Cooperative Information Systems). MIT Press, Cambridge MA, USA.
- D. Allemang and J. Hendler. Semantic web for the working ontologist: modeling in RDF, RDFS and OWL. Morgan Kaufmann Elsevier, Amsterdam, NL, 2008.