Web Ontology Language: OWL

Bojan Furlan

A Semantic Web Primer,
G. Antoniou, F. van Harmelen

Requirements for Ontology Languages

- Ontology languages allow users to write explicit, formal conceptualizations of domain models
- The main requirements are:
  - a well-defined syntax
  - efficient reasoning support
  - a formal semantics
  - sufficient expressive power
  - convenience of expression

Tradeoff between Expressive Power and Efficient Reasoning Support

- The richer the language is, the more inefficient the reasoning support becomes
- Sometimes it crosses the border of *noncomputability*
- We need a compromise:
  - A language supported by reasonably efficient reasoners
  - A language that can express large classes of ontologies and knowledge.

Reasoning About Knowledge in Ontology Languages

- 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, too

Reasoning About Knowledge in Ontology Languages (2)

- Consistency
  - X instance of classes A and B, but A and B are disjoint
  - This is an indication of an error in the ontology
- Classification
  - Certain property-value pairs are a sufficient condition for membership in a class A; if an individual x satisfies such conditions, we can conclude that x must be an instance of A
    - (X teaches Course => X is Lecturer)

Uses for Reasoning

- Reasoning support is important for
  - checking the consistency of the ontology and the knowledge
  - checking for unintended relationships between classes
  - automatically classifying instances in classes
- Checks like the preceding ones are valuable for
  - designing large ontologies, where multiple authors are involved
  - integrating and sharing ontologies from various sources

Reasoning Support for OWL

- Semantics is a prerequisite for reasoning support
- Formal semantics and reasoning support are usually provided by
  - mapping an ontology language to a known logical formalism
  - using automated reasoners that already exist for those formalisms
- OWL is (partially) mapped on a description logic, and makes use of reasoners such as FaCT and RACER
- Description logics are a subset of predicate logic for which efficient reasoning support is possible

Limitations of the Expressive Power of RDF Schema

- Local scope of properties
  - **rdfs:range** defines the range of a property (e.g. eats) for all classes
  - In RDF Schema we cannot declare range restrictions that apply to some classes only
  - E.g. we cannot say that cows eat only plants, while other animals may eat meat, too

Limitations of the Expressive Power of RDF Schema (2)

- Disjointness of classes
  - Sometimes we wish to say that classes are disjoint (e.g. **male** and **female**)
- Boolean combinations of classes
  - Sometimes we wish to build new classes by combining other classes using union, intersection, and complement
  - E.g. **person** is the disjoint union of the classes **male** and **female**

Limitations of the Expressive Power of RDF Schema (3)

- Cardinality restrictions
  - E.g. a person has exactly two parents, a course is taught by at least one lecturer
- Special characteristics of properties
  - Transitive property (like "greater than")
  - Unique property (like "is mother of")
  - A property is the inverse of another property (like "eats" and "is eaten by")

Combining OWL with RDF Schema

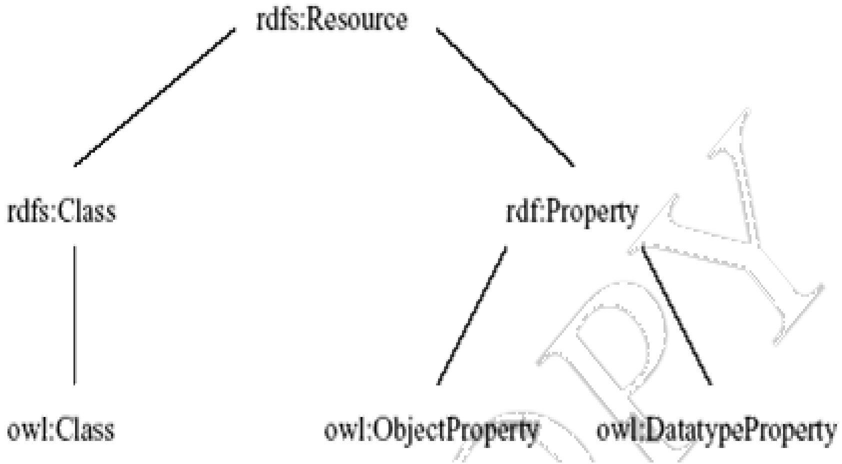
- Ideally, OWL would extend RDF Schema
  - Consistent with the layered architecture of the Semantic Web
- **But** simply extending RDF Schema would work against obtaining expressive power and efficient reasoning
  - Combining RDF Schema with logic leads to uncontrollable computational properties

Three Species of OWL

- W3C’sWeb Ontology Working Group defined OWL as three different sublanguages:
  - OWL Full
  - OWL DL
  - OWL Lite
- Each sublanguage geared toward fulfilling different aspects of requirements

OWL Compatibility with RDF Schema

- All varieties of OWL use RDF for their syntax
- Instances are declared as in RDF, using RDF descriptions
- and typing information OWL constructors are specialisations of their RDF counterparts



OWL Compatibility with RDF Schema (2)

- Semantic Web design aims at downward compatibility with corresponding reuse of software across the various layers
- The advantage of full downward compatibility for OWL is only achieved for OWL Full, at the cost of computational intractability

OWL Syntactic Varieties

- OWL builds on RDF and uses RDF's XML-based syntax
- Other syntactic forms for OWL have also been defined:
  - An alternative, more readable XML-based syntax
  - An abstract syntax, that is much more compact and readable than the XML languages
  - A graphic syntax based on the conventions of UML

OWL XML/RDF Syntax: Header

```
<rdf:RDF
    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#">
```

- An OWL ontology may start with a collection of assertions for housekeeping purposes using **owl:Ontology** element

owl: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>
```

- **owl:imports** is a transitive property

Classes

- Classes are defined using **owl:Class**
  - **owl:Class** is a subclass of **rdfs:Class**
- Disjointness is defined using **owl:disjointWith**

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

Classes (2)

- **owl:equivalentClass** defines equivalence of classes

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

- **owl:Thing** is the most general class, which contains everything
- **owl:Nothing** is the empty class

Properties

- In OWL there are two kinds of properties
  - Object properties, which relate objects to other objects
    - E.g. is-TaughtBy, supervises
  - Data type properties, which relate objects to datatype values
    - E.g. phone, title, age, etc.

Datatype Properties

- OWL makes use of XML Schema data types, using the layered architecture of the SW

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

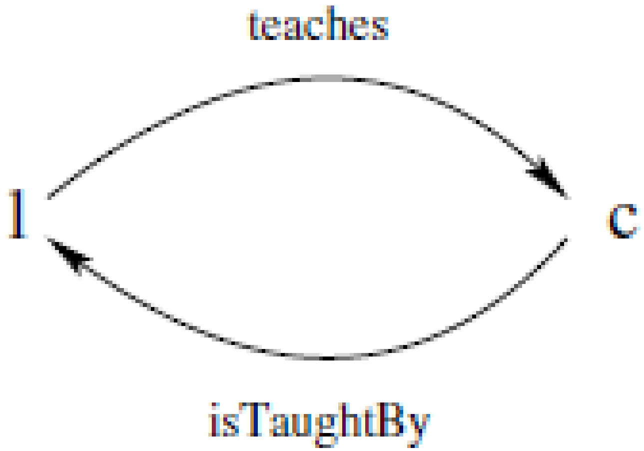
Object Properties

- User-defined data types

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

Inverse Properties

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



Equivalent Properties

owl:equivalentProperty

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

Property Restrictions

- In OWL we can declare that the class C satisfies certain conditions
  - All instances of C satisfy the conditions
- This is equivalent to saying that C is subclass of a class C', where C' collects all objects that satisfy the conditions
  - C' can remain anonymous

Property Restrictions (2)

- A (restriction) class is achieved through an **owl:Restriction** element
- This element contains an **owl:onProperty** element and one or more restriction declarations
- One type defines cardinality restrictions (at least one, at most 3,…)

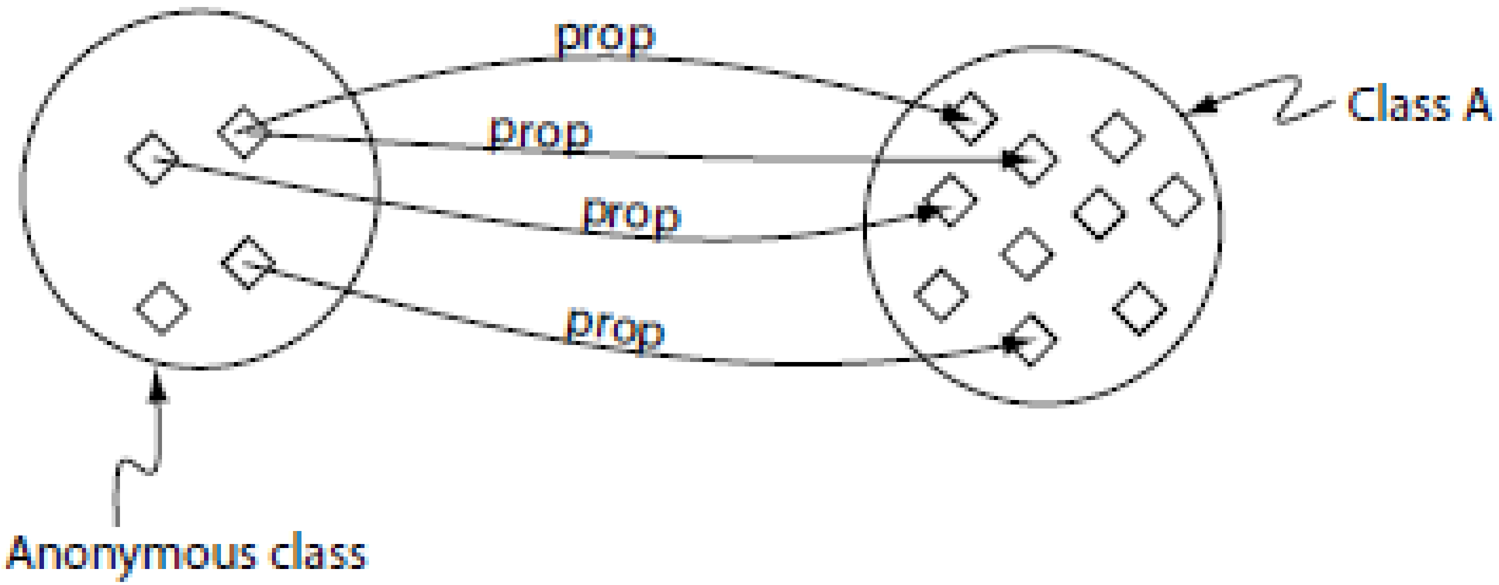
Property Restrictions (3)

- The other type defines restrictions on the kinds of values the property may take
  - **owl:allValuesFrom** specifies universal quantification
  - **owl:someValuesFrom** specifies existential quantification
  - **owl:hasValue** specifies a specific value

owl:allValuesFrom

```
<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>
```

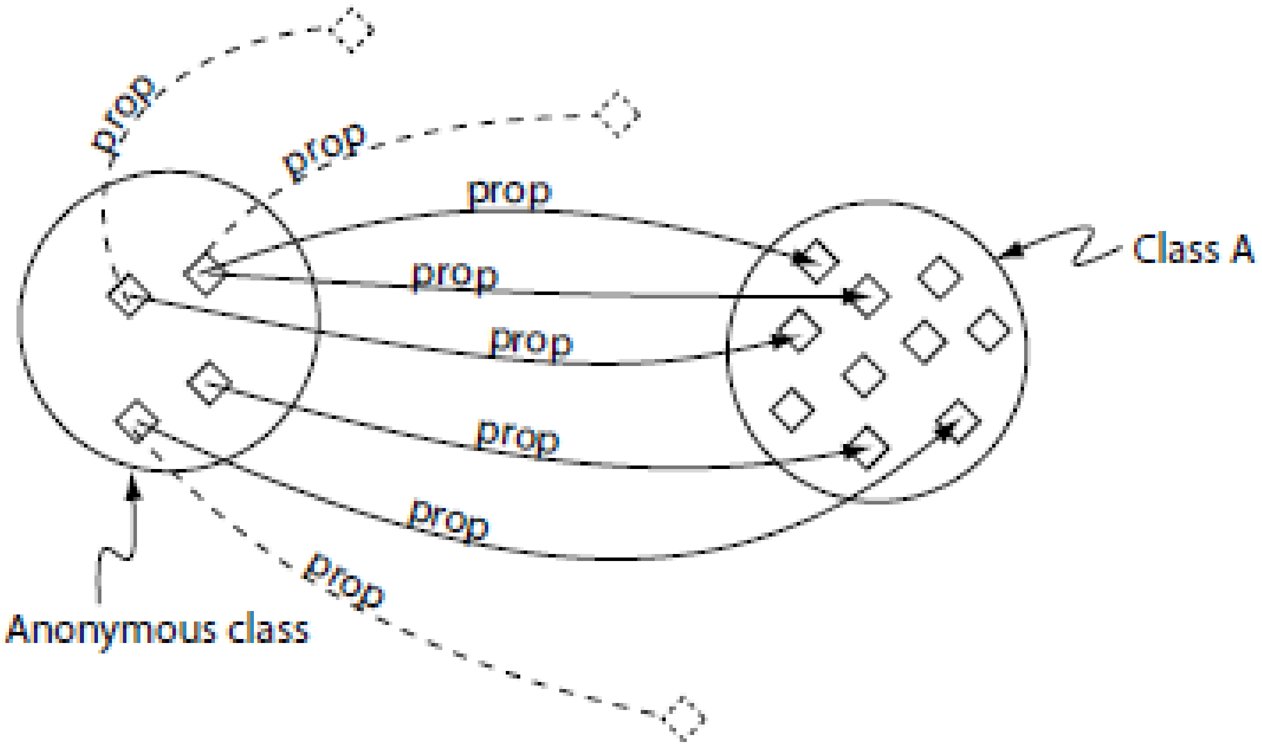
owl:allValuesFrom (illustration)



owl:someValuesFrom

```
<owl:Class rdf:about="#academicStaffMember">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#teaches"/>
      <owl:someValuesFrom rdf:resource=
            "#undergraduateCourse"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

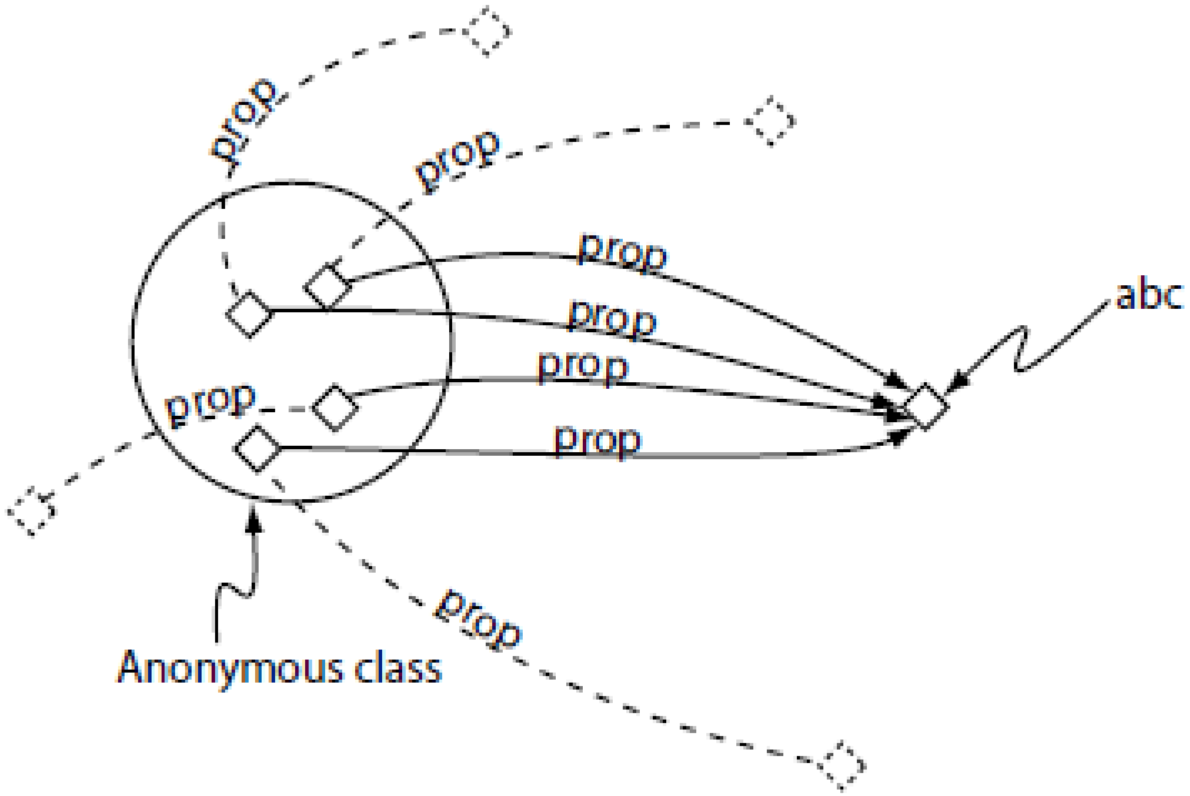
owl:someValuesFrom (illustration)



owl:hasValue

```
<owl:Class rdf:about="#mathCourse">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource=
                "#isTaughtBy"/>
            <owl:hasValue rdf:resource=
                "#949352"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

owl:hasValue (illustration)



Cardinality Restrictions

- We can specify minimum and maximum number using **owl:minCardinality** and **owl:maxCardinality**
- It is possible to specify a precise number by using the same minimum and maximum number
- For convenience, OWL offers also **owl:cardinality**

Cardinality Restrictions (2)

```
<owl:Class rdf:about="#course">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isTaughtBy"/>
            <owl:minCardinality rdf:datatype=
                "&xsd;nonNegativeInteger">
            1
            </owl:minCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Special Properties

- **owl:TransitiveProperty (**(transitive property)
  - E.g. “has better grade than”, “is ancestor of”
- **owl:SymmetricProperty** (symmetry)
  - E.g. “has same grade as”, “is sibling of”
- **owl:FunctionalProperty** defines a property that has at most one value for each object
  - E.g. “age”, “height”, “directSupervisor”
- **owl:InverseFunctionalProperty** defines a property for which two different objects cannot have the same value
  - E.g. “Social ID” (JMBG)

Special Properties (2)

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

Boolean Combinations

- We can combine classes using Boolean operations (union, intersection, complement)

```
<owl:Class rdf:about="#undergraduate">
    <rdfs:subClassOf>
        <owl:Restriction>
                <owl:complementOf rdf:resource=
                        "#graduate"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Boolean Combinations (2)

```
<owl:Class rdf:ID="peopleAtUni">
   <owl:unionOf rdf:parseType="Collection">
         <owl:Class rdf:about="#staffMember"/>
         <owl:Class rdf:about="#student"/>
   </owl:unionOf>
</owl:Class>
```

- The new class is not a subclass of the union, but rather equal to the union
  - We have stated an equivalence of classes

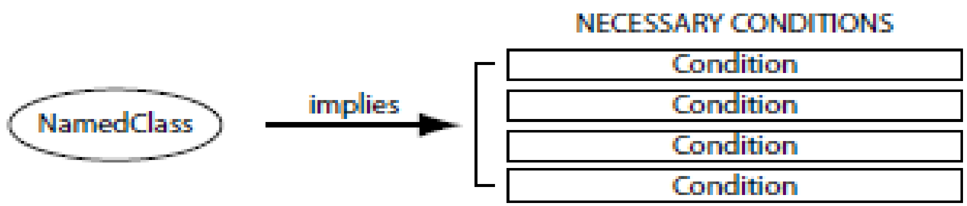
Boolean Combinations (3)

```
<owl:Class rdf:ID="facultyInCS">
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#faculty"/>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#belongsTo"/>
            <owl:hasValue rdf:resource=
                "#CSDepartment"/>
        </owl:Restriction>
    </owl:intersectionOf>
</owl:Class>
```

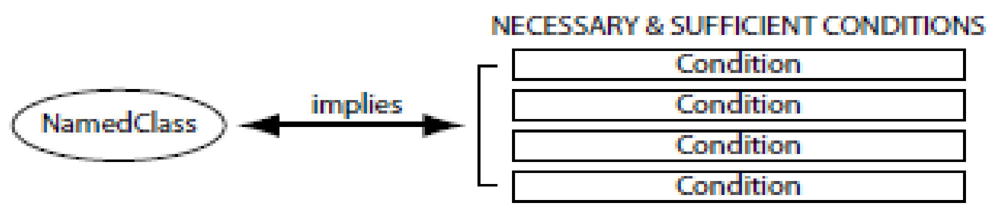
Nesting of Boolean Operators

```
<owl:Class rdf:ID="adminStaff">
	<owl:intersectionOf rdf:parseType="Collection">
		<owl:Class rdf:about="#staffMember"/>
		<owl: Class>
			<owl:complementOf>
				<owl: Class>
					<owl:unionOf  rdf:parseType="Collection">
						<owl:Class rdf:about="#faculty"/>
						<owl:Class rdf:about=#techSupportStaff"/>
					</owl:unionOf>
				</owl: Class>
			</owl:complementOf>
		</owl: Class>
	</owl:intersectionOf>
</owl:Class>
```

Necessary And Sufficient Conditions (Primitive and Defined Classes)



If an individual is a member of 'NamedClass' then it must satisfy the conditions. However if some individual satisfies these necessary conditions, we cannot say that it is a member of 'Named Class' (the conditions are not 'sufficient' to be able to say this) - this is indicated by the direction of the arrow.

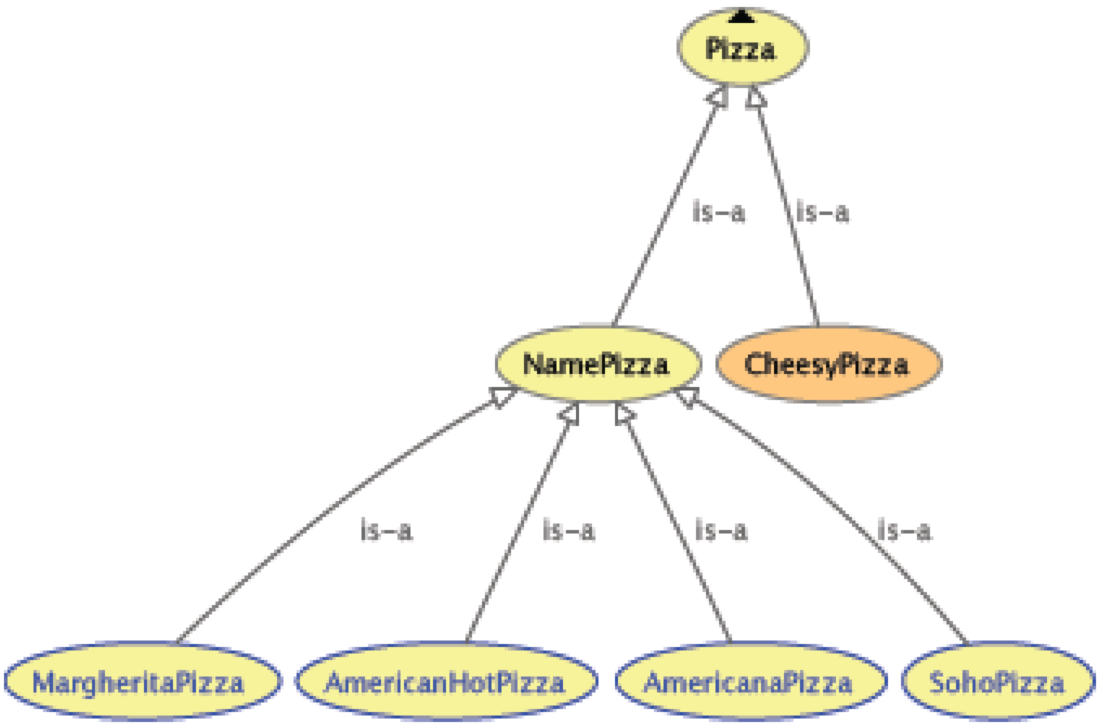


If an individual is a memeber of 'NamedClass' then it must satisfy the conditions. If some individual satisfies the condtions then the individual must be a member of 'NamedClass' - this is indicated by the double arrow.

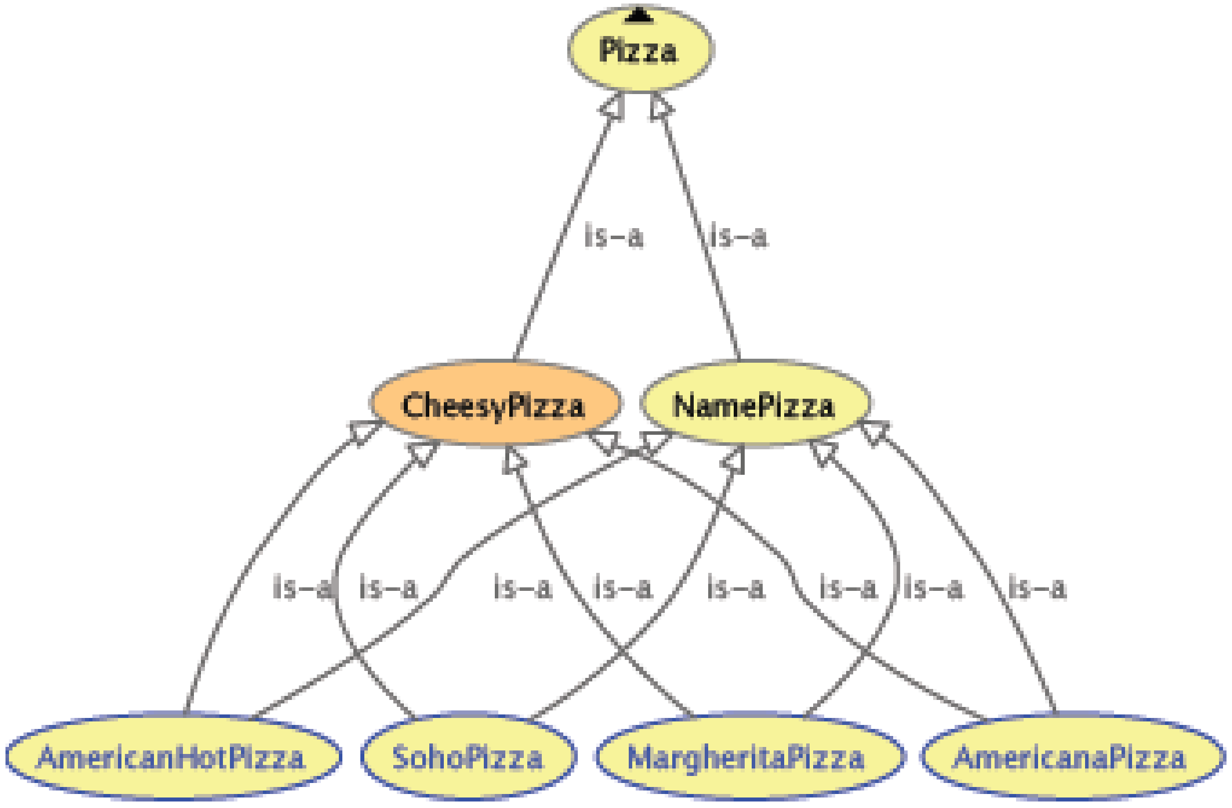
Figure 4.45: Necessary And Sufficient Conditions

Asserted Hierarchy

- CheesyPizza⇔is Pizza and HasTopping some Cheese



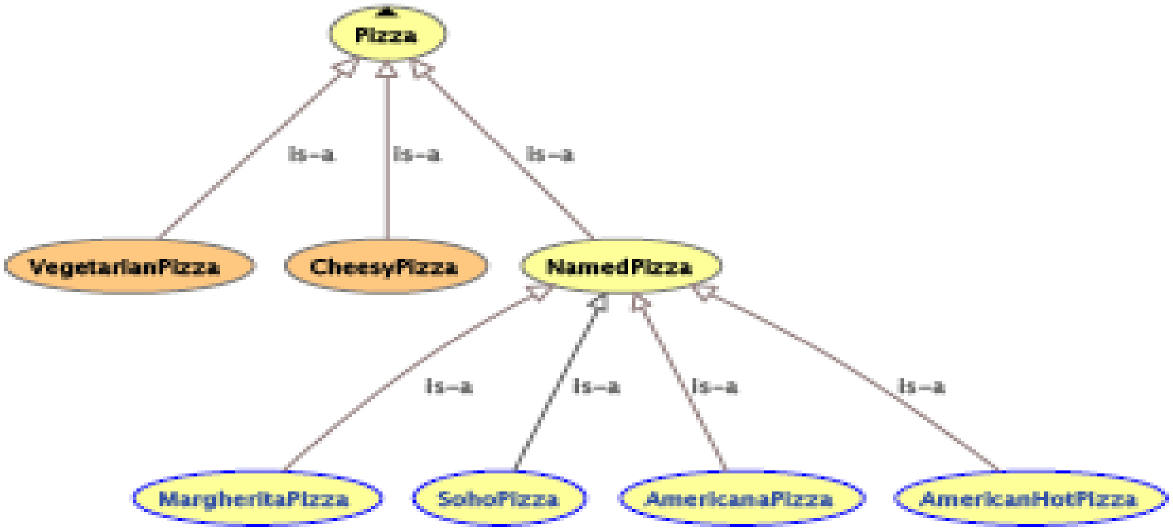
Inferred Hierarchy (reasoner)



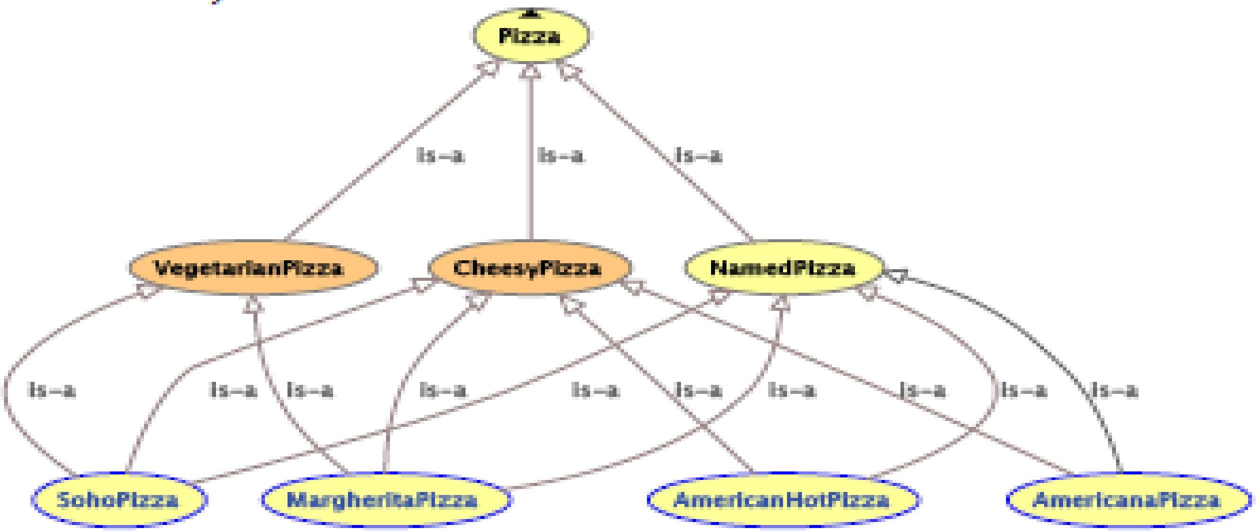
Closure Axiom

- VegetarianPizza⇔hasTopping some (Vegetables or Cheese)
- Not Correct! – There are some pizzas with vegetables that are nonVegetarian
- VegetarianPizza => hasTopping some (Vegetables or Cheese) and only (Vegetables or Cheese)

Asserted Hierarchy



Inferred Hierarchy



Enumerations with owl:oneOf

```
<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>
```

Declaring Instances

- Instances of classes are declared as in RDF:

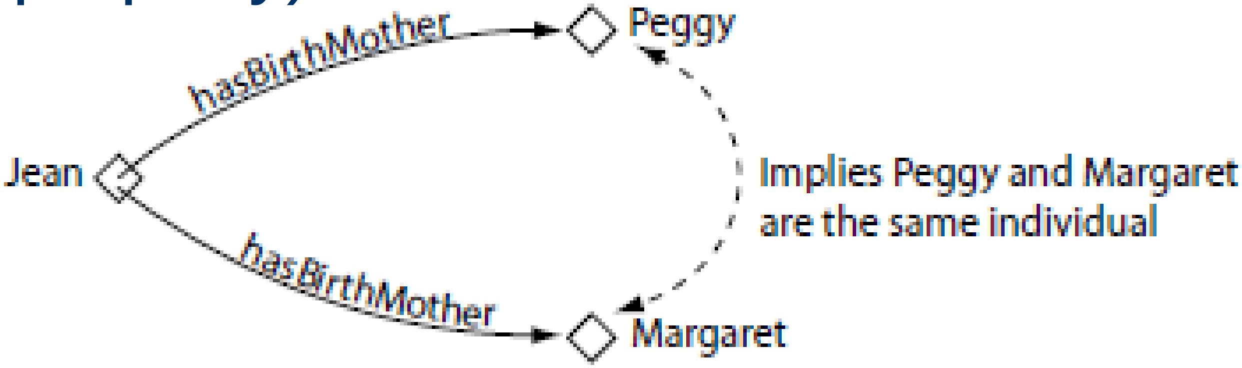
```
<rdf:Description rdf:ID="949352">
    <rdf:type rdf:resource=
        "#academicStaffMember"/>
</rdf:Description>
```

- Equivalently

```
<academicStaffMember rdf:ID="949352“/>
```

No Unique-Names Assumption

- OWL does not adopt the unique-names assumption of database systems
  - If two instances have a different name or ID does not imply that they are different individuals
- E.g. Every person has at most one Mother (functional property)



Distinct Objects

- To ensure that different individuals are indeed recognized as such, we must explicitly assert their inequality:

```
<lecturer rdf:about="949318">
    <owl:differentFrom rdf:resource="949352"/>
</lecturer>
```

Distinct Objects (2)

- OWL provides a shorthand notation to assert the pairwise inequality of all individuals in a given list

```
<owl:allDifferent>
    <owl:distinctMembers rdf:parseType="Collection">
        <lecturer rdf:about="949318"/>
        <lecturer rdf:about="949352"/>
        <lecturer rdf:about="949111"/>
    </owl:distinctMembers>
</owl:allDifferent>
```

Open-world assumption

- We cannot conclude some statement x to be false simply because we cannot show x to be true
- We may not deduce falsity from the absence of truth
- OWL uses Open-world assumption

Open-world assumption example

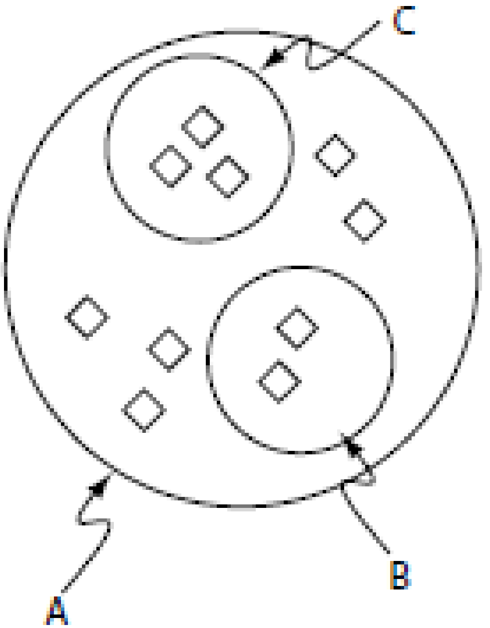
- **Question:** "Did it rain in Tokyo yesterday?"
- **Answer:** "I don’t know that it rained , but that’s not enough reason to conclude that it didn’t rain"

Closed-world assumption (CWA)

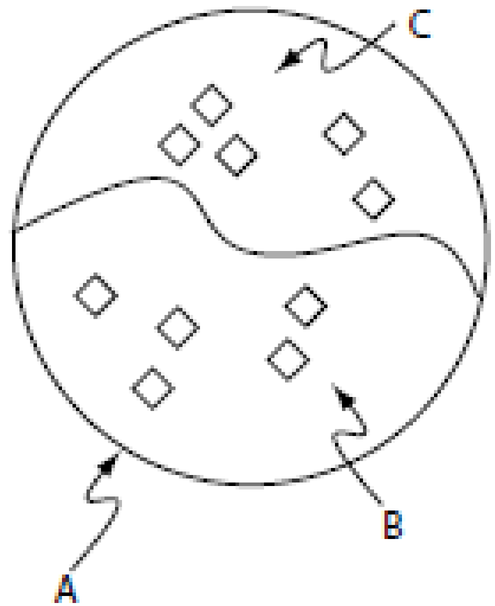
- Closed-world assumption allow deriving falsity from the inability to derive truth
  - (eg. Databases)
- Example:
  - **Question:** " Was there a big earthquake disaster in Tokyo yesterday? "
  - **Answer:** " I don’t know that there was, but if there had been such a disaster, I’d have heard about it. Therefore I conclude that there wasn’t such a disaster"

Covering axiom

- Eg. Person is Male or Female



Without a covering axiom
(B and C are subclasses of A)



With a covering axiom
(B and C are subclasses of A
and A is a subclass of B union C)

Covering axiom (Example)

- Male and Female are disjoint and are subclasses of Person

```
<owl:Class rdf:ID="person">
  <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#male"/>
      <owl:Class rdf:about="# female"/>
  </owl:unionOf>
</owl:Class>
```

OWL DLP use

- Systems such as databases and logic-programming systems have tended to support closed worlds and unique names
- Knowledge representation systems and theorem plovers support open worlds and non-unique names

Tutorials

- Getting Started with Protege 4.x OWL, http://protegewiki.stanford.edu/wiki/Protege4GettingStarted
- Horridge M., *A Practical Guide To Building OWL Ontologies Using Protege 4 and CO-ODE Tools*, Ed 1.2, 2009. http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/
- DL Query tab, http://protegewiki.stanford.edu/wiki/DLQueryTab