

# **1. Introduction**

# Outline

## ◆ Introduction

- ◆ XML, DTD
- ◆ Introduction to Semantic Web
- ◆ Semantic Web applications
- ◆ Ontology Engineering
- ◆ Text Information Retrieval
- ◆ Ontology Modeling Framework

# References

- ◆ Web – W3C specification
- ◆ John Dominque, Dieter Fensel, James A. Hendler, **Handbook of Semantic Web Technologies**, Springer, 2011
- ◆ Frank Van Harmelen, Vladimir Lifschitz, Bruce Porter, **Handbook of Knowledge Representation, Foundations of AI**, 2008

# **XML & DTD**

# **What is XML**

- ◆ **XML stands for eXtensible Markup Language.**
- ◆ **A markup language is used to provide information about a document.**
- ◆ **Tags are added to the document to provide the extra information.**
- ◆ **HTML tags tell a browser how to display the document.**
- ◆ **XML tags give a reader some idea what some of the data means.**

# **What is XML Used For?**

- ◆ **XML documents are used to transfer data from one place to another often over the Internet.**
- ◆ **XML subsets are designed for particular applications.**
- ◆ **One is RSS (Rich Site Summary or Really Simple Syndication ). It is used to send breaking news bulletins from one web site to another.**
- ◆ **A number of fields have their own subsets. These include chemistry, mathematics, and books publishing.**
- ◆ **Most of these subsets are registered with the W3Consortium and are available for anyone's use.**

# **Advantages of XML**

- ◆ **XML is text (Unicode) based.**
  - Takes up less space.
  - Can be transmitted efficiently.
- ◆ **One XML document can be displayed differently in different media.**
  - Html, video, CD, DVD,
  - You only have to change the XML document in order to change all the rest.
- ◆ **XML documents can be modularized. Parts can be reused.**

# Example of an HTML Document

```
<html>
  <head><title>Example</title></head>
  <body>
    <h1>This is an example of a page.</h1>
    <h2>Some information goes here.</h2>
  </body>
</html>
```

# Example of an XML Document

```
<?xml version="1.0"?>  
<address>  
  <name>Abebe kebde</name>  
  <email>abebekebede@gmail.com</email>  
  <phone>00251911616161</phone>  
  <birthday>1985-03-22</birthday>  
</address>
```

# Difference Between HTML and XML

- ◆ HTML tags have a fixed meaning and browsers know what it is.
- ◆ XML tags are different for different applications, and users know what they mean.
- ◆ HTML tags are used for display.
- ◆ XML tags are used to describe documents and data.

# XML Rules

- ◆ **Tags are enclosed in angle brackets.**
- ◆ **Tags come in pairs with start-tags and end-tags.**
- ◆ **Tags must be properly nested.**
  - ◆ `<name><email>...</name></email>` is not allowed.
  - ◆ `<name><email>...</email><name>` is.
- ◆ **Tags that do not have end-tags must be terminated by a '/'.**
  - ◆ `<br />` is an html example.

# More XML Rules

- ◆ **Tags are case sensitive.**
  - <address> is not the same as <Address>
- ◆ **XML in any combination of cases is not allowed as part of a tag.**
- ◆ **Tags may not contain '<' or '&'.**
- ◆ **Tags follow Java naming conventions, except that a single colon and other characters are allowed. They must begin with a letter and may not contain white space.**
- ◆ **Documents must have a single *root* tag that begins the document.**

# Encoding

- ◆ XML (like Java) uses Unicode to encode characters.
- ◆ Unicode comes in many flavors. The most common one used in the West is UTF-8.
- ◆ UTF-8 is a variable length code. Characters are encoded in 1 byte, 2 bytes, or 4 bytes.
- ◆ The first 128 characters in Unicode are ASCII.
- ◆ In UTF-8, the numbers between 128 and 255 code for some of the more common characters used in western Europe, such as ã, á, å, or ç.
- ◆ Two byte codes are used for some characters not listed in the first 256 and some Asian ideographs.
- ◆ Four byte codes can handle any ideographs that are left.
- ◆ Those using non-western languages should investigate other versions of Unicode.

# **Well-Formed Documents**

- ◆ An XML document is said to be well-formed if it follows all the rules.
- ◆ An XML parser is used to check that all the rules have been obeyed.
- ◆ Recent browsers such as Internet Explorer 5 and Netscape 7 come with XML parsers.
- ◆ Parsers are also available for free download over the Internet. One is Xerces, from the Apache open-source project.
- ◆ Java 1.4 also supports an open-source parser.

# XML Example Revisited

```
<?xml version="1.0"/>
<address>
  <name>Abebe kebde</name>
  <email>abebekebede@gmail.com</email>
  <phone>00251911616161</phone>
  <birthday>1985-03-22</birthday>
</address>
```

- ◆ **Markup for the data aids understanding of its purpose.**
- ◆ **A flat text file is not nearly so clear.**

Abebe kebde

[abebekebede@gmail.com](mailto:abebekebede@gmail.com)

00251911616161

1985-03-22

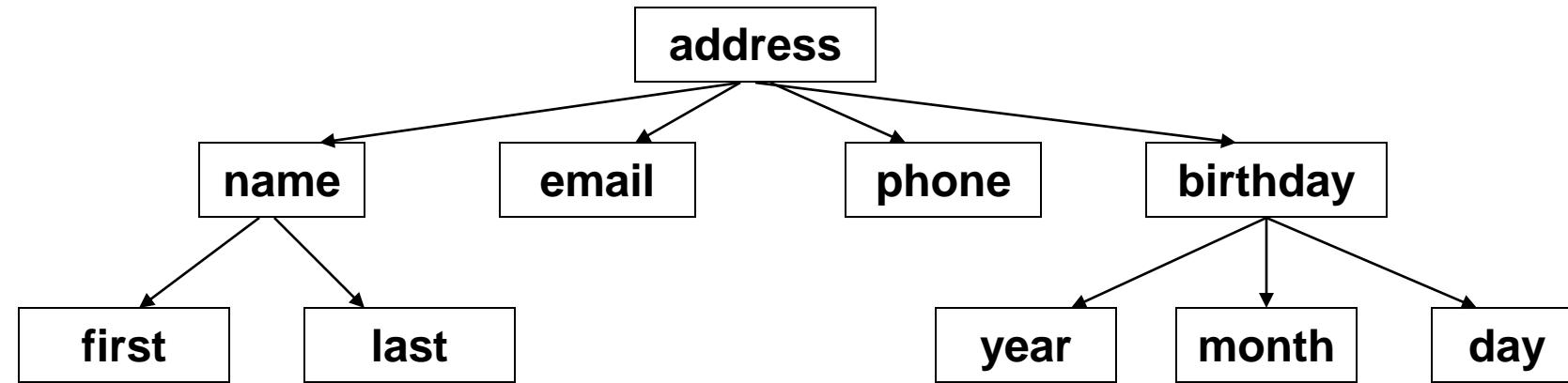
- ◆ **The last line looks like a date, but what is it for?**

# Expanded Example

```
<?xml version = “1.0” ?>
<address>
    <name>
        <first>Abebe </first>
        <last>kebde</last>
    </name>
    <email>abebekebede@gmail.com</email>
    <phone>00251911616161</phone>
    <birthday>
        <year>1985</year>
        <month>03</month>
        <day>22</day></birthday>
    </address>
```

# XML Files are Trees

---



# XML Trees

- ◆ An XML document has a **single root node**.
- ◆ **The tree is a general ordered tree.**
  - ◆ A parent node may have any number of children.
  - ◆ Child nodes are ordered, and may have siblings.
- ◆ **Preorder traversals are usually used for getting information out of the tree.**

# Validity

- ◆ A well-formed document has a tree structure and obeys all the XML rules.
- ◆ A particular application may add more rules in either a DTD (document type definition) or in a schema.
- ◆ Many specialized DTDs and schemas have been created to describe particular areas.
- ◆ These range from disseminating news bulletins (RSS) to chemical formulas.
- ◆ DTDs were developed first, so they are not as comprehensive as schema.

# Document Type Definitions - DTDs

- ◆ A DTD describes the tree structure of a document and something about its data.
- ◆ There are two data types, PCDATA and CDATA.
  - ◆ PCDATA is parsed character data.
  - ◆ CDATA is character data, not usually parsed.
- ◆ A DTD determines how many times a node may appear, and how child nodes are ordered.

# DTD for address Example

```
<!ELEMENT address (name, email, phone, birthday)>
<!ELEMENT name (first, last)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT birthday (year, month, day)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT month (#PCDATA)>
<!ELEMENT day (#PCDATA)>
```

# Schemas

- ◆ **Schemas are themselves XML documents.**
- ◆ **They were standardized after DTDs and provide more information about the document.**
- ◆ **They have a number of data types including string, decimal, integer, boolean, date, and time.**
- ◆ **They divide elements into simple and complex types.**
- ◆ **They also determine the tree structure and how many children a node may have.**

# Schema for First address Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="address">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="email" type="xs:string"/>
      <xs:element name="phone" type="xs:string"/>
      <xs:element name="birthday" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

# Explanation of Example Schema

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

- ◆ ISO-8859-1, Latin-1, is the same as UTF-8 in the first 128 characters.

```
<xm:schema xmlns:xm="http://www.w3.org/2001/XMLSchema">
```

- ◆ [www.w3.org/2001/XMLSchema](http://www.w3.org/2001/XMLSchema) contains the schema standards.

```
<xm:element name="address">
```

```
  <xm:complexType>
```

- ◆ This states that address is a complex type element.

```
    <xm:sequence>
```

- ◆ This states that the following elements form a sequence and must come in the order shown.

```
<xm:element name="name" type="xm:string"/>
```

- ◆ This says that the element, name, must be a string.

```
<xm:element name="birthday" type="xm:date"/>
```

- ◆ This states that the element, birthday, is a date. Dates are always of the form yyyy-mm-dd.

# XSLT- Extensible Stylesheet Language Transformations

- ◆ XSLT is used to transform one xml document into another, often an html document.
- ◆ The Transform classes are now part of Java 1.4.
- ◆ A program is used that takes as input one xml document and produces as output another.
- ◆ If the resulting document is in html, it can be viewed by a web browser.
- ◆ This is a good way to display xml data.

# A Style Sheet to Transform address.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="address">
    <html><head><title>Address Book</title></head>
    <body>
      <xsl:value-of select="name"/>
      <br/><xsl:value-of select="email"/>
      <br/><xsl:value-of select="phone"/>
      <br/><xsl:value-of select="birthday"/>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

# The Result of the Transformation

---

Abebe kebde

abebekebede@gmail.com

00251911616161

1985-03-22

# Parsers

- ◆ There are two principal models for parsers.
- ◆ **SAX – Simple API for XML**
  - ◆ Uses a call-back method
  - ◆ Similar to javax listeners
- ◆ **DOM – Document Object Model**
  - ◆ Creates a parse tree
  - ◆ Requires a tree traversal

# **Introduction to Semantic Web**

# What is the Semantic Web?

- ◆ The Semantic Web is not a separate web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation.
- ◆ “The Semantic Web is a Web of actionable information- information derived from data through a **semantic theory** for interpreting the symbols.”
- ◆ “The semantic theory provides an account of ‘meaning’ in which the logical connection of terms establishes interoperability between systems”

# What is the Semantic Web?

- ◆ **The Semantic Web builds upon**
  - the principles and technologies of the web
  - It reuses the Web's global indexing and naming scheme
  - Semantic Web documents can be accessed through standard Web browsers as well as through semantically aware applications
  - the Web is a shared resource, and therefore, within a machine-readable Web, meaning should be shared too.

# What is the Web?

- ◆ **Web is a global document repository.**
- ◆ **Principles of the Web**
  - ◆ Openness: anyone within the web can be provider or consumer of information. it incorporates:
    - Accessibility: accessed remotely from a wide variety of hardware and software platforms
    - Nonproprietary
    - Consensual control
    - Usable
  - ◆ Interoperability: The Web is neutral to hardware and software platforms
  - ◆ Decentralized authorship and editorship: Content can appear, becoming modified, or be removed in a non-controlled fashion.
  - ◆ Enabling n:m relationship for maximum interaction

- ◆ **It is simple.**
- ◆ **Functional services of the Web**
  - ◆ A worldwide addressing schema: using URLs – resource identification and describing its network location
  - ◆ A transport layer- HTTP – supports remote access to content over a network layer (TCP-IP). HTTP functions as a request –respond protocol in client-server computing model.
  - ◆ A platform-independent interface – enable users to easily access any online resource

# What Are the Problems with the Web?

- ◆ Huge collection of documents
- ◆ Accessing data
  - Documents are indexed and accessed via plain text (i.e. string matching algorithms). For example: “Paris” can denote: the capital of France; towns in Canada, Kiribati, and the USA.
  - The current paradigm is dominated by returning single “best fit” documents for a search.
  - Underlying data are not available. A significant number of websites are generated through databases but the underlying data are hidden behind the presented HTML. (Dark Web)
  - Enabling delegation of tasks such as the integration of information, data analysis, and sense making to machines, at least partially

# What Are the Problems with the Web?

- ◆ Semantic Web extends the Web with “meaning” supporting access to data at web-scale and enabling the delegation of certain classes of tasks.
- ◆ The Web has documents at the center
- ◆ The Semantic Web places data and the semantics of data at its core.

# What Are Semantics?

- ◆ Semantic technology provides machine-understandable (or better machine-processable) descriptions of data, programs, and infrastructure, enabling computers to reflect on these artifacts.

# Semantic Web Languages

## ◆ HTML

- It provides a number of ways to express the semantics of data. An obvious one is the META tag  
`<META name = "Author" lang= "fr" content = "Arnaud Le Hors">`
- Ontobroker used the attribute of the anchor tag to encode semantic information.
- But, HTML was not designed to provide descriptions of documents beyond that of informing the browser on how to render the contents.
- Semantic HTML

## ◆ The Extensible Markup Language (XML)

- It is a generic way to structure documents on the Web. It generalizes HTML by allowing user-defined tags.
- It is flexible but reduces the possibilities for the type of semantic interpretation that was possible with the predefined tags of HTML.

# Semantic Web Languages

## ◆ The Resource Description Framework (RDF)

- It is a simple data model for semantically describing resources on the Web.
- Binary properties interlink terms forming a directed graph.
- Terms as well as properties are described using URIs.
- As a property can be a URI, it can be used as a term interlinked to another property.
- Unlike most logical languages or databases, it is not possible to distinguish the language or schema from statements in the language or schema.
- Example: <rdf:type, rdf:type, rdf:Property>

## ◆ RDF schema (RDFS)

- It uses basic RDF statements and defines a simple ontology language.
- It defines entities such as rdfs:class, rdfs:subclass, rdfs:subproperty, rdfs:domain, and rdfs:range, enabling one to model classes, properties with domain and range restrictions, and hierarchies of classes and properties.
- It is RDF + some more definitions (statements) in RDF.

# Semantic Web Languages

## ◆ The Web Ontology Language OWL

- It extends vocabulary to a full-fledged spectrum of Descriptions Logics defined in RDF, namely, OWL Lite, OWL DL, and OWL Full.
- It allows mechanisms for defining properties to be inverse, transitive, symmetric, or functional. Properties can be used to define the membership of instances for classes or hierarchies of classes and of properties.

## ◆ OWL2

- Defined some of the issues around OWL.
- OWL Lite had been defined as an over expressive Description Logic. Three sub-languages:
  - OWL2EL - all the standard reasoning tasks of description logic
  - OWL2QL enables efficient query answering over large instance populations
  - OWL2RL restricts the expressiveness with respect to extensibility toward rule languages.

# Semantic Web Languages

- ◆ **The Rule Interchange Format (RIF)**
  - It complements OWL with a language framework centered on the rule paradigm.
  - RIF-BLD – defines a simple logic-oriented rule language
  - RIF-PRD captures most of the aspects of production rule systems
  - RIF-Core is the intersection of both of these languages
- ◆ **SPARQL**
  - It is a query language for the graph-based data model of RDF.
  - SPARQL has been developed without considering RDFS, OWL, and RIF

# Semantic Web Languages

```
<person>
  <name>Sir Tim</name>
  <phone number>01-444444 </phone number>
</person>
Concept – Person
Property: Name, Phone number
```

How do we know that Sir Tim is then name of a person?

- ◆ **Microformats** : are predefined formats to add meta information to elements in HTML and XML. It provides a language structure to present information and provide domain specific terminologies as well.
- ◆ **hCard** can be used for representing people, companies, organizations, and places, using vCard, a file format standard for electronic business cards.
- ◆ **RDFa** provides a set of XHTML attributes to include RDF metadata directly into HTML and XML documents.
- ◆ **GRDDL** has been developed as a mechanism for Gleaning Resource Descriptions from Dialects of Languages to derive **RDFa** definitions from **Microformats** so as to integrate information from heterogeneous sources.

# The Semantic Web: why?

- ◆ The traditional web is web of documents and it is meant mainly for humans and has two major problems
- ◆ **Problem 1:** web documents do not distinguish between information content and presentation
  - ◆ (“solved” by XML)
- ◆ **Problem 2:** different web documents may represent in different ways semantically related pieces of information
  - ◆ this leads to hard problems for “intelligent” information search on the Web

# Separating content and presentation

Problem 1: web documents do not distinguish between information content and presentation

- ◆ Cause of the problem: the HTML language
- ◆ Current Technology solution: – stylesheets (HTML, XML), XML
- ◆ stylesheets allow for separating formatting attributes from the information presented

```
.g, body, html, input, .std, h1 {  
    font-size: small;  
    font-family: arial,sans-serif;  
}
```

# Separating content and presentation

- ◆ XML: eXtensible Mark-up Language
- ◆ XML documents are written through a user defined set of tags
- ◆ tags are used to express the “semantics” of the various pieces of information

## Example: HTML

```
<html>
  <body>
    <H1> Seminar: Article Review Presentation Schedule</H1>
    <UL>
      <LI>fekadegetahun@gmail.com</LI> <LI>aaucs_gs03@yahoogroups.com</LI>
      <LI>CoSc 705 - Seminar: Article Review Presentation Schedule</LI >
      <LI> The presentations of the seminar course are scheduled from July 4 - 8,
          2016. Please find attached the presentation schedule</LI>
    </UL>
  </body>
</html>
```

## Example: XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <from>fekadegetahun@gmail.com </from>
  <to>aaucs\_gs03@yahoogroups.com </to>
  <subject> CoSc 705 - Seminar: Article Review Presentation Schedule </subject>
  <body> The presentations of the seminar course are scheduled from July 4 - 8,
        2011. Please find attached the presentation schedule </body>
</note>
```

# Example: XML - DTD

```

<!DOCTYPE paper[
  <!ELEMENT paper ((author+ | publisher), version, length?, url*)>
  <!ATTLIST paper title CDATA #REQUIRED
    category CDATA #IMPLIED>
  <!ELEMENT author (firstName, middleName?, lastName)
  <!ELEMENT publisher (#PCDATA | (firstName,
    middleName?, lastName)>
  <!ELEMENT length (#PCDATA)>
  <!ELEMENT version (#PCDATA)>
  <!ELEMENT url (homepage, download+)>
  <!ELEMENT homepage (#PCDATA)>
  <!ELEMENT download (#PCDATA)>
  <!ELEMENT fisrtName (#PCDATA)>
  <!ELEMENT middleName (#PCDATA)>
  <!ELEMENT lastName (#PCDATA)> ]>

```

a. Sample DTD  $D$  with two *Or* ('|') operators

```

<!DOCTYPE paper[
  <!ELEMENT paper (author+, version, length?, url*)>
  <!ATTLIST paper title CDATA #REQUIRED
    category CDATA #IMPLIED >
  <!ELEMENT author (firstName, middleName?, lastName)
  <!ELEMENT version (#PCDATA)>
  <!ELEMENT length (#PCDATA)>
  <!ELEMENT url (download+, homepage)>
  <!ELEMENT download (#PCDATA)>
  <!ELEMENT homepage (#PCDATA)>
  <!ELEMENT fisrtName (#PCDATA)>
  <!ELEMENT middleName (#PCDATA)>
  <!ELEMENT lastName (#PCDATA) ]>

```

b. First conjunctive DTD corresponding to  $D, D_1$

```

<!DOCTYPE paper[
  <!ELEMENT paper (publisher, version, length?, url*)>
  <!ATTLIST paper title CDATA #REQUIRED
    category CDATA #IMPLIED >
  <!ELEMENT publisher (#PCDATA)>
  <!ELEMENT version (#PCDATA)>
  <!ELEMENT length (#PCDATA)>
  <!ELEMENT url (homepage, download+)>
  <!ELEMENT homepage (#PCDATA)>
  <!ELEMENT download (#PCDATA)> ]>

```

c. Second conjunctive DTD  $D_2$  corresponding to  $D$

```

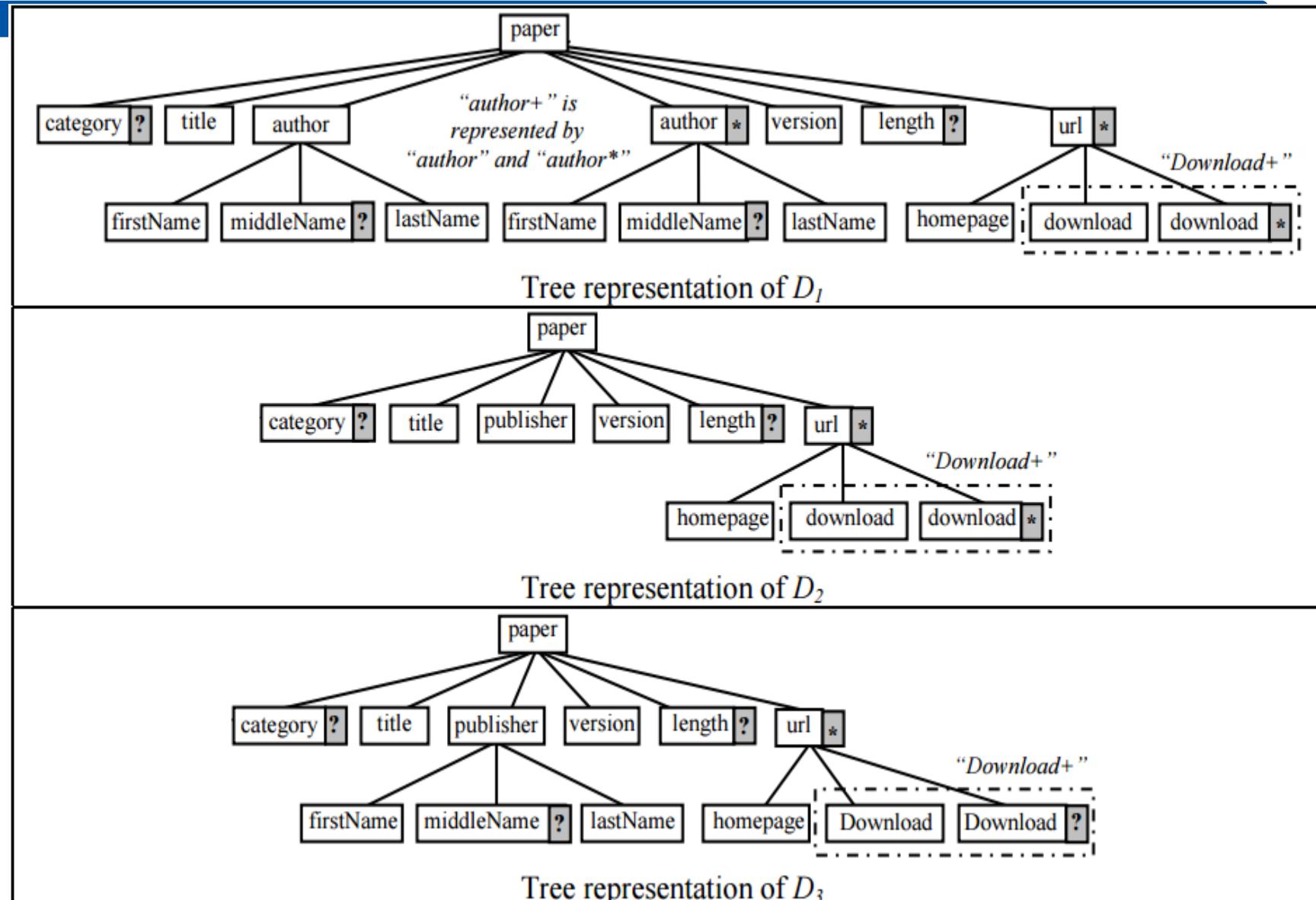
<!DOCTYPE paper[
  <!ELEMENT paper (publisher, version, length?, url*)>
  <!ATTLIST paper title CDATA #REQUIRED
    category CDATA #IMPLIED >
  <!ELEMENT publisher (firstName, middleName?, lastName)>
  <!ELEMENT version (#PCDATA)>
  <!ELEMENT length (#PCDATA)>
  <!ELEMENT url (download+, homepage)>
  <!ELEMENT homepage (#PCDATA)>
  <!ELEMENT download (#PCDATA)>
  <!ELEMENT fisrtName (#PCDATA)>
  <!ELEMENT middleName (#PCDATA)>
  <!ELEMENT lastName (#PCDATA) ]>

```

d. Third conjunctive DTD  $D_3$  corresponding to  $D$

# Limitations of XML

- ◆ XML does not solve all the problems:
- ◆ legacy HTML documents
- ◆ different XML documents may express information with the same meaning but using different tags



# The need for a “Semantic” Web

- Problem 2: different web documents may represent in different ways semantically related pieces of information different XML documents do not share the “semantics” of information
- idea: annotate (mark-up) pieces of information to express the “meaning” of such a piece of information the meaning of such tags is shared! ⇒ shared semantics

```
<?XML>
<Academy>
  <Department>
    <Laboratory>
      <Professor> </Professor>
      <Student> </Student>
    </Laboratory>
  </Department>
</Academy>
```

Sample A

```
<?XML>
<College>
  <Department>
    <Laboratory>
      <Lecturer> </Lecturer>
    </Laboratory>
  </Department>
</College>
```

Sample B

```
<?XML>
<Factory>
  <Department>
    <Laboratory>
      <Supervisor> </Supervisor>
    </Laboratory>
  </Department>
</Factory>
```

Sample C

# The Semantic Web initiative

- ◆ Sir Tim Berners-Lee started provide a layered approach to structure the Semantic Web
- ◆ **Viewpoint:**
  - ◆ the Web = a web of data
- ◆ **Goal:**
  - ◆ to provide a common framework to share data on
  - ◆ the Web across application boundaries
- ◆ **Main ideas:**
  - ◆ Ontology
  - ◆ Standards
  - ◆ “layers”

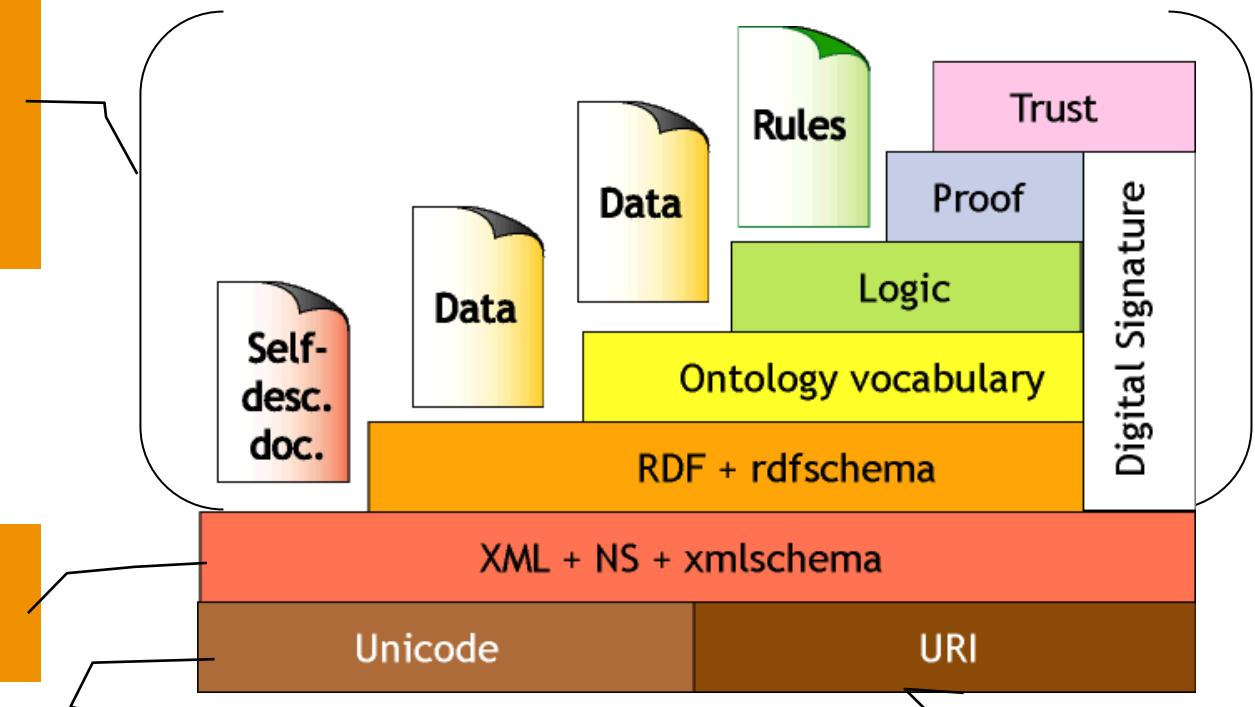
# The Semantic Web Tower

five layers of semantics:  
RDF, OWL, RIF, and layers  
for proof and trust

XML+ NS + XMLSchema provide  
syntactic descriptions of structured  
objects.

**Unicode** a means to encode text

**URIs** to refer to resources



# The XML Layer

- ◆ **XML (eXtensible Markup Language)**
- ◆ **XML documents are written through a user-defined set of tags and their are domain-specific markup**
- ◆ **tags are used to express the “semantics” of the various pieces of information**
- ◆ **URI (Uniform Resource Identifier)**
  - ◆ universal naming for Web resources
  - ◆ same URI = same resource
  - ◆ URIs are the “ground terms” of the SW
- ◆ **W3C standards**

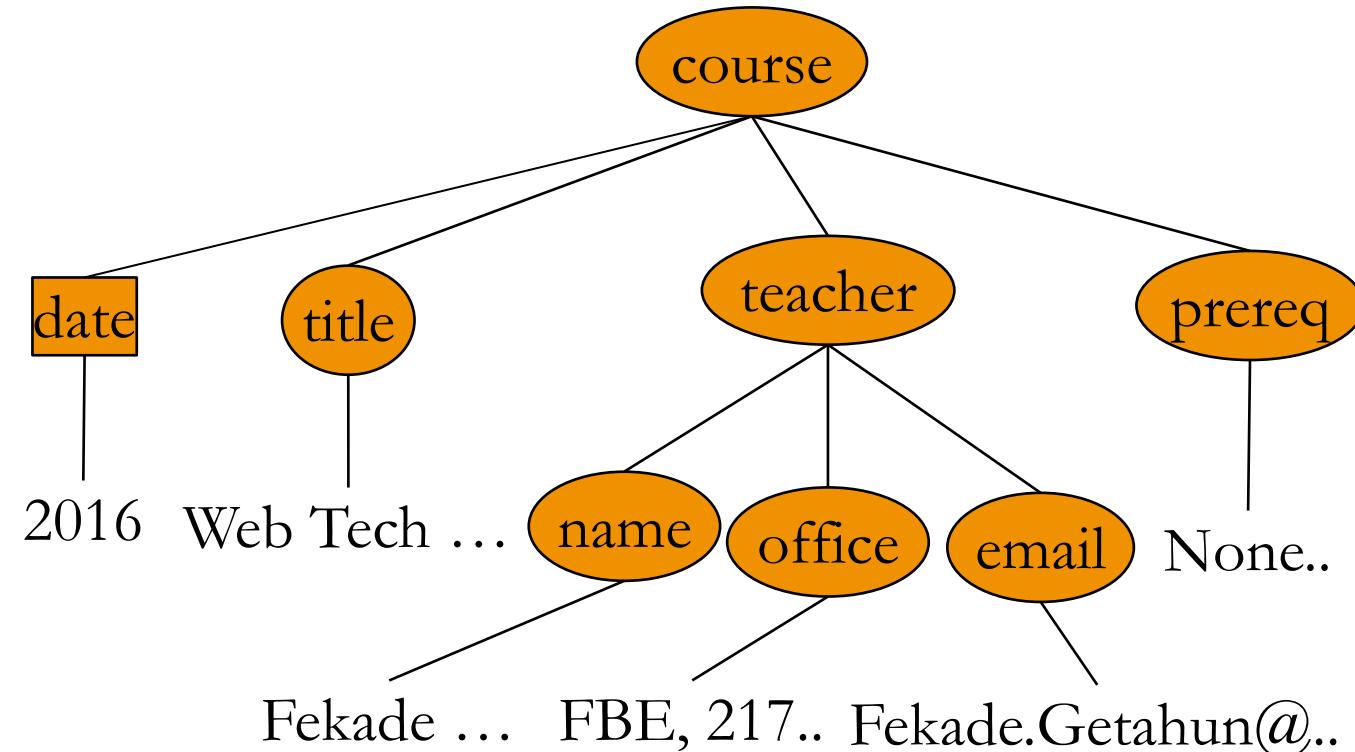
# The XML Layer – example

```
<course date="2016">
  <title>Web Technologies</title>
  <teacher>
    <name>Fekade Getahun</name>
    <office>FBE, 217</office>
    <email>fekade.getahun@aau.edu.et</email>
  </teacher>
  <prereq>none</prereq>
</course>
```

# The XML Layer – example

- **XML: document = labelled tree**
- **node = label + attributes/ values + contents**

```
<course date="2016">
  <title>Web Technologies</title>
  <teacher>
    <name>Fekade Getahun</name>
    <office>FBE, 217</office>
    <email>feakde.getahun@aau.edu.et</email>
  </teacher>
  <prereq>none</prereq>
</course>
```



- ◆ XML Schema – grammar for describing legal trees and datatypes
- ◆ E.g. DTD

```
<!DOCTYPE paper[  
<!ELEMENT paper ((author+ | publisher), version, length?, url*)>  
<!ATTLIST paper title CDATA #REQUIRED  
          category CDATA #IMPLIED>  
<!ELEMENT author (firstName, middleName?, lastName)>  
<!ELEMENT publisher (#PCDATA | (firstName,  
           middleName?, lastName))>  
<!ELEMENT length (#PCDATA)>  
<!ELEMENT version (#PCDATA)>  
<!ELEMENT url (homepage, download+)>  
<!ELEMENT homepage (#PCDATA)>  
<!ELEMENT download (#PCDATA)>  
<!ELEMENT firstName (#PCDATA)>  
<!ELEMENT middleName (#PCDATA)>  
<!ELEMENT lastName (#PCDATA)> ]>
```

```
<!DOCTYPE paper[  
<!ELEMENT paper (publisher, version, length?, url*)>  
<!ATTLIST paper title CDATA #REQUIRED  
          category CDATA #IMPLIED >  
<!ELEMENT publisher (#PCDATA)>  
<!ELEMENT version (#PCDATA)>  
<!ELEMENT length (#PCDATA)>  
<!ELEMENT url (homepage, download+)>  
<!ELEMENT homepage (#PCDATA)>  
<!ELEMENT download (#PCDATA)> ]>
```

### Element Type Declaration

elementdecl	<code>::= '&lt;!ELEMENT' S Name S contentspec S&gt;'</code>
contentspec	<code>::= 'EMPTY'   'ANY'   Mixed   children</code>

### Element-content Models

children	<code>::= (choice   seq) ('?'   '*'   '+') ?</code>
cp	<code>::= (Name   choice   seq) ('?'   '*'   '+') ?</code>
choice	<code>::= '(' S? cp ( S? '   ' S? cp )* S? ')'</code>
seq	<code>::= '(' S? cp ( S? ', ' S? cp )* S? ')'</code>

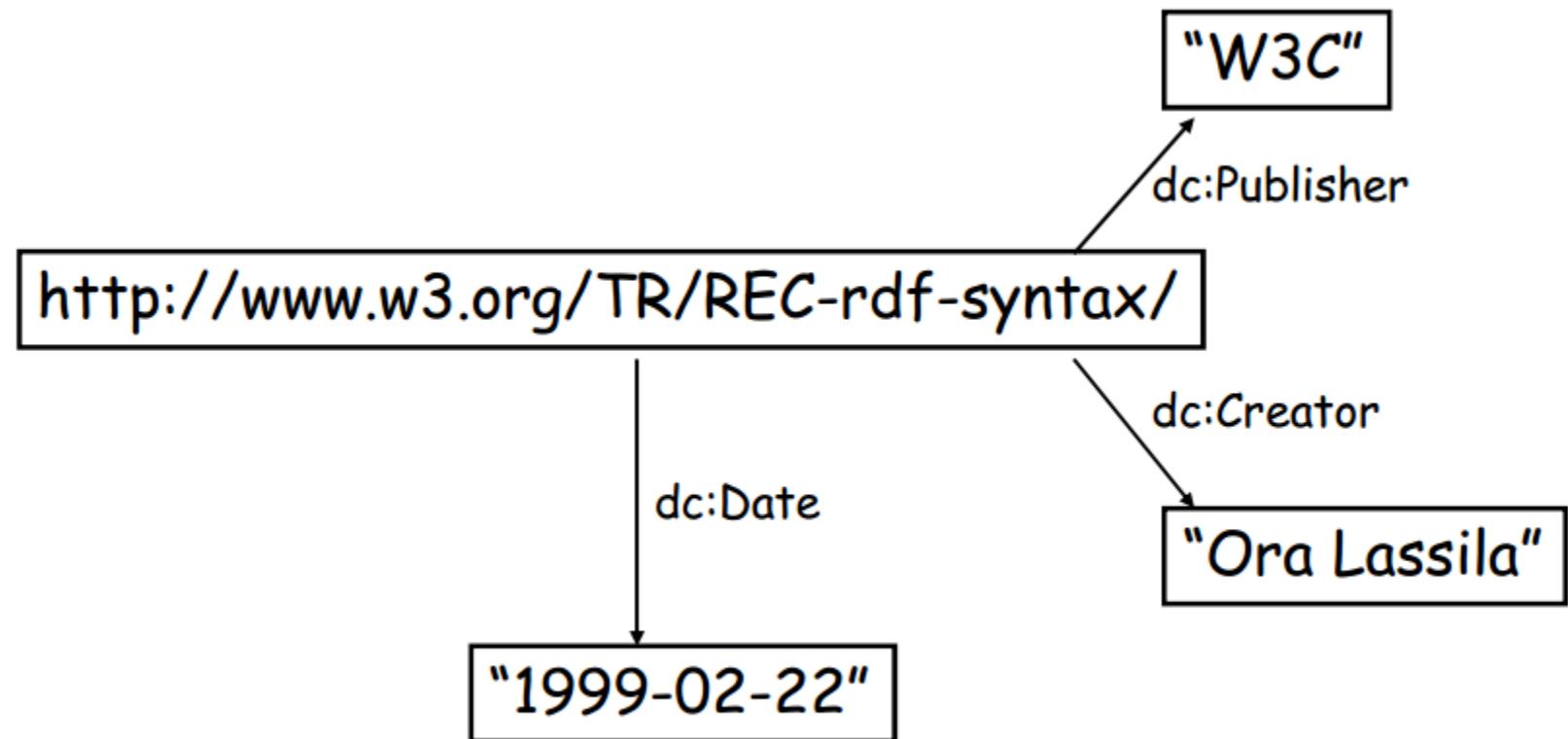
```
<!DOCTYPE paper[  
<!ELEMENT paper (publisher, version, length?, url*)>  
<!ATTLIST paper title CDATA #REQUIRED  
          category CDATA #IMPLIED >  
<!ELEMENT publisher (firstName, middleName?, lastName)>  
<!ELEMENT version (#PCDATA)>  
<!ELEMENT length (#PCDATA)>  
<!ELEMENT url (download+, homepage)>  
<!ELEMENT homepage (#PCDATA)>  
<!ELEMENT download (#PCDATA)>  
<!ELEMENT firstName (#PCDATA)>  
<!ELEMENT middleName (#PCDATA)>  
<!ELEMENT lastName (#PCDATA)> ]>
```

# The RDF + RDFS layer

- ◆ RDF = a simple conceptual data model
- ◆ W3C standard (1999)
- ◆ RDF model = set of RDF triples
- ◆ triple = expression (statement)
  - ◆ (subject, predicate, object)
  - ◆ subject is a resource
  - ◆ predicate is a property (of the resource)
  - ◆ object is the value (of the property)
- ◆ **an RDF model is a graph**

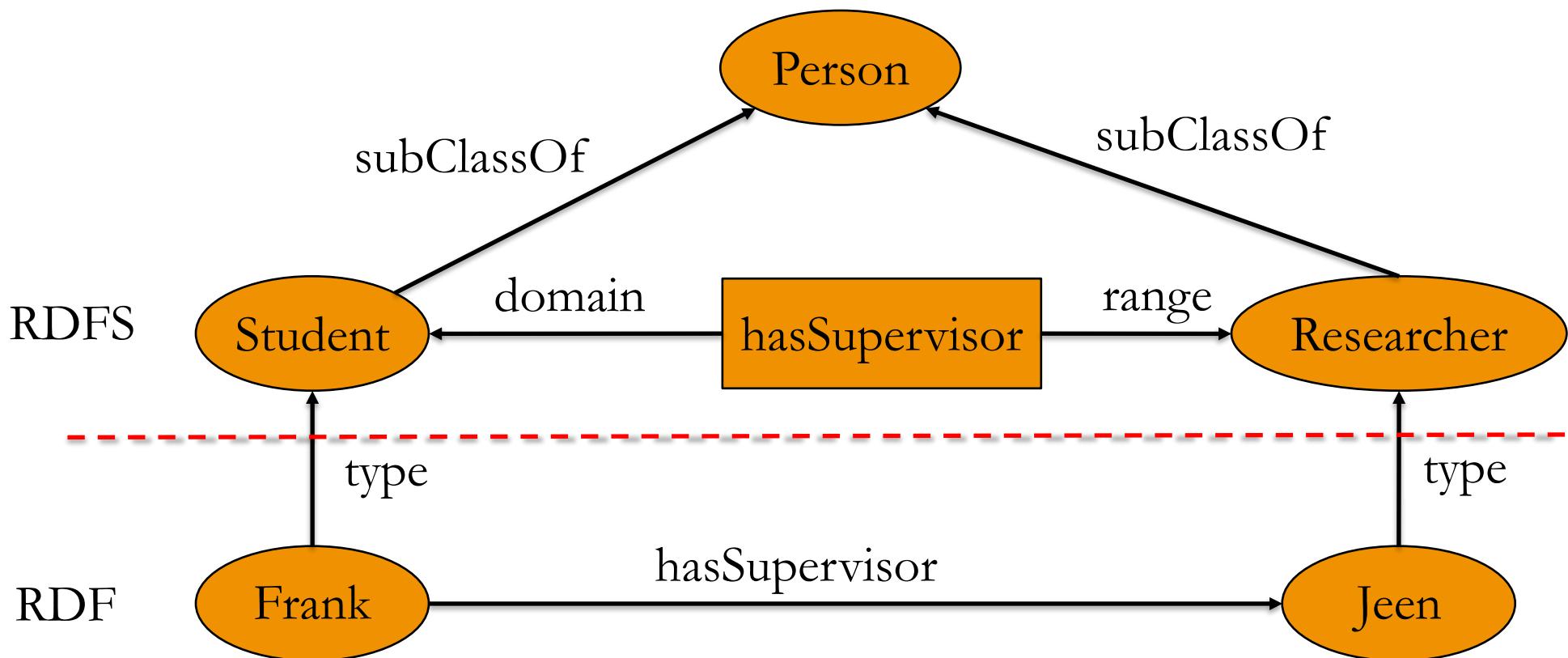
# The RDF + RDFS layer

---



# The RDF + RDFS layer

- ◆ RDFS is RDF Schema
- ◆ It contains “vocabulary” for RDF: Class, domain, range, subClassOf, subPropertyOf
- ◆ It is a W3C standard (2004)



# The Ontology - OWL layer

- ◆ Ontology is a shared conceptualization of a domain of interest
- ◆ It is a conceptual model (more expressive than RDF + RDFS)
- ◆ It is expressed in a true knowledge representation language i.e. OWL (Web Ontology Language), the standard language for ontologies
- ◆ The ontology can be
  - ◆ A simple (shallow) that contain shared vocabulary
  - ◆ Deep ontology that represent (complex) relationships between “terms”

# Ontologies: example

```
class-def animal
class-def plant
    subclass-of NOT animal
class-def tree
    subclass-of plant
class-def branch
    slot-constraint is-part-of
        has-value tree
        max-cardinality 1
class-def defined carnivore
    subclass-of animal
    slot-constraint eats
        value-type animal
class-def defined herbivore
    subclass-of animal, NOT carnivore
    slot-constraint eats
        value-type plant
```

# Ontologies and Description Logics

- ◆ OWL is based on a fragment of first-order predicate logic (FOL)
- ◆ Description Logics (DLs) is a subclasses of FOL. It has
  - ◆ only unary and binary predicates
  - ◆ function-free
  - ◆ quantification allowed only in restricted form
  - ◆ (variable-free syntax)
  - ◆ decidable reasoning
- ◆ DLs are one of the most prominent languages for Knowledge Representation

# The proof/ rule layer

- ◆ rule: informal notion
- ◆ rules are used to perform inference over ontologies
- ◆ rules are used as a tool for capturing further knowledge (not expressible in OWL ontologies)

# Ontologies: the role of logic

- ◆ ontology = logical theory
- ◆ why?
  - ◆ Declarative
  - ◆ formal semantics
  - ◆ reasoning (sound and complete inference techniques)
- ◆ well-established correspondence between conceptual modeling formalisms and logic

# Rule-based formalisms

- ◆ Prolog
- ◆ Logic programming
- ◆ Constraint (logic) programming
- ◆ Production rules
- ◆ Datalog
- ◆ • ...
- ◆ RIF (Rule Interchange Format): W3C recommendation (2010)

# The Trust layer

- ◆ It is the top layer of the Semantic web
- ◆ It provides support for provenance/ trust
- ◆ To judge the value of the data item and how to use the data appropriately, one has to know the origin of the data.
- ◆ provenance: it answers the following main questions
  - ◆ which pieces of data were chosen and composed together?
  - ◆ where do they come from? who created them?
  - ◆ or which inference rules were used to create implicit statements?
  - ◆ can I trust this information? trust is related to data source, authorship, certainty, ...
- ◆ It is largely unexplored issue and there is no standardization effort to handle it.

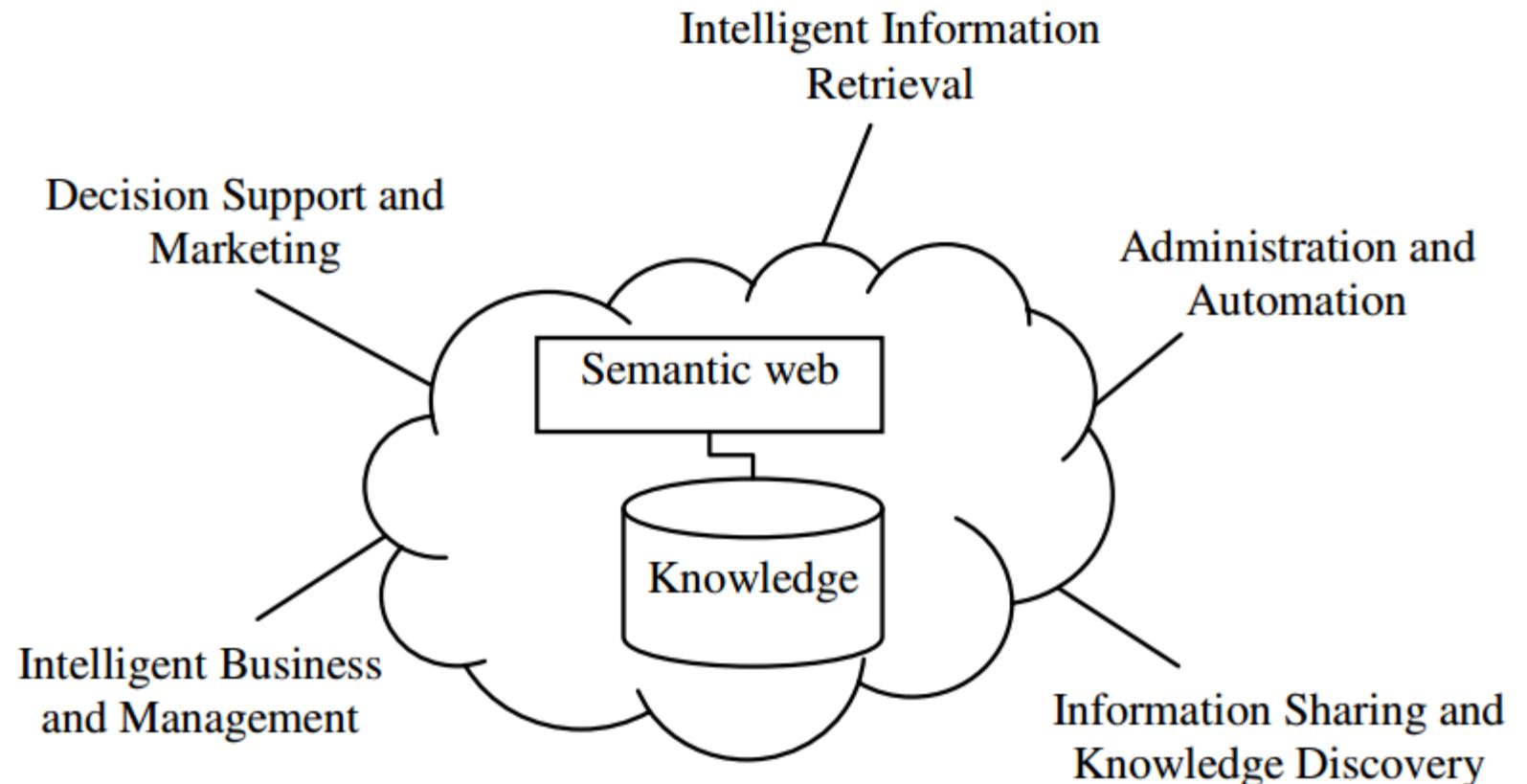


Inference in the Open Provenance Model

# **Semantic Web Applications**

# Applications

- ◆ Intelligent Business and Management (e.g. e-commerce – matchmaking for e-business)



# Semantic Web Applications

- ◆ **Main categories**
  - ◆ Search (also browsing, personalization) e.g. Taalee search engine
  - ◆ Integration (also interoperability)
  - ◆ Analysis (also visualization)
- ◆ drug discovery process [Borkum & Frey, Journal of Cheminformatics, 2014:  
**Usage and applications of Semantic Web techniques and technologies to support chemistry research**]
- ◆ Integrating applications
- ◆ Model checking, consistency and collaboration for Engineering of products e.g. [Stefan Feldmann, Konstantin Kernschmidt, Birgit Vogel-Heuser, Springer, 2016: , Applications of Semantic Web Technologies for the Engineering of Automated Production Systems—Three Use Cases]

## **2. Knowledge Representation and Reasoning**

# Knowledge Representation KR

- ◆ KR is the field of study within AI concerned with using formal symbols to represent a collection of propositions believed by some putative agent. (...)
- ◆ There could be infinite number of propositions believed, only a finite number of which are ever represented.
- ◆ Reasoning is the formal manipulation of **the symbols** representing a collection of believed **propositions** to produce representations of the new ones.
- ◆ E.g. “John loves Mary” and “Mary is coming to the party,”  
**Reasoning:** “Someone John loves is coming to the party.” (**logical inference**)
- ◆ What is Knowledge? How can it be ascribed?
- ◆ Knowledge is what makes possible rational behavior: if a **system** has knowledge that one of its actions will lead to one of its goals, then it will select that action.  
rational behavior =  $f(\text{knowledge})$

# Knowledge Representation KR

*Randall Davis, Howard Shrobe, Peter Szolovits, MIT*

- ◆ A knowledge representation is most fundamentally a **surrogate**, a **substitute** for the thing itself, used to enable an entity to determine consequences by reasoning about the world.
- ◆ Two questions:
  - 1) what is its intended identify. There must be some form of correspondences specified between the surrogate and tis intended referent in the word (i.e. the semantic representation)
  - 2) fidelity: how close is the surrogate to the real thing? What attributes fo the original does it capture and make it explicit and which are omitted?

# Knowledge Representation KR

- ◆ It is a set of ontological commitments, i.e., an answer to the question: *In what terms should I think about the world?*
- ◆ Any representations are imperfect approximations to reality: hence attends some things and ignoring others.
- ◆ Commitment determines brings some part of the world into focus at the expense of blurring others
- ◆ Commitment to a particular view of the world thus starts with the choice of a representation technology, and accumulates as subsequent choices are made about how to see the world in those terms.

- ◆ A KR is not a data structure.
- ◆ (rules, frames, semantic net ... are representations but not data structure)
- ◆ For instance, semantic net is a representation while a graph is a data structure.

- ◆ It is a **fragmentary theory of intelligent reasoning**, expressed in terms of **three components**:
  - the representation's fundamental **conception of intelligent reasoning**;
  - the **set of inferences** the representation *sanctions* (*indicates what can be inferred*);
  - the set of inferences it *recommends* (*concerned with what should be inferred*).
- ◆ It is a **medium of human expression**, i.e., a language in which we say things about the world
  - How well does the representation function as a medium of expression? How general is it? How precise? Does it provide expressive adequacy? etc.

- ◆ If **A** represents **B**, then **A** stands for **B** and is usually more easily accessible than **B**.
- ◆ We are interested in **symbolic representations**
- ◆ *Symbolic representations of propositions or statements* that are believed by some agent.

# Reasoning

- ◆ Reasoning is the use of symbolic representations of some statements in order to derive new ones.
- ◆ While statements are abstract objects, their representations are concrete objects and can be easily manipulated.
- ◆ Reasoning can be as easy as *mechanical symbol manipulation*.
  - ◆ If we charge high fees for university, only the rich will enroll.
  - ◆ We charge high fees for university. Therefore, only the rich will enroll.
- ◆ Reasoning should scale well: we need efficient reasoning algorithms.

# Knowledge Representation KR ...

- ◆ The system is an agent that has knowledge and on the basis of this knowledge it chooses to execute some actions to achieve goals.
- ◆ Knowledge is evaluated in terms of (competent) **rational behavior** that is based on the reasoning capabilities of the system.
- ◆ Commonsense knowledge : **knowledge about the everyday physical world: about objects, shape, space, movement, substances, time, etc.**
- ◆ “**The common knowledge about the world that is possessed by every schoolchild and the methods for making obvious inferences from this knowledge are called common sense.**” (E. Davis, **Representations of Commonsense Knowledge**, 1990).

# Knowledge representation (KR) and reasoning

- ◆ A specific approach to build a rational (intelligent) system, based on the following two ideas.
- ◆ **Declarative representation:** An explicit representation of the knowledge of the world, an explicit model of the world;  
**e.g. facts about the world, laws that govern the environment and its evolution or govern the effects of actions in the environment.**
- ◆ **Reasoning:** Rules that allow the system to reason on the model of the environment it has (the explicit representation of the knowledge of the environment it has) to derive ‘new’ knowledge that can influence its behavior to achieve its goals.

# Knowledge based systems

- ◆ A knowledge based system has two main components:
  - 1) **a formal system** (representation language and reasoning mechanism);
  - 2) **a knowledge base** (representation of the system's knowledge about the world).

# Formal system

1. **A representation language:** atomic symbols + grammar for complex formulas apt to code the information one wants to represent.
2. **A semantics of this language:** symbols denote something.
3. **General rules** (effective procedures) that (syntactically) manipulate formulas of the language according to their semantics.
4. **The rules allow to infer new valid** (with respect to the chosen semantics) formulas.
  - truth preserving inference.
  - Logical knowledge. Tautologies (truths starting from an empty set of hypotheses), knowledge about logical connectives:
    - Bopo is a man or Bopo it is not a man .
    - $\text{Man}(\text{Bopo}) \vee \neg \text{Man}(\text{Bopo})$  vs  $P(a) \vee \neg P(a)$  vs  $p \vee \neg p$

# Knowledge base

- ◆ A set of propositions written in the chosen language.
- ◆ **Factual knowledge.** Knowledge about specific individuals in the domain of quantification:
  - *Socrates is a man.*
  - *Socrates is taller than Plato.*
- ◆ **Terminological knowledge.** Knowledge about general facts concerning concepts, general rules that govern the world:
  - *Human beings are mortal.*
  - *Human beings are either male or female.*
  - *If something moves (in the atmosphere) it warms up.*

# Knowledge base

Two generic functions:

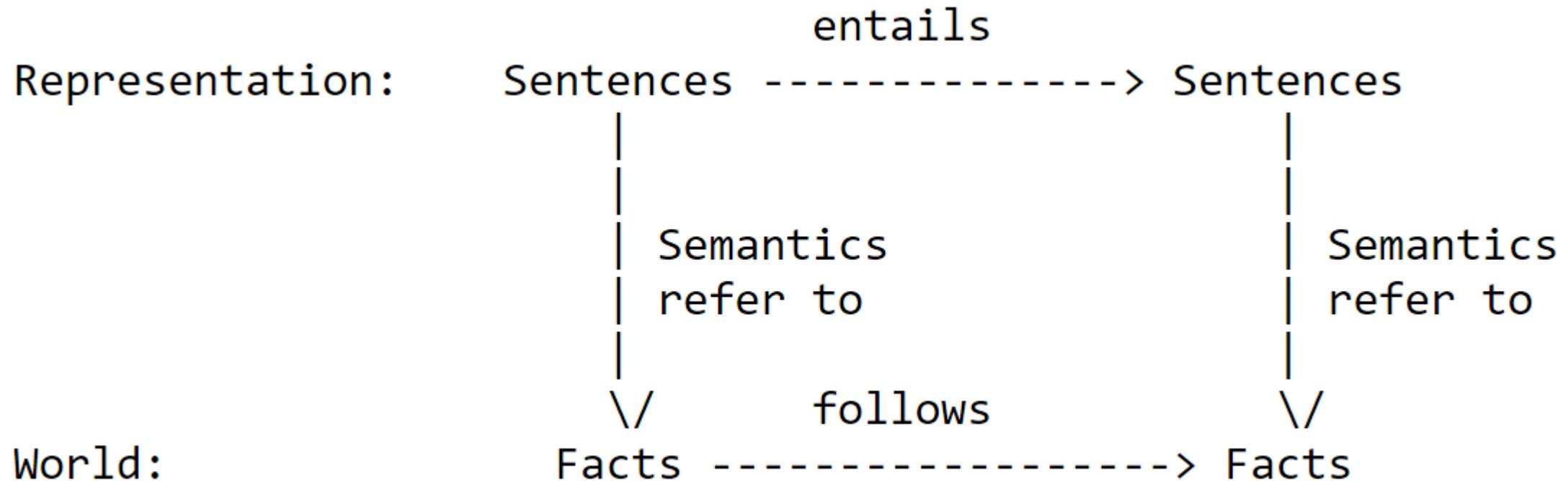
- ◆ TELL - add new sentences (facts) to the KB
  - ◆ “Tell it what it needs to know”
- ◆ ASK - query what is known from the KB
  - ◆ “Ask what to do next”

# Why logic?

- ◆ Logic is a formal system in which the formulas or sentences have true or false values.
- ◆ A logic includes:
  - Syntax: Specifies the symbols in the language and how they can be combined to form sentences. Hence facts about the world are represented as sentences in logic
  - Semantics: Specifies what facts in the world a sentence refers to. Hence, also specifies how you assign a truth value to a sentence based on its meaning in the world. A fact is a claim about the world, and may be true or false.
  - Inference Procedure: Mechanical method for computing (deriving) new (true) sentences from existing sentences

# Why logic?

- Logic is a reliable tool to represent and reason on explicit knowledge.



# Why logic?

- ◆ Propositions in natural language (e.g. English) are most often ambiguous and context dependent. They can be understood differently and this leads to different conclusions, even incompatible conclusions.
- ◆ To understand when conclusions are incompatible and to avoid them, we must find a language that has no ambiguity and is context independent.
- ◆ Logic defines its own languages (formal languages) and explicitly establishes how the semantics (meaning) of the expressions can be characterized.

# What is logic?

A formal system

- ◆ Language
  - ◆ Syntax: atomic symbols + a grammar to build formulas out of them. It provides the basic elements of the language and the grammatical rules to build (complex) sentences. It uses set of symbols – vocabulary, set of grammatical rules to combine the symbols
  - ◆ Semantics: interpretation of the formulas (i.e. give meaning to sentences). Introduce the notion of structure
- ◆ Reasoning rules: syntactic procedures that manipulate formulas to infer new valid, according to the chosen semantics, formulas.
  - ◆ Different forms of reasoning can be formalized
    - Deductive Reasoning: the conclusion is a “consequence” of the premises (what is given or known)
    - Inductive Reasoning: the conclusion is a “generalization”
    - Abductive Reasoning: the conclusion is a possible “explanation”
    - Default Reasoning: the conclusion is a “typical” consequence; it is defeasible
    - Analogical Reasoning: the conclusion is obtained by analogy with another situation/ problem

# Deductive reasoning

- ◆ *Very important*
- ◆ Assume we have a set  $\Phi = \{p_1, p_2, \dots, p_n, \dots\}$  of propositions (statements) that are known or important for a given phenomenon we aim to study. The logician wants to know which propositions follow from  $\Phi$  using a deductive reasoning method.
- ◆ **Idea:** proposition **p follows from  $\Phi$** , if there exists a (finite) chain of inferences that starts with finitely many propositions in  $\Phi$  (thus, the premises are all in  $\Phi$ ) and ends with p.
- ◆ A correct chain of inferences that starts with propositions in  $\Phi$  and ends with p is called **a proof of p from  $\Phi$** . If there is a proof of p from  $\Phi$ , we say that p is derived from (is deduced from, follows from, is a consequence of)  $\Phi$ .
- ◆ Example of correct chain: “***if I think, then I exist***” and from  
“***I think***”, it follows that “***I exist***”

# Deductive reasoning

There are two ways to make correct inferences:

- ◆ **Deduction or Syntactic inference** (Proof Theory). The derived proposition is obtained by applying some accepted syntactic rules to propositions in  $\Phi$ . (which syntactic rules?)
- ◆ **Entailment or Semantic inference** (Model Theory). The derived proposition is true whenever the premises are true. (what's true?)

# Propositional logic/ Boolean logic

- ◆ A simple language that is useful for showing key ideas and definitions.
- ◆ A **sentence** (also called a formula or well-formed formula or wff) is defined as:
  - 1) A symbol
  - 2) If S is a sentence, then  $\sim S$  is a sentence, where " $\sim$ " is the "not" logical operator
  - 3) If S and T are sentences, then  $(S \vee T)$ ,  $(S \wedge T)$ ,  $(S \Rightarrow T)$ , and  $(S \Leftrightarrow T)$  are sentences, where the four logical connectives correspond to "or," "and," "implies," and "if and only if," respectively
  - 4) A finite number of applications of (1)-(3)
    - For example, P means "It is hot", Q means "It is humid", R means "It is raining"
    - $(P \wedge Q) \Rightarrow R$  (here meaning "If it is hot and humid, then it is raining")
    - $Q \Rightarrow P$  (here meaning "If it is humid, then it is hot")

# Propositional logic/ Boolean logic

- ◆ Rule for the generation of all the wffs (inductive definition):
  - Each propositional letter is a wff
  - If P is a wff then  $(\neg P)$  is a wff
  - If P and Q are wffs then  $(P \wedge Q), (P \vee Q), (P \rightarrow Q), (P \leftrightarrow Q)$  are wffs
  - Nothing else is a wff
- ◆ Classify the formula as wffs or not
  - $((\neg A) \vee B) \rightarrow C$
  - $((A \wedge B) \leftrightarrow \neg(C \vee (\neg A)))$
  - $((\neg A) \wedge A)$
  - $((A \vee B) \neg C)$
  - $\neg A \vee B \rightarrow C$  Not
  - $A \rightarrow B \rightarrow C$  Not

# Propositional logic – Semantics

- ◆ Semantics is defined by a Valuation function  $V$  whose domain is the set of wffs and whose range is {false, true}.

For non-atomic wffs, the effect of  $V$  is defined by induction:

$$V(\neg\varphi) = \text{true iff } V(\varphi) = \text{false}$$

$$V(\varphi \wedge \psi) = \text{true iff } V(\varphi) = \text{true and } V(\psi) = \text{true}$$

$$V(\varphi \vee \psi) = \text{true iff } V(\varphi) = \text{true or } V(\psi) = \text{true}$$

$$V(\varphi \rightarrow \psi) = \text{true iff } V(\varphi) = \text{false or } V(\psi) = \text{true}$$

$$V(\varphi \leftrightarrow \psi) = \text{true iff } V(\varphi) = V(\psi)$$

- ◆ This doesn't tell us how to determine the truth of atomic wffs. It is arbitrary; each function  $V$  characterizes a different model or world.
- ◆ Truth tables provide an exhaustive list of all possible models for the truth of a set of propositions.

# Propositional logic: Interpretations

$P$	$Q$	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
False	False	True	False	False	True	True
False	True	True	False	True	True	False
True	False	False	False	True	False	False
True	True	False	True	True	True	True

A **model** is an interpretation (i.e., an assignment of truth values to symbols) of a set of sentences such that each sentence is True.

A model is just a formal mathematical structure that "stands in" for the world.

# Equivalence, Validity, Satisfiability

Two sentences are logically equivalent iff true in same models:

$$\alpha \equiv \beta \text{ if and only if } \alpha \models \beta \text{ and } \beta \models \alpha$$

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \text{ commutativity of } \wedge$$

$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \text{ commutativity of } \vee$$

$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \text{ associativity of } \wedge$$

$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \text{ associativity of } \vee$$

$$\neg(\neg\alpha) \equiv \alpha \text{ double-negation elimination}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha) \text{ contraposition}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta) \text{ implication elimination}$$

$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \text{ biconditional elimination}$$

$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \text{ de Morgan}$$

$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \text{ de Morgan}$$

$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \text{ distributivity of } \wedge \text{ over } \vee$$

$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \text{ distributivity of } \vee \text{ over } \wedge$$

# Logical relations: Equivalence, Validity, Satisfiability

- ◆ A sentence is valid (Tautology) if it is true in all models or interpretations/
  - e.g. True,  $A \vee \neg A$ ,  $A \Rightarrow A$ ,  $(A \wedge (A \Rightarrow B)) \Rightarrow B$
- ◆ Equivalence:  $\varphi$  and  $\psi$  are logically equivalent iff for every valuation function  $V$  (i.e., every model),  $V(\varphi) = V(\psi)$ . Notation:  $\varphi \equiv \psi$
- ◆ Validity is connected to inference via the Deduction Theorem
  - $KB \vdash \alpha$  iff  $(KB \Rightarrow \alpha)$  is valid
- ◆ A sentence is satisfiable if it is True in some model
  - e.g.  $A \vee B$ , C
- ◆ A sentence is unsatisfiable if it is True in no models (i.e. false under all interpretations)
  - e.g.  $A \wedge \neg A$
- ◆ Satisfiability is connected to inference via the following
  - $KB \models \alpha$  (all models of  $KB$  are models of  $\alpha$  as well) iff  $(KB \wedge \neg \alpha)$  is unsatisfiable (proof by contradiction)

# Logical relations: Equivalence, Validity, Satisfiability

---

- ◆ Check the validity of the following formula

- $A \wedge A \equiv A$
- $A \vee B \equiv B \vee A$
- $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$
- $\neg(A \wedge B) \equiv \neg A \vee \neg B$
- $A \rightarrow B \equiv \neg A \vee B$
- $A \rightarrow B \equiv \neg B \rightarrow \neg A$
- $A \rightarrow (B \rightarrow C) \equiv A \wedge B \rightarrow C$

# Question

- ◆  $P = \text{"It is hot"}, Q = \text{"It is humid"}, R = \text{"It is raining"}$
- ◆ If  $\text{KB} = (((P \wedge Q) \Rightarrow R) \wedge (Q \Rightarrow P) \wedge Q)$ , then  $\text{KB} \Rightarrow ? R$

P	Q	R	$(P \wedge Q) \Rightarrow R$	$Q \Rightarrow P$	$(Q \Rightarrow P) \wedge Q$	KB	$\text{KB} \Rightarrow R$	$\text{KB} \Rightarrow Q$
T	T	T	T	T	T	T	T	T
T	T	F	F	T	T	F	T	T
T	F	T	T	T	F	F	T	T
T	F	F	T	T	F	F	T	T
F	T	T	T	F	F	F	T	T
F	T	F	T	F	F	F	T	T
F	F	T	T	T	F	F	T	T
F	F	F	T	T	F	F	T	T

there is only one model of KB, when P, Q, and R are all True. And in this case R is also True, so R is entailed by KB.

# First-Order Logical Language/ Predicate Logic

- ◆ **Limitation of propositional logic** : it applies only to atomic propositions. There is no way to talk about properties that apply to categories of objects, or about relationships between those properties.
- ◆ Unable to account for classical syllogisms such as:
  - P: Every man is mortal,
  - Q: Socrates is a man,
  - therefore Socrates is mortal.
  - all dogs are animals;
  - all animals have four legs;
  - therefore all dogs have four legs
- ◆ Need to analyze the internal structure of propositions
- ◆ Need to refer to entities, with terms, and to their properties and relations, with predicates

# First-Order Logical Language/ Predicate Logic

- ◆ Unable to account for classical syllogisms such as:
  - ◆ **All women like to shop.**
  - ◆ **John likes to shop. Then John is a woman.**
- ◆ **If X smokes everyday, x increases risk of lung cancer. X does not smoke everyday. Then x has no increased risk of lung cancer. But what if X's husband is a chain smoker.**
- ◆ **All birds lay eggs.**
- ◆ **A chicken lays eggs. Then a chicken is a bird.**
- ◆ **All dogs bark. The basenji, mute and cannot bark. A basenji is not a dog.**

# First-Order Logical Language/ Predicate Logic

- ◆ **Limitation of propositional logic :** it applies only to atomic propositions. There is no way to talk about properties that apply to categories of objects, or about relationships between those properties.
- ◆ Need to analyze the internal structure of propositions
- ◆ Need to refer to entities, with terms, and to their properties and relations, with predicates

# First-Order Logical Language/ Predicate Logic

- ◆ Instead of a vocabulary of letters for atomic propositions, we have:
  - ◆ Predicate (represents a property of or relations between terms that can be true or false) constants: P1, P2, P3... (or Loves, Mortal, Human...) often a special binary predicate is distinguished “=” (identity)
  - ◆ Terms, (are simply names for objects) to serve as arguments of predicates (recursive definition because of functions):
    - Individual constants: a, b, c... (= functions of arity 0)
    - Variables: x, y, z...
    - Function constants with arity n: f1, f2,... applied to n terms
  - ◆ Atomic propositions are then of the form
    - $P(t_1, \dots, t_n)$  where P is an n-ary predicate and  $t_i$  a term
    - e.g. Human(s) is an atomic proposition with a unary predicate,
    - Loves(j, m) with a binary predicate
  - ◆ Quantifiers are used to bind variables

# First-Order Logical Language/ Predicate Logic

- ◆ Predicate logic includes a richer ontology:
  - ◆ objects (terms)
  - ◆ properties (unary predicates on terms)
  - ◆ relations (n-ary predicates on terms)
  - ◆ functions (mappings from terms to other terms)
- ◆ Allows more flexible and compact representation of knowledge
  - ◆ E.g.  $\text{Move}(x, y, z)$  for person x moved from location y to z.

# Syntax for First-Order Logic

**Sentence** → AtomicSentence | Sentence Connective Sentence | Quantifier Variable Sentence |  $\neg$ Sentence | (Sentence)

**AtomicSentence** → Predicate(Term, Term, ...) | Term=Term

**Term** → Function(Term, Term, ...) | Constant | Variable

**Connective** →  $\vee$  |  $\wedge$  |  $\Rightarrow$  |  $\Leftrightarrow$

**Quanitfier** →  $\exists$  |  $\forall$

**Constant** → A | John | Car1

**Variable** → x | y | z | ...

**Predicate** → Brother | Owns | ...

**Function** → father-of | plus | ...

# First-Order Logical Language/ Predicate Logic

- ◆ Quantifiers are used to bind variables
  - $\forall$ , the universal quantifier “for all”
  - $\exists$ , the existential quantifier “there is at least one”
  - If  $\varphi$  is a wff and  $x$  a variable, then  $\forall x\varphi$  and  $\exists x\varphi$  are wffs
- ◆ Every man is mortal:  $\forall x(\text{Man}(x) \rightarrow \text{Mortal}(x))$
- ◆ “There are white cats” (there is at least one white cat)  
 $\exists x(\text{Cat}(x) \wedge \text{White}(x))$  valid  
 $\exists x(\text{Cat}(x) \rightarrow \text{White}(x))$  not valid
- ◆ • “Some chairs are broken” (there is at least one broken chair)  
 $\exists x(\text{Chair}(x) \wedge \text{Broken}(x))$   
 $\exists x(\text{Chair}(x) \rightarrow \text{Broken}(x))$  not valid

# First-Order Logical Language/ Predicate Logic

## For all ... (inherently shows implication )

- ◆ “Any cook knows how to cook pizza” (also with every / each / a)  
“All the cooks know how to cook pizza” (also with bare plural)  
“If somebody is a cook, then s/he knows how to cook pizza”  
$$\forall x(\text{Cook}(x) \rightarrow \text{KnowsCooking}(x, \text{pizza}))$$
- ◆ “Everybody is a cook and knows how to cook pizza”  
$$\forall x(\text{Cook}(x) \wedge \text{KnowsCooking}(x, \text{pizza}))$$
- ◆ “Chianti is the only good wine, if any” that is,  
“if a wine is good, then it is Chianti”  
$$\forall x(\text{Wine}(x) \wedge \text{Good}(x) \rightarrow \text{Chianti}(x))$$

# First-Order Logical Language/ Predicate Logic

## Represent the expression using FOL

- “Whoever owns a dog loves animals”

$$\forall x(\exists y(\text{Dog}(y) \wedge \text{Own}(x, y)) \rightarrow \text{LoveAnimals}(x))$$

# Grammar of First-Order Logical Language/ Predicate Logic $L$

## Terms

### Definition of grammar of FOL language $L$

- ◆ The set Term of terms of  $L$ , also called  $T(V)$  with  $V$  the vocabulary, is the set generated by the following rules:
  1. Every individual constant and individual variable is in Term
  2. If  $t_1, \dots, t_n \in \text{Term}$  and  $f$  is a function constant with arity  $n$ , then  $f(t_1, \dots, t_n) \in \text{Term}$ .
  3. Nothing else is in Term.
- ◆ Examples:  $x, 0, \text{George}, \text{FatherOf}(\text{George}), \text{Prod}(x, y)$  where
  - $x$  and  $y$  are variables of  $L$ ;
  - $0, \text{George}, 5, 7$  are constants of  $L$ ;
  - $\text{FatherOf}$  and  $\text{Prod}$  are functions of  $L$ .

# Grammar of First-Order Logical Language/ Predicate Logic $L$

The set Atom of atoms or atomic formulas is generated by the following rules:

1  $T, \perp \in \text{Atom}$

2 If  $t_1, t_2 \in \text{Term}$  then  $t_1 = t_2 \in \text{Atom}$

3 If  $t_1, \dots, t_n \in \text{Term}$  and  $P$  is a predicate constant with arity  $n$ , then  $P(t_1, \dots, t_n) \in \text{Atom}$

- Examples:  $x = y, P(x), Q(x, c), \text{FatherOf}(\text{FirstSonOf}(x)) = x$

John is happy:  $H(j)$

John loves Mary:  $L(j,m)$

John introduced Mary to Sue:  $I(j,m,s)$

John is Bill:  $j=b$

# Grammar of First-Order Logical Language/ Predicate Logic $L$

4. If  $\varphi$  is a formula and  $x$  is a variable, then  $\forall x\varphi$ , and  $\exists x\varphi$  are formulas too.

- **Examples:**

Everyone is happy:  $\forall x H(x)$

Someone loves John:  $\exists x L(x, j)$

Everyone loves someone:  $\forall x \exists y L(x, y)$ ,  $\exists y \forall x L(x, y)$

5 If  $\varphi$  is a formula, then  $\neg\varphi$  is a formula.

6 If  $\varphi$  and  $\psi$  are formulas, then  $(\varphi \wedge \psi)$ ,  $(\varphi \vee \psi)$ ,  $(\varphi \rightarrow \psi)$ , and  $(\varphi \leftrightarrow \psi)$  are formulas too

7. Nothing else is in Atom

# Order of precedence for connections

0  $\forall, \exists$

1  $\neg$

2  $\vee$  and  $\wedge$

3  $\rightarrow$  and  $\leftrightarrow$

$\forall x\varphi \rightarrow \psi$  is the same as  $((\forall x\varphi) \rightarrow \psi)$

$\exists y\forall z\varphi \wedge \neg\exists x\psi$  is the same as  $((\exists y(\forall z\varphi)) \wedge (\neg(\exists x\psi)))$

We write  $\forall xy$  instead of  $\forall x\forall y$ , and  $\exists xy$  for  $\exists x\exists y$ .

# Order of precedence for connections

Understand the difference and similarity between the following formula

$\forall x(\exists y(\text{Loves}(x, y)))$  : everybody loves somebody, i.e. everyone has someone whom they love.

$\forall x(\exists y(\text{Loves}(y, x)))$ : there is someone who is loved by everyone in the universe

$\exists x(\forall y(\text{Loves}(x, y)))$ : there is someone who loves everyone in the universe

$\exists x(\forall y(\text{Loves}(y, x)))$ : everybody loves somebody

# **Knowledge Representation on the Web: The Semantic Web vision**

# Limitations of the old Web

- ◆ **Resource content accessible to humans, not to machines**
  - humans must use their own (usually linguistic) skills to understand the resource content
  - humans must use their own **strategies** to browse, surf, and extract information
  - humans must use their own **knowledge to aggregate** information from several resources
  - humans must use their own **reasoning capacities** to deduce new information from these
  - only humans can fully interpret the content within the resources
- ◆ **What is missing is semantics for machines**

# Endowing machines with semantic capacities

- ◆ Most web content appear to us humans through texts in natural language
- ◆ Are Natural Language Processing (NLP) tools enough?
- ◆ No: on-the-fly natural language understanding still too costly and very limited
- ◆ NL is highly expressive (higher order) and **highly ambiguous**
- ◆ Add metadata around the human-directed content, i.e., content specifically intended for machines, as explicit and unambiguous as possible
- ◆ Add structured data within the content, for both humans and machines (e.g. infoboxes in Wikipedia)

## Addis Ababa University

From Wikipedia, the free encyclopedia

Coordinates: 9°2'48"N 38°45'33"E

Addis Ababa University (Amharic: አዲስ አበባ ዘመንናት<sup>2</sup>) is a state university in Addis Ababa, the capital of Ethiopia. Originally called the University College of Addis Ababa at its establishment in 1950, it was later renamed Haile Selassie I University in 1962 after the Ethiopian Emperor Haile Selassie I. The institution received its current name in 1975.

### Contents [hide]

- 1 History
- 2 Campuses and Colleges
- 3 Notable alumni
- 4 References
- 5 Further reading
- 6 External links

### History [edit]



Front entrance to Addis Ababa

Addis Ababa University was founded as a two-year college in 1950 by a Canadian Jesuit, Dr Lucien Matte, S.J., at the request of Haile Selassie. It began operations the following year. Over the following two years an affiliation with the University of London was developed. The writer and theorist Richard Cummings served as a member of the Faculty of Law in the 1960s.

As part of their sweeping changes, the Derg ordered Addis Ababa University temporarily closed on March 4, 1975 and dispatched its 50,000 students to the countryside to help build

### Addis Ababa University



Former names	University College of Addis Ababa (1950–1962) Haile Selassie I University (1962–1975)
Type	State university
Established	1950
President	Dr. Admasu Tsegaye
Students	48,673 (2013/14) <sup>[1]</sup>
Location	Addis Ababa, Ethiopia

# The Semantic Web or WWW 3.0

- ◆ Termed coined in 2001
- ◆ The Semantic Web – A new form of Web content that is meaningful to computers to unleash a revolution of new possibilities, Tim Berners-Lee, James Hendler and Ora Lassila, Scientific American, 2001
- ◆ The Semantic Web = a Web with a meaning
  - ◆ Syntax
  - ◆ Semantics

# The Semantic Web or WWW 3.0

- ◆ Syntax: new standards to allow content exchange between machines **RDF, RDFS, OWL** (a DL), **SPARQL**
- ◆ Semantics:
  - ◆ make sure all the machines “understand” this information in the same way,
  - ◆ make sure that using the same language they talk about the same things and facts and that they share the necessary knowledge to exploit these facts
    - ground their language to things and facts in the reality, or, rather,
    - relate language elements across sources: **Linked Open Data**
    - fix a vocabulary and constrain its interpretation in a formal framework to minimize misunderstandings and provide reasoning capacities:
    - **Knowledge Representation**, and more specifically,
    - **Ontologies and Description Logics**
- ◆ The Semantic Web not yet fully there, but in progress

### **3 Web Data Semantic**

# Introduction

# The Web (“old” version, 1.0 and 2.0)

## ◆ Resources

- distributed on the internet, interlinked, multi-platform
- dynamic, interactive, participative
- still mainly textual, but always more multimedia

## ◆ Tim Berners-Lee

- 1980 hypertext
- 1991 first browser
- 1994 foundation of the W3C, World Wide Web Consortium

## ◆ W3C

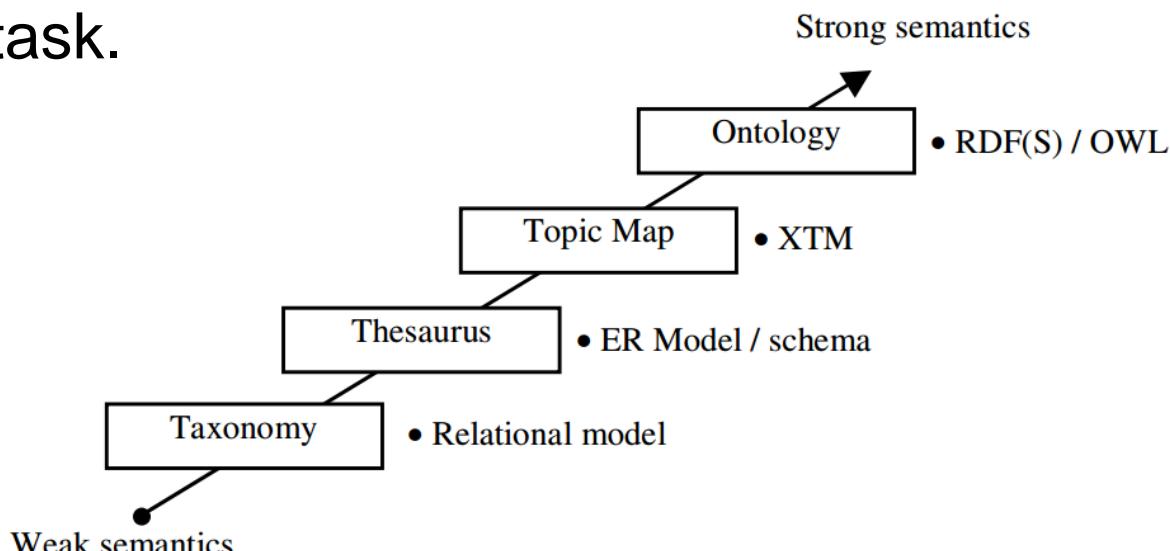
- The main international standards organization for the World Wide Web (beyond the internet protocol suite)
- HTTP, HTML, XML, . . .

# Semantic Web

- ◆ The semantic web is designed for not only providing web data for human uses, but also creating the data that the machines can process.
- ◆ The main vision of semantic web is to create machines-processable data and define how machines act to the data and make a web system become more intelligent.
- ◆ Reasons for semantic web
  - ◆ web information is overloaded
  - ◆ the amount of web data is too much for human consumption
    - ◆ Machines are needed a lots of information processing before it deliver to us.
    - ◆ This information processing such as information filtering, searching, and recommendations require high machine-intelligence

# Semantic Modeling

- ◆ Semantic modeling in information technologies refer to mapping or formalizing human knowledge to some kinds of language syntax
- ◆ Human knowledge are usually expressed in unstructured natural language which is very difficult for computer processing, therefore we need some structured language syntax to model the underlying “semantics” behind the natural language.
- ◆ The main idea of semantic modeling is to associate a term in a statement with a concept in the real world that the term refers to. Various technologies have been developed to handle the semantic modeling task.



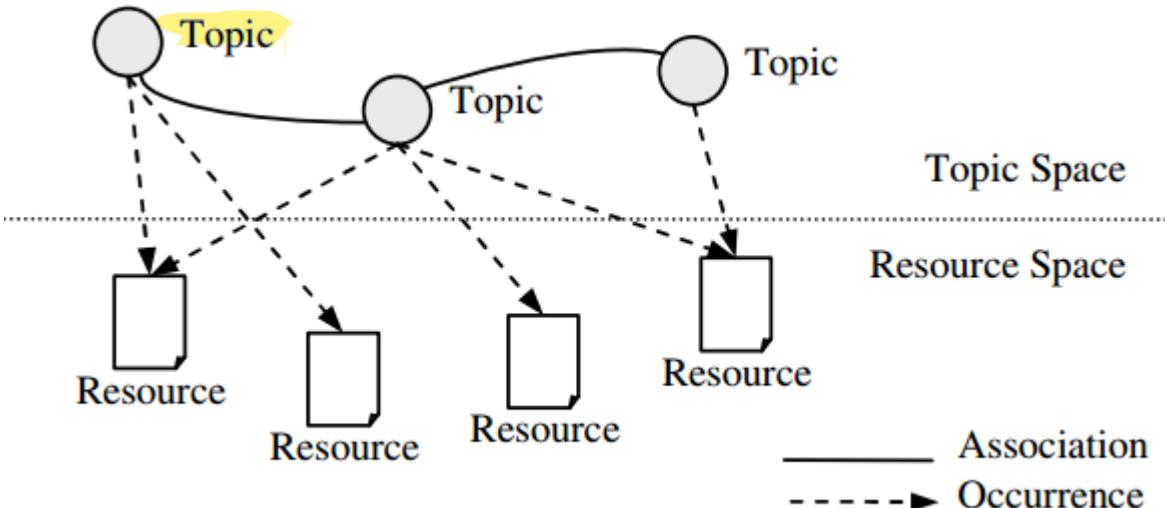
# Semantic Modeling

- ◆ **Taxonomy:** Taxonomy describes knowledge in hierarchical structure or in the semantics of the parent/child relationship or class and sub-class relation.
- ◆ **Thesaurus:** is “controlled vocabulary arranged in a known order and structured so that equivalence, homographic, hierarchical, and associative relationship among terms are displayed clearly and identified by standardized relationship indicators”. Therefore, it describes knowledge more than the taxonomy. WordNet is an example of thesaurus for English and HowNet for Chinese.

Relationship Type	Example
Equivalence	
Synonymy	“HK”/ “Hong Kong”
Homographic	
Homonym	“Mouse” (animal)/ “Mouse” (input device)
Hierarchical	
Hypernym	“Mouse” / “Mammal” (child-of)
Hyponym	“Mammal” / “Mouse” (parent-of)
Meronym	“Window” / “House” (part-of)
Holonym	“House” / “Window” (has-part)
Associative	
Cause-effect	“Accident”/ “Injury”
Attribute-host	“Color”/ “Cloth”
Material-product	“Grapes”/ “Wine”
Location-event	“Hospital”/ “Medical treatment”
Event-role	“Medical treatment” / “Patient”

# Semantic Modeling

- ◆ **Topic Maps:** is an ISO international standard for the representation of structured information model.
- ◆ It is used to represent the relationships between abstract concepts and information resources.



Topic Maps model composed of two separated spaces:

1. Topic space – consists of topics that represent concepts in the real world, and
2. Resource space – consists of resource files (web pages, text documents, multimedia files, etc.).

Topics related together by association connection to form concepts and it relates resource file by occurrence connection

# Semantic Modeling

- ◆ **Ontology:** the strongest semantic modelling technique
- ◆ In CS, an ontology precisely defines a term about a specific domain, represents an area of knowledge, and standardizes the meaning.

## Components of an Ontology

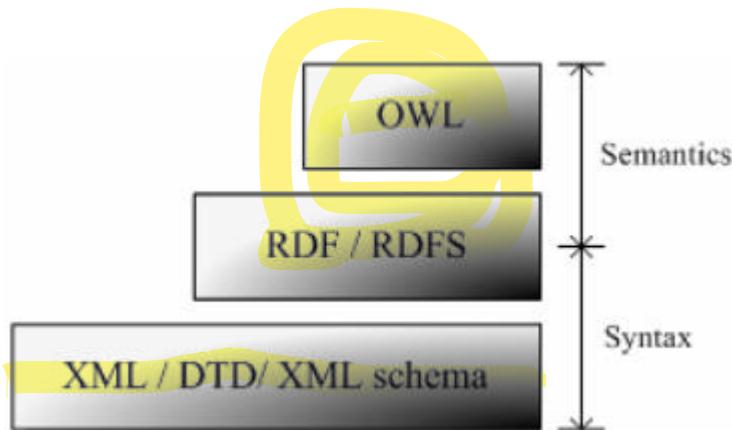
Component	Description
Classes	Set of concepts that describe objects
Instances	Particular things of objects
Relationships	Associations about meaning among those things
Properties	Property values of those things
Functions	Functions and processes describing those things
Constraints	Description logic and rules describing those things

According to Gruber (1993), “an ontology is an explicit specification of a conceptualization”.

Ontology usually consists of a set of vocabulary (concepts), taxonomy, relationships, properties, etc.

# *Ontology Languages for the Semantic Web*

- ◆ Ontology language is the markup language which can be used to model the **data semantic architecture** in the **data layer of the Semantic Web architectures**.
- ◆ The language available to markup the ontology and data semantic for semantic web includes XML, RDF, RDFS, DAML + OIL and OWL.



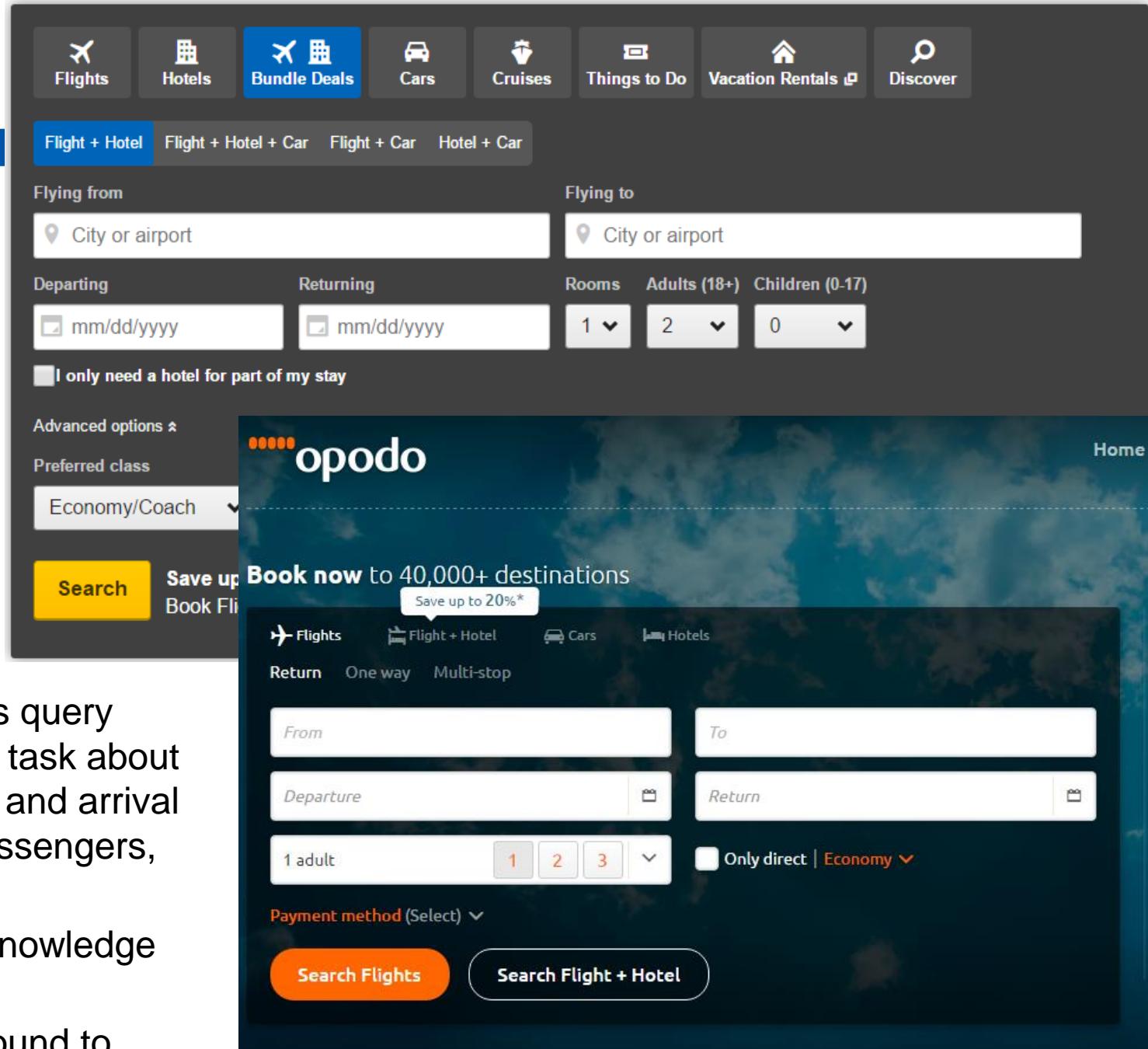
# **Assignment- development and implementation of web based**

- ◆ hotel price and quality comparison search engine G4
- ◆ cultural heritage search engine G1
- ◆ Amharic music search engine G3
- ◆ Clue based Music searching G2

**Ontology**

# Motivation for ontology

- ◆ Flight search engine (e.g. Expedia, Wegolo, Opodo...): A single interface to
  - ◆ query different airline booking systems,
  - ◆ compare the results,
  - ◆ sometimes aggregate them (mixed airlines complex routes), and perform booking.
- ◆ These websites and the various airlines sites query communicate without misunderstandings for task about what are: flights, airports, airlines, departure and arrival points, dates and times, seats, bookings, passengers, fares. . .
- ◆ Thus, they share a common language and knowledge about those “things”
- ◆ Generally, web services need a common ground to achieve interoperability



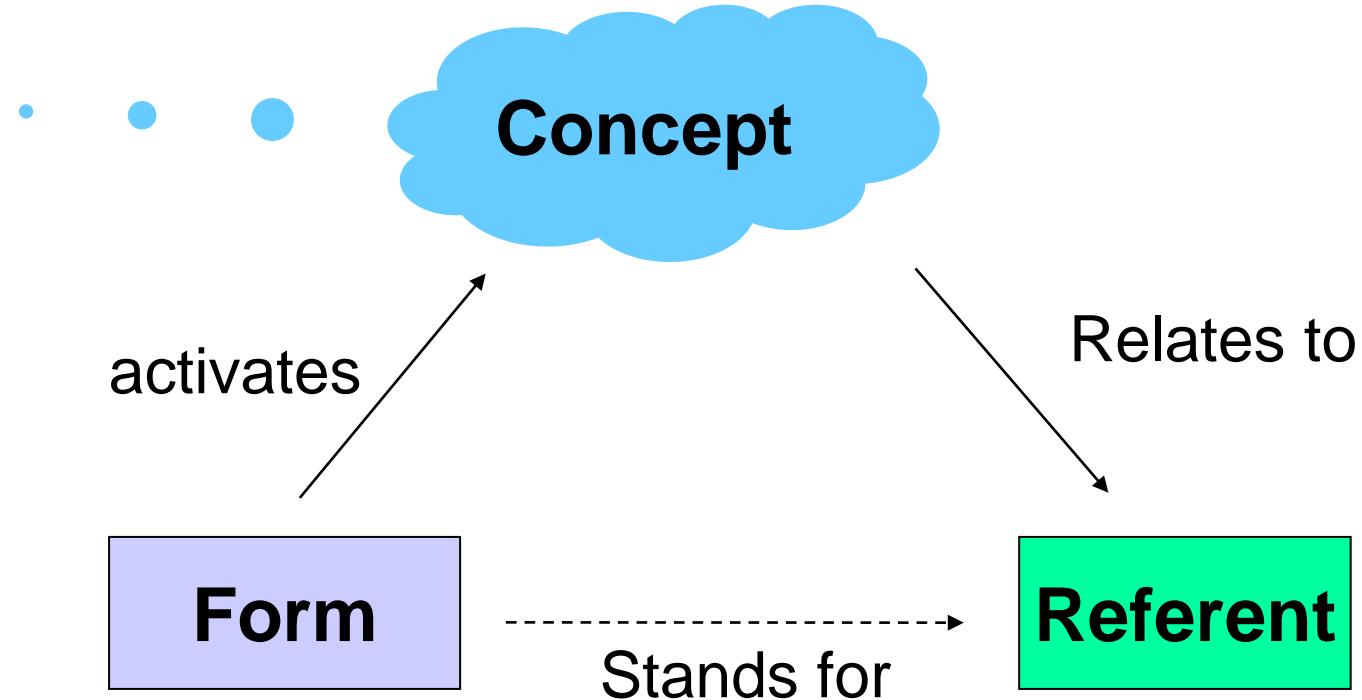
# What sort of content should machines share?

- ◆ A vocabulary
- ◆ Facts expressed with this vocabulary
- ◆ Not enough
  - ◆ Something is missing to make sure the machines all give the same / a similar meaning to the shared vocabulary, i.e., to constrain its interpretation: Knowledge about the domain, described through the vocabulary
- ◆ More specifically, an ontology expressed in a logical framework
- ◆ NB: Facts can be seen as being part or not of an ontology or a knowledge base, different views can be seen around. Here we will consider that facts are not part of the ontology, but will sometimes display instances on taxonomic graphs.

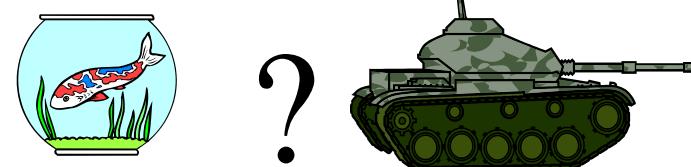
# Ontology: Origins and History Ontology in Philosophy

- A philosophical discipline - a branch of philosophy that deals with the nature and the organisation of reality
- Science of Being (Aristotle, Metaphysics, IV, 1)
- Tries to answer the questions:
  - *What characterizes being?*
  - *Eventually, what is being?*

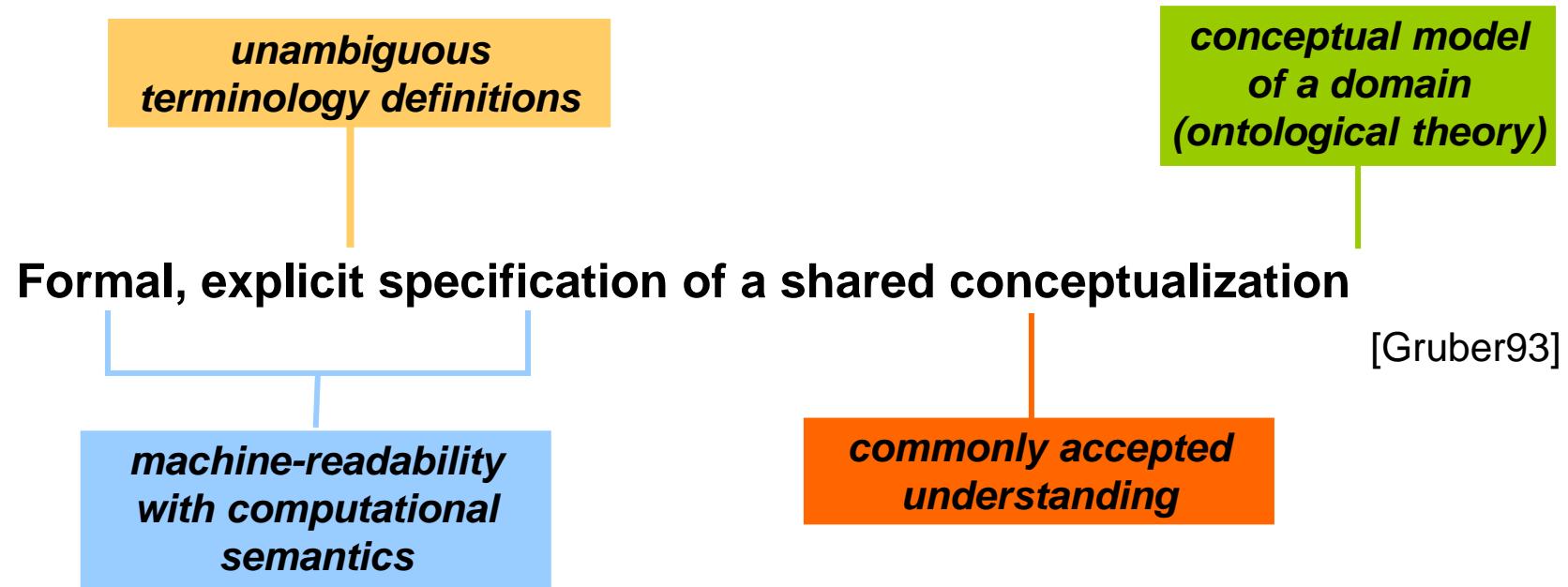
# Ontology in Linguistics



***“Tank”***  
[Ogden, Richards, 1923]



# Ontology Definition



# What is a conceptualization

- ◆ Formal structure of (a piece of) reality *as perceived and organized by an agent, independently of:*
  - the **vocabulary** used the actual occurrence of a specific **situation**
- ◆ Different situations involving the same objects, described by different vocabularies, may share the same conceptualization.

# Ontology in Computer Science

- ◆ An ontology is an engineering artifact:
  - It is constituted by a specific vocabulary used to describe a certain reality, plus
  - a set of explicit assumptions regarding the intended meaning of the vocabulary.
- ◆ Thus, an ontology describes a formal specification of a certain domain:
  - Shared understanding of a domain of interest
  - Formal and machine manipulable model of a domain of interest

$$\mathcal{O} := (C, R, f)$$

$R_i \in R$ ,  $C_i, C_j \in C$  - concept

  $C_i R_i C_j \in \mathcal{O}$   
relation.

# Structure of an Ontology

Ontologies typically have two distinct components:

## Names for important concepts in the domain

- Elephant is a concept whose members are a kind of animal
  - Herbivore is a concept whose members are exactly those animals who eat only plants or parts of plants
  - Adult\_Elephant is a concept whose members are exactly those elephants whose age is greater than 20 years
- 
- ◆ Background knowledge/constraints on the domain
    - Adult\_Elephants weigh at least 2,000 kg
    - All Elephants are either African\_Elephants or Indian\_Elephants
    - No individual can be both a Herbivore and a Carnivore

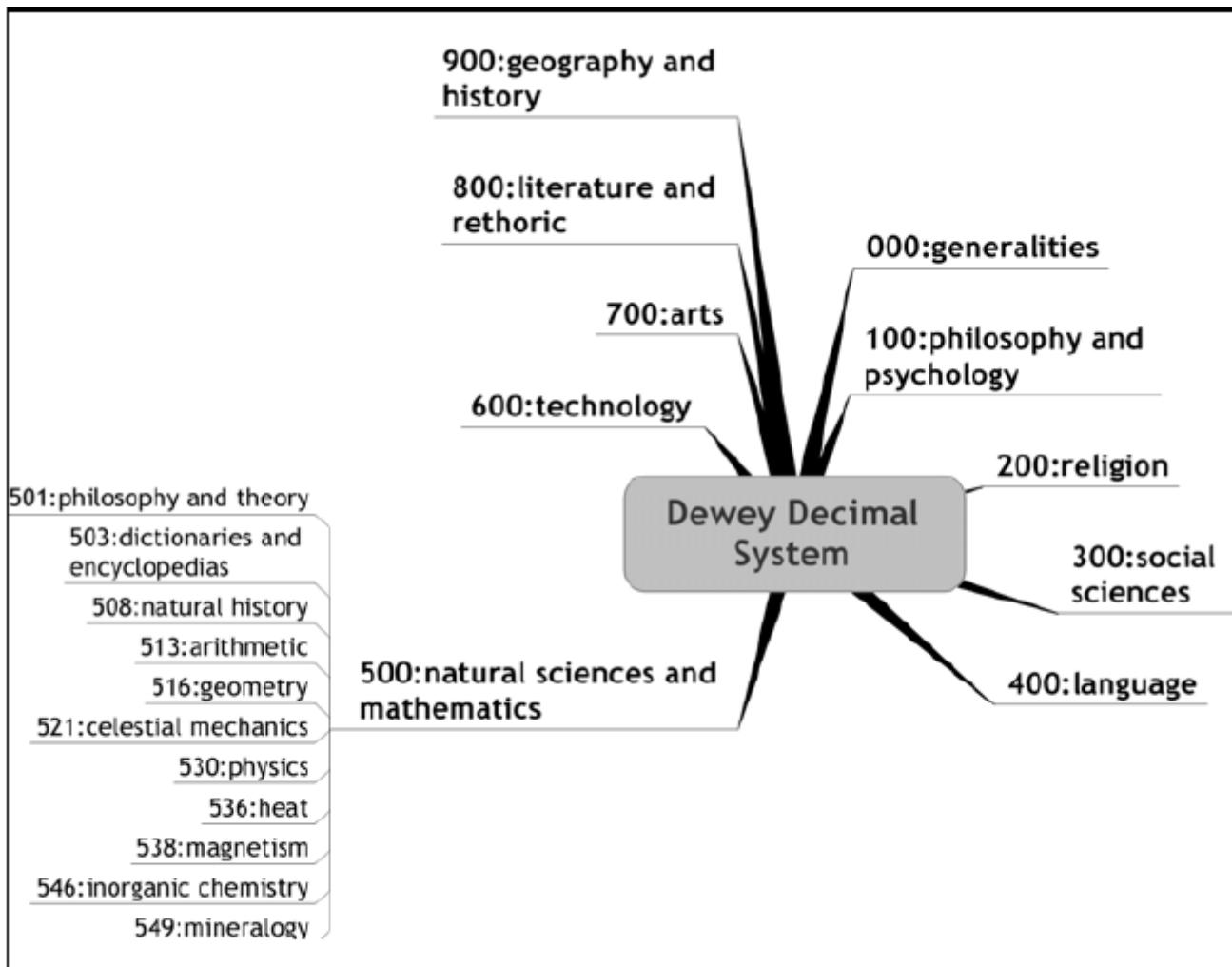
# Examples

---

Kingdom: Animalia  
Fil: Chordata  
Subfilo: Vertebrata  
Class: Mammalia  
Subclass: Theria  
Order: Primata  
Suborder: Anthropoidea  
Family: Hominidae  
Genera: Homo  
Species: Sapiens

**Fig. 2.2** Linnaean taxonomy of the living beings — human classification.

# Examples



**Fig. 2.3** Subject taxonomy used by the Dewey Decimal System (partial).

## WordNet Search - 3.0

- [WordNet home page](#) - [Glossary](#) - [Help](#)

Word to search for:

Display Options:  ▾

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations

Display options for sense: (gloss) "an example sentence"

### Noun

- S: (n) tank, army tank, armored combat vehicle, armoured combat vehicle (an enclosed armored military vehicle; has a cannon and moves on caterpillar treads)
- S: (n) tank, storage tank (a large (usually metallic) vessel for holding gases or liquids)
- S: (n) tank, tankful (as much as a tank will hold)
- S: (n) tank car, tank (a freight car that transports liquids or gases in bulk)
- S: (n) cooler, tank (a cell for violent prisoners)

### Verb

- S: (v) tank (store in a tank by causing (something) to flow into it)
- S: (v) tank (consume excessive amounts of alcohol)
- S: (v) tank (treat in a tank) "*tank animal refuse*"

# Ontology Example

## Concept

conceptual entity of the domain

## Attribute

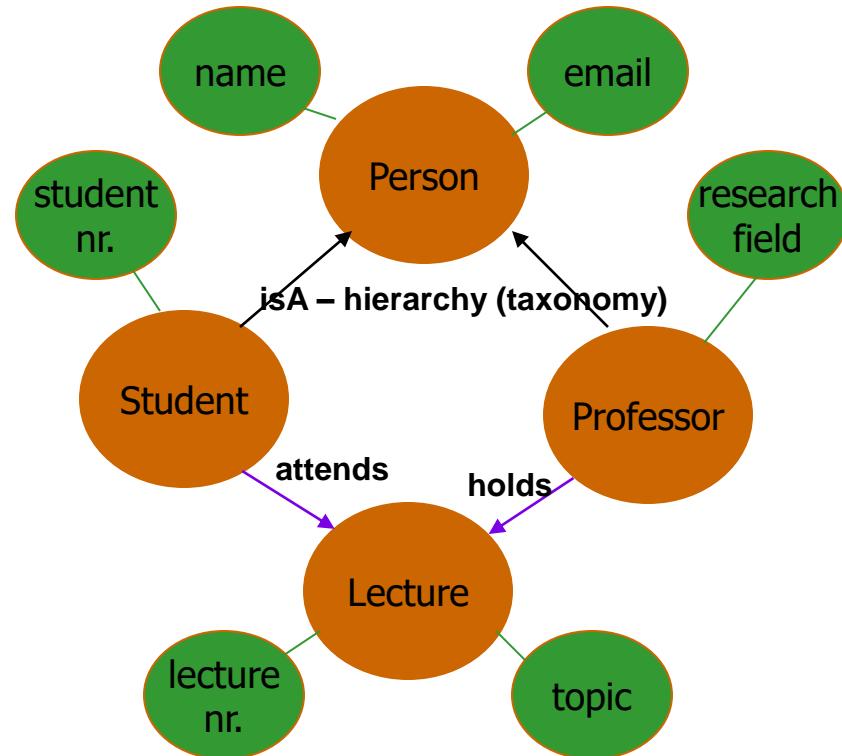
property of a concept

## Relation

relationship between concepts or properties

## Axiom

coherent description between Concepts / Properties  
/ Relations via logical expressions



$\text{holds}(\text{Professor}, \text{Lecture}) \Rightarrow \text{Lecture}.\text{topic} \in \text{Professor}.\text{researchField}$

# What is an ontology in practice?

- ◆ A theoretical or computational artefact that models knowledge of a domain through a vocabulary of concepts:
  - ◆ classes to categorize existing individuals in this domain (entities, “things” that populate the domain) and relations establishing links between those individuals.
  - ◆ Classes: what types of individuals exist (in general / in my domain of interest)? what distinctions are significant between them?
    - e.g., tables, chairs, computers, programs, students, teachers, humans, courses, exams, disciplines, universities, classrooms, days, numbers. . .
  - ◆ Classes correspond to unary predicates in FOL
    - Human(Daniel) encodes the fact that the individual Daniel is an instance of the class Human and
    - University(AAU) that AAU is an instance of University

# What is an ontology in practice?

## ◆ Relations: in which ways may those individuals be related?

- e.g., humans may sit on chairs, students may be enrolled at universities, a course may be taught by a teacher to students ...
- In FOL, Relations correspond to n-ary predicates ( $n > 1$ )
  - **enrolledAt**(Daniel, AAU) encodes the fact that the individuals Daniel and AAU are related by the binary relation **enrolledAt**
  - **teaches**(Mesfin, CoSc 605, Daniel) encodes the fact that the individuals Mesfin, CoSC 605, and Daniel are related by the ternary relation **teaches**

# What is an ontology in practice?

- ◆ **formalizes generic, necessary knowledge of this domain and constrains the interpretation of the vocabulary through axioms:**

- Taxonomic links between classes (“IS-A”, subsumption)

e.g., a student is a human:

*Student IS-A Human* ;  $\forall x(\text{Student}(x) \rightarrow \text{Human}(x))$

- Characterization of the relation arguments, especially the domain and range of binary relations

e.g., only students and universities can be related through enrollment:

$\forall xy(\text{enrolledAt}(x, y) \rightarrow \text{Student}(x) \wedge \text{University}(y))$

- More complex constraints on classes and relations (existence, unicity...)

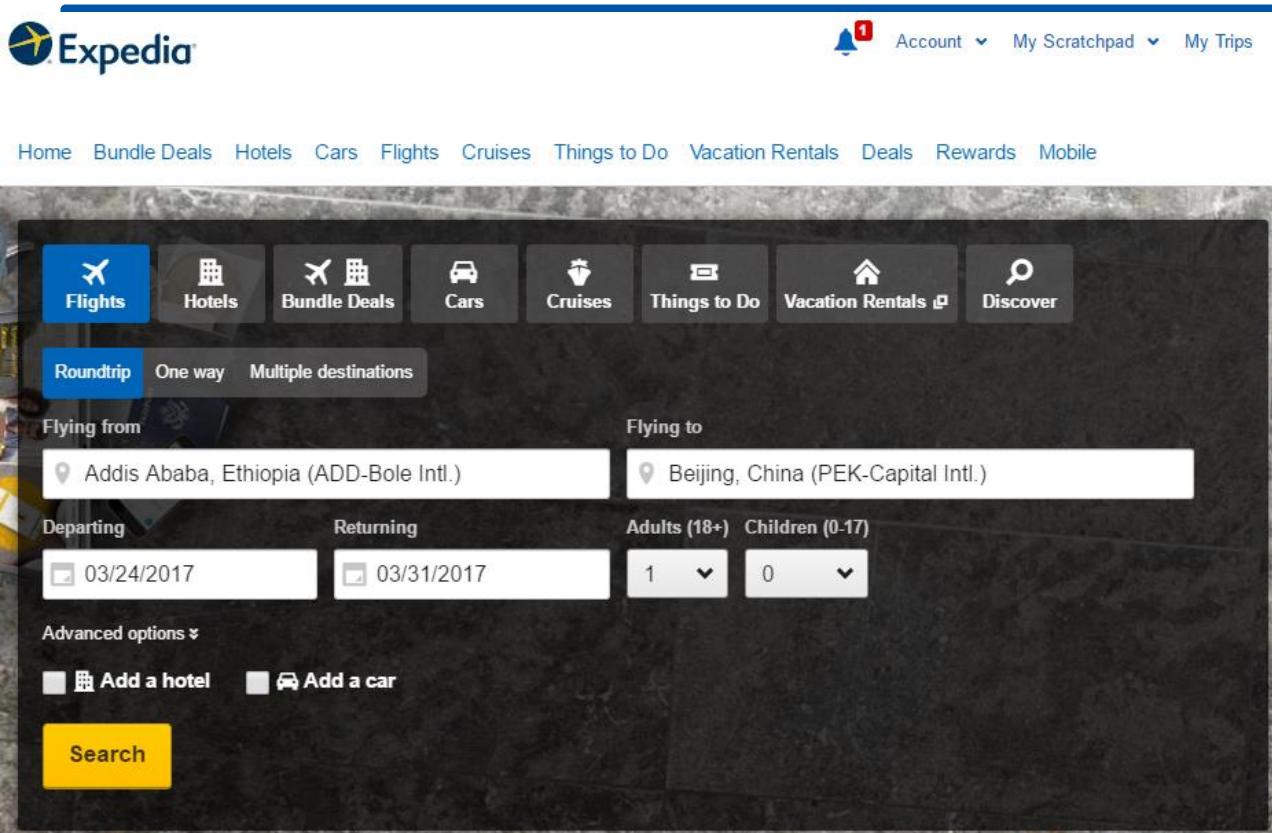
$\forall x(\text{Student}(x) \rightarrow \exists y(\text{enrolledAt}(x, y) \wedge \text{University}(y)))$

# What is an ontology in practice?

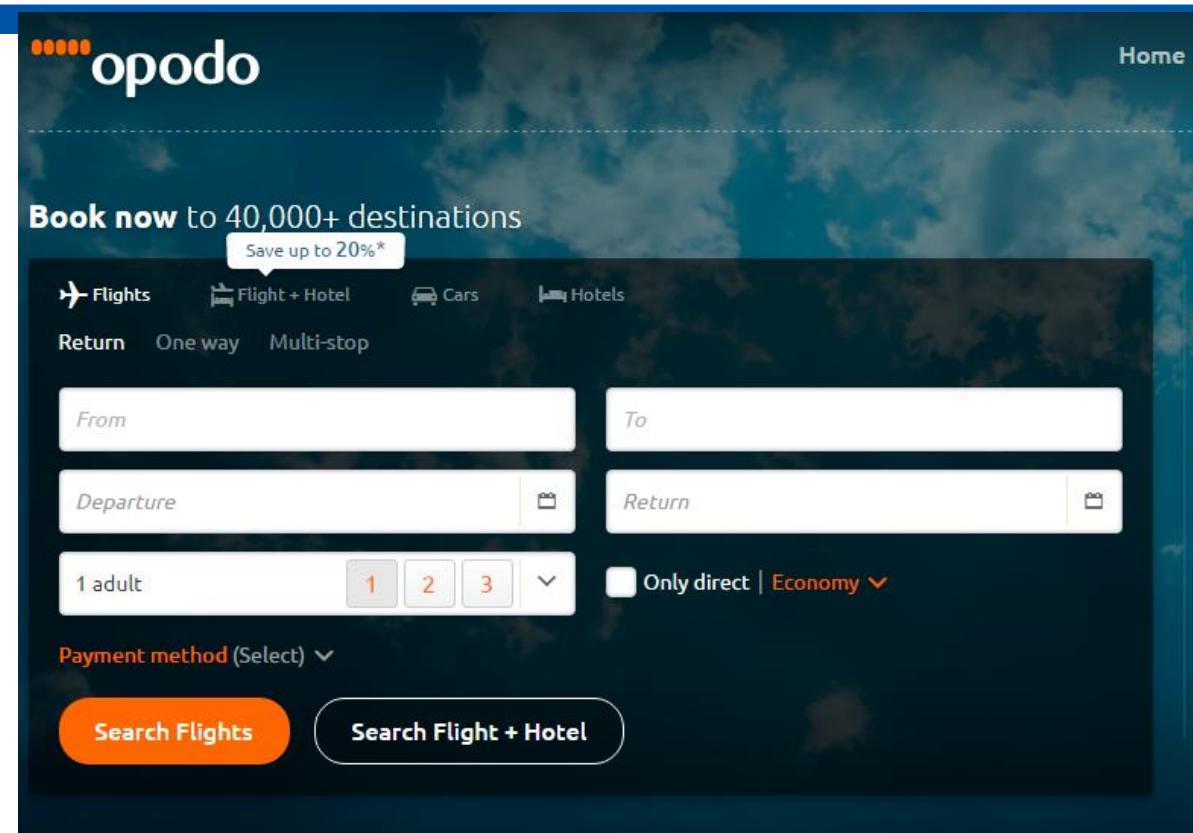
- ◆ An ontology can not represent contingent propositions : i.e. proposition is neither necessarily true nor necessarily false.

*Example:*  $\forall x(Human(x) \rightarrow (Student(x) \vee Teacher(x)))$

# Basic ontology – flight search engines



The Expedia interface shows a dark-themed search form. At the top, there are tabs for Flights, Hotels, Bundle Deals, Cars, Cruises, Things to Do, Vacation Rentals, Deals, Rewards, and Mobile. Below these are buttons for Roundtrip, One way, and Multiple destinations. The 'From' field contains 'Addis Ababa, Ethiopia (ADD-Bole Intl.)' and the 'To' field contains 'Beijing, China (PEK-Capital Intl.)'. Under 'Departing' and 'Returning', dates are set to 03/24/2017 and 03/31/2017 respectively. The passenger count is set to 1 adult and 0 children. There are 'Advanced options', 'Add a hotel', and 'Add a car' buttons, and a prominent yellow 'Search' button.



The Opodo interface features a dark background with a blue header bar. It includes a 'Save up to 20%\*' offer, navigation links for Flights, Flight + Hotel, Cars, and Hotels, and buttons for Return, One way, and Multi-stop. The 'From' and 'To' fields are empty. The 'Departure' and 'Return' fields are also empty. The passenger count is set to 1 adult, with options for 1, 2, or 3 adults. A checkbox for 'Only direct | Economy' is checked. Below the search fields are 'Payment method (Select)' dropdowns and two large orange buttons: 'Search Flights' and 'Search Flight + Hotel'.

Which domain of entities, which classes?

Flight, RoundTripF, OneWayF, MultipleDestF, Airport, Town, Country, Date, Day, Month, Year, Calendars, Human, Adult, Child, Infant, Currency  
+ Options, Hotel, Car...

Round Trip One Way Multi-city

Addis Ababa, Ethiopia (ADD)

Nearby airports

Beijing, China (PEK)

Nearby airports

03/24/2017

03/31/2017

Search

1 Traveler, All Airlines, Economy / Coach Show options ▾

## Select your departure to Beijing Fri, Mar 24

Prices are roundtrip per person, include all taxes and fees, but do not include baggage fees.

Watch Airline Fares

Price (Lowest) ▾

Finalizing prices...



11:45p - 8:15p +1  
Multiple Airlines  
15h 30m  
ADD - PEK  
1 stop  
2h 5m in CAN

Select to price

Select

Shenzhen Airlines 3602 operated by Ethiopian Airlines

Stops

- Nonstop (1) \$1,594
- 1 Stop (21) \$3,454
- 2+ Stops (44) \$4,144



11:45p - 9:20p +1  
Multiple Airlines  
16h 35m  
ADD - PEK  
1 stop  
3h 5m in CAN

Select to price

Select

Shenzhen Airlines 3602 operated by Ethiopian Airlines

Alliance Group

- Star Alliance (57) \$1,594
- OneWorld (1)
- SkyTeam (7) \$4,144



11:45p - 10:20p +1  
Multiple Airlines  
17h 35m  
ADD - PEK  
1 stop  
4h 5m in CAN

Select to price

Select

Shenzhen Airlines 3602 operated by Ethiopian Airlines

Airlines included

- Ethiopian Airlines (43) \$1,594



4:00p - 4:35p +1  
Multiple Airlines  
19h 35m  
ADD - PEK  
2 stops  
DEL, ICN

Select to price

Select

Air India 7568 operated by Ethiopian Airlines

Air India 7148 operated by Asiana Airlines

## Additional Entities:

Time, Fare, Airline, DirectF, 1StopF, 2+StopF, Duration, and then, FlightID, Passenger, Booking, Seat...

### Custom Flight Search Engine

Round Trip     One Way

From

Enter a city or airport

Depart

mm/dd/yyyy 

To

Enter a city or airport

Return

mm/dd/yyyy 

Search Now!

Classes: Flight, RoundTripF, OneWayF,

Relations: departFrom, goesTo, departsOn, returnsOn

# Taxonomy of Classes

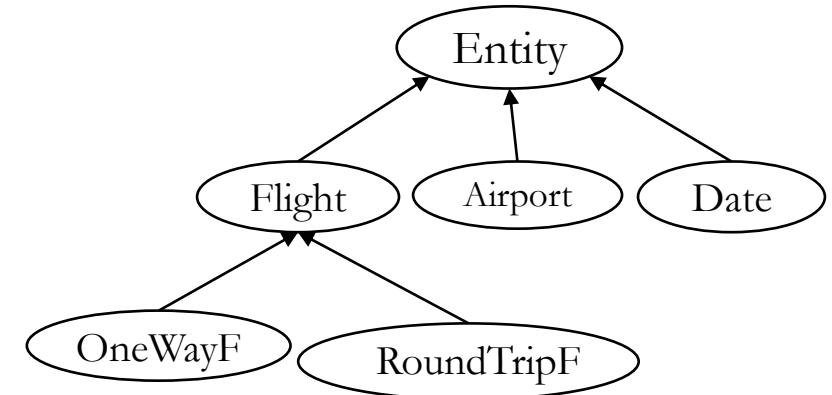
- Any oneway flight is a flight, any roundtrip flight is a flight

$\forall x(\text{OneWayF}(x) \rightarrow \text{Flight}(x))$

OneWayF IS-A Flight

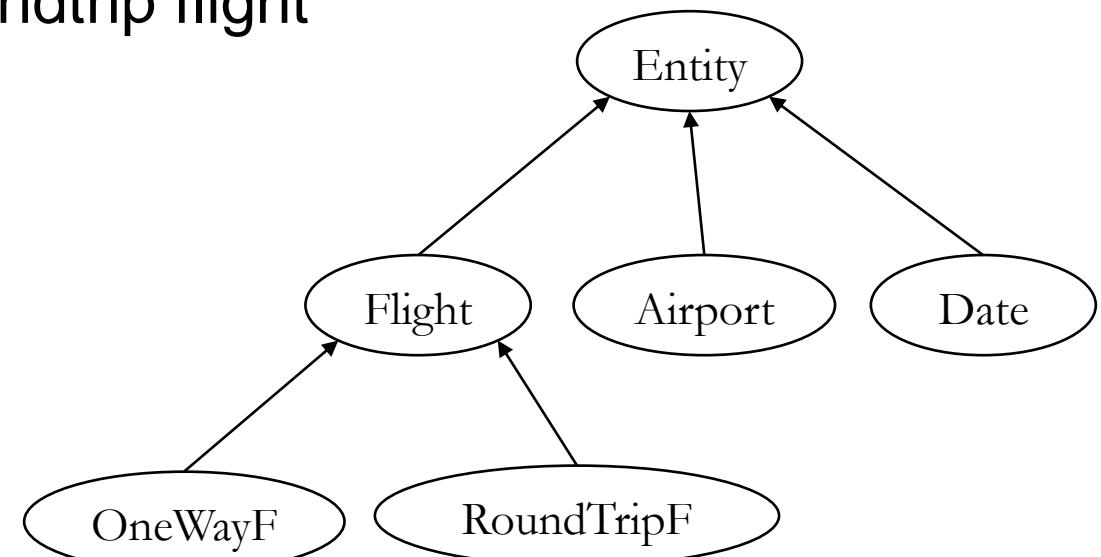
$\forall x(\text{RoundTripF}(x) \rightarrow \text{Flight}(x))$

RoundTripF IS-A Flight



# Taxonomy of Classes: Disjointness

- ◆ Definition- Disjointness: Two classes are disjoint iff their “taxonomic overlap”, i.e. the set of common individuals is empty.
- ◆ Nothing is both a oneway flight and a roundtrip flight  
 $\forall x(\text{OneWayF}(x) \rightarrow \neg \text{RoundTripF}(x))$
- ◆ Nothing is both a flight and an airport  
 $\forall x(\text{Flight}(x) \rightarrow \neg \text{Airport}(x))$
- ◆ Nothing is both a flight and a date  
 $\forall x(\text{Flight}(x) \rightarrow \neg \text{Date}(x))$
- ◆ Nothing is both an airport and a date  
 $\forall x(\text{Airport}(x) \rightarrow \neg \text{Date}(x))$



# Domain and Range of relations

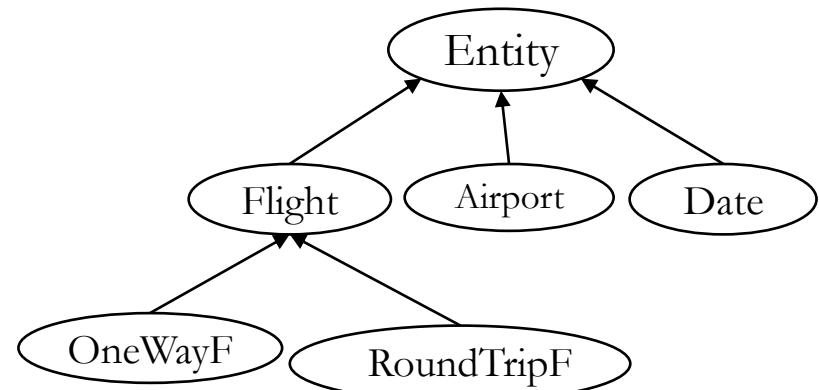
- **departsFrom** and **goesTo** relate instances of Flight to instances of Airport

$$\forall xy(\text{departsFrom}(x, y) \rightarrow (\text{Flight}(x) \wedge \text{Airport}(y)))$$
$$\forall xy(\text{goesTo}(x, y) \rightarrow (\text{Flight}(x) \wedge \text{Airport}(y)))$$

- **departsOn** relates instances of Flight to instances of Date

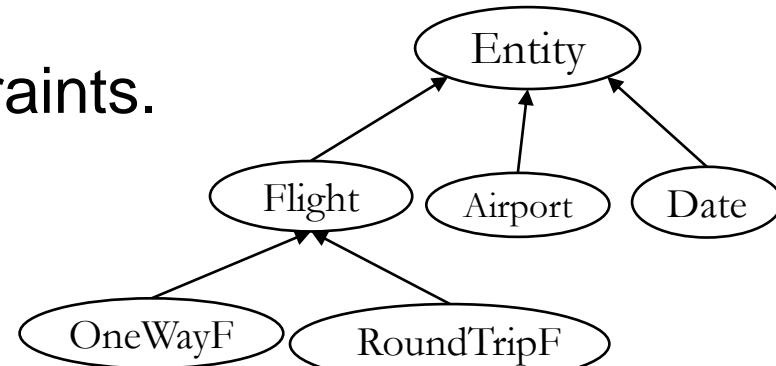
$$\forall xy(\text{departsOn}(x, y) \rightarrow (\text{Flight}(x) \wedge \text{Date}(y)))$$

- **returnsOn** relates instances of RoundTripF to instances of Date

$$\forall xy(\text{returnsOn}(x, y) \rightarrow (\text{RoundTripF}(x) \wedge \text{Date}(y)))$$


# Existence and unicity

- **Existence constraints** make sure that any flight departs from and goes to some airport, at a given date, etc.
- add existence constraints to the domain and range constraints.

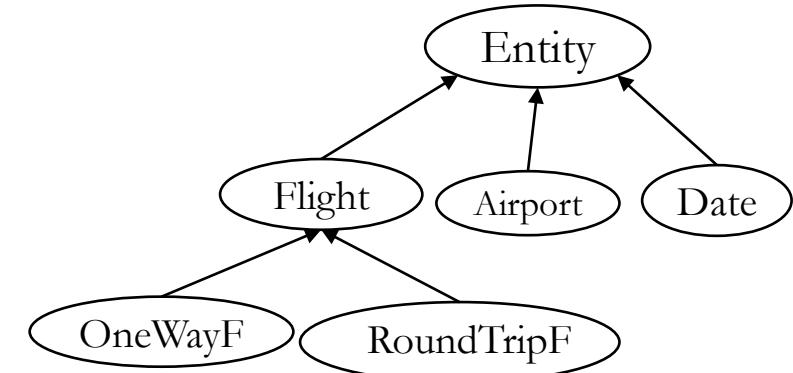
$$\forall x(\text{Flight}(x) \rightarrow \exists yzt(\text{departsFrom}(x, y) \wedge \text{goesTo}(x, z) \wedge \text{departsOn}(x, t)))$$
$$\forall x(\text{RoundTripF}(x) \rightarrow \exists t \text{ returnsOn}(x, t))$$


- **Unicity constraint** ascertains that the existence of a **binary functional** relation. Here, all the relations are **functional**, i.e., any **flight departs** from and **goes to** a unique airport, on a unique date, etc.

$$\forall xyz((\text{departsFrom}(x, y) \wedge \text{departsFrom}(x, z)) \rightarrow y = z)$$
$$\forall xyz((\text{goesTo}(x, y) \wedge \text{goesTo}(x, z)) \rightarrow y = z)$$
$$\forall xyz((\text{departsOn}(x, y) \wedge \text{departsOn}(x, z)) \rightarrow y = z)$$
$$\forall xyz((\text{returnsOn}(x, y) \wedge \text{returnsOn}(x, z)) \rightarrow y = z)$$

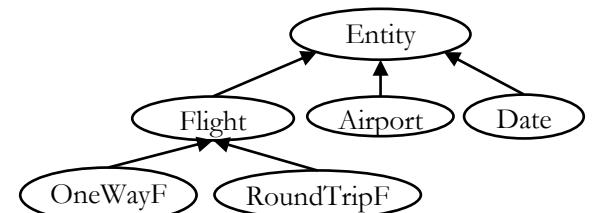
# Non-identity

- ◆ No flight leaves and arrives at the same place

$$\forall xyz((\text{departsFrom}(x, y) \wedge \text{goesTo}(x, z)) \rightarrow \neg y = z)$$


# What is a roundtrip flight?

- ◆ A roundtrip flight is composed of two oneway flights with matching airports
  - New dedicated ternary relation: isComposedOf
  - Constraint on argument classes (selection restrictions)
$$\forall xyz(\text{isComposedOf}(x, y, z) \rightarrow (\text{RoundTripF}(x) \wedge \text{OneWayF}(y) \wedge \text{OneWayF}(z)))$$
  - Constraint on departure and arrival airports
$$\forall xyzab((\text{isComposedOf}(x, y, z) \wedge \text{departsFrom}(x, a) \wedge \text{goesTo}(x, b)) \rightarrow (\text{departsFrom}(y, a) \wedge \text{goesTo}(y, b) \wedge \text{departsFrom}(z, b) \wedge \text{goesTo}(z, a)))$$
  - Constraint on dates
$$\forall xyzab((\text{isComposedOf}(x, y, z) \wedge \text{departsOn}(x, a) \wedge \text{returnsOn}(x, b)) \rightarrow (\text{departsOn}(y, a) \wedge \text{departsOn}(z, b)))$$



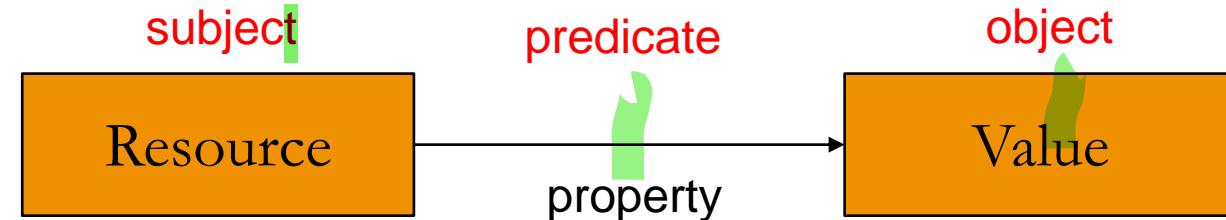
# **Resource Description Framework - RDF**

- ◆ It is a framework to publish statements on the Web about anything.
- ◆ It allows anyone to describe resources, in particular Web resources, such as the author, creation date, subject, and copyright of an image.
- ◆ RDF is a data model
  - it is domain-neutral, application-neutral and ready for internationalization
  - it can be viewed as directed, labeled graphs or as an object-oriented model (object/attribute/value)
  - It is an abstract, conceptual layer independent of XML
  - XML is a transfer syntax for RDF, not a component of RDF
  - RDF data might never occur in XML form
  - it provides interoperability between applications that exchange machine-understandable information.

- ◆ Resources are a core concept on the Semantic Web:
  - ◆ everything one might refer to is considered a resource. Everything that can be identified by a URI is considered as a resource.
- ◆ Descriptions of resources are essential for understanding and reasoning about them.
  - ◆ In the most general case, a description is a set of attributes, features, and relations concerning the resource.
- ◆ The Framework means it provides models, languages, and syntaxes for these descriptions.
- ◆ RDF provides a standard data structure and a model to encode data and metadata about any subject on the Web

# RDF model

- ◆ RDF model is a set of RDF triples
- ◆ A triple is an expression (statement) of the format  
**(subject, predicate, object)**
- ◆ Subject: **resource** or **blank node**
- ◆ Predicate: **property** (of the resource)
- ◆ Object: **value** (of the property) – literal or blank node



## Triplet data model

---

- ◆ Example: represent the assertion “Fabien has written a page doc.html about music” can be broken down into two RDF statements:  
(doc.html, author, Fabien) and  
(doc.html, theme, music).
- ◆ The logical view of the RDF triples is that RDF makes an **open-world assumption** (i.e. what is not known to be true is simply unknown - the absence of a triple is not significant) as opposed to the **closed-world assumption** of classical systems (i.e. what is not known to be true must be false).

# Graph-Oriented data model

---

- ◆ RDF **triples** can be seen as **two vertices** and one arc of a graph describing and linking resources.
- ◆ RDF is a decentralized data representation model relying on distributed triples that form a global graph.
- ◆ The identity of the resources is based on the URI mechanism: if two resources have the same URI they are one and the same node in the graph.
- ◆ An RDF graph is:
  - ◆ **A multi-graph:** a graph that can contain both multiple edges and loops between its vertices
  - ◆ **A directed graph:** every edge is oriented going from the end-vertex representing the subject to the end-vertex representing the object
  - ◆ **A labeled graph:** edges are labeled with URIs, vertices are labeled with URIs, blank node identifiers, or literals

English: "The report doc.html has the authors Fabien and York, the theme music and is 23-pages long."

Logic predicates:

Report(doc.html)

creator(doc.html,Fabien)

creator(doc.html,York)

theme(doc.html,Music)

nbPages(doc.html, 23)

Triples:

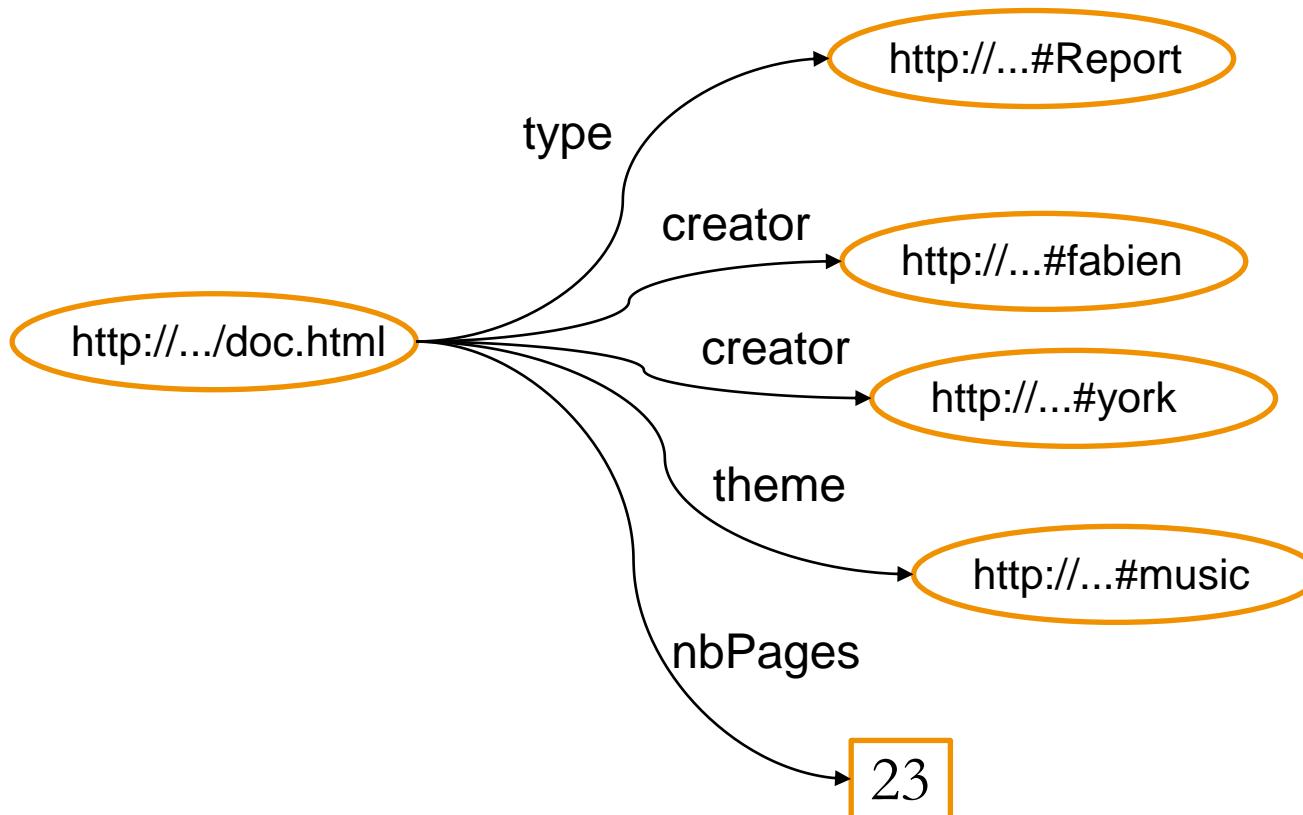
(doc.html, type, Report)

(doc.html, creator, Fabien)

(doc.html, creator, York)

(doc.html, theme, Music)

(doc.html, nbPages, "23")



## Example – Graph-oriented data model

Describe a Person identified by

<http://www.w3.org/People/EM/contact#me>, whose name is Eric Miller, whose email address is [em@w3.org](mailto:em@w3.org), and whose title is Dr.

<http://www.w3.org/2000/10/swap/pim/contact#Person>

<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>

<http://www.w3.org/2000/10/swap/pim/contact#fullName>

Eric Miller

<http://www.w3.org/People/EM/contact#me>

<http://www.w3.org/2000/10/swap/pim/contact#mailbox>

<http://www.w3.org/2000/10/swap/pim/contact#personaltitle>

Dr.

<mailto:em@w3.org>

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
           xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">
  <contact:Person
    rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>
</rdf:RDF>
```

## Example:

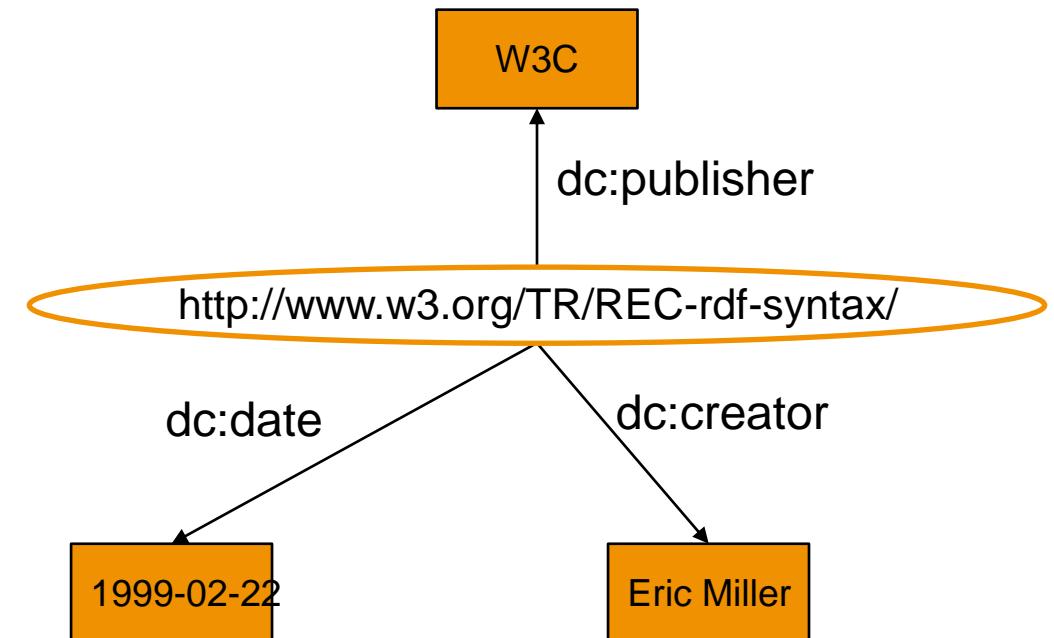
Describe “the document at <http://www.w3c.org/TR/REC-rdf-syntax> is created by Eric Miller on 1999-02-22 and is published by W3C”

Triple:

<http://www.w3c.org/TR/REC-rdf-syntax> creator Eric Miller

<http://www.w3c.org/TR/REC-rdf-syntax> date 1999-02-22

<http://www.w3c.org/TR/REC-rdf-syntax> publisher W3C



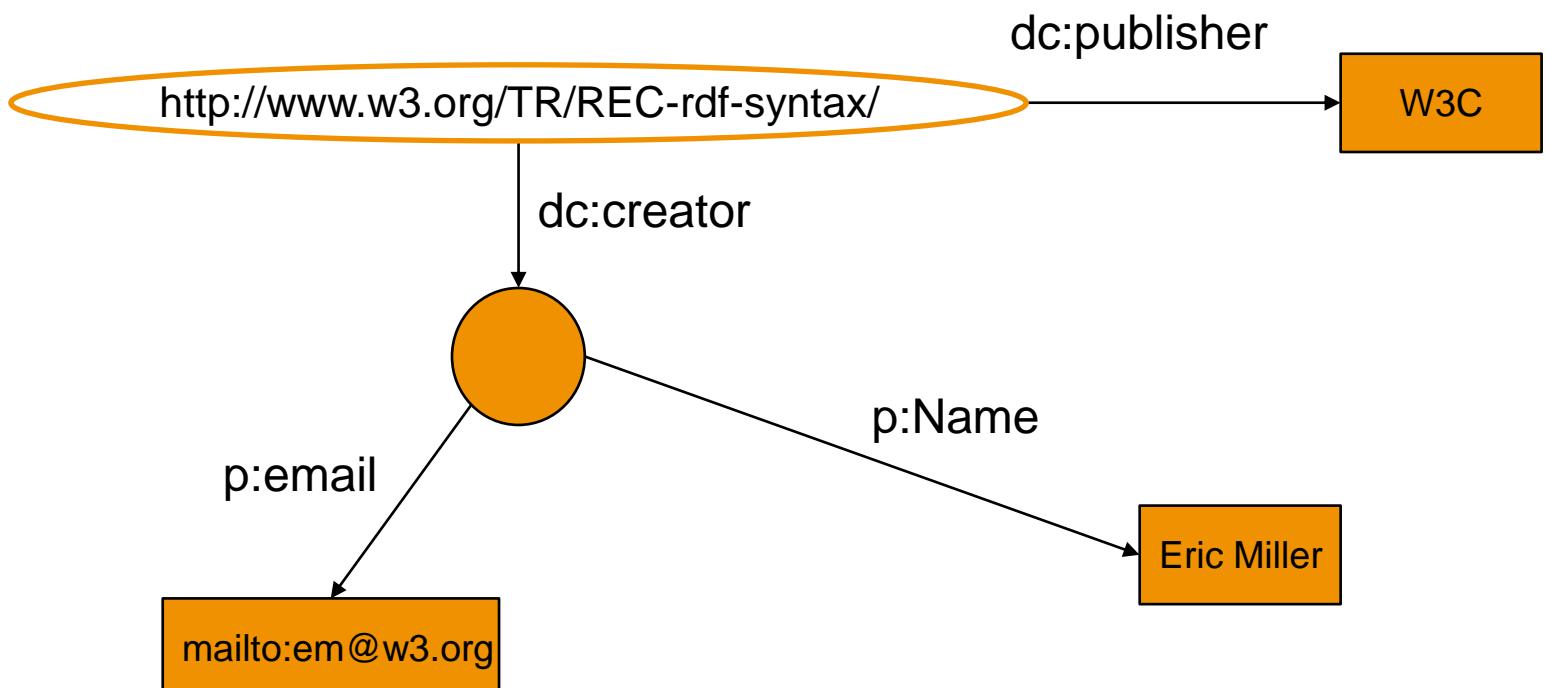
# Node and edge labels in RDF graphs

---

- ◆ node and edge labels:
  - URI
  - literal (string)
  - blank node (anonymous label)
- ◆ but:
  - a literal can only appear in object positions
  - a blank node can only appear in subject or object positions
- ◆ remark: URIs are used as predicates, i.e., graph nodes can be used as edge labels (RDF has meta-modeling abilities)

# Complex values

- values of properties need not be simple strings
- the value of a property can also be a graph node (corresponding to a resource)
- using blank nodes, arbitrarily complex tree and graph structures are possible

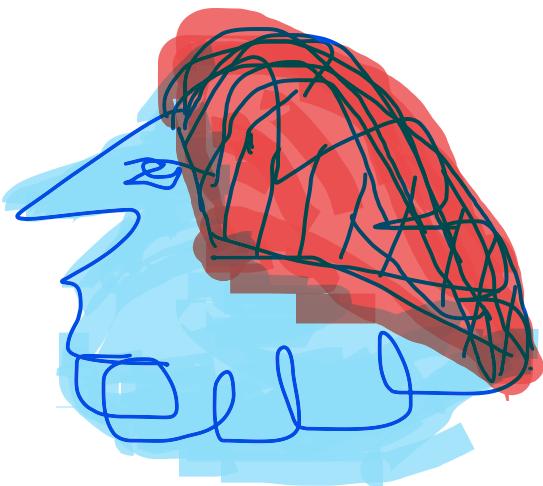


# Blank nodes

- blank node (bnode) = RDF graph node with “anonymous label” (i.e., not associated with an URI)

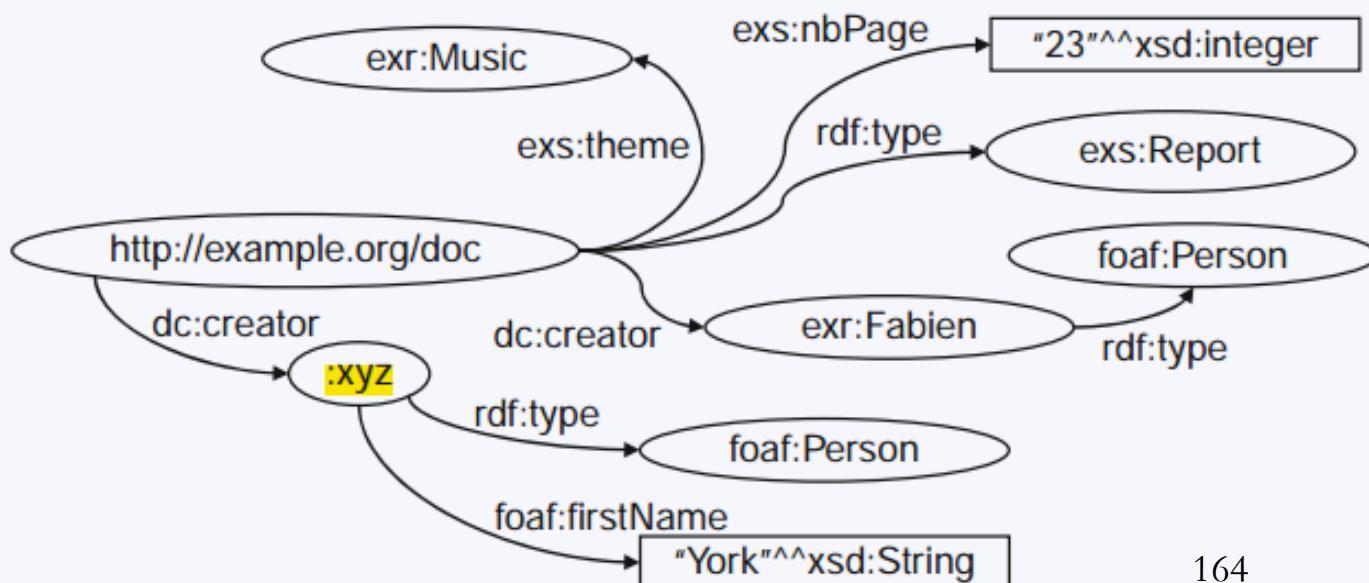
Example: "Jean has a friend born the 21st of April"

ex:Jean foaf:knows    \_:p1  
  \_:p1    foaf:birthDate 04-21  
  \_:p1 is the blank node (bnode)  
◆ bnode can be used both  
  as subjects and objects



## Triples

(<http://example.org/doc.html> , rdf:type , exs:Report)  
(<http://example.org/doc.html> , dc:creator , exr:Fabien)  
(<http://example.org/doc.html> , dc:creator , \_:xyz)  
(\_:xyz, foaf:firstName, "York"^^xsd:String)  
(<http://example.org/doc.html> , exs:theme , exr:Music)  
(<http://example.org/doc.html> , exs:nbPages , "23"^^xsd:integer)



## RDF Graphs as XML Trees

---

- ◆ **RDF/ XML syntax is the standardized way of serializing RDF triplet.**
- ◆ The principal aim of RDF/XML is to be machine-processable and compliant to the de facto standard of Web document formatting, XML.
- ◆ **RDF/ XML syntax allows RDF documents to be easily exchanged between very different types of systems and applications.**

# RDF/ XML

- ◆ There are several possibilities to express the same RDF graph (i.e. the serialization is not unique)

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:exs= "http://example.org/schema#">
<rdf:Description rdf:about="http://example.org/doc.html">
    <rdf:type rdf:resource="http://example.org/schema#Report"/>
    <exs:theme rdf:resource="http://example.org#Music"/>
    <exs:theme rdf:resource="http://example.org#History"/>
    <exs:nbPages rdf:datatype= "http://www.w3.org/2001/XMLSchema#int">
        23
    </exs:nbPages>
</rdf:Description>
</rdf:RDF>

<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:exs= "http://example.org/schema#">
<exs:Report rdf:about="http://example.org/doc.html" exs:nbPages="23">
    <exs:theme rdf:resource="http://example.org#Music"/>
    <exs:theme rdf:resource="http://example.org#History"/>
</exs:Report>
</rdf:RDF>
```

## N-Triples – N3 format

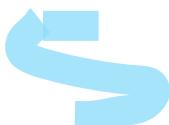
- ◆ N-Triples requires one triple to be written per line.
- ◆ Every triple is provided as subject, predicate, and object that are separated by whitespaces and terminated by a dot (.).

```
<http://example.org/doc.html> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://example.org/schema#Report>.
<http://example.org/doc.html> <http://example.org/schema#theme> <http://example.org#Music>.
<http://example.org/doc.html> <http://example.org/schema#theme> <http://example.org#History>.
<http://example.org/doc.html> <http://example.org/schema#nbPages> "23"^^
  <http://www.w3.org/2001/XMLSchema#integer>.
```

# Triple Serialization in Turtle

- ◆ Turtle (Terse RDF Triple Language) is a compact textual format for serializing an RDF graph, is less constrained than N-Triples
- ◆ Turtle, for example, does not have the restriction of single-lined statement as in N-Triples, and provides various abbreviations.
- ◆ Further shortcuts are supported for repeated subjects, or multiple objects.
- ◆ To abbreviate multiple statements with the same subject, Turtle provides a semicolon (;) notation which makes it possible to list predicate–object pairs for the same subject.
- ◆ Multiple statements with the same subject–predicate pair but different objects can be similarly abbreviated using the comma (,) notation.

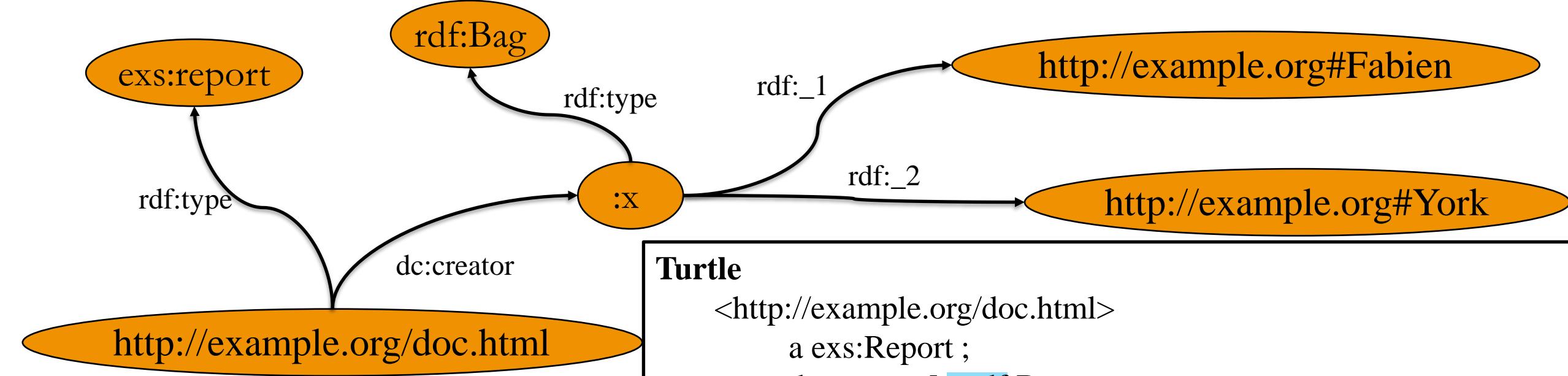
```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.  
@prefix exs: <http://example.org/schema#>.  
<http://example.org/doc.html> a exs:Report ;  
exs:theme <http://example.org#Music> ,<http://example.org#History> ;  
exs:nbPages "23"^^xsd:int .
```



# Containers

---

- ◆ Containers are collections: allow grouping of resources (or literal values)
- ◆ It is possible to make statements about the container (as a whole) or about its members individually
- ◆ Different types of containers exist
  - ◆ rdf:Bag defines an unordered group of resources or literals.
  - ◆ rdf:Seq defines an ordered group of resources or literals.
  - ◆ rdf:Alt represents a group of resources or literals that are alternatives; that is, only one of the values can be selected.
- ◆ It is also possible to create collections based on URI patterns
- ◆ Duplicate values are permitted (no mechanism to enforce unique value constraints)



### Turtle

```
<http://example.org/doc.html>
  a exs:Report ;
  dc:creator [ a rdf:Bag ;
    rdf:_1 <http://example.org#Fabien> ;
    rdf:_2 <http://example.org#York> ].
```

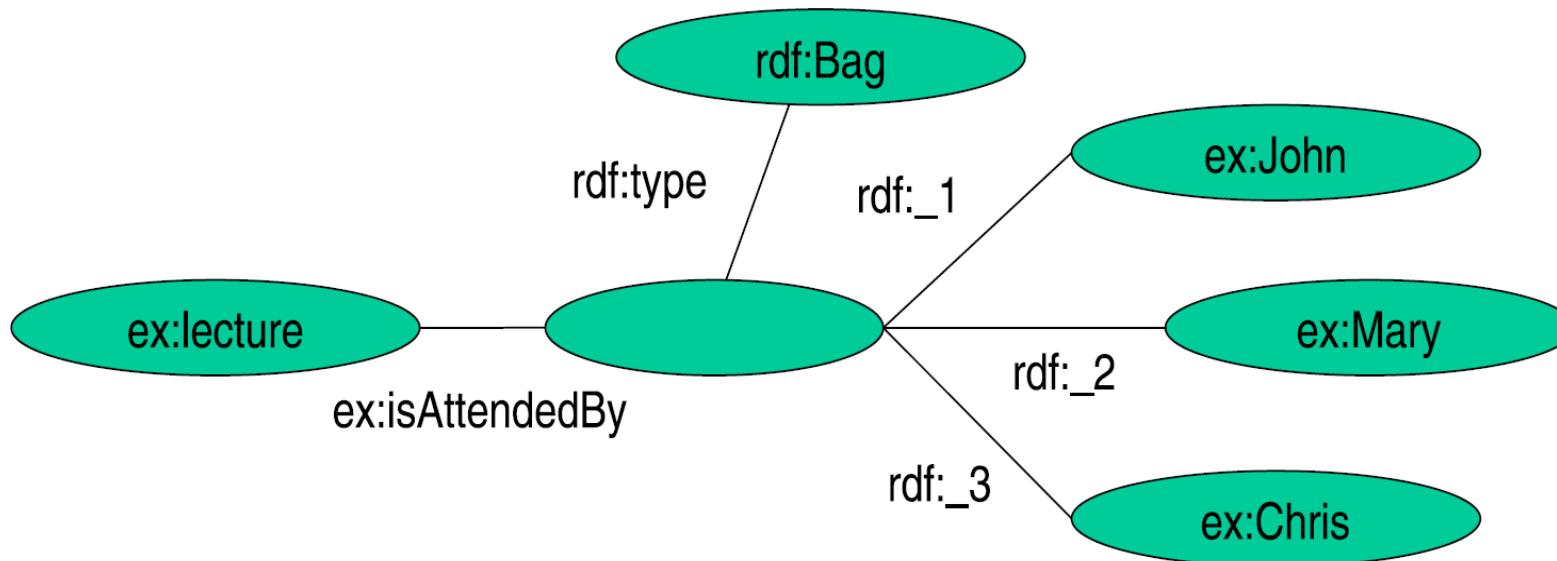
### RDF/XML

```
<exs:Report rdf:about="http://example.org/doc.html">
  <dc:creator>
    <rdf:Bag>
      <rdf:li rdf:resource="http://example.org#Fabien"/>
      <rdf:li rdf:resource="http://example.org#York"/>
    </rdf:Bag>
  </dc:creator>
</exs:Report>
```

# RDF Containers: Bag

---

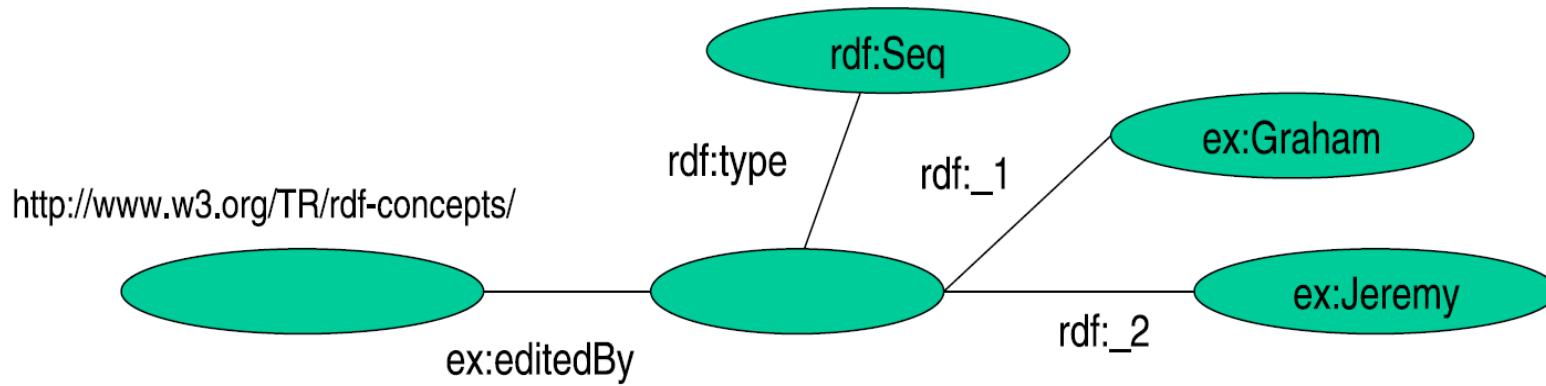
***The lecture is attended by John, Mary and Chris***



# RDF Containers: Seq

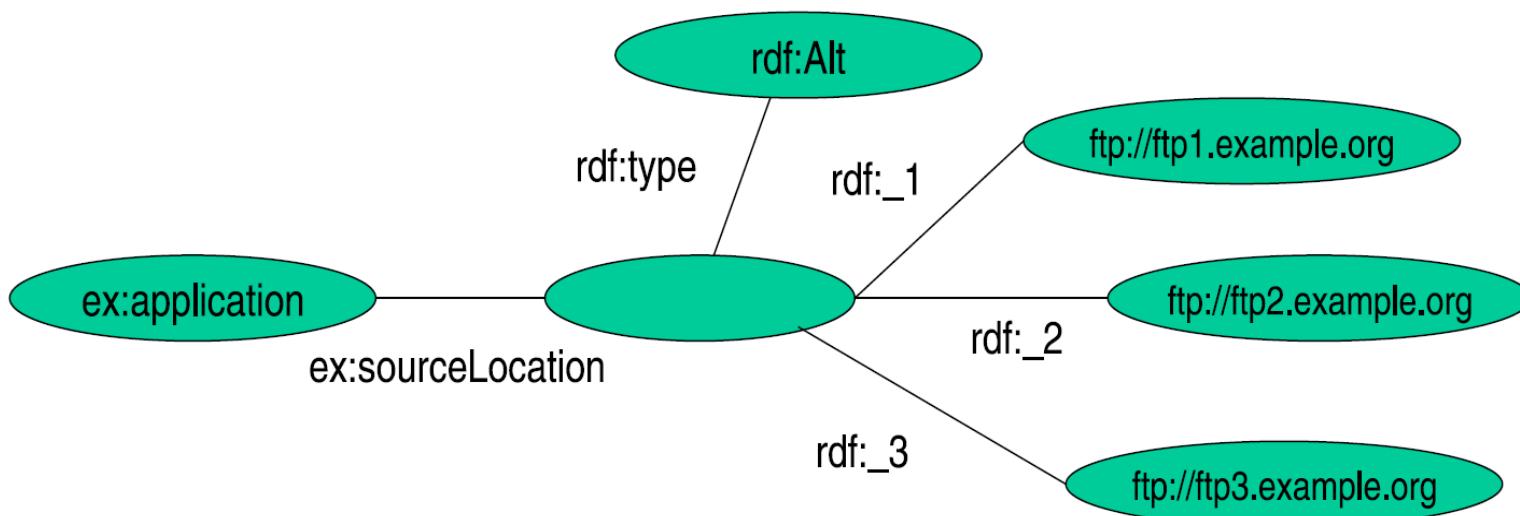
---

***“[RDF-Concepts] is edited by Graham and Jeremy  
(in that order)”***



# RDF Containers: Alt

***“The source code for the application may be found at  
ftp1.example.org, ftp2.example.org, ftp3.example.org”***



# Reification

- ◆ It is a mechanism to make statement about statements
- ◆ Reification in RDF is represented using an **RDF statement** as the subject (or object) of another RDF statement

## Examples

```
<rdf:Description rdf:about="#949352">
    <uni:name>Grigoris Antoniou</uni:name>
</rdf:Description>
```

**reifies as**

```
<rdf:Statement rdf:ID="StatementAbout949352">
    <rdf:subject rdf:resource="#949352"/>
    <rdf:predicate rdf:resource="http://www.mydomain.org/uni-ns#name"/>
    <rdf:object>Grigoris Antoniou</rdf:object>
</rdf:Statement>
```

# Reification

---

- ◆ RDF provides a built-in predicate vocabulary for reification:
  - ◆ rdf:subject
  - ◆ rdf:predicate
  - ◆ rdf:object
  - ◆ rdf:statement
- ◆ Using this vocabulary (i.e., these URIs from the `rdf:namespace`) it is possible to represent a triple through a blank node

# Reification

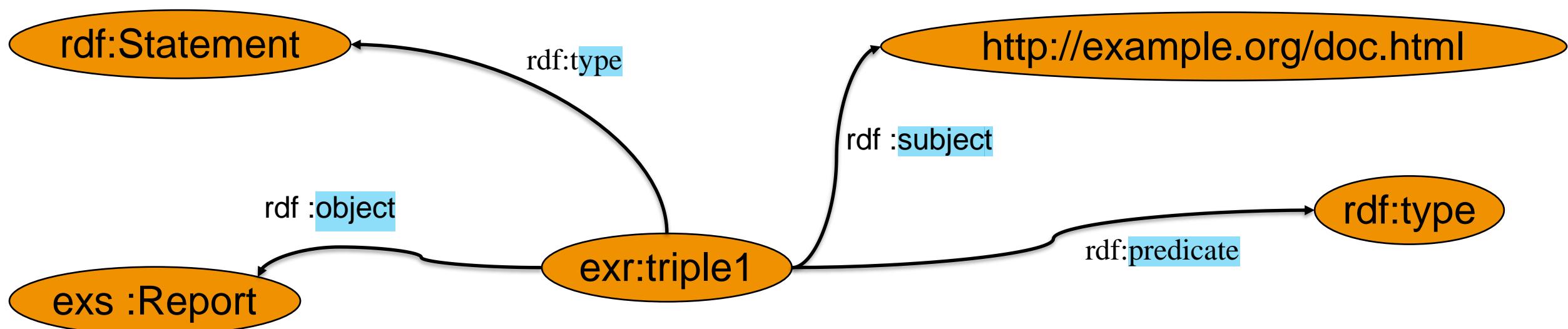
## Examples

Triple

`<http://example.org/doc.html> a exs:Report.`

Reified triple in Turtle syntax

```
exr:triple1 a rdf:Statement ;  
    rdf:subject <http://example.org/doc.html>;  
    rdf:predicate rdf:type ;  
    rdf:object exs:Report.
```



## **The RDF Schema (RDFS)**

# RDF Vocabulary Description Language

- ◆ Types in RDF:

```
<#john, rdf:type, #Student>
```

- ◆ What is a “#Student”?

- ◆ We know that “#Student” identifies a category (a concept or a class), but this is only implicitly defined in RDF

# RDF Vocabulary Description Language

- ◆ We need a language for defining RDF types:
  - Define classes:
    - “**#Student** is a class”
  - Relationships between classes:
    - “**#Student** is a sub-class of **#Person**”
  - Properties of classes:
    - “**#Person** has a property **hasName**”
- ◆ RDF Schema is such a language

# RDF Vocabulary Description Language

- ◆ **Classes:**

```
<#Student, rdf:type, #rdfs:Class>
```

- ◆ **Class hierarchies:**

```
<#Student, rdfs:subClassOf, #Person>
```

- ◆ **Properties:**

```
<#hasName, rdf:type, rdf:Property>
```

- ◆ **Property hierarchies:**

```
<#hasMother, rdfs:subPropertyOf, #hasParent>
```

- ◆ **Associating properties with classes (a):**

- “The property `#hasName` only applies to `#Person`”

```
<#hasName, rdfs:domain, #Person>
```

- ◆ **Associating properties with classes (b):**

- “The type of the property `#hasName` is `#xsd:string`”

```
<#hasName, rdfs:range, xsd:string>
```

# RDFS Vocabulary

- ◆ RDFS Extends the RDF Vocabulary
- ◆ RDFS vocabulary is defined in the namespace:

<http://www.w3.org/2000/01/rdf-schema#>

## RDFS Classes

- `rdfs:Resource`
- `rdfs:Class`
- `rdfs:Literal`
- `rdfs:Datatype`
- `rdfs:Container`
- `rdfs:ContainerMembershipProperty`

## RDFS Properties

- `rdfs:domain`
- `rdfs:range`
- `rdfs:subPropertyOf`
- `rdfs:subClassOf`
- `rdfs:member`
- `rdfs:seeAlso`
- `rdfs:isDefinedBy`
- `rdfs:comment`
- `rdfs:label`

# RDFS Principles

## ◆ Resource

- All resources are implicitly instances of `rdfs:Resource`

## ◆ Class

- Describe sets of resources
- Classes are resources themselves - e.g. Webpages, people, document types
  - Class hierarchy can be defined through `rdfs:subClassOf`
  - Every class is a member of `rdfs:Class`

## ◆ Property

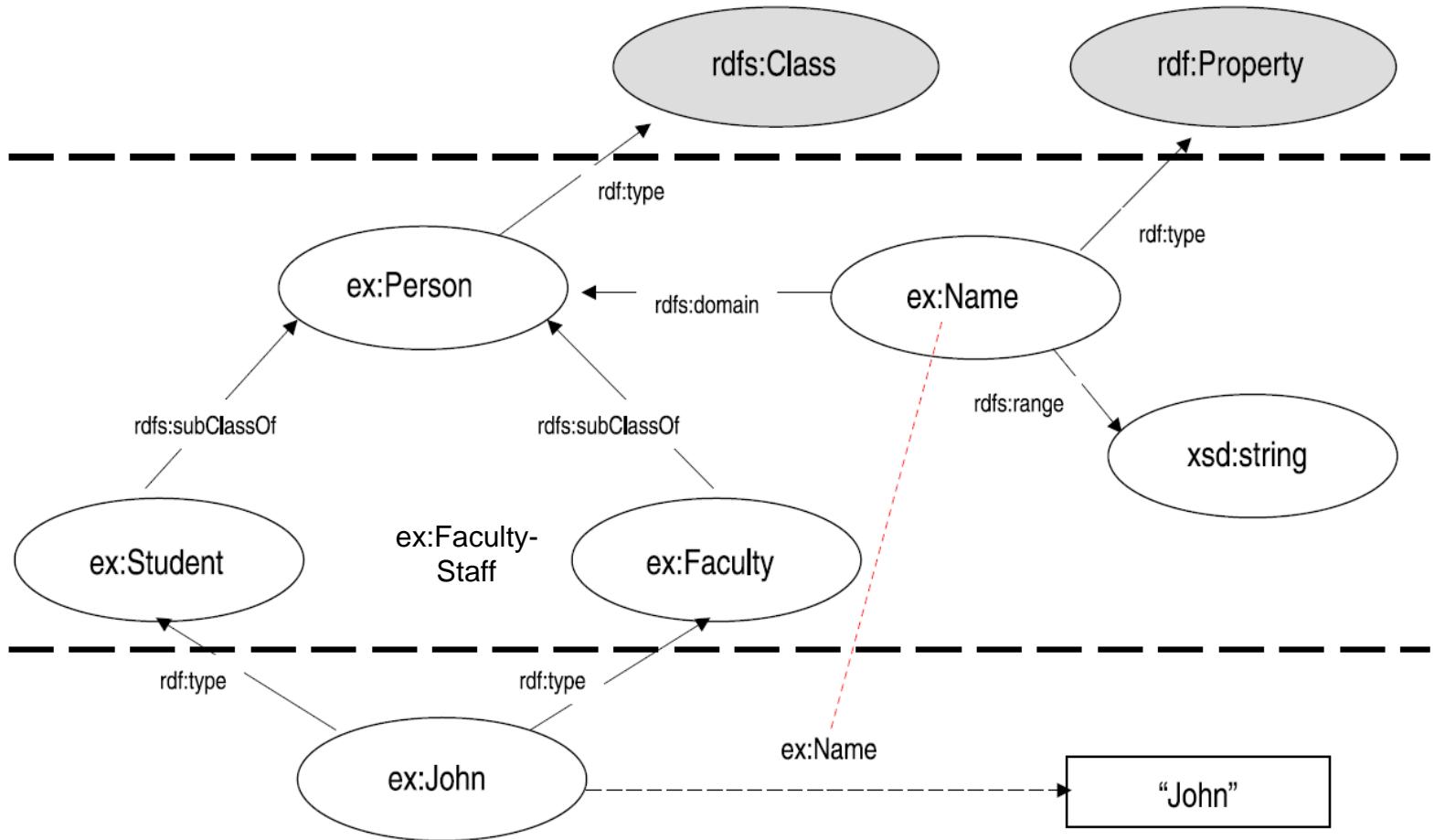
- Subset of RDFS Resources that are properties
  - **Domain:** class associated with property: `rdfs:domain`
  - **Range:** type of the property values: `rdfs:range`
  - Property hierarchy defined through: `rdfs:subPropertyOf`

# RDFS Example

Vocabulary Layer

RDFS Layer

RDF Layer



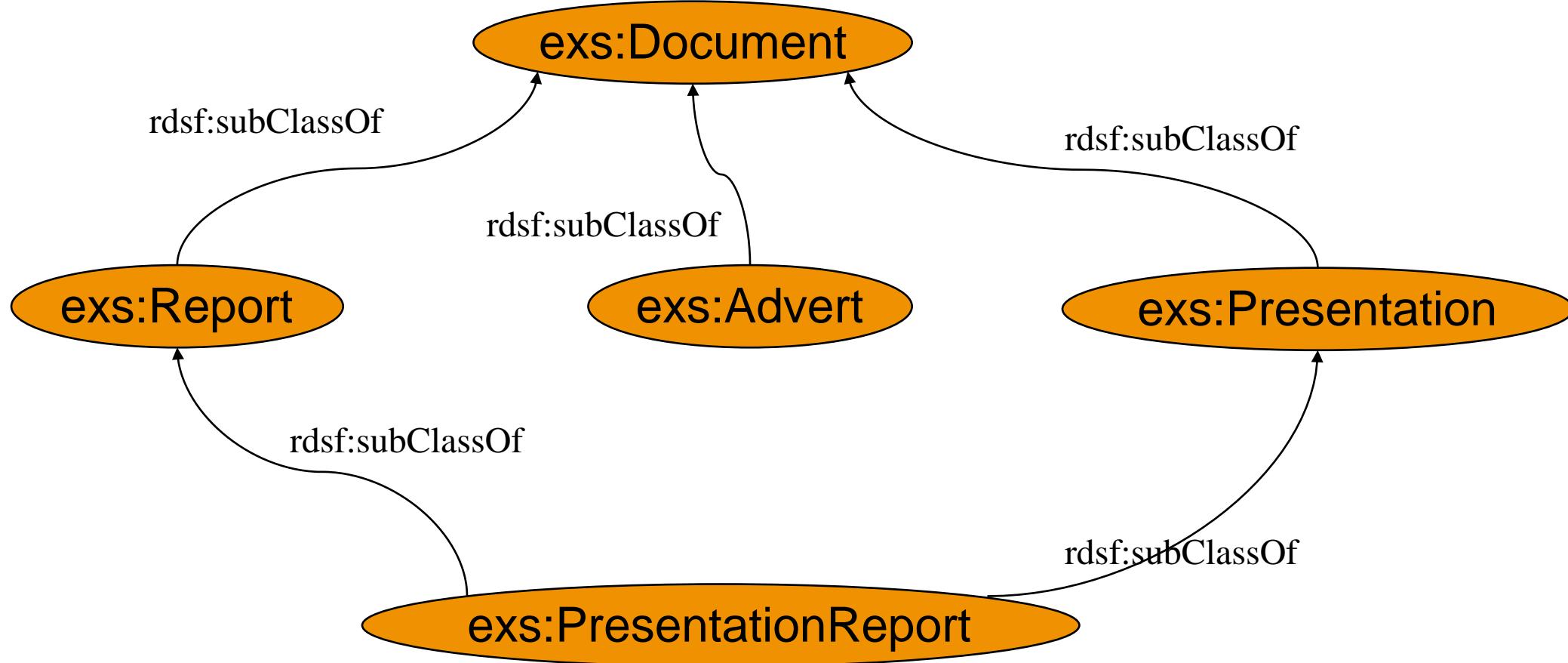
## Triples

exs:Report, rdfs:subClassOf, exs:Document.

exs:Advert, rdfs:subClassOf, exs:Document.

exs:Presentation, rdfs:subClassOf, exs:Document.

exs:PresentationReport, rdfs:subClassOf, exs:Report, exs:Presentation.



## Triples

exs:Researcher rdfs:subClassOf foaf:Person.

foaf:name rdfs:domain foaf:Person.

exs:TimBernersLee a exr:Researcher;

    foaf:name "Tim Berners-Lee";

    foaf:interest <<http://www.w3.org/sw/>>,  
        <<http://www.w3.org/RDF/>>.

exs:BlogEntry rdfs:subClassOf foaf:Document.

<<http://example.org/blog/4>> a exs:BlogEntry;

    exs:mainSubject exs:TimBernersLee;

    foaf:maker, exs:TimBernersLee;

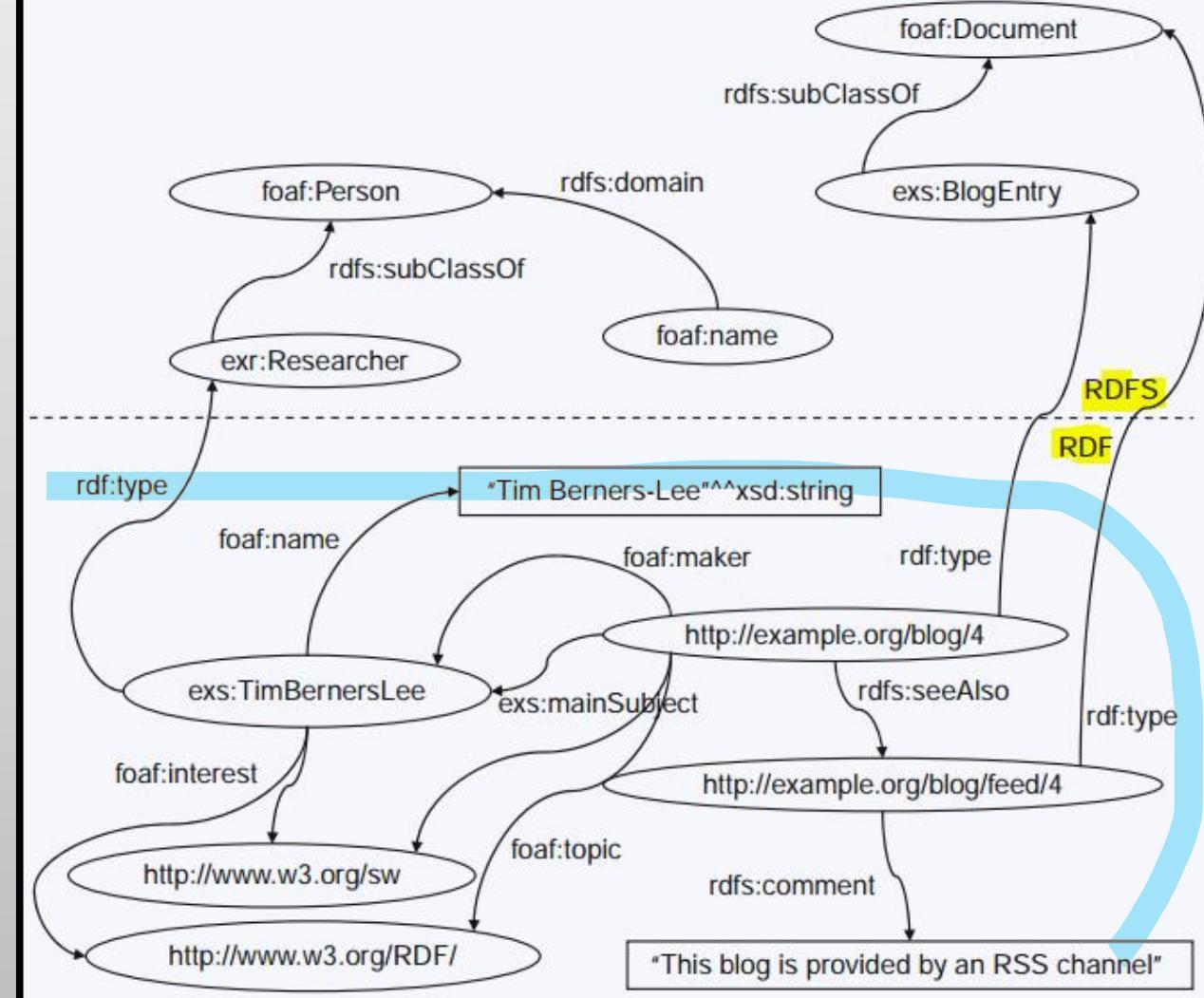
    foaf:topic <<http://www.w3.org/sw/>>,

        <<http://www.w3.org/RDF/>> ;

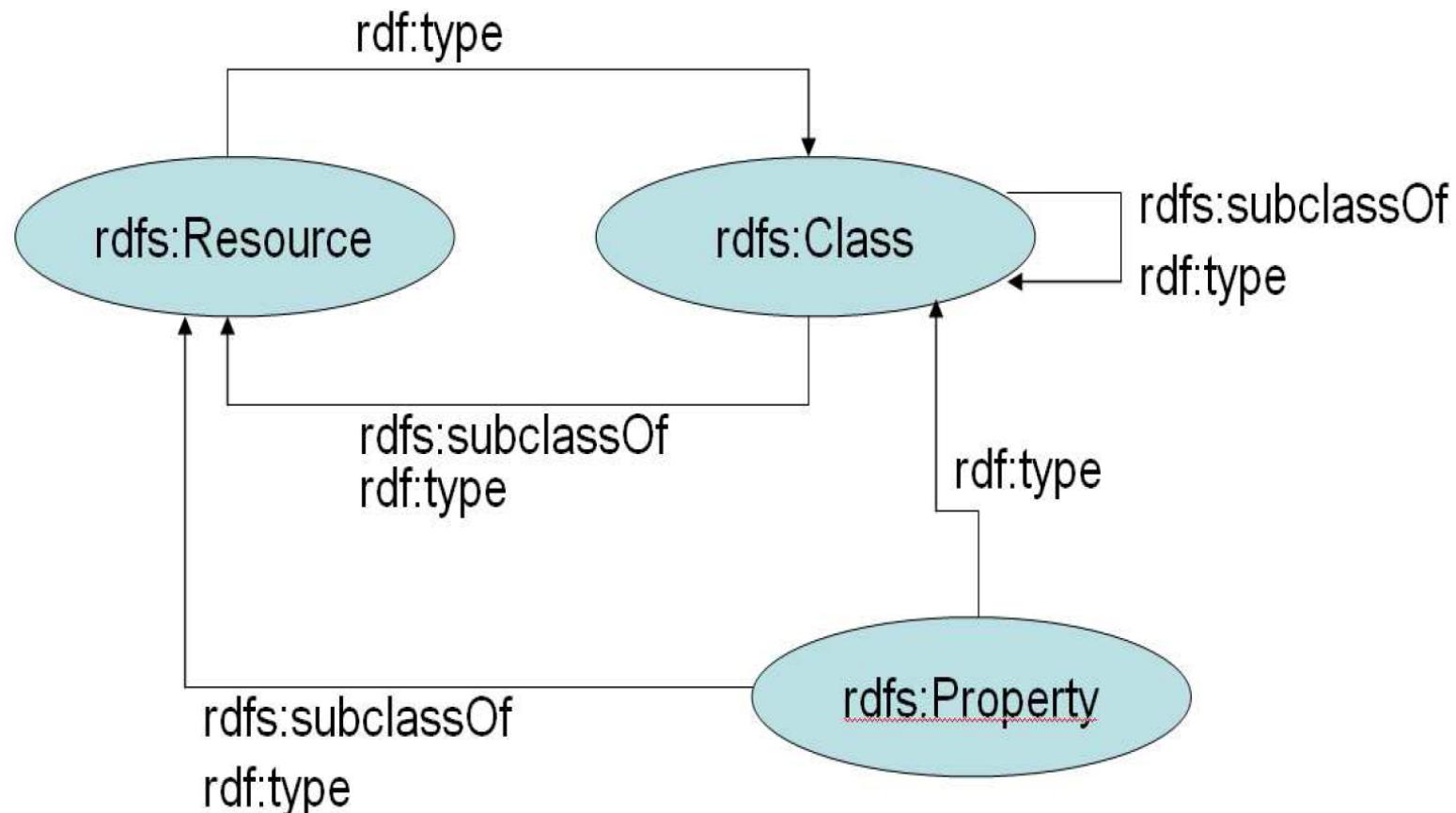
    rdfs:seeAlso <<http://example.org/blog/feed/4>>.

<<http://example.org/blog/feed/4>> a foaf:Document;

    rdfs:comment "This blog is provided by an  
        RSS channel.".



# RDFS Vocabulary Example



# RDFS Metadata Properties

- ◆ **Metadata is “data about data”**
- ◆ **Any meta-data can be attached to a resource, using:**

- ◆ **rdfs:comment**

- Human-readable description of the resource, e.g.
    - `<ex:Person>, rdfs:comment, "A person is any human being"`

- ◆ **rdfs:label**

- Human-readable version of the resource name, e.g.
    - `<ex:Person>, rdfs:label, "Human being"`

- ◆ **rdfs:seeAlso**

- Indicate additional information about the resource, e.g.
    - `<ex:Person>, rdfs:seeAlso, <http://xmlns.com/wordnet/1.6/Human>`

- ◆ **rdfs:isDefinedBy**

- A special kind of rdfs:seeAlso, e.g.
    - `<ex:Person>, rdfs:isDefinedBy, <http://xmlns.com/wordnet/1.6/Human>`

# **RDF(S) SEMANTICS**

- ◆ RDFS stands for RDF schema and is a semantic extension to RDF.
- ◆ It provides mechanisms for describing groups of related resources and the relationships between these resources ([www.w3.org/TR/rdf-schema/](http://www.w3.org/TR/rdf-schema/)).
- ◆ It is a lightweight language to declare and describe the resource types (called classes) and resource relationship and attribute types (called properties).
- ◆ RDFS allows one to name and define vocabularies used in labeling RDF graphs:
  - ◆ naming the classes of existing resources;
  - ◆ naming relation types existing between instances of these classes and giving their signatures, that is, the type of resources they connect.
- ◆ RDFS defines inferences to be applied using these hierarchies of types and the signatures of properties.
- ◆ RDFS allows one to declare the taxonomic skeleton of an ontology in a universal language, with universal identifiers and semantics

# Semantics

- ◆ RDF(S) Semantics
  - ◆ Makes meaning explicit
  - ◆ Defines what follows from an RDF graph
- ◆ Semantic notions
  - ◆ Subgraph
  - ◆ Instance
  - ◆ Entailment

# Subgraph

- ◆  $E$  is a subgraph of  $S$  if and only if  $E$  predicates are a subset of  $S$  predicates

- $\langle \#john, \#hasName, _:johnsname \rangle$
- $\langle _:johnsname, \#firstName, "John" \text{^^xsd:string} \rangle$
- $\langle _:johnsname, \#lastName, "Smith" \text{^^xsd:string} \rangle$

- ◆ Subgraphs:

- $\langle \#john, \#hasName, _:johnsname \rangle$
- $\langle _:johnsname, \#firstName, "John" \text{^^xsd:string} \rangle$
- $\langle _:johnsname, \#lastName, "Smith" \text{^^xsd:string} \rangle$

# Instance

- ◆ **S' is an instance of S if and only if some blank nodes in S are replaced with blank nodes, literals or URIs**
  - ◆ `<#john>, <#hasName>, _:johnsname`
  - ◆ `_:johnsname, <#firstName>, "John"^^xsd:string`
  - ◆ `_:johnsname, <#lastName>, "Smith"^^xsd:string`
- ◆ **Instances:**
  - ◆ `<#john>, <#hasName>, <#abc>`
  - ◆ `<#abc>, <#firstName>, "John"^^xsd:string`
  - ◆ `<#abc>, <#lastName>, "Smith"^^xsd:string`
  - ◆ `<#john>, <#hasName>, _:x`
  - ◆ `_:x, <#firstName>, "John"^^xsd:string`
  - ◆ `_:x, <#lastName>, "Smith"^^xsd:string`
  - ◆ `<#john>, <#hasName>, _:johnsname`
  - ◆ `_:johnsname, <#firstName>, "John"^^xsd:string`
  - ◆ `_:johnsname, <#lastName>, "Smith"^^xsd:string`
- ◆ **Every graph is an instance of itself!**

# Entailment

- ◆ **Entailment or logical implication** is a relation between sentences of a formal language
- ◆ **S entails E if E logically follows from S**
  - Written:  $S \models E$
- ◆ **A graph entails all its subgraphs**
  - If  $S'$  is a subgraph of  $S$ :  $S \models S'$
- ◆ **All instances of a graph S entail S**
  - If  $S''$  is an instance of  $S$ :  $S'' \models S$

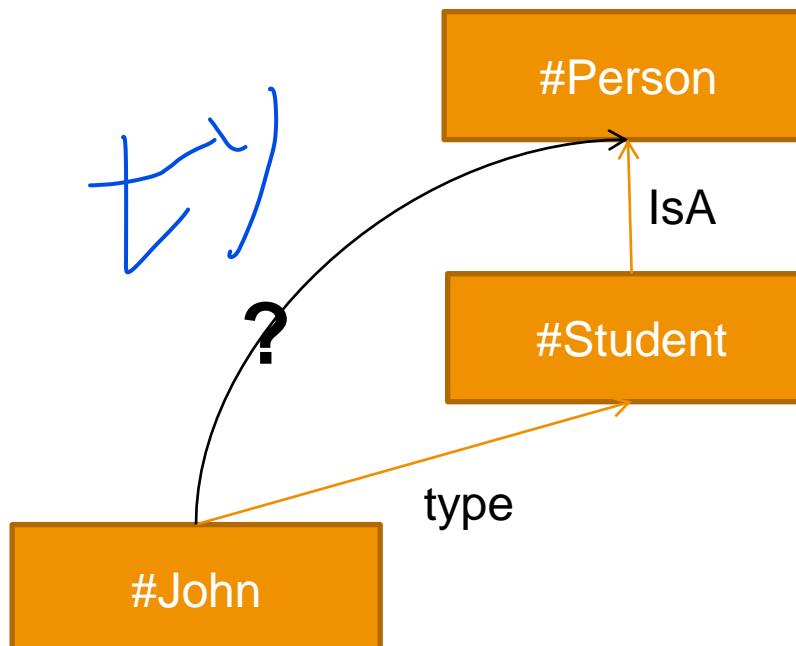
$\models$  Intuition

# RDFS Entailment

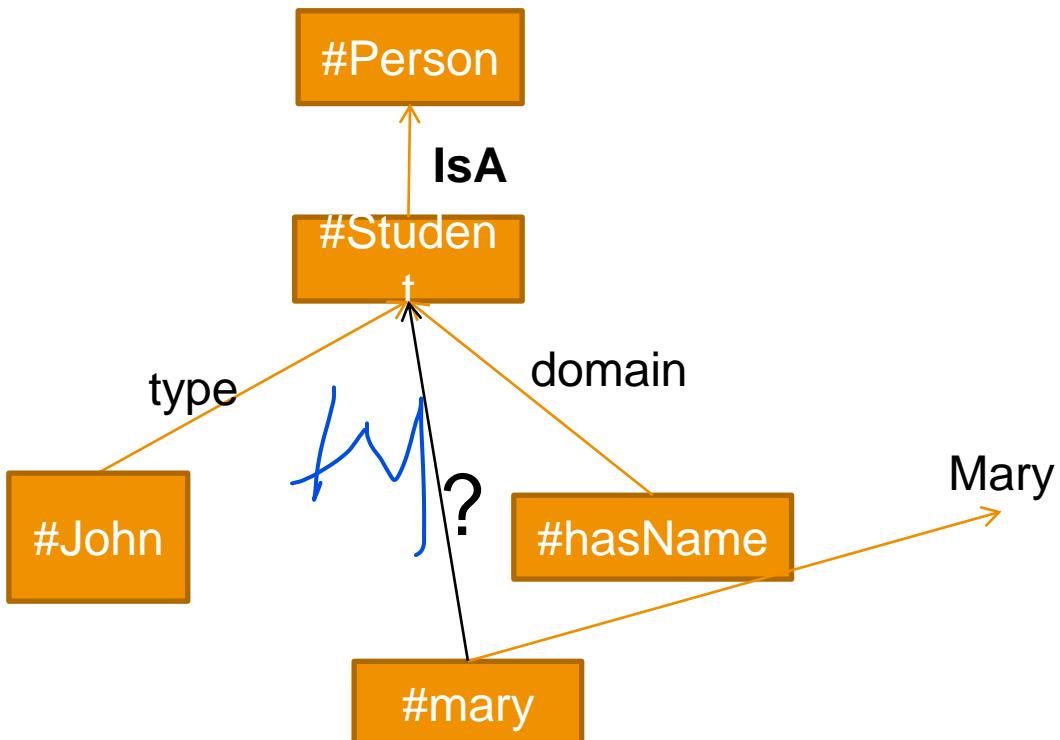
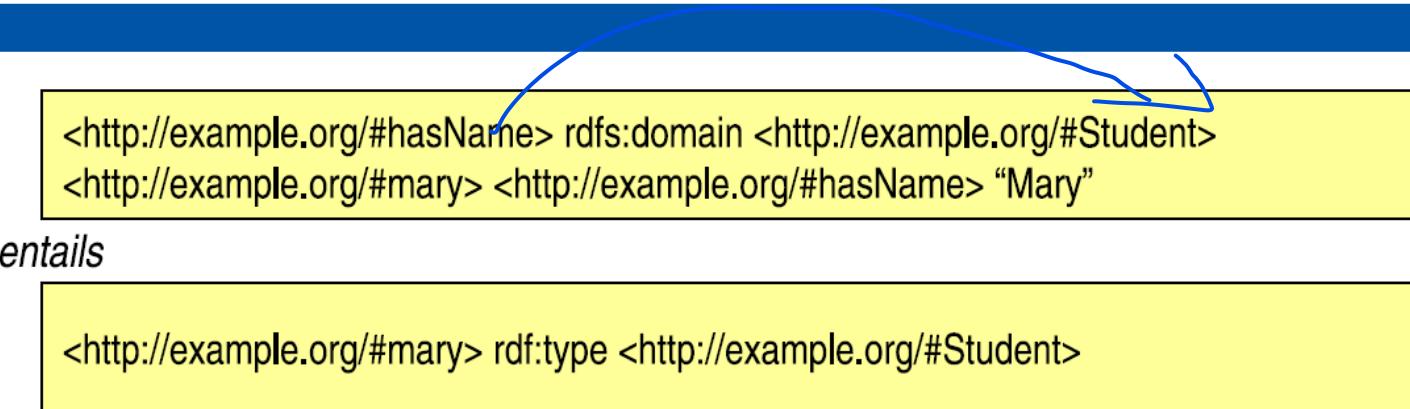
```
<http://example.org/#john> rdf:type <http://example.org/#Student>
<http://example.org/#Student> rdfs:subClassOf <http://example.org/#Person>
```

*entails*

```
<http://example.org/#john> rdf:type <http://example.org/#Person>
```



# RDFS Entailment

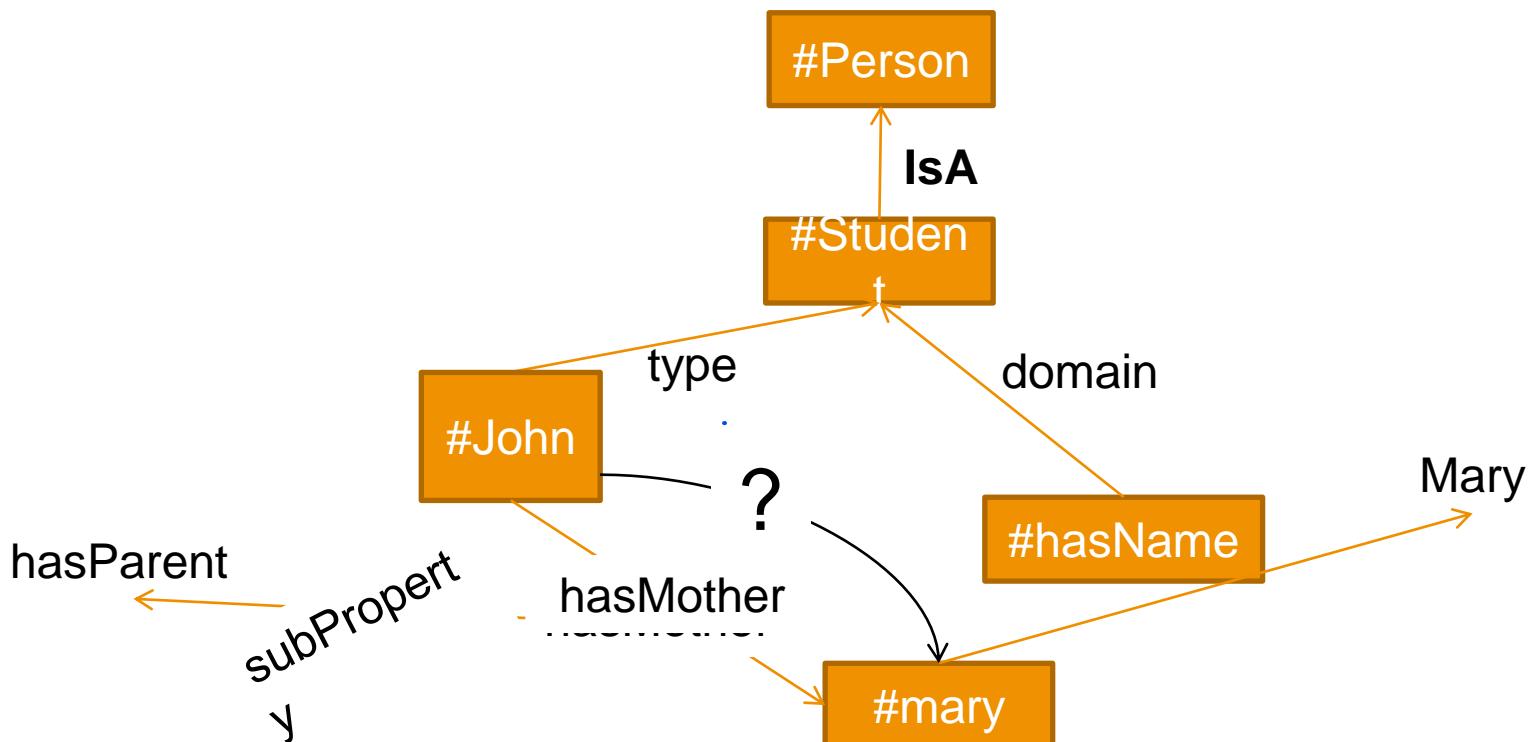


# RDFS Entailment

```
<http://example.org/#john> <http://example.org/#hasMother> <http://example.org/#mary>
<http://example.org/#hasMother> rdfs:subPropertyOf <http://example.org/#hasParent>
```

entails

```
<http://example.org/#john> <http://example.org/#hasParent> <http://example.org/#mary>
```



# Entailment Rules

- ◆ Semantics defined through *entailment rules*
- ◆ Rule:
  - ◆ If  $S$  contains `<triple pattern>` then add `<triple>`
- ◆ Executing all entailment rules yields *realization* of  $S$
- ◆  $S$  entails  $E$  if  $E$  is a subgraph of the realization of  $S$
- ◆ Axiomatic triple are always added

# RDF Entailment

- ◆ if  $E$  contains  $\langle A, B, C \rangle$  then add  
 $\langle B, \text{rdf:type}, \text{rdf:Property} \rangle$
- ◆ if  $E$  contains  $\langle A, B, l \rangle$  ( $l$  is a valid XML literal) then add  
 $\langle _:X, \text{rdf:type}, \text{rdf:XMLLiteral} \rangle$

where  $_:x$  identifies to blank node allocated to  $l$

# RDFS Entailment 1

- **everything in the subject is a resource**
  - if  $E$  contains  $\langle A, B, C \rangle$  then add  $\langle A, \text{ rdf:type, rdfs:Resource} \rangle$
- **every non-literal in the object is a resource**
  - if  $E$  contains  $\langle A, B, C \rangle$  ( $C$  is not a literal) then add  $\langle C, \text{ rdf:type, rdfs:Resource} \rangle$
- **every class is subclass of `rdfs:Resource`**
  - if  $E$  contains  $\langle A, \text{ rdf:type, rdfs:Class} \rangle$  then add  $\langle A, \text{ rdfs:subClassOf, rdfs:Resource} \rangle$
- **inheritance:**
  - if  $E$  contains  $\langle A, \text{ rdf:type, B} \rangle, \langle B, \text{ rdfs:subClassOf, C} \rangle$  then add  $\langle A, \text{ rdf:type, C} \rangle$
- **`rdfs:subClassOf` is transitive**
  - if  $E$  contains  $\langle A, \text{ rdfs:subClassOf, B} \rangle, \langle B, \text{ rdfs:subClassOf, C} \rangle$  then add  $\langle A, \text{ rdfs:subClassOf, C} \rangle$

# RDFS Entailment 2

- **rdfs:subClassOf is reflexive**
  - if E contains  $\langle A, \text{rdf:type}, \text{rdfs:Class} \rangle$  then add  $\langle A, \text{rdfs:subClassOf}, A \rangle$
- **rdfs:subPropertyOf is transitive**
  - if E contains  $\langle A, \text{rdfs:subPropertyOf}, B \rangle, \langle B, \text{rdfs:subPropertyOf}, C \rangle$  then add  $\langle A, \text{rdfs:subPropertyOf}, C \rangle$
- **rdfs:subPropertyOf is reflexive**
  - if E contains  $\langle P, \text{rdf:type}, \text{rdf:Property} \rangle$  then add  $\langle P, \text{rdfs:subPropertyOf}, P \rangle$
- **domain of properties**
  - if E contains  $\langle P, \text{rdfs:domain}, C \rangle, \langle A, P, B \rangle$  then add  $\langle A, \text{rdf:type}, C \rangle$
- **range of properties**
  - if E contains  $\langle P, \text{rdfs:range}, C \rangle, \langle A, P, B \rangle$  then add  $\langle B, \text{rdf:type}, C \rangle$

# RDFS Entailment 3

- ◆ **every literal is a member of rdfs:Literal**
  - ◆ if E contains  $\langle A, B, I \rangle$  ( $I$  is a plain literal) then add  $\langle \_X, \text{rdf:type}, \text{rdfs:Literal} \rangle$
- ◆ **every datatype is subclass of rdfs:Literal**
  - ◆ if E contains  $\langle A, \text{rdf:type}, \text{rdfs:Datatype} \rangle$  then add  $\langle A, \text{rdfs:subClassOf}, \text{rdfs:Literal} \rangle$

# Axiomatic Semantics

- ◆ **the meaning of the modeling primitives of RDF and RDF Schema**
  - By translating into first-order logic
  - make the semantics unambiguous and machine accessible
  - provide a basis for reasoning support by automated reasoners manipulating logical formulas

# The Approach

- ◆ All language primitives in RDF and RDF Schema are represented by constants:
  - ◆ Resource, Class, Property, subClassOf, etc.
- ◆ A few predefined predicates are used as a foundation for expressing relationships between the constants
- ◆ We use predicate logic with equality
- ◆ Variable names begin with ?
- ◆ All axioms are implicitly universally quantified
  
- ◆ Predicate (Subject, Object)
- ◆ Predicate(?S, ?o)

# An Auxiliary Axiomatisation of Lists

- ◆ **Function symbols:**

- ◆ **Function symbols:**
- **nil** (empty list)
- **cons(x,I)** (adds an element to the front of the list)
- **first(I)** (returns the first element)
- **rest(I)** (returns the rest of the list)

- ◆ **Predicate symbols:**

- ◆ **Predicate symbols:**
- **item(x,I)** (tests if an element occurs in the list)
- **list(I)** (tests whether I is a list)

- ◆ **Lists are used to represent containers in RDF**

# Basic Predicates

- ◆ **PropVal(P,R,V)**
  - A predicate with 3 arguments, which is used to represent an RDF statement with resource **R**, property **P** and value **V**
  - An RDF statement (triple) **(P,R,V)** is represented as **PropVal(P,R,V)**.
- ◆ **Type(R,T)**
  - Short for **PropVal(type,R,T)**
  - Specifies that the resource **R** has the type **T**
- ◆ **Type(?r,?t)  $\leftrightarrow$  PropVal(type,?r,?t)**

# RDF Classes

- ◆ **Constants: Class, Resource, Property, Literal**
  - ◆ All classes are instances of **Class**

**Type(Class,Class)**

**Type(Resource,Class)**

**Type(Property,Class)**

**Type(Literal,Class)**

## RDF Classes (2)

- Resource is the most general class: every class and every property is a resource

Type(?p, Property) → Type(?p, Resource)

Type(?c, Class) → Type(?c, Resource)

- The predicate in an RDF statement must be a property

PropVal(?p,?r,?v) → Type(?p,Property)

# The type Property

- ◆ type is a property

PropVal(<sup>N</sup>~~type~~, ~~type~~, Property)

- ◆ type can be applied to resources (domain) and has a class as its value (range)

Type(?r, ?c)  $\rightarrow$  (Type(?r, Resource)  $\wedge$  Type(?c, Class))

<#John; rdf:type; #student>

# The Auxiliary FuncProp Property

- ◆ P is a functional property if, and only if,
  - it is a property, and
  - there are no x, y<sub>1</sub> and y<sub>2</sub> with P(x,y<sub>1</sub>), P(x,y<sub>2</sub>) and y<sub>1</sub>≠y<sub>2</sub>

Type(?p, FuncProp)  $\leftrightarrow$

$$\begin{aligned} & (\text{Type}(\textit{?p}, \text{Property}) \wedge \\ & \forall \textit{?r} \, \forall \textit{?v1} \, \forall \textit{?v2} \\ & \quad (\text{PropVal}(\textit{?p}, \textit{?r}, \textit{?v1}) \wedge \text{PropVal}(\textit{?p}, \textit{?r}, \textit{?v2}) \rightarrow \textit{?v1} = \textit{?v2})) \end{aligned}$$

# Containers

- ◆ Containers are lists:

$\text{Type}(\text{?c}, \text{Container}) \rightarrow \text{list}(\text{?c})$

- ◆ Containers are bags or sequences or alternatives:

$\text{Type}(\text{?c}, \text{Container}) \leftrightarrow$

$(\text{Type}(\text{?c}, \text{Bag}) \vee \text{Type}(\text{?c}, \text{Seq}) \vee \text{Type}(\text{?c}, \text{Alt}))$

- ◆ Bags and sequences are disjoint:

$\neg(\text{Type}(\text{?x}, \text{Bag}) \wedge \text{Type}(\text{?x}, \text{Seq}))$

## Containers (2)

- ◆ For every natural number  $n > 0$ , there is the selector  $_n$ , which selects the  $n^{\text{th}}$  element of a container
- ◆ It is a functional property:

$\text{Type}(_n, \text{FuncProp})$

- ◆ It applies to containers only:

$\text{PropVal}(_n, ?c, ?o) \rightarrow \text{Type}(?c, \text{Container})$

# Subclass

- ◆ `subClassOf` is a property:

`Type(subClassOf,Property)`

- ◆ If a class C is a subclass of a class C', then all instances of C are also instances of C':

`PropVal(subClassOf,?c,?c') \leftrightarrow`

$$(\text{Type}(\text{?c}, \text{Class}) \wedge \text{Type}(\text{?c}', \text{Class}) \wedge \\ \forall \text{?x } (\text{Type}(\text{?x}, \text{?c}) \rightarrow \text{Type}(\text{?x}, \text{?c}')))$$

`<C; rdfs:subClassOf; C'>????`

`<#Person; rdss:subClassOf; #student>???`

# Subproperty

- $P$  is a subproperty of  $P'$ , if  $P'(x,y)$  is true whenever  $P(x,y)$  is true:

Type(subPropertyOf,Property)

PropVal(subPropertyOf,?p,?p')  $\leftrightarrow$   
 $(Type(?p,Property) \wedge Type(?p',Property) \wedge$   
 $\forall ?r \ \forall ?v \ (PropVal(?p,?r,?v) \rightarrow PropVal(?p',?r,?v)))$

# Domain and Range

- ♦ If the domain of P is D, then for every  $P(x,y)$ ,  $x \in D$

$\text{PropVal}(\text{domain}, ?p, ?d) \rightarrow$

$\forall ?x \forall ?y (\text{PropVal}(?p, ?x, ?y) \rightarrow \text{Type}(?x, ?d))$

- ♦ If the range of P is R, then for every  $P(x,y)$ ,  $y \in R$

$\text{PropVal}(\text{range}, ?p, ?r) \rightarrow$

$\forall ?x \forall ?y (\text{PropVal}(?p, ?x, ?y) \rightarrow \text{Type}(?y, ?r))$

# Problems with RDFS

- ◆ **RDFS too weak to describe resources in sufficient detail**

- ◆ **RDFS too weak to describe resources in sufficient detail**
  - ◆ No localized range and domain constraints
    - Can't say that the range of hasChild is person when applied to persons and elephant when applied to elephants
  - ◆ No existence/cardinality constraints
    - Can't say that all instances of person have a mother that is also a person, or that persons have exactly 2 parents
  - ◆ No transitive, inverse or symmetrical properties
    - Can't say that isPartOf is a transitive property, that hasPart is the inverse of isPartOf or that touches is symmetrical
  - ◆ Disjointness of classes: Sometimes we wish to say that classes are disjoint (e.g. male and female). But in RDF Schema we can only state subclass relationships, e.g., female is a subclass of person.
  - ◆ 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 but RDF Schema does not allow such definition



**Web Ontology Language : OWL**

# 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
- ◆ **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  $A$

# 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

# Three Species of OWL

- ◆ W3C's Web 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 Full

- ◆ It uses all the OWL languages primitives
- ◆ It allows the combination of these primitives in arbitrary ways with RDF and RDF Schema
- ◆ OWL Full is fully upward-compatible with RDF, both syntactically and semantically
- ◆ Any legal RDF document is also a legal OWL Full document, and any valid RDF/RDF Schema conclusion is also a valid OWL Full conclusion.
- ◆ OWL Full is so powerful that it is undecidable
  - ◆ – No complete (or efficient) reasoning support

- ◆ **OWL DL (Description Logic) is a sublanguage of OWL Full that restricts application of the constructors from OWL and RDF**
  - Application of OWL's constructors' to each other is disallowed
  - Therefore it corresponds to a well studied description logic
- ◆ **OWL DL permits efficient reasoning support**
- ◆ **But we lose full compatibility with RDF:**
  - Not every RDF document is a legal OWL DL document.
  - Every legal OWL DL document is a legal RDF document.

- ◆ An even further restriction limits OWL DL to a subset of the language constructors
  - ◆ E.g., OWL Lite excludes enumerated classes, disjointness statements, and arbitrary cardinality.
- ◆ The advantage of this is a language that is easier to
  - ◆ grasp, for users
  - ◆ implement, for tool builders
- ◆ The disadvantage is restricted expressivity

# 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 that does not follow the RDF conventions and is thus more easily read by human users.
  - ◆ 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

- ◆ OWL documents are usually called OWL ontologies and are RDF documents. The root element of an OWL ontology is an **rdf:RDF** element, which also specifies a number of namespaces:

```
<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/XMLSchema#">
```

- ◆ An OWL ontology may start with a collection of assertions for housekeeping purposes using **owl:Ontology** element, which contains comments, version control, and inclusion of other ontologies.

## OWL XML/RDF Syntax: Header (2)

### Example:

```
<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**, lists other ontologies whose content is assumed to be part of the current ontology.
- ◆ **owl:imports is a transitive property**, if ontology A imports ontology B, and ontology B imports ontology C, then ontology A also imports ontology C.

# Classes

- ◆ **Classes are defined using owl:Class**

- ◆ owl:Class is a subclass of rdfs:Class

```
<owl:Class rdf:ID="associateProfessor">  
    <rdfs:subClassOf rdf:resource="#academicStaffMember"/>
```

```
</owl: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.
- ◆ **OWL does not have any predefined data types, nor does it provide special definition facilities. Instead, it allows one to use XML Schema data types, thus making use of the layered architecture of the Semantic Web**

# 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/XMLSchema#nonNegativeInteger"/>  
</owl:DatatypeProperty>
```

# Object Properties

- ◆ User-defined data types will usually be collected in an XML schema and then used in an OWL ontology.

- ◆ Example:

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

- ◆ More than one domain and range may be declared. In this case the intersection of the domains, respectively ranges, is taken.

# 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

- Equivalence of properties can be defined through the use of the element `owl:equivalentProperty`.

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

# Property Restrictions

- ◆ **rdfs:subClassOf** is used to specify a class C as a subclass of another class C'; then every instance of C is also an instance of C'.
- ◆ Suppose we wish to declare, instead, that the class C satisfies certain conditions, that is, 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. That is exactly how it is done in OWL.
- ◆ Note that, in general, 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,...)
- ◆ The other type defines restrictions on the kinds of values the property may take
  - ◆ owl:allValuesFrom specifies universal quantification
  - ◆ owl:hasValue specifies a specific value
  - ◆ owl:someValuesFrom specifies existential quantification

## **owl:allValuesFrom**

- ◆ Represent the following text using OWL “First year courses to be taught by professors only”

```
<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:hasValue

- Represent the statement “Courses on mathematics are taught by David Billington”

```
<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:someValuesFrom

- ◆ “ To declare that all academic staff members must teach at least one undergraduate course”

```
<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>
```

# 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

```
<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 Voter Id is assigned to one person only.

## 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

- ◆ One can combine classes using Boolean operations (union, intersection, complement)
- ◆ Example: let us say courses and staff members are disjoint classes

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

- ◆ The same can be done using owl:disjointWith

## Boolean Combinations (2)

- It is possible for a staff member to be a student

```
<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

## 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

- Let define admin staff to be those staff members that are neither faculty nor technical support staff

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

## **owl:oneOf**

- ◆ **Enumerations with owl:oneOf**

```
<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>
```

# Declaring Instances

- ◆ Instances of classes are declared as in RDF:

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

- ◆ Providing further information

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

# No Unique-Names Assumption

- ◆ Unlike typical DBMS, 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
- ◆ Represent: each course is taught by atmost one staff member, and that a given course is taught by two staff members

```
<owl:ObjectProperty rdf:ID="isTaughtBy">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
</owl:ObjectProperty>
<course rdf:ID="CIT1111">
  <isTaughtBy rdf:resource="#949318"/>
  <isTaughtBy rdf:resource="#949352"/>
</course>
```

# Distinct Objects

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

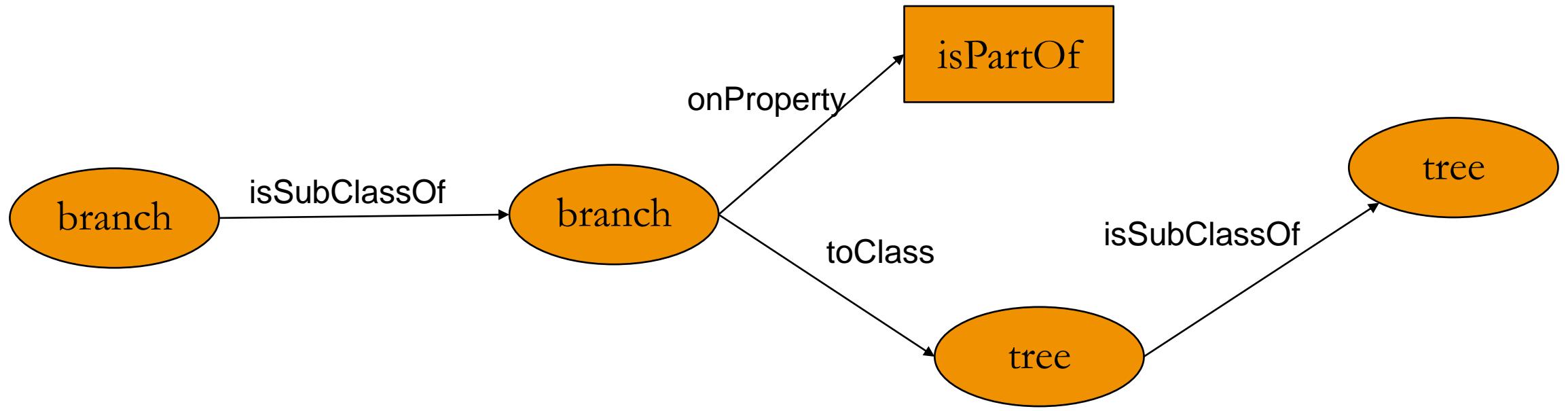
```
<lecturer rdf:about="949318">  
  <owl:differentFrom rdf:resource="949352"/>  
</lecturer>
```

## Distinct Objects (2)

- ◆ **owl:allDifferent asserts pairwise inequality of all individuals in a list**

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

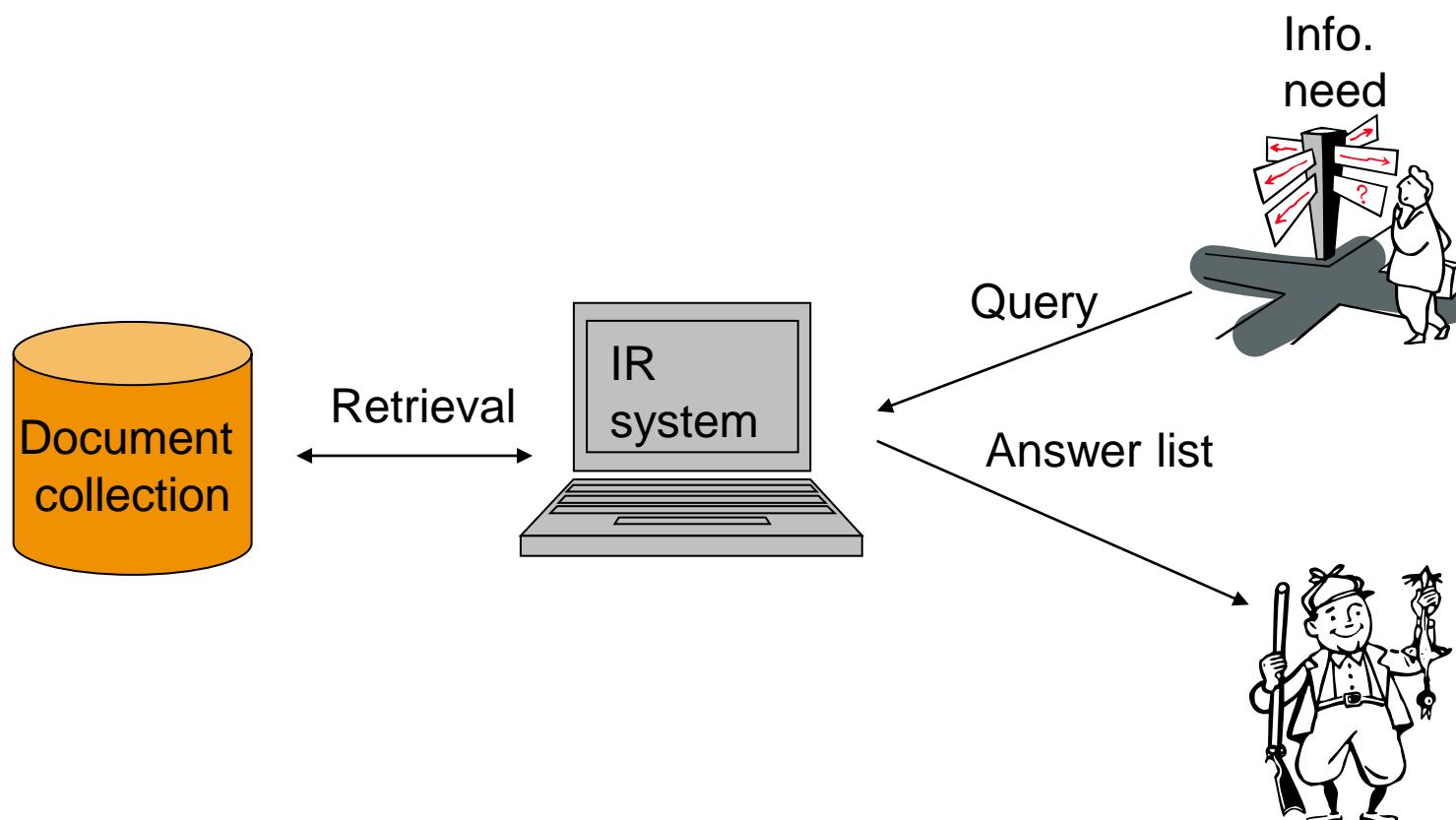
- ◆ **Note that owl:distinctMembers can only be used in combination with owl:allDifferent.**



# **Text Information Retrieval**

# The problem of IR

- Goal = find documents *relevant* to an information need from a large document set



# Example

Google Addis Ababa University Department of Computer science

All Maps Images News Videos More Settings Tools

About 127,000 results (0.75 seconds)

**Department of Computer Science | College of Natural Sciences**  
www.aau.edu.et/cns/department-of-computer-science/ ▾  
Overview Programs Staff Seminars and Conferences Research Consultancy and trainings Facilities Partnerships and...

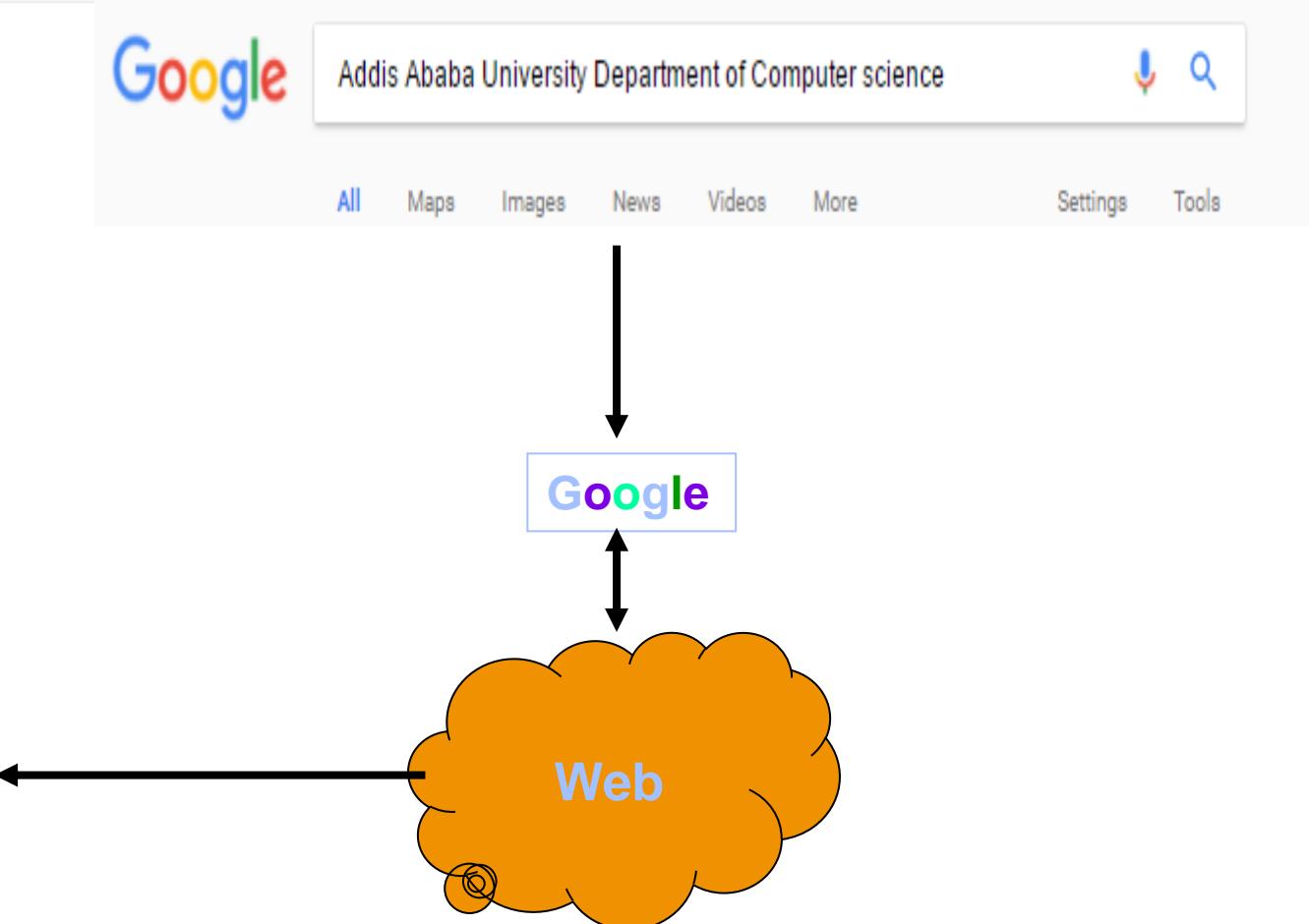
**Computer Science | Addis Ababa University**  
www.aau.edu.et/natural-sciences/computer-science/ ▾  
College/Institution: CNS Department/School/Center: Department of Computer Science Program title: B.Sc. in Computer Science Program duration (in years)

**Overview | College of Natural Sciences**  
www.aau.edu.et/cns/department-of-computer-science/overview-of-computer/ ▾  
In addition, the Department also started to offer Computer Science as a minor ... of the University Senate in August 2002, the Computer Science program was ...

**Department of Computer Science, Addis Ababa University - Home ...**  
https://www.facebook.com › Places › Addis Ababa, Ethiopia ▾  
★★★★★ Rating: 3.5 - 13 votes  
Department of Computer Science, Addis Ababa University, Addis Ababa, Ethiopia. 1642 likes · 20 talking about this · 41 were here. The Department of...

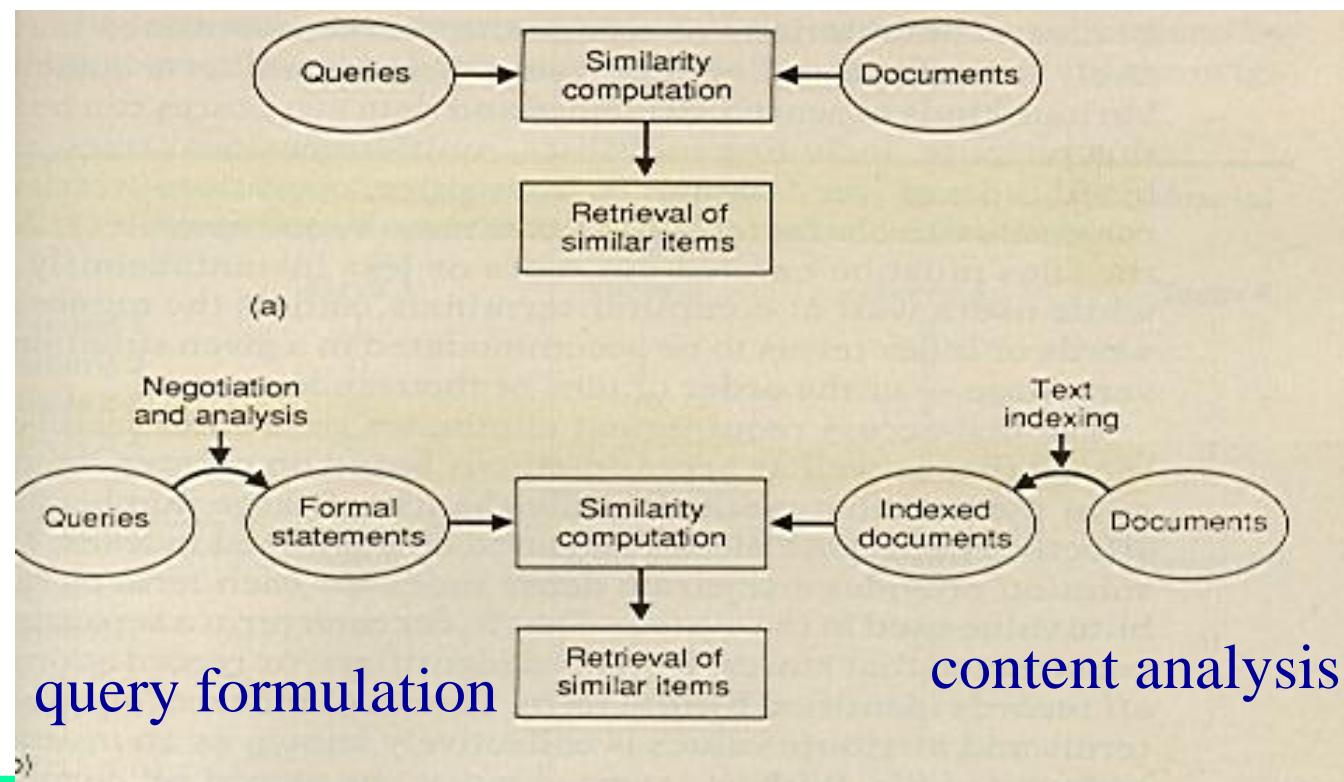
**Addis Ababa University - Department of Computer Science**  
https://www.researchgate.net/.../Addis\_Ababa\_University/department/Department\_of\_Co... ▾  
Find researchers and browse publications, fulltexts, contact details and general information related to the Department of Computer Science at Addis Ababa ...

**Addis Ababa University - Department of Computer Science - Members**  
https://www.researchgate.net/.../Addis\_Ababa\_University/department/Department\_of\_Co...  
Find researchers and browse publications, fulltexts, contact details and general information related to the Department of Computer Science at Addis Ababa ...

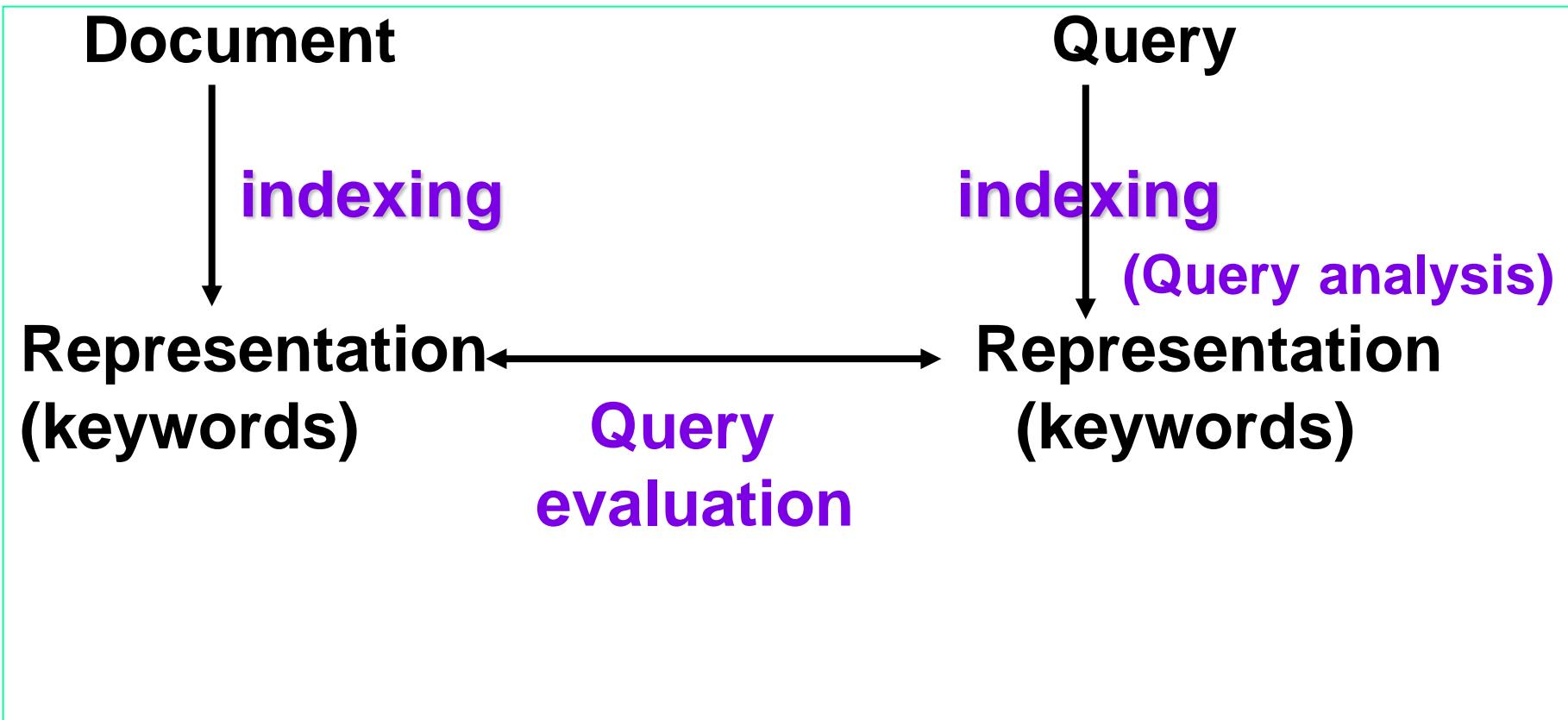


# Text Retrieval System

- Content identifiers (keywords, index terms, descriptors) characterize the stored texts.



# Indexing-based IR



# Main problems in IR

---

- ◆ **Document and query indexing**
  - How to best represent their contents?
- ◆ **Query evaluation (or retrieval process)**
  - To what extent does a document correspond to a query?
- ◆ **System evaluation**
  - How good is a system?
  - Are the retrieved documents relevant? (precision)
  - Are all the relevant documents retrieved? (recall)

# Possible Representation

---

- ◆ **Document representation**

- ◆ **Document representation**
  - unweighted index terms (term vectors)
  - weighted index terms
  - ...

- ◆ **Query**

- ◆ **Query**
  - unweighted or weighted index terms
  - Boolean combinations (or, and, not)
  - ...

- ◆ **Search operation must be effective**

# Document indexing

- ◆ Goal = Find the important meanings and create an internal representation
- ◆ Factors to consider:
  - Accuracy to represent meanings (semantics)
  - Exhaustiveness (cover all the contents)
  - Facility for computer to manipulate
- ◆ What is the best representation of contents?
  - Char. string (char trigrams): not precise enough
  - Word: good coverage, not precise
  - Phrase: poor coverage, more precise
  - Concept: poor coverage, precise



# Inverted Files

---

- ◆ File is represented as an array of indexed documents.

	Term 1	Term 2	Term 3	Term 4
Doc 1	1	1	0	1
Doc 2	0	1	1	1
Doc 3	1	0	1	1
Doc 4	0	0	1	1

# Inverted-file process

---

- ◆ The document-term array is inverted (transposed).

	Doc 1	Doc 2	Doc 3	Doc 4
Term 1	1	0	1	0
Term 2	1	1	0	0
Term 3	0	1	1	1
Term 4	1	1	1	1

# Term Weighting Model

- ◆ Use of bag of words/ terms is the common approach of text information retrieval.
- ◆ Term importance is the main measurement in this approach as every single term may have different importance (weight) to the information domain
- ◆ Each document  $d_i$  is represented as collection of weighted terms e.g.  $d_i = \langle t_1, \dots, t_n \rangle$  Weight can be terms frequency – tf, idf, tf-idf depending on the application.

Doc	terms
d1	t6
d2	t1, t2, t5
d3	t2 ,t5, t8
d4	t1, t4, t6
d5	t1, t7
d6	t3, t7
d7	t1, t3

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}$$

# Term Weighting Model

- Weight can be term frequency – tf,

$$tfidf_{t,d} = tf_{t,d} * \log\left(\frac{|D|}{df_t}\right)$$

Doc	terms
d1	t6
d2	t1, t2, t5
d3	t2 ,t5, t8
d4	t1, t4, t6
d5	t1, t7
d6	t3, t7
d7	t1, t3

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

# weighting schema

---

- ◆ **tf = term frequency**

- frequency of a term/keyword in a document

**The higher the tf, the higher the importance (weight) for the doc.**

- ◆ **df = document frequency**

- no. of documents containing the term
  - distribution of the term

- ◆ **idf = inverse document frequency**

- the unevenness of term distribution in the corpus
  - the specificity of term to a document

**The more the term is distributed evenly, the less it is specific to a document**

$$\text{weight}(t,D) = \text{tf}(t,D) * \text{idf}(t)$$

## Some common *tf\*idf* schemes

---

- ◆  $tf(t, D) = freq(t, D)$                        $idf(t) = \log(N/n)$
- ◆  $tf(t, D) = \log[freq(t, D)]$                    $n = \#docs$  containing  $t$
- ◆  $tf(t, D) = \log[freq(t, D)] + 1$      $N = \#docs$  in corpus
- ◆  $tf(t, D) = freq(t, d) / Max[f(t, d)]$

$$\text{weight}(t, D) = tf(t, D) * idf(t)$$

- ◆ Normalization: Cosine normalization,  $/max$ , ...

# Stopwords / Stoplist

- ◆ function words do not bear useful information for IR: of, in, about, with, I, although, ...
- ◆ Stoplist: contain stopwords, not to be used as index
  - Prepositions
  - Articles
  - Pronouns
  - Some adverbs and adjectives
  - Some frequent words (e.g. document)
- ◆ The removal of stopwords usually improves IR effectiveness
- ◆ A few “standard” stoplists are commonly used.

# Stemming

---

## ◆ Reason:

- Different word forms may bear similar meaning (e.g. search, searching): create a “standard” representation for them

## ◆ Stemming:

- Removing some endings of word

computer  
compute  
computes  
computing  
computed  
computation

{ comput

# Porter algorithm

- ◆ **Step 1: plurals and past participles**
    - ◆ SSES -> SS caresses -> caress
    - ◆ (\*v\*) ING -> motoring -> motor
  - ◆ **Step 2: adj->n, n->v, n->adj, ...**
    - ◆ (m>0) OUSNESS -> OUS callousness -> callous
    - ◆ (m>0) ATIONAL -> ATE relational -> relate
  - ◆ **Step 3:**
    - ◆ (m>0) ICATE -> IC triplicate -> triplic
  - ◆ **Step 4:**
    - ◆ (m>1) AL -> revival -> reviv
    - ◆ (m>1) ANCE -> allowance -> allow
  - ◆ **Step 5:**
    - ◆ (m>1) E -> probate -> probat
    - ◆ (m > 1 and \*d and \*L) -> single letter controll -> control

# Lemmatization

- ◆ transform to standard form according to syntactic category.

E.g. verb + *ing* → verb

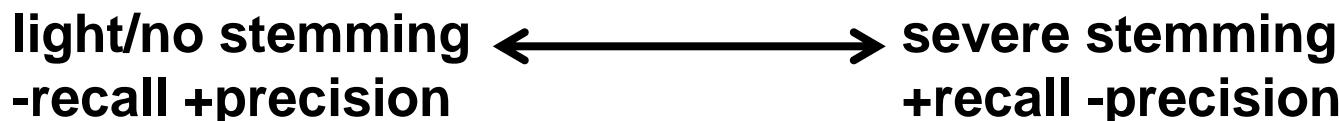
noun + *s* → noun

- Need POS tagging
- More accurate than stemming, but needs more resources

- ◆ crucial to choose stemming/ lemmatization rules

noise v.s. recognition rate

- ◆ compromise between precision and recall



# Result of indexing

---

- ◆ **Each document is represented by a set of weighted keywords (terms):**

$$D_1 \rightarrow \{(t_1, w_1), (t_2, w_2), \dots\}$$

e.g.  $D_1 \rightarrow \{(\text{comput}, 0.2), (\text{architect}, 0.3), \dots\}$

$$D_2 \rightarrow \{(\text{comput}, 0.1), (\text{network}, 0.5), \dots\}$$

- ◆ **Inverted file:**

$$\text{comput} \rightarrow \{(D_1, 0.2), (D_2, 0.1), \dots\}$$

Inverted file is used during retrieval for higher efficiency.

- ◆ **The problems underlying retrieval**

- Retrieval model
    - How is a document represented with the selected keywords?
    - How are document and query representations compared to calculate a score?

## ◆ **1-word query:**

The documents to be retrieved are those that include the word

- Retrieve the inverted list for the word
- Sort in decreasing order of the weight of the word

## ◆ **Multi-word query?**

- Combining several lists
- How to interpret the weight? => IR Model

## ◆ Matching score model

- Document D = a set of weighted keywords
- Query Q = a set of non-weighted keywords
- $R(D, Q) = \sum_i w(t_i, D)$

where  $t_i$  is in Q.

# Boolean model

- Document = Logical conjunction of keywords
- Query = Boolean expression of keywords
- $R(D, Q) = D \rightarrow Q$

e.g.  $D = t_1 \wedge t_2 \wedge \dots \wedge t_n$   
 $Q = (t_1 \wedge t_2) \vee (t_3 \wedge \neg t_4)$   
 $D \rightarrow Q$ , thus  $R(D, Q) = 1$ .

## Problems:

- $R$  is either 1 or 0 (unordered set of documents)
- many documents or few documents
- End-users cannot manipulate Boolean operators correctly  
E.g. documents about *kangaroos* and *koalas*

# Extensions to Boolean model (for document ordering)

- ◆  $D = \{..., (t_i, w_i), ...\}$ : **weighted keywords**

- ◆ **Interpretation:**

- ◆  $D$  is a member of class  $t_i$  to degree  $w_i$ .

- ◆ **A possible Evaluation:**

$$R(D, Q_1 \wedge Q_2) = \min(R(D, Q_1), R(D, Q_2));$$

$$R(D, Q_1 \vee Q_2) = \max(R(D, Q_1), R(D, Q_2));$$

$$R(D, \neg Q_1) = 1 - R(D, Q_1).$$

## Boolean Query ...

---

- ◆ Transform a Boolean expression into ***disjunctive normal form.***
- ◆ For example:  $Q = T_1 \text{ and } (T_2 \text{ or } T_3)$   
=  $(T_1 \text{ and } T_2) \text{ or } (T_1 \text{ and } T_3)$
- ◆ For each conjunct, compute the minimum term weight of any document term in that conjunct.
- ◆ The document weight is the maximum of all the conjunct weights.

## Boolean Query ...

- ◆ Example:  $Q=(T1 \text{ and } T2) \text{ or } T3$

Document Vectors	Conjunct Weights $(T1 \text{ and } T2)$	$(T3)$	Query Weight $(T1 \text{ and } T2) \text{ or } T3$
$D1=(T1,0.2;T2,0.5;T3,0.6)$	0.2	0.6	0.6
$D2=(T1,0.7;T2,0.2;T3,0.1)$	0.2	0.1	0.2

D1 is preferred.

# Vector space model

---

- ◆ **Vector space = all the keywords encountered**

$\langle t_1, t_2, t_3, \dots, t_n \rangle$

- ◆ **Document**

$D = \langle a_1, a_2, a_3, \dots, a_n \rangle$

$a_i = \text{weight of } t_i \text{ in } D$

- ◆ **Query**

$Q = \langle b_1, b_2, b_3, \dots, b_n \rangle$

$b_i = \text{weight of } t_i \text{ in } Q$

- ◆  **$R(D, Q) = \text{Sim}(D, Q)$**

# Matrix representation

Document space	$t_1$	$t_2$	$t_3$	...	$t_n$	← Term vector space
$D_1 a_{11}$	$a_{12}$	$a_{13}$	...		$a_{1n}$	
$D_2 a_{21}$	$a_{22}$	$a_{23}$	...		$a_{2n}$	
$D_3 a_{31}$	$a_{32}$	$a_{33}$	...		$a_{3n}$	
...						
$D_m a_{m1}$	$a_{m2}$	$a_{m3}$	...		$a_{mn}$	
$Q$	$b_1$	$b_2$	$b_3$	...	$b_n$	

# Some formulas for Sim

**Dot Product**

$$Sim(D, Q) = \sum (a_i * b_i)$$

**Cosine**

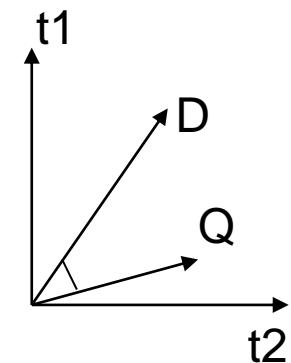
$$Sim(D, Q) = \frac{\sum_i (a_i * b_i)}{\sqrt{\sum_i a_i^2 * \sum_i b_i^2}}$$

**Dice**

$$Sim(D, Q) = \frac{2 \sum_i (a_i * b_i)}{\sum_i a_i^2 + \sum_i b_i^2}$$

**Jaccard**

$$Sim(D, Q) = \frac{\sum_i (a_i * b_i)}{\sum_i a_i^2 + \sum_i b_i^2 - \sum_i (a_i * b_i)}$$



## Implementation (space)

---

- ◆ Matrix is very sparse: a few 100s terms for a document, and a few terms for a query, while the term space is large (~100k)
- ◆ Stored as:  
 $D_1 \rightarrow \{(t_1, a_1), (t_2, a_2), \dots\}$   
 $t_1 \rightarrow \{(D_1, a_1), \dots\}$

# Implementation (time)

---

- ◆ The implementation of VSM with dot product:

- ◆ Naïve implementation:  $O(m*n)$
  - ◆ Implementation using inverted file:

Given a query =  $\{(t1,b1), (t2,b2)\}$ :

1. find the sets of related documents through inverted file for  $t1$  and  $t2$
2. calculate the score of the documents to each weighted term

$$(t1,b1) \rightarrow \{(D1,a1 * b1), \dots\}$$

3. combine the sets and sum the weights ( $\Sigma$ )

- ◆  $O(|Q|*n)$

## Other similarities

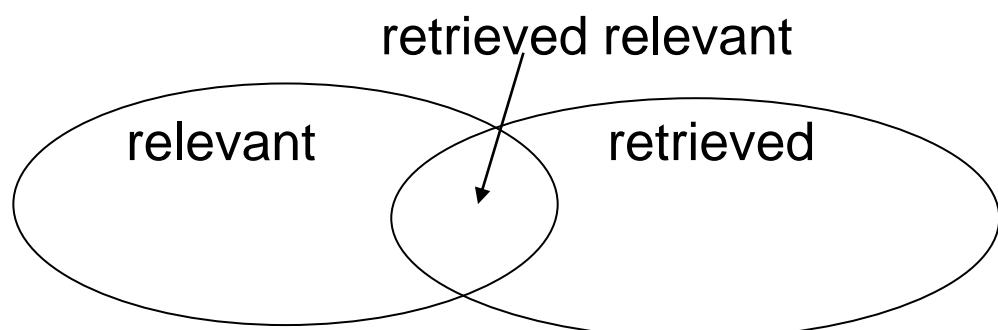
- ◆ Cosine:

$$Sim(D, Q) = \frac{\sum_i (a_i * b_i)}{\sqrt{\sum_j a_j^2 * \sum_j b_j^2}} = \sum_i \frac{a_i}{\sqrt{\sum_j a_j^2}} \frac{b_i}{\sqrt{\sum_j b_j^2}}$$

- use  $\sqrt{\sum_j a_j^2}$  and  $\sqrt{\sum_j b_j^2}$  to normalize the weights after indexing
- Dot product  
(Similar operations do not apply to Dice and Jaccard)

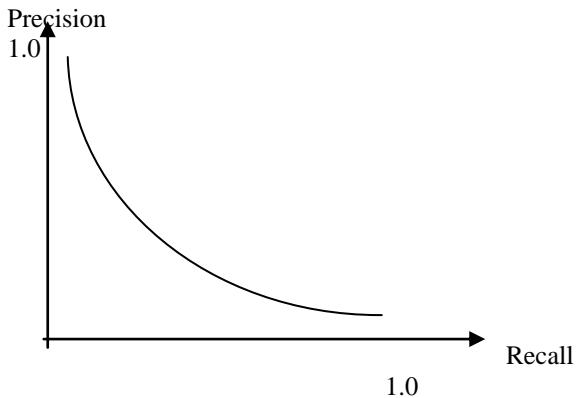
# System evaluation

- ◆ **Efficiency: time, space**
- ◆ **Effectiveness:**
  - ◆ How is a system capable of retrieving relevant documents?
  - ◆ Is a system better than another one?
- ◆ **Metrics often used (together):**
  - ◆ Precision = retrieved relevant docs / retrieved docs
  - ◆ Recall = retrieved relevant docs / relevant docs



# General form of precision/recall

---

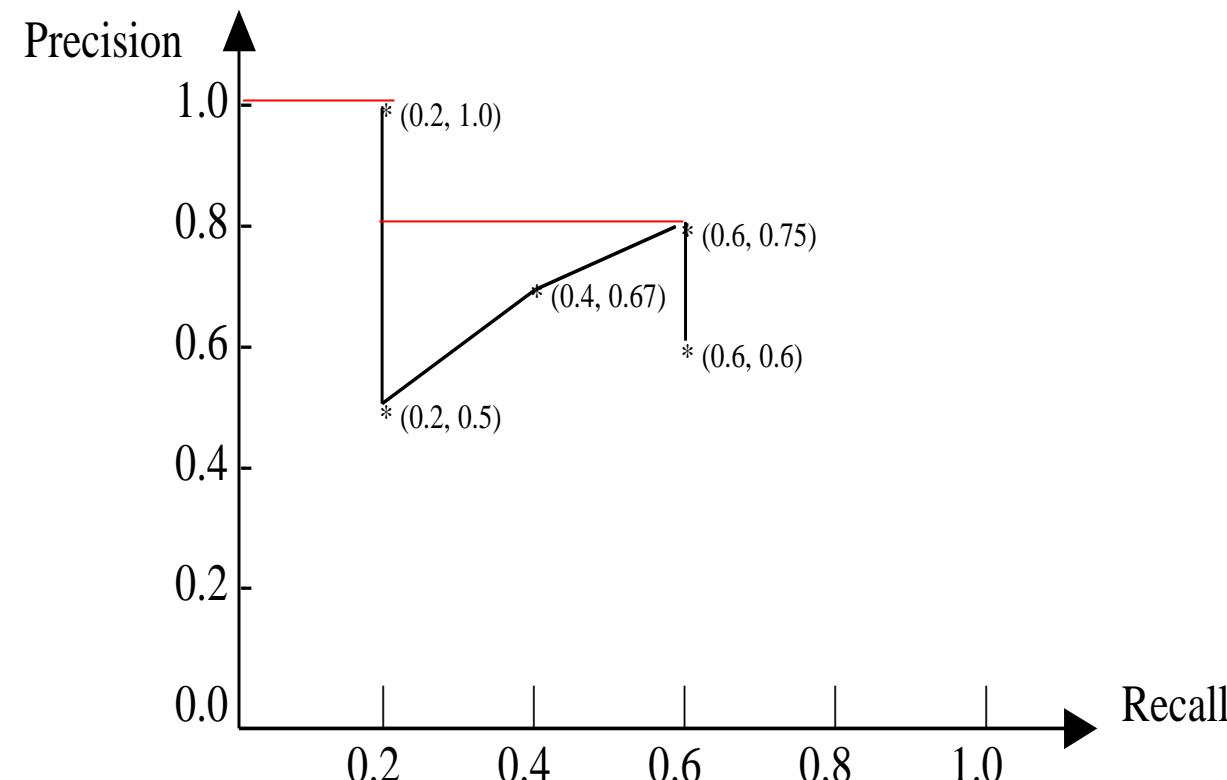


- Precision change w.r.t. Recall (not a fixed point)
- Systems cannot compare at one Precision/Recall point
- Average precision (on 11 points of recall: 0.0, 0.1, ..., 1.0)

# An illustration of P/R calculation

List	Rel?
Doc1	Y
Doc2	
Doc3	Y
Doc4	Y
Doc5	
...	

Assume: 5 relevant docs.



# Query expansion

---

- ◆ A query contains part of the important words
- ◆ Add new (related) terms into the query
  - ◆ Manually constructed knowledge base/ thesaurus (e.g. Wordnet)
    - | Q = information retrieval
    - | Q' = (information + data + knowledge + ...)  
(retrieval + search + seeking + ...)
  - ◆ Corpus analysis:
    - | two terms that often co-occur are related (Mutual information)
    - | Two terms that co-occur with the same words are related (e.g. T-shirt and coat with wear, ...)

# Global vs. local context analysis

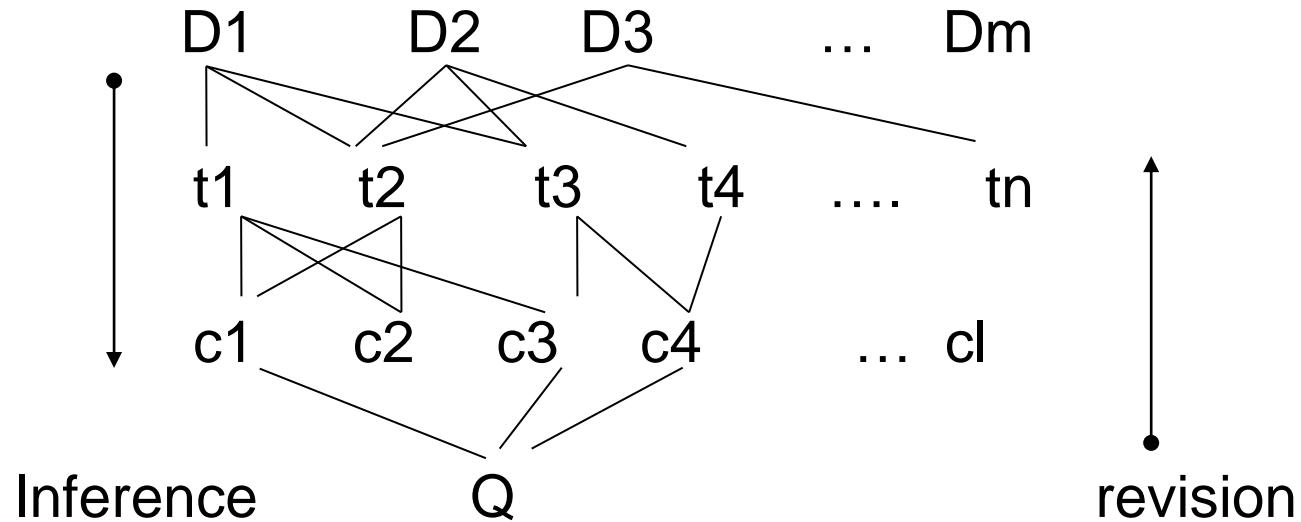
---

- ◆ Global analysis: use the whole document collection to calculate term relationships
- ◆ Local analysis: use the query to retrieve a subset of documents, then calculate term relationships
  - ◆ Combine pseudo-relevance feedback and term co-occurrences
  - ◆ More effective than global analysis

# Theory ...

# ◆ Bayesian networks

- $P(Q|D)$



## ◆ Language models

# Logical models

---

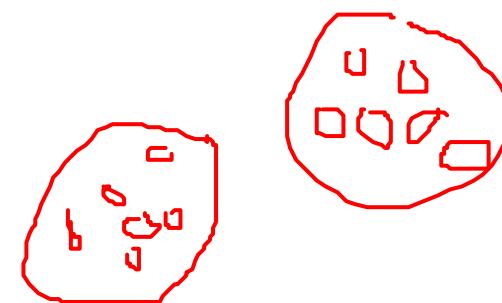
- ◆ How to describe the relevance relation as a logical relation?  
 $D \Rightarrow Q$
- ◆ What are the properties of this relation?
- ◆ How to combine uncertainty with a logical framework?
- ◆ The problem: What is relevance?

# **Text Similarity**

# Why similarity?

Used everywhere in NLP

- ◆ Information retrieval (Query vs Document)
- ◆ Text classification (Document vs Category)
- ◆ Word-sense disambiguation
- ◆ Automatic evaluation
  - ◆ Machine translation
  - ◆ Text summarization



# Word Similarity

T = w<sub>1</sub>, ..., w<sub>n</sub>

- ◆ Finding similarity between words is a fundamental part of text similarity.
- ◆ Words can be similar if:
  - They mean the same thing (synonyms)
  - They do not mean the opposite (antonyms)
  - They are used in the same way (red, green)
  - They are used in the same context (doctor, hospital, scalpel)
  - One is a type of another (poodle, dog, mammal)
- ◆ Lexical hierarchies like WordNet can be useful.

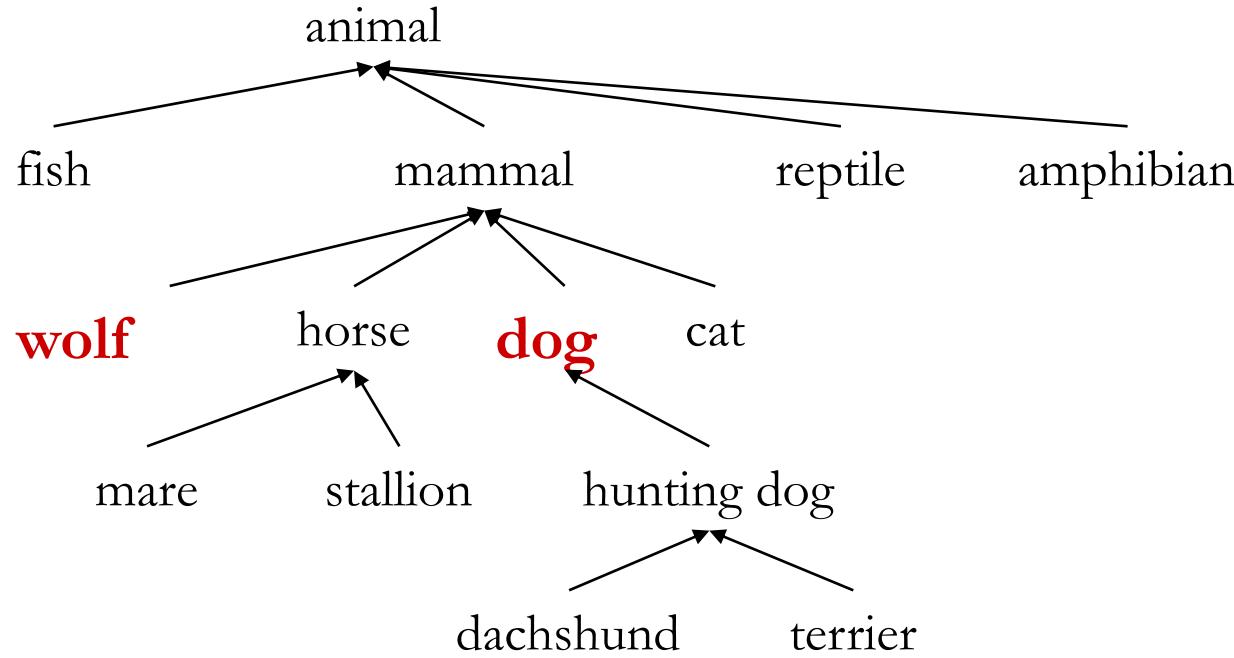
# Syntactic similarity

- String Edit distance:  $\text{Sim}(w_1 \text{ and } w_2)$  is min number of edit script (i.e., edit, insert, delete, move...) needed to transform  $w_1$  to  $w_2$ .

Abelae  $\Rightarrow$  Edit  $\Rightarrow$  |  
Abeba

Wolf  $\geq$  Edit  $\triangleright = 2$ ,  
dog  $\leq$  Edit  $\triangleright = 3$ ,  
 $\text{Sim} =$  Edit  $\triangleright + \cancel{\phi}$  |

# WordNet-like Hierarchy



# Format of Wordnet Entries

The noun “bass” has 8 senses in WordNet.

1. bass<sup>1</sup> - (the lowest part of the musical range)
2. bass<sup>2</sup>, bass part<sup>1</sup> - (the lowest part in polyphonic music)
3. bass<sup>3</sup>, basso<sup>1</sup> - (an adult male singer with the lowest voice)
4. sea bass<sup>1</sup>, bass<sup>4</sup> - (the lean flesh of a saltwater fish of the family Serranidae)
5. freshwater bass<sup>1</sup>, bass<sup>5</sup> - (any of various North American freshwater fish with lean flesh (especially of the genus Micropterus))
6. bass<sup>6</sup>, bass voice<sup>1</sup>, basso<sup>2</sup> - (the lowest adult male singing voice)
7. bass<sup>7</sup> - (the member with the lowest range of a family of musical instruments)
8. bass<sup>8</sup> - (nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes)

The adjective “bass” has 1 sense in WordNet.

1. bass<sup>1</sup>, deep<sup>6</sup> - (having or denoting a low vocal or instrumental range)  
*“a deep voice”; “a bass voice is lower than a baritone voice”;*  
*“a bass clarinet”*

- The set of near-synonyms for a WordNet sense is called a **synset (synonym set)**; it's their version of a sense or a concept

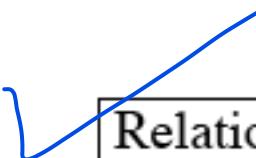
Example: **chump** as a noun to mean

‘a person who is gullible and easy to take advantage of’

{chump<sup>1</sup>, fool<sup>2</sup>, gull<sup>1</sup>, mark<sup>9</sup>, patsy<sup>1</sup>, fall guy<sup>1</sup>, sucker<sup>1</sup>,  
soft touch<sup>1</sup>, mug<sup>2</sup>}

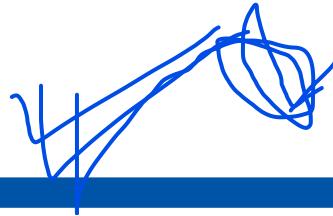
- Each of these senses share this same gloss
- Thus for WordNet, the meaning of this sense of **chump** is this list.

# WordNet Noun Relations



Relation	Also called	Definition	Example
Hypernym	Superordinate	From concepts to superordinates	<i>breakfast</i> <sup>1</sup> → <i>meal</i> <sup>1</sup>
Hyponym	Subordinate	From concepts to subtypes	<i>meal</i> <sup>1</sup> → <i>lunch</i> <sup>1</sup>
Member Meronym	Has-Member	From groups to their members	<i>faculty</i> <sup>2</sup> → <i>professor</i> <sup>1</sup>
Has-Instance		From concepts to instances of the concept	<i>composer</i> <sup>1</sup> → <i>Bach</i> <sup>1</sup>
Instance		From instances to their concepts	<i>Austen</i> <sup>1</sup> → <i>author</i> <sup>1</sup>
Member Holonym	Member-Of	From members to their groups	<i>copilot</i> <sup>1</sup> → <i>crew</i> <sup>1</sup>
Part Meronym	Has-Part	From wholes to parts	<i>table</i> <sup>2</sup> → <i>leg</i> <sup>3</sup>
Part Holonym	Part-Of	From parts to wholes	<i>course</i> <sup>7</sup> → <i>meal</i> <sup>1</sup>
Antonym		Opposites	<i>leader</i> <sup>1</sup> → <i>follower</i> <sup>1</sup>

# WordNet Verb Relations



Relation	Definition	Example
Hypernym	From events to superordinate events	$\text{fly}^9 \rightarrow \text{travel}^5$
Troponym	From a verb (event) to a specific manner elaboration of that verb	$\text{walk}^1 \rightarrow \text{stroll}^1$
Entails	From verbs (events) to the verbs (events) they entail	$\text{snore}^1 \rightarrow \text{sleep}^1$
Antonym	Opposites	$\text{increase}^1 \iff \text{decrease}^1$

# WordNet Hierarchies

```
Sense 3
bass, basso --
(an adult male singer with the lowest voice)
=> singer, vocalist, vocalizer, vocaliser
    => musician, instrumentalist, player
        => performer, performing artist
            => entertainer
                => person, individual, someone...
                    => organism, being
                        => living thing, animate thing,
                            => whole, unit
                                => object, physical object
                                    => physical entity
                                        => entity
                => causal agent, cause, causal agency
                    => physical entity
                        => entity
```

```
Sense 7
bass --
(the member with the lowest range of a family of
musical instruments)
=> musical instrument, instrument
    => device
        => instrumentality, instrumentation
            => artifact, artefact
                => whole, unit
                    => object, physical object
                        => physical entity
                            => entity
```

# Word similarity

- ◆ **Synonymy is a binary relation**
  - ◆ Two words are either synonymous or not
- ◆ **We want a looser metric**
  - ◆ Word similarity or
  - ◆ Word distance
- ◆ **Two words are more similar**
  - ◆ If they share more features of meaning
- ◆ **Actually these are really relations between senses:**
  - ◆ Instead of saying “bank is like fund”
  - ◆ We say
    - ◆ Bank1 is similar to fund3
    - ◆ Bank2 is similar to slope5
- ◆ **We'll compute them over both words and senses**

# Concept/ Semantic similarity

- ◆ Two categories

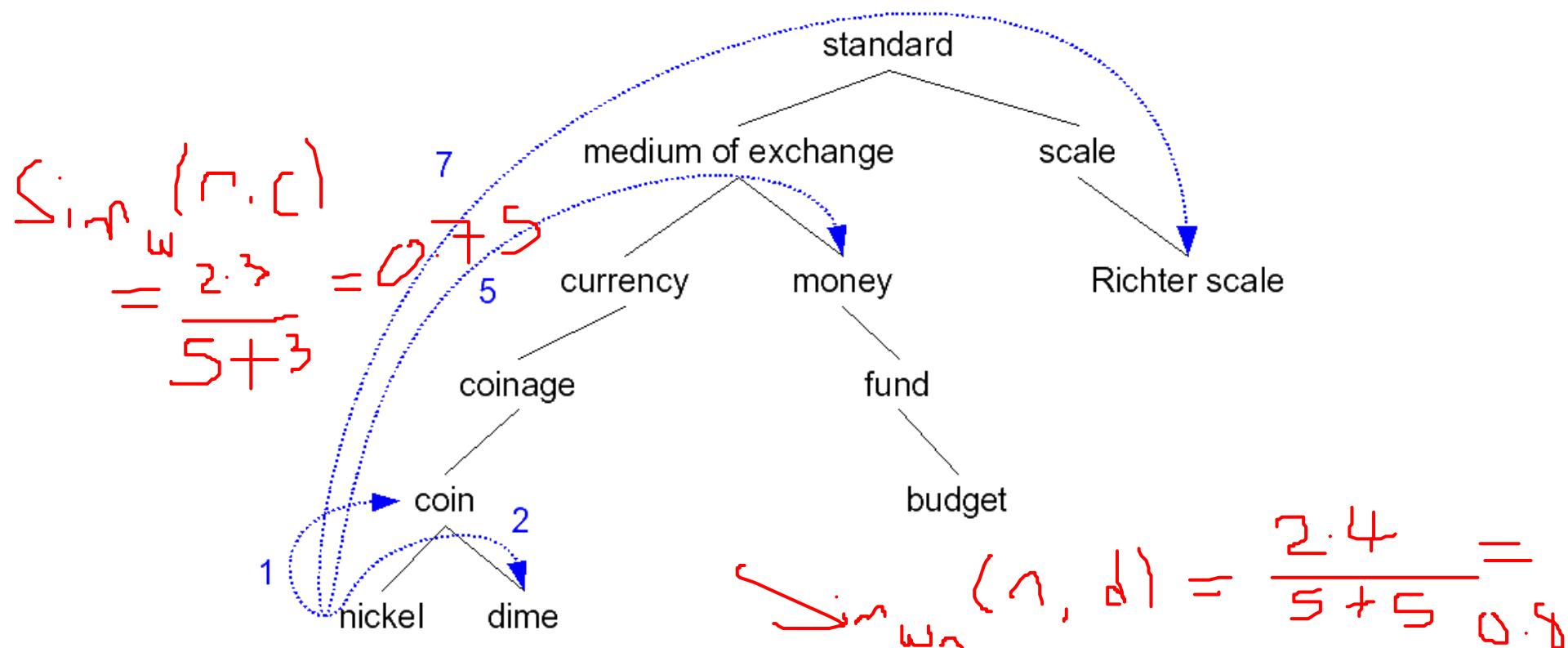
- ◆ Distance-based approaches
- ◆ Information content-based approaches

# Distance-based approaches

- ◆ The distance-based approaches use the distance/path-length between concepts in semantic knowledge as basic parameter.

# Path based similarity

- Two words are similar if nearby in thesaurus hierarchy (i.e. short path between them)

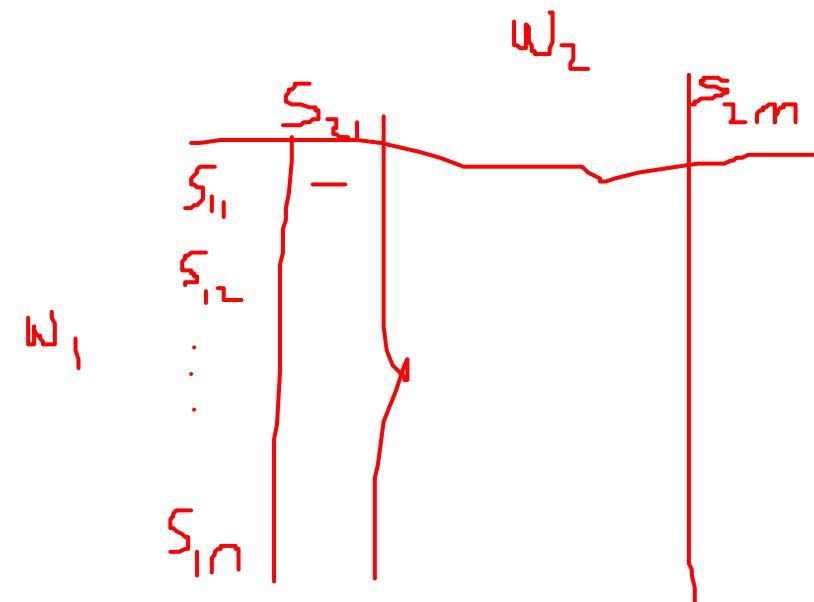


# Path-based similarity

- ◆  $\text{pathlen}(c_1, c_2) = \text{number of edges in the } \underline{\text{shortest path}} \text{ between the sense nodes } c_1 \text{ and } c_2$

$$\text{Sim}_{\text{Rada \& Bicknell}}(C_1, C_2) = \text{len}(C_1, C_2)$$

- ◆  $\text{simpath}(c_1, c_2) = -\log \text{pathlen}(c_1, c_2)$
- ◆  $\text{wordsim}(w_1, w_2) =$ 
  - ◆  $\max_{c_1 \in \text{senses}(w_1), c_2 \in \text{senses}(w_2)} \text{sim}(c_1, c_2)$



## Problem with basic path-based similarity

- ◆ Assumes each link represents a uniform distance
- ◆ Nickel to money seem closer than nickel to standard
- ◆ Instead:
  - ◆ Want a metric which lets us represent the cost of each edge independently

- ◆ Leacock & Chodorow (LEACOCK, C. and Chodorow, M., 1998) propose scaled concept-based measure by including the maximum depth of the semantic knowledge as a path length normalization factor. It is denoted as:

$$Sim_{\text{Leacock \& Chodorow}}(C_1, C_2) = \frac{-\log(\text{len}(C_1, C_2))}{2 \times D}$$

- ◆ Wu & Palmer (WU, Z. and Palmer, M. S., 1994) evaluate a conceptual similarity between pair of concepts in hierarchy-based Knowledge Base using their most common ancestor.
- ◆ The similarity measure takes into consideration the depth of the *least common ancestor concept* as well as the distance separating each concept from the least common ancestor. It is denoted as:

$$Sim_{Wu \& Palmer}(C_1, C_2) = \frac{2 \times \text{depth}(C)}{\text{len}(C_1, C) + \text{len}(C, C_2) + 2 \times \text{depth}(C)}$$

# Information content similarity metrics

- ◆ Let's define  $P(C)$  as:

- The probability that a randomly selected word in a corpus is an instance of concept  $c$
- Formally: there is a distinct random variable, ranging over words, associated with each concept in the hierarchy
- $P(\text{root})=1$
- The lower a node in the hierarchy, the lower its probability

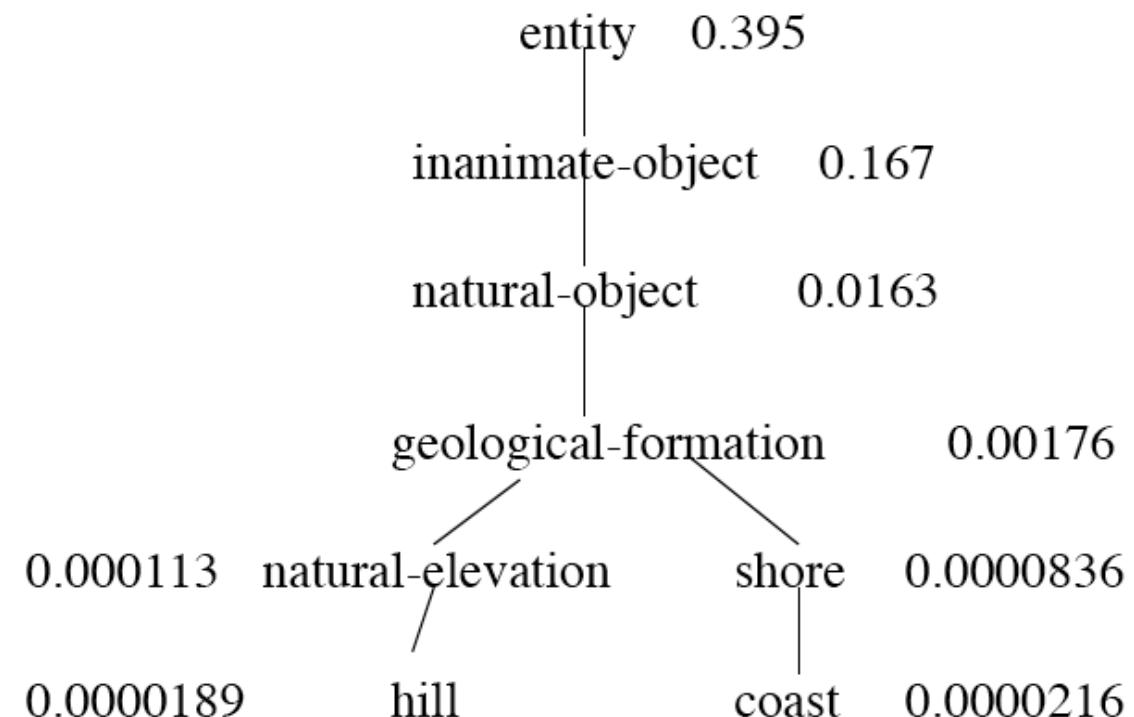
# Information content similarity

- ◆ Train by counting in a corpus
  - ◆ instance of “dime” could count toward frequency of *coin*, *currency*, *standard*, etc
- ◆ More formally:

$$P(c) = \frac{\sum count(w)}{N}$$

# Information content similarity

- WordNet hierarchy augmented with probabilities  $P(C)$



## Information content: definitions

- ◆ **Information content:**

- ◆  $IC(c) = -\log P(c)$

- ◆ **Lowest common subsumer LCS( $c_1, c_2$ )**

- ◆ i.e. the lowest node in the hierarchy that subsumes (is a hypernym of) both  $c_1$  and  $c_2$

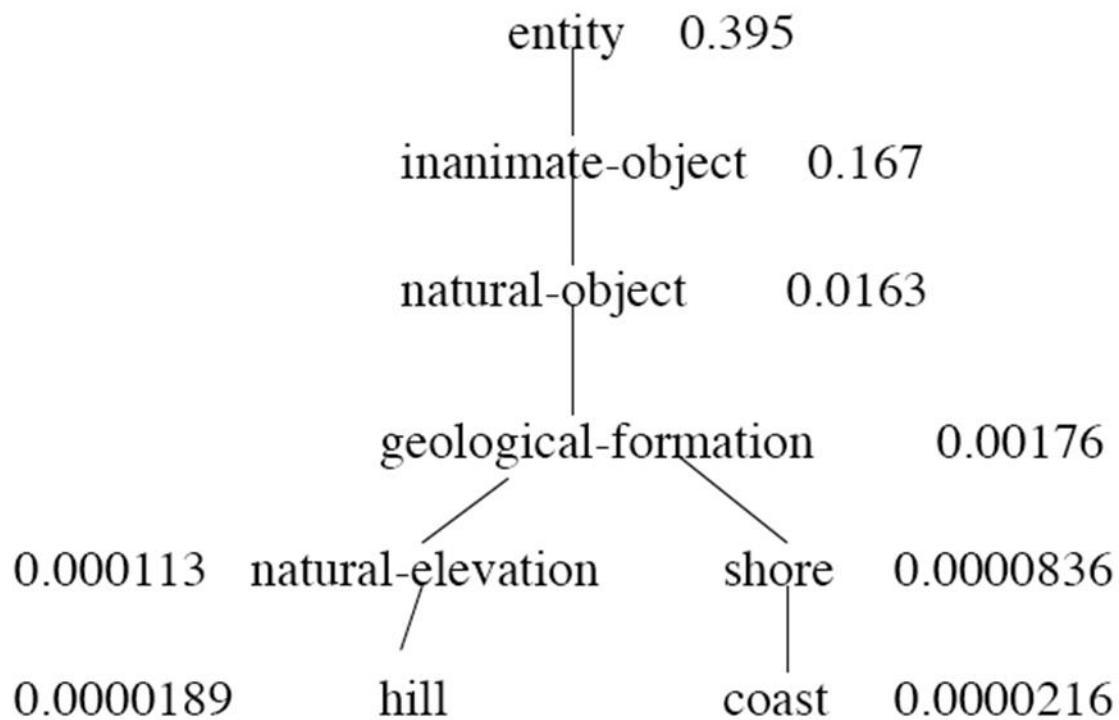
## Resnik (RESNIK, P., 1995),

- The similarity between two words is related to their common information in ISA taxonomy (i.e. info content of the lowest common subsumer of the two nodes)
- **The more two words have in common, the more similar they are**

$$Sim_{\text{Resnik}}(C_1, C_2) = -\log P(LCS(C_1, C_2))$$

## Example

- ◆  $\text{Sim}(\text{hill}, \text{shore})?$
- ◆  $\text{Sim}(\text{hill}, \text{coast})?$



- ◆ Drawback of the this method?

## Lin's universal similarity measure (LIN, D., 1998)

- ◆ The similarity between two **concepts defined** as a ratio of the amount of information needed to state **their commonality** and the information needed to fully state each of them. It is denoted as:

$$Sim_{Lin}(C_1, C_2) = \frac{2 \times IC(C)}{IC(C_1) + IC(C_2)}$$

- ◆ Example:

$$\begin{aligned} Sim_{Lin}(hill, coast) &= \frac{2 \times \log p(\text{geographic-information})}{\log p(hill) + \log p(coast)} \\ &= .59 \end{aligned}$$

- ◆ The semantic distance between two concepts is qualified with the computational evidence derived from the distributional analysis of the corpus data.
- ◆ It is denoted as:

$$Dist_{jc}(C_1, C_2) = IC(C_1) + IC(C_2) - 2 \times IC(C)$$

# Vector space model

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

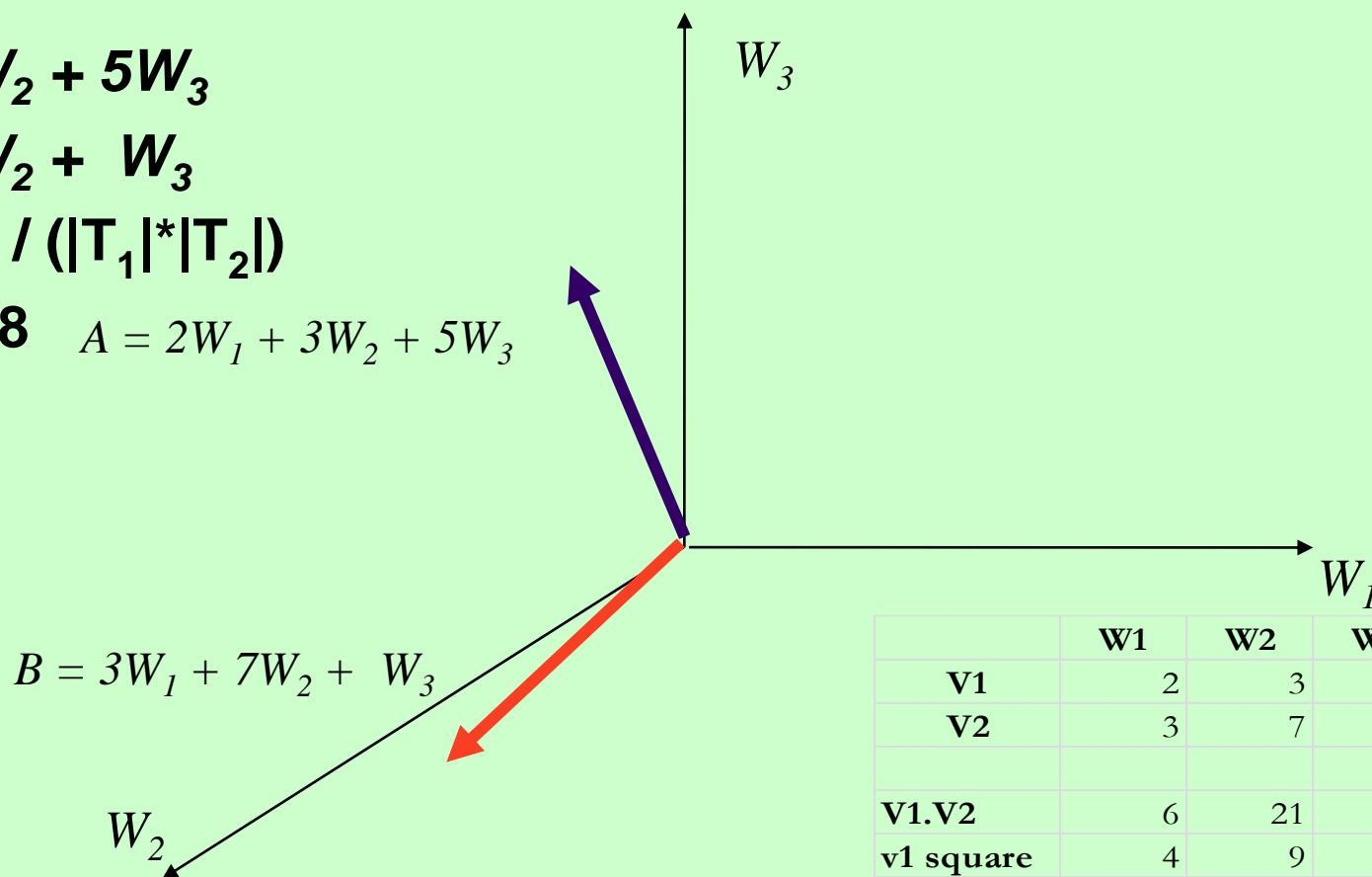
**Example:**

$$A = 2W_1 + 3W_2 + 5W_3$$

$$B = 3W_1 + 7W_2 + W_3$$

$$\cos \Theta = \mathbf{T}_1 \cdot \mathbf{T}_2 / (\|\mathbf{T}_1\| * \|\mathbf{T}_2\|)$$

$$= 0.6758 \quad A = 2W_1 + 3W_2 + 5W_3$$



	W1	W2	W3		
V1	2	3	5		
V2	3	7	1		sum
V1.V2	6	21	5	32	
v1 square	4	9	25	6.1644	
v2 square	9	49	1	7.6811	
			cos(V1, V2) =	0.6758	

# Document similarity

Hurricane Gilbert swept toward the Dominican Republic Sunday, and the Civil Defense alerted its heavily populated south coast to prepare for high winds, heavy rains and high seas.

The storm was approaching from the southeast with sustained winds of 75 mph gusting to 92 mph .

"There is no need for alarm," Civil Defense Director Eugenio Cabral said in a television alert shortly before midnight Saturday .

Cabral said residents of the province of Barahona should closely follow Gilbert 's movement .

An estimated 100,000 people live in the province, including 70,000 in the city of Barahona , about 125 miles west of Santo Domingo .

Tropical Storm Gilbert formed in the eastern Caribbean and strengthened into a hurricane Saturday night

The National Hurricane Center in Miami reported its position at 2a.m. Sunday at latitude 16.1 north , longitude 67.5 west, about 140 miles south of Ponce, Puerto Rico, and 200 miles southeast of Santo Domingo.

The National Weather Service in San Juan , Puerto Rico , said Gilbert was moving westward at 15 mph with a "broad area of cloudiness and heavy weather" rotating around the center of the storm.

The weather service issued a flash flood watch for Puerto Rico and the Virgin Islands until at least 6p.m. Sunday.

Strong winds associated with the Gilbert brought coastal flooding , strong southeast winds and up to 12 feet to Puerto Rico 's south coast.

## Document Vectors for selected terms

• **Document2**

- Gilbert: 2
- Hurricane: 1
- Rains: 0
- Storm: 1
- Winds: 2

• **Document1**

- Gilbert: 3
- Hurricane: 2
- Rains: 1
- Storm: 2
- Winds: 2

Cosine Similarity: 0.9439

# Jacquard Coefficient

- ◆  $\text{Sim}(t_1, t_2) = t_1 \cap t_2 / t_1 \cup t_2$

# Example

- ◆ T1: The dog eat the boy
- ◆ T2: the boy eat the dog
- ◆ T3: The animal eat the dog
- ◆ T1 identical to T2

	Dog	Eat	Boy	Animal
T1	1	1	1	0
T3	1	1	0	1

	Dog	Eat	Boy	Animal
T1	1	1	1	0.8
T3	1	1	0.2	1

# **SEMANTIC SEARCH**

# SEMANTIC SEARCH

- ◆ Semantic search seeks to improve search accuracy by *understanding searcher's intent* and the *contextual meaning of terms* as they appear in the searchable dataspace, whether on the Web or within a closed system, to generate more relevant results.

# MOTIVATION FOR SEMANTIC SEARCH

## Search

About 12,000,000 results (0.43 seconds)

[Everything](#)[Images](#)[Maps](#)[Videos](#)[News](#)[More](#)[Mumbai, Maharashtra](#)[Change location](#)[The web](#)[Pages from India](#)[More search tools](#)[Six Escalation Scenarios to World Nuclear War](#)[www.carolmoore.net/nuclearwar/alternatescenarios.html](http://www.carolmoore.net/nuclearwar/alternatescenarios.html) +1

URGENT: Scenario 3 - Israel Bombs Iranian Nuclear Plants ... Or they could deliver nuclear bombs to destroys nuclear and/or non-nuclear military ... It is likely that the U.S., Russia, China, Israel, India and Pakistan will use some of their ...

[Nuclear weapons and Israel - Wikipedia, the free encyclopedia](#)[en.wikipedia.org/wiki/Nuclear\\_weapons\\_and\\_Israel](http://en.wikipedia.org/wiki/Nuclear_weapons_and_Israel) +1

Israel has never officially admitted to having nuclear weapons, instead ..... to destroy three Iranian nuclear facilities with low-yield nuclear bunker-busters that ... on the P-1 centrifuge, the design A. Q. Khan stole in 1976 and took to Pakistan. ...

[The Day Pakistan and Israel Came Close to War - YouTube](#)[www.youtube.com/watch?v=JEyX7FQqlHQ](http://www.youtube.com/watch?v=JEyX7FQqlHQ) +1

29 Apr 2009 - 10 min - Uploaded by A2Kaid

... a Documentary/Research video Produced by Me that explains the Covert Joint Indo-Israeli operation to destroy Pakistan's ...

[More videos for destroying nuclear plants in pakistan »](#)[Germany Opposes United States on China-Pakistan Nuclear Deal ...](#)[armscontrolnow.org/.../germany-opposes-united-states-on-china-paki...](http://armscontrolnow.org/.../germany-opposes-united-states-on-china-paki...) +1

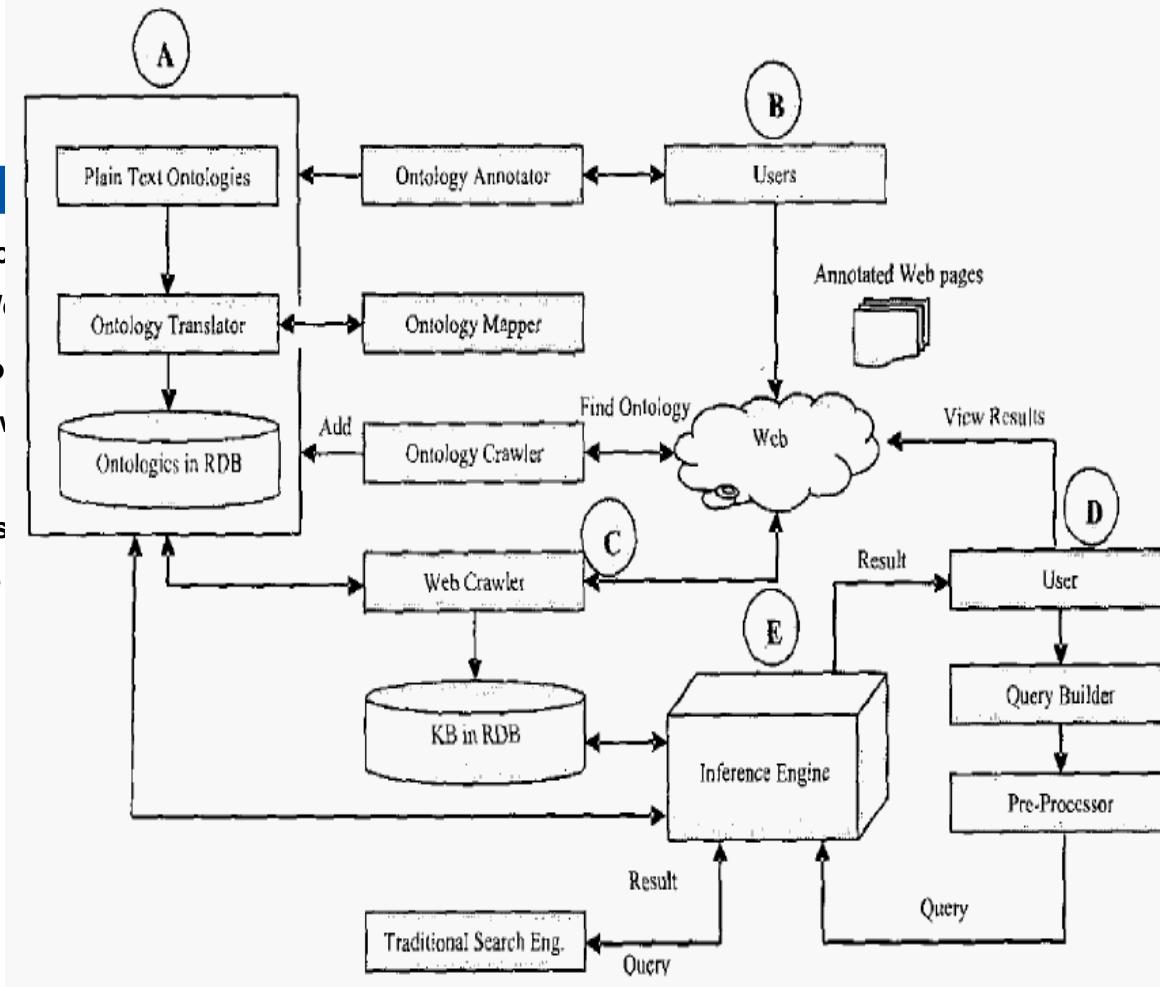
21 Jun 2011 – Pakistan, which is not a member of the Nuclear Nonproliferation Treaty and has ... Thus, the proposed supply of additional nuclear plants at the ... other countries then first they should destroy all their nuclear bomb, dis arm all ...

## Characteristics for the semantic search frameworks

Feature \ System	SHOE	TAP	AquaLog	Falcon-S	DOSE	Squiggle	KIM	OWLIR	SemSearch
Ontology	Y	TAP	Y	soccer	Y	N	KIMO	Y	Y
Inference	shallow	shallow	Y	template	taxonomy	Y	Y	Y	Y
Query Processing	N	N	Y	N	Y	N	N	N	Y
Query Expansion or Refining	N	N	Y	N	Y	Y	N	N	Y
Relevance Feedback	N	N	Y	N	Y	Y	N	N	N
Ranking	-	-	-	Y	tf-idf	-	-	-	Y
RDF/OWL Repository	-	-	-	sesame	jena	sesame	sesame	sesame	sesame
String Matching	-	label	Y	N	tf-idf	N	N	N	Y
Indexing	N	Y	Y	Y	Y	Y	Y	Y	Y

# Architecture

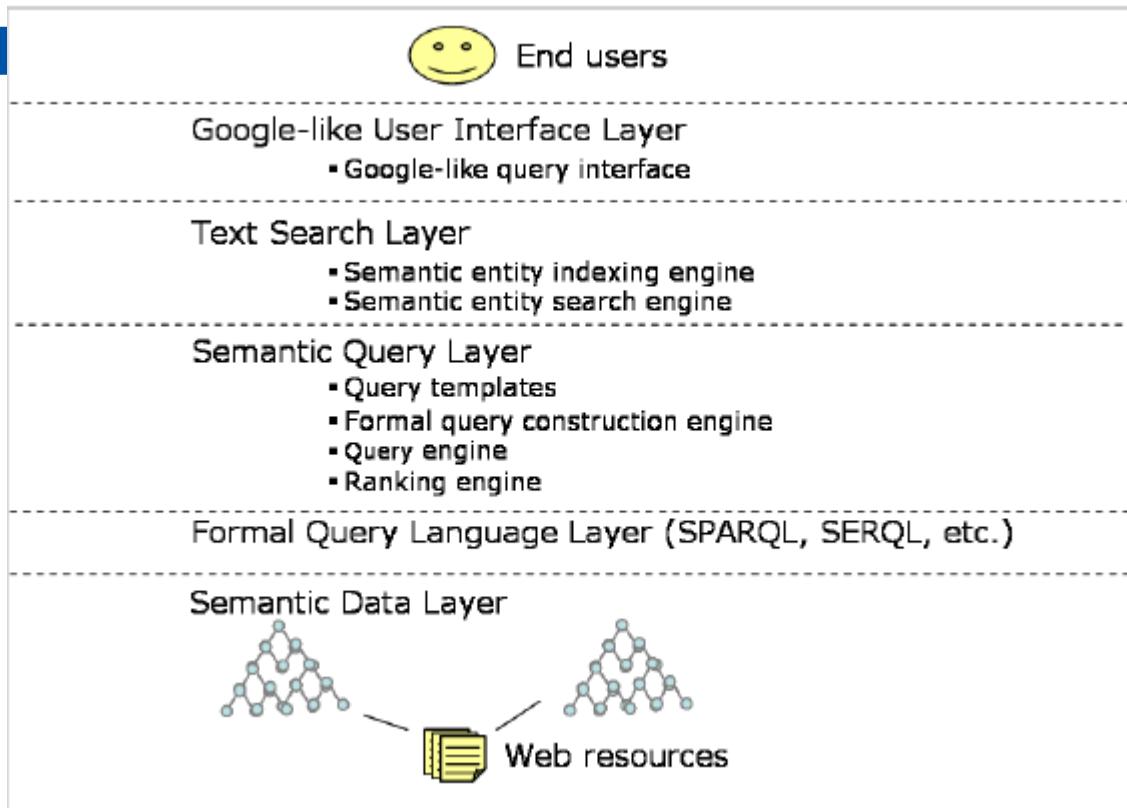
- ♦ A: Ontologies are constructed from plain text ontologies on the Web.
- ♦ B: Users use Ontology Annotator to annotate Web pages.
- ♦ C: Web Crawler crawls annotated Web pages.
- ♦ D: Users construct semantic queries.
- ♦ E: Inference Engine finds results and displays them on the Web.



into relational database tables. Ontology Crawler finds new ontologies on the Web. Ontology Mapper maps the instances of these ontologies in these Web pages. Inference Engine after pre-processing by Query Pre-processor. The results are knowledge base. The results are finally sent to the user to be viewed on the Web.

## Conceptual Semantic Search Engine Architecture

(Qazi Mudassar Ilyas et al. Conceptual semantic search engine, IEEE,2004 )

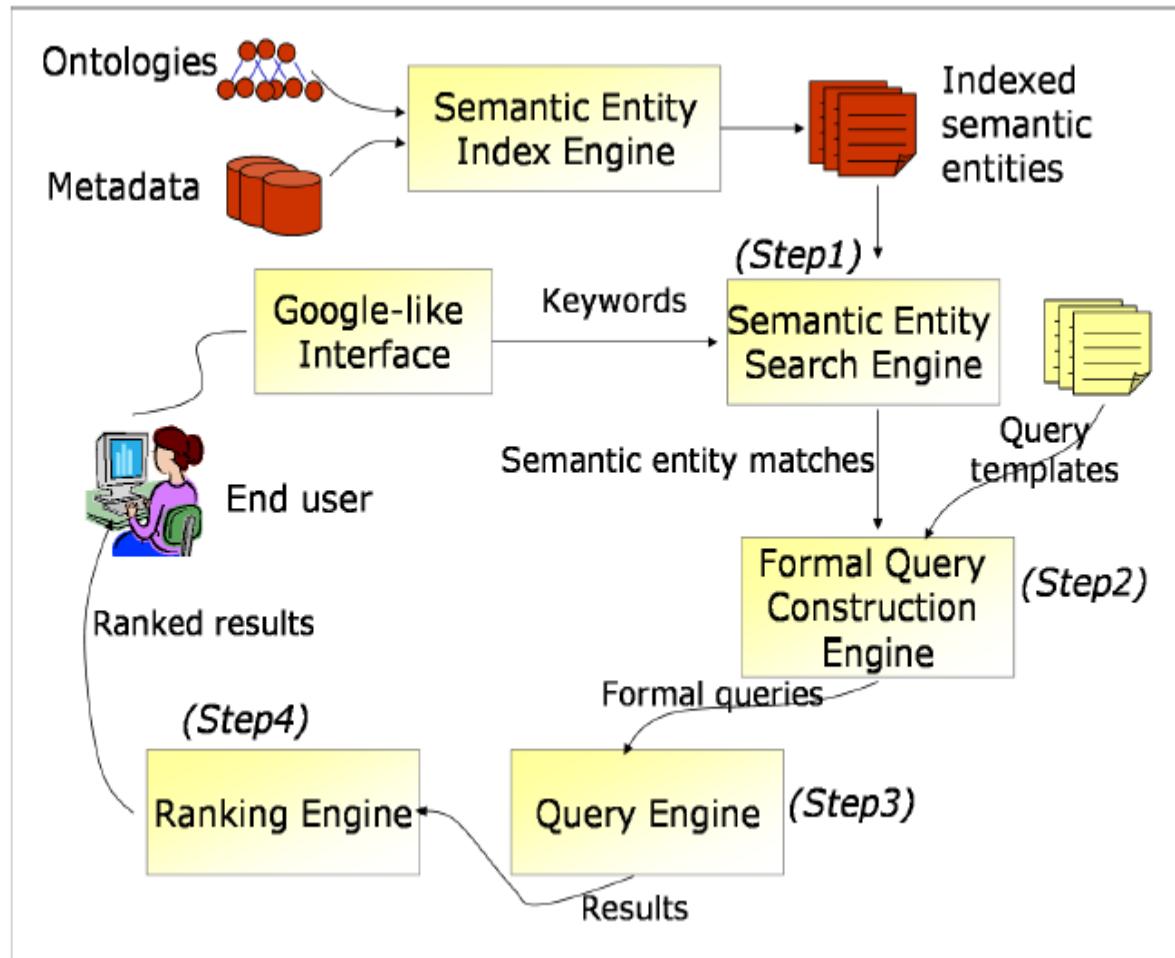


## An overview of the SemSearch architecture

(Lei Y. et al, SemSearch: A Search Engine for the Semantic Web)

# Four steps in SemSearch

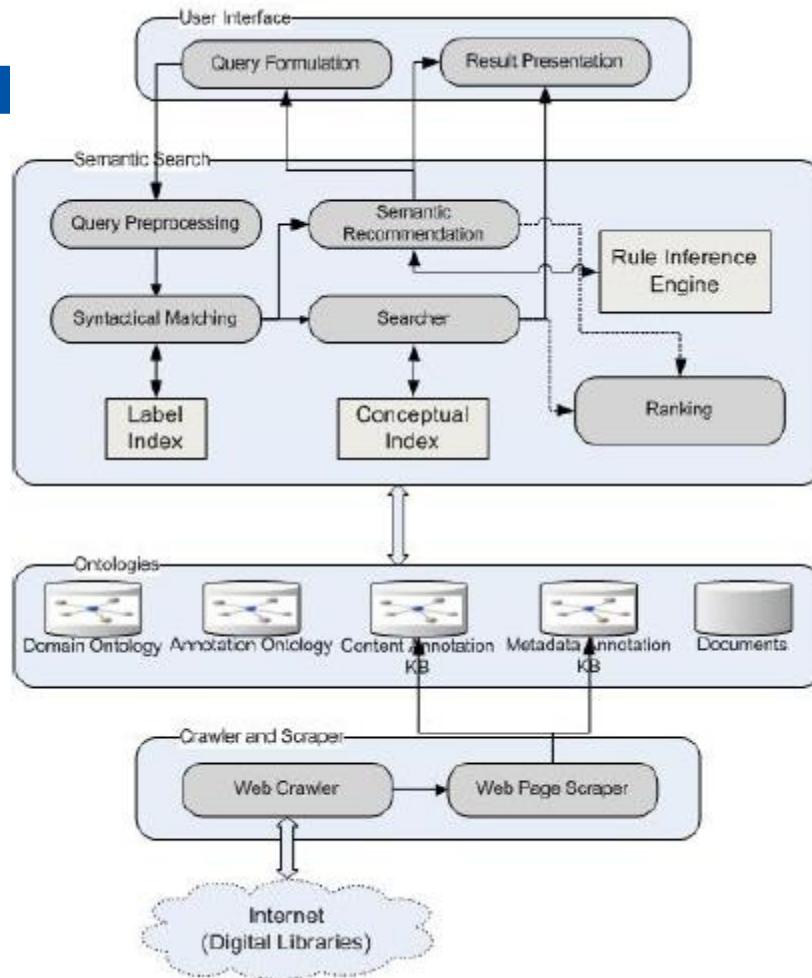
- An overall diagram of the SemSearch search engine



- Step1. Making sense of the user query, which is to find out the semantic meanings of the keywords specified in a user query.

Step2.

332



The architecture of the IRIS semantic search system  
 (WangWei, Payam M. Barnaghi, Andrzej Bargiela, The Anatomy and Design  
 of A Semantic Search Engine, Elsevier, 2008)

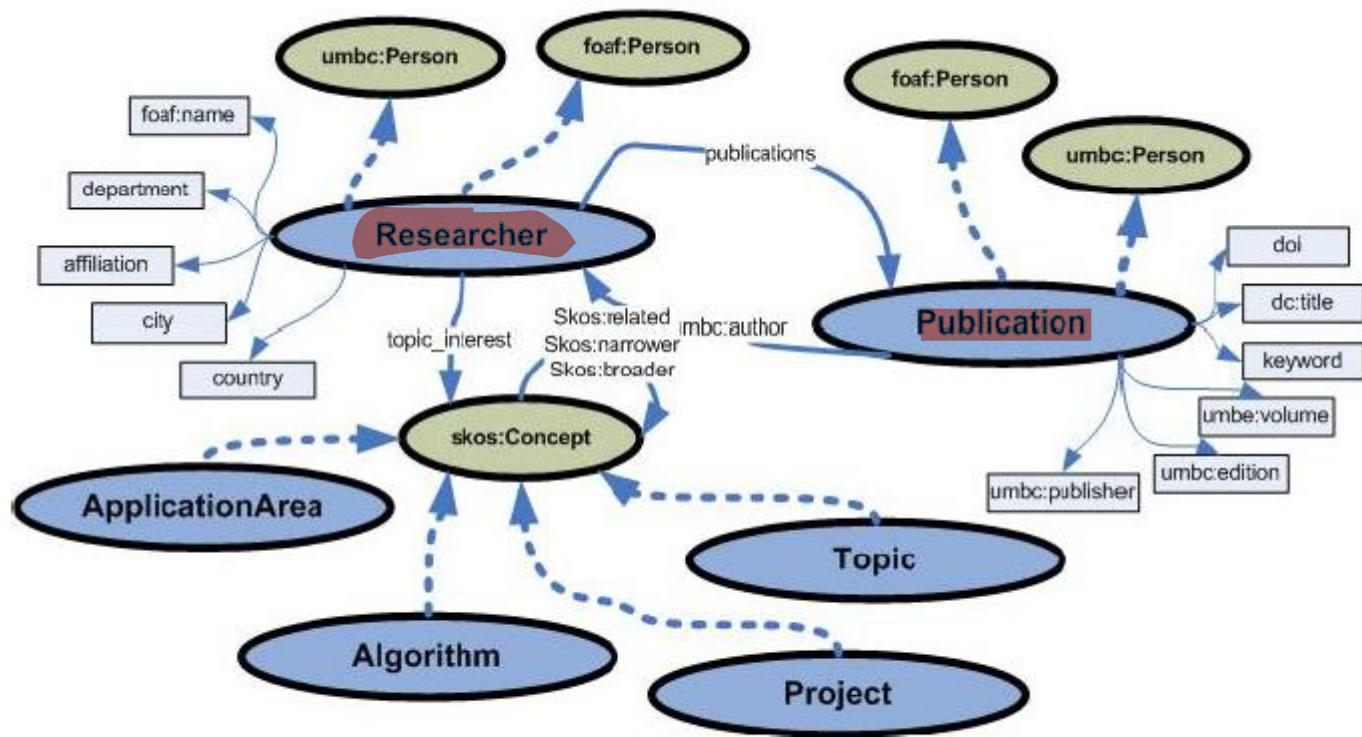


Fig. 2. The IRIS annotation ontology.

## **Querying the Semantic Web: SPARQL**

# Overview

- ◆ SPARQL [1] is a query language designed for the Semantic Web and is the language recommended by W3C for this use.
- ◆ SPARQL is completely integrated into the Semantic Web and uses other W3C recommendations.
- ◆ It assumes that data are represented as RDF graphs, that resources are identified by IRIs, and that RDF literals are typed with XML Schema

[1] Prud'hommeaux, E., Seaborne, A. (ed.): SPARQL query language for RDF, W3C Recommendation. World Wide Web Consortium (2008)

- ◆ SPARQL stands for “**SPARQL Protocol and RDF Query Language**”.
- ◆ In addition to the language, W3C has also defined:
  - The **SPARQL Protocol** for **RDF specification**: it defines the remote protocol for issuing SPARQL queries and receiving the results.
  - The **SPARQL Query Results XML Format specification**: it defines an XML document format for representing the results of SPARQL queries.

# SPARQL Basics

- ◆ SPARQL is based on matching **graph patterns** against RDF graphs.
- ◆ What is a graph pattern?
- ◆ To define graph patterns, we must first define triple patterns:
  - ◆ A **triple pattern** is like an **RDF triple**, but with the option of a variable in place of RDF terms (i.e., IRIs, literals or blank nodes) in the subject, predicate or object positions.
  - ◆ Example:

```
<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> ?title .
```

- ◆ ?title is a **variable**.

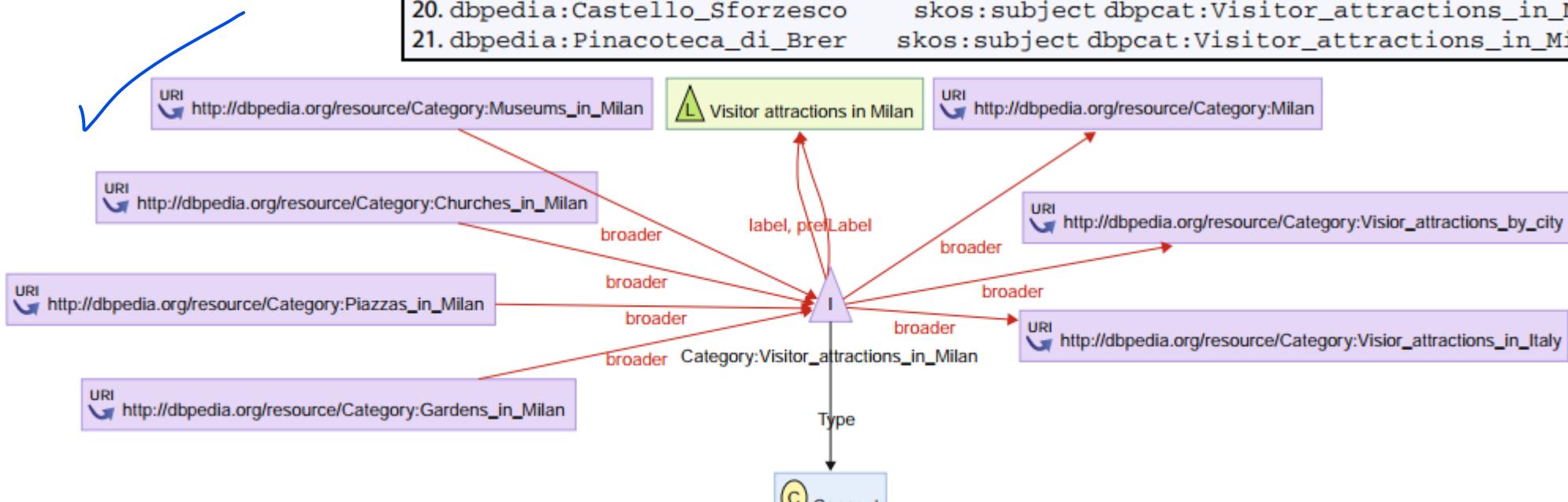
# SPARQL Basics

- ◆ SPARQL is basic
- ◆ What is a graph?

```

1. @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2. @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3. @prefix skos: <http://www.w3.org/2004/02/skos/core#> .
4. @prefix dbpedia: <http://dbpedia.org/resource/> .
5. @prefix dbpcat: <http://dbpedia.org/resource/Category:> .
6. dbpcat:Visitor_attractions_in_Milan
7.   rdf:type skos:Concept ;
8.   rdfs:label "Visitor attractions in Milan"@en ;
9.   skos:prefLabel "Visitor attractions in Milan"@en ;
10.  skos:broader dbpcat:Milan ,
11.    dbpcat:Visitor_attractions_in_Italy ,
12.    dbpcat:Visitor_attractions_by_city .
13. dbpcat:Museums_in_Milan      skos:broader dbpcat:Visitor_attractions_in_Milan .
14. dbpcat:Churches_in_Milan    skos:broader dbpcat:Visitor_attractions_in_Milan .
15. dbpcat:Piazzas_in_Milan    skos:broader dbpcat:Visitor_attractions_in_Milan .
16. dbpcat:Gardens_in_Milan    skos:broader dbpcat:Visitor_attractions_in_Milan .
17. dbpedia:Biblioteca_Ambrosiana skos:subject dbpcat:Visitor_attractions_in_Milan .
18. dbpedia:Brera_Academ      skos:subject dbpcat:Visitor_attractions_in_Milan .
19. dbpedia:Via_Montenapoleone skos:subject dbpcat:Visitor_attractions_in_Milan .
20. dbpedia:Castello_Sforzesco  skos:subject dbpcat:Visitor_attractions_in_Milan .
21. dbpedia:Pinacoteca_di_Brera skos:subject dbpcat:Visitor_attractions_in_Milan .

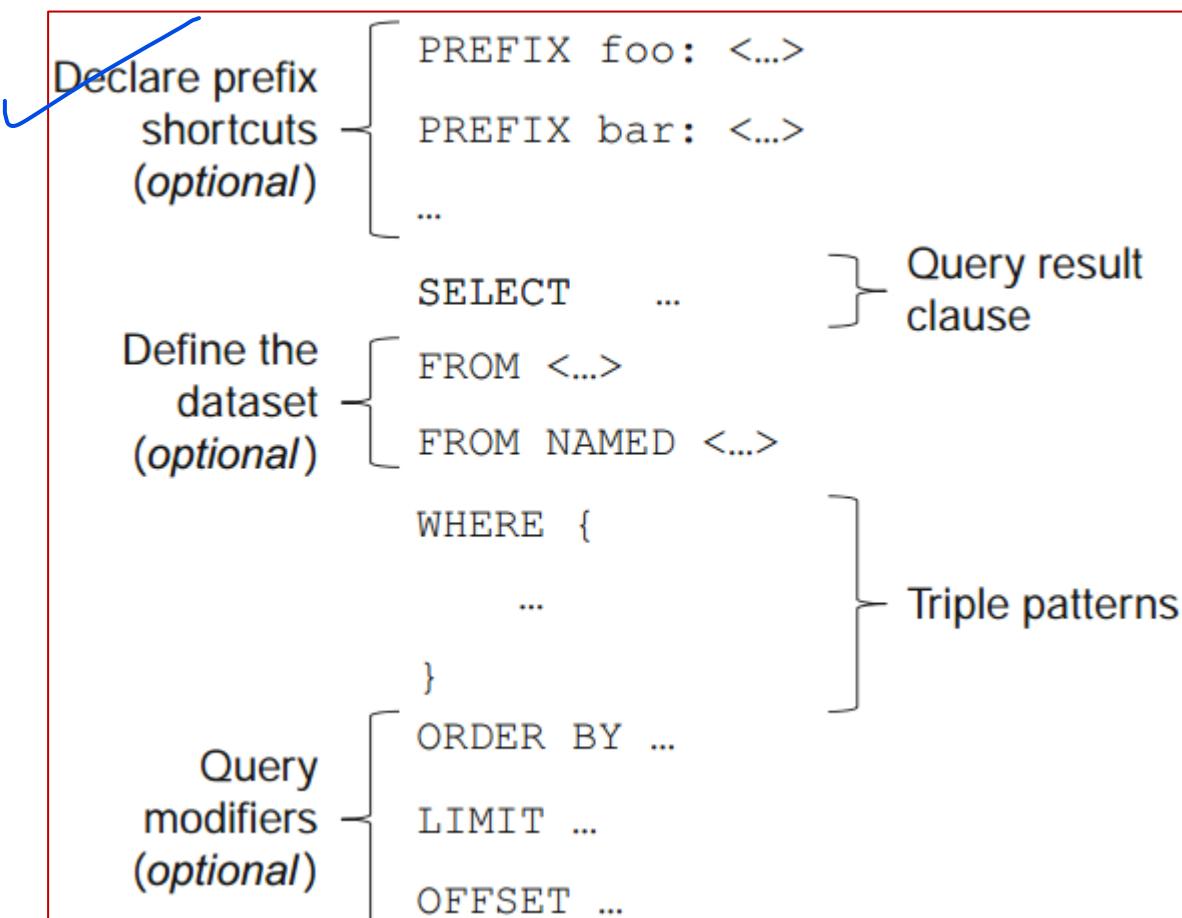
```



# Anatomy of a SPARQL Query

- ◆ A SPARQL query is composed of five parts:

- zero or more prefix declarations,
- a query result clause,
- zero or more FROM or FROM NAMED clauses,
- a WHERE clause,
- and zero or more query modifiers.



## Anatomy of a SPARQL Query (2)

- ◆ Query result clause: specifies the form of the result.
- ◆ A SPARQL query can take four forms: **SELECT**, **ASK**, **CONSTRUCT**, and **DESCRIBE**.
  - **SELECT** queries provide answers in a tabular form as if the SPARQL query were a SQL query executed against a relational database.
  - The **ASK** form checks whether the SPARQL endpoint can provide at least one result; if it does, the answer to the query is **YES**, otherwise the answer is **NO**.
  - The **CONSTRUCT** form is similar to the SELECT form, but it provides the answer to the query as an RDF graph.
  - The **DESCRIBE** form is conceived to retrieve information from a SPARQL endpoint without knowing the vocabulary in use, producing as result an **RDF graph**.
- ◆ The optional set of **FROM** or **FROM NAMED** clauses define the dataset against which the query is executed.
- ◆ The **WHERE** clause is the core of a SPARQL query. It is specified in terms of a set of triple patterns.

# Basic Pattern

- ◆ SPARQL is a
- ◆ A query contains
- ◆ A triple pattern object may be
- ◆ Syntactically a example, ?name
- ◆ A triple pattern substituted for

```

1. @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2. @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3. @prefix skos: <http://www.w3.org/2004/02/skos/core#> .
4. @prefix dbpedia: <http://dbpedia.org/resource/> .
5. @prefix dbpcat: <http://dbpedia.org/resource/Category:> .
6. dbpcat:Visitor_attractions_in_Milan
7.   rdf:type skos:Concept ;
8.   rdfs:label "Visitor attractions in Milan"@en ;
9.   skos:prefLabel "Visitor attractions in Milan"@en ;
10.  skos:broader dbpcat:Milan ,
11.      dbpcat:Visitor_attractions_in_Italy ,
12.      dbpcat:Visitor_attractions_by_city .
13.  dbpcat:Museums_in_Milan      skos:broader dbpcat:Visitor_attractions_in_Milan .
14.  dbpcat:Churches_in_Milan    skos:broader dbpcat:Visitor_attractions_in_Milan .
15.  dbpcat:Piazzas_in_Milan    skos:broader dbpcat:Visitor_attractions_in_Milan .
16.  dbpcat:Gardens_in_Milan    skos:broader dbpcat:Visitor_attractions_in_Milan .
17.  dbpedia:Biblioteca_Ambrosiana skos:subject dbpcat:Visitor_attractions_in_Milan .
18.  dbpedia:Brera_Academy      skos:subject dbpcat:Visitor_attractions_in_Milan .
19.  dbpedia:Via_Montenapoleone  skos:subject dbpcat:Visitor_attractions_in_Milan .
20.  dbpedia:Castello_Sforzesco   skos:subject dbpcat:Visitor_attractions_in_Milan .
21.  dbpedia:Pinacoteca_di_Brera skos:subject dbpcat:Visitor_attractions_in_Milan .

```

Example: ?s skos:broader dbpcat:Visitor\_attractions\_in\_Milan .

## Basic Patterns (2) (Example)

Part of the RDF graph that described the category "Visitor attractions in Milan" in DBpedia

```
1. @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
2. @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .  
3. @prefix skos: <http://www.w3.org/2004/02/skos/core#> .  
4. @prefix dbpedia: <http://dbpedia.org/resource/> .  
5. @prefix dbpcat: <http://dbpedia.org/resource/Category:> .  
6. dbpcat:Visitor_attractions_in_Milan  
7. rdf:type skos:Concept ;  
8. rdfs:label "Visitor attractions in Milan"@en ;  
9. skos:prefLabel "Visitor attractions in Milan"@en ;  
10. skos:broader dbpcat:Milan ,  
11. dbpcat:Visitor_attractions_in_Italy ,  
12. dbpcat:Visitor_attractions_by_city .  
13. dbpcat:Museums_in_Milan skos:broader dbpcat:Visitor_attractions_in_Milan .  
14. dbpcat:Churches_in_Milan skos:broader dbpcat:Visitor_attractions_in_Milan .  
15. dbpcat:Piazzas_in_Milan skos:broader dbpcat:Visitor_attractions_in_Milan .  
16. dbpcat:Gardens_in_Milan skos:broader dbpcat:Visitor_attractions_in_Milan .  
17. dbpedia:Biblioteca_Ambrosiana skos:subject dbpcat:Visitor_attractions_in_Milan .  
18. dbpedia:Brera_Academ skos:subject dbpcat:Visitor_attractions_in_Milan .  
19. dbpedia:Via_Montenapoleone skos:subject dbpcat:Visitor_attractions_in_Milan .  
20. dbpedia:Castello_Sforzesco skos:subject dbpcat:Visitor_attractions_in_Milan .  
21. dbpedia:Pinacoteca_di_Brer skos:subject dbpcat:Visitor_attractions_in_Milan .
```

Example triple pattern 1: ?s skos:broader dbpcat:Visitor\_attractions\_in\_Milan .

Example triple pattern 2: ?s ?p dbpcat:Visitor\_attractions\_in\_Milan .

# Basic graph pattern

- ◆ It is a set of triple patterns.
- ◆ A basic graph pattern matches a sub-graph of the RDF data when the variables in the graph pattern substitute the RDF terms.

```
?s rdf:type skos:Concept .  
?s skos:broader ?o .
```

1. @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
2. @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3. @prefix skos: <http://www.w3.org/2004/02/skos/core#> .
4. @prefix dbpedia: <http://dbpedia.org/resource/> .
5. @prefix dbpcat: <http://dbpedia.org/resource/Category:> .
6. dbpcat:Visitor\_attractions\_in\_Milan
7.        rdf:type skos:Concept ;
8.        rdfs:label "Visitor attractions in Milan"@en ;
9.        skos:prefLabel "Visitor attractions in Milan"@en ;
10.      skos:broader dbpcat:Milan ,
11.            dbpcat:Visitor\_attractions\_in\_Italy ,
12.            dbpcat:Visitor\_attractions\_by\_city .
13. dbpcat:Museums\_in\_Milan        skos:broader dbpcat:Visitor\_attractions\_in\_Milan .
14. dbpcat:Churches\_in\_Milan      skos:broader dbpcat:Visitor\_attractions\_in\_Milan .
15. dbpcat:Piazzas\_in\_Milan      skos:broader dbpcat:Visitor\_attractions\_in\_Milan .
16. dbpcat:Gardens\_in\_Milan      skos:broader dbpcat:Visitor\_attractions\_in\_Milan .
17. dbpedia:Biblioteca\_Ambrosiana skos:subject dbpcat:Visitor\_attractions\_in\_Milan .
18. dbpedia:Brera\_Academ         skos:subject dbpcat:Visitor\_attractions\_in\_Milan .
19. dbpedia:Via\_Montenapoleone skos:subject dbpcat:Visitor\_attractions\_in\_Milan .
20. dbpedia:Castello\_Sforzesco    skos:subject dbpcat:Visitor\_attractions\_in\_Milan .
21. dbpedia:Pinacoteca\_di\_Brer skos:subject dbpcat:Visitor\_attractions\_in\_Milan .

# Sample query

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?s ?o
FROM <http://dbpedia.org/data/Category:Visitor_attractions_in_Milan.rdf>
WHERE {?s rdf:type skos:Concept.
      ?s skos:broader ?o . }
```

?s	?o
dbcat:Visitor_attractions_in_Milan	dbcat:Visitor_attractions_by_city
dbcat:Visitor_attractions_in_Milan	dbcat:Visitor_attractions_in_Italy
dbcat:Visitor_attractions_in_Milan	dbcat:Milan

# Matching RDF Literals

- ◆ Literals in RDF contains language and data-type information and hence the matching **may not return matched graph.** For instance
- ◆ “Milan Cathedral”@en (label in English), “Milan Cathedral”@it (in Italy) , “40000”^^xsd:integer (no of population)

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?s
FROM <http://dbpedia.org/data/Milan_Cathedral.rdf>
WHERE { ?s rdfs:label “Milan Cathedral”. }
```

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?s
FROM <http://dbpedia.org/data/Milan_Cathedral.rdf>
WHERE { ?s rdfs:label “Milan Cathedral”@en. }
```

# RDF Term Constraints

- ◆ SPARQL allows restrictions of the solutions by constructing constraints within **FILTER clauses**.
- ◆ An RDF term bound to a variable appears in the results if the constraint in the FILTER expression, applied to the term, evaluates to TRUE.

```
PREFIX dbpprop: <http://dbpedia.org/property/>
SELECT ?s
FROM <http://dbpedia.org/data/Milan_Cathedral.rdf>
WHERE { ?s dbpprop:capacity ?c .
        FILTER (?c > 50000 ) }
```

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpprop: <http://dbpedia.org/property/>
SELECT ?s
FROM <http://dbpedia.org/data/Milan_Cathedral.rdf>
WHERE { ?s rdfs:label ?l .
        FILTER (regex(str(?l), 'Cathedral' ) || (?c > 50000 ) ) }
```

# Optional patterns

```
1. PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
2. PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3. PREFIX dbcat: <http://dbpedia.org/resource/Category:>
4. PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
5. SELECT ?s ?label ?lat ?long
6. FROM <http://dbpedia.org/data/Milan_Cathedral.rdf>
7. FROM <http://dbpedia.org/data/Basilica_of_Sant%27Ambrogio.rdf>
8. WHERE { ?s skos:subject dbcat:Churches_in_Milan.
9.           ?s rdfs:label ?label .
10.          FILTER langMatches( lang(?label) , "EN" ) .
11.          ?s geo:lat ?lat .
12.          ?s geo:long ?long . }
```

?s	?label	?lat	?long
dbpedia:Basilica_of_Sant%27Ambrogio	"Basilica of Sant' Ambrogio"@en	45.4624	9.1758

# Optional patterns

```
1. PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
2. PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3. PREFIX dbcat: <http://dbpedia.org/resource/Category:>
4. PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
5. SELECT ?s ?label ?lat ?long
6. FROM <http://dbpedia.org/data/Milan_Cathedral.rdf>
7. FROM <http://dbpedia.org/data/Basilica_of_Sant%27Ambrogio.rdf>
8. WHERE { ?s skos:subject dbcat:Churches_in_Milan .
9.           ?s rdfs:label ?label .
10.          FILTER langMatches( lang(?label) , "EN" ) .
11.          OPTIONAL {
12.            ?s geo:lat ?lat .
13.            ?s geo:long ?long . }
14. }
```

?s	?label	?lat	?long
dbpedia:Milan_Cathedral	"Milan Cathedral"@en	null	Null
dbpedia:Basilica_of_Sant% 27Ambrogio	'Basilica of Sant' Ambrogio'@en	45.4624	9.1758

# Union – to provide alternative graph pattern

```
1. PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
2. PREFIX dbcat: <http://dbpedia.org/resource/Category:>
3. PREFIX yago: <http://dbpedia.org/class/yago/>
4. SELECT ?s
5. FROM <http://dbpedia.org/data/Milan_Cathedral.rdf>
6. WHERE {
7.     { ?s skos:subject dbcat:Churches_in_Milan . }
8.     UNION
9.     { ?s skos:subject yago:ChurchesInMilan . }
10. }
```

- ◆ **Projection modifier: Select**
- ◆ **Order Modifier: Order by**
- ◆ **Distinct and reduced modifier: Distinct and reduced**
- ◆ **Limit and offset modifier: Limit, offset**

An example of query that it is possible to issue against one of the RKB Explorer repository of CORDIS projects (<http://cordis.rkbexplorer.com/sparql>)

```
1. PREFIX akts: <http://www.aktors.org/ontology/support#>
2. PREFIX akt: <http://www.aktors.org/ontology/portal#>
3. SELECT DISTINCT ?name ?tel
4. WHERE { ?areaOfInterest akts:has-pretty-name ?areaLabel .
5.           FILTER regex(str(?areaLabel), "information systems") .
6.           ?project akt:addresses-generic-area-of-interest ?
7.                     areaOfInterest .
8.           ?project akt:has-project-leader ?projectLeader .
9.           ?projectLeader akt:full-name ?name .
10.          ?projectLeader akt:has-telephone-number ?tel .
11.          FILTER regex(str(?tel), "'+'39-02") .
12.          LIMIT 10
```

# Query Forms **CONSTRUCT**, **ASK**, and **DESCRIBE**

- ◆ The **CONSTRUCT** query form returns an RDF graph specified by the query designer as graph template.
- ◆ Example: the graph template at row 4 states that a church is an ex:Big\_Church if it has a capacity larger than 30,000 people.

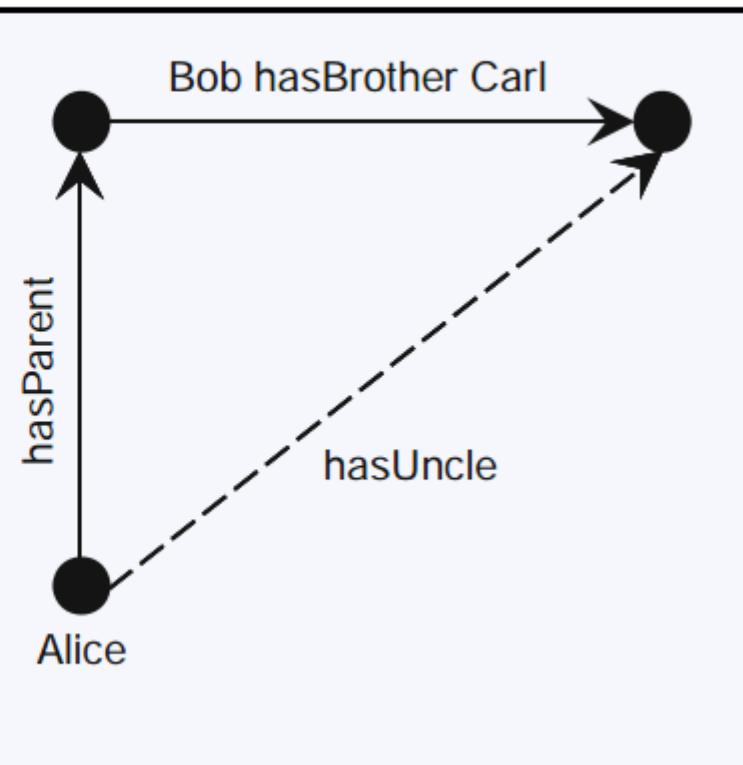
**SPARQL for constructing a triple stating that Milan\_Cathedral has type Big\_Church**

a)	<pre>1. PREFIX dbpprop: &lt;http://dbpedia.org/property/&gt; 2. PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; 3. PREFIX ex: &lt;http://ex.org/resources/&gt; 4. CONSTRUCT { ?s rdf:type ex:Big_Church . } 5. FROM &lt;http://dbpedia.org/data/Milan_Cathedral.rdf&gt; 6. WHERE { ?s dbpprop:capacity ?c .            FILTER (?c &gt; 30000) }</pre>
b)	<pre>dbpedia:Milan_Cathedral rdf:type ex:Big_Church</pre>

# Query Forms **CONSTRUCT**, **ASK**, and **DESCRIBE**

- ◆ The **CONSTRUCT** query form returns an RDF graph specified by the query designer as graph template.

The SPARQL construct query (b) has the expressive power as the logical rule “uncle”

a)	1. Alice hasParent Bob . 2. Bob hasBrother Carl .	 <p>Bob hasBrother Carl</p> <p>Alice</p> <p>hasParent</p> <p>hasUncle</p>
b)	1. CONSTRUCT { ?x hasUncle ?z . } 2. WHERE { ?x hasParent ?y . 3. ?y hasBrother ?z . }	
c)	1. Alice hasUncle Carl .	

# ASK Form

- ◆ ASK is applicable to check if a given query can be answered by a SPARQL endpoint, without being interested in the actual result.
- ◆ When an ASK query is issued to a SPARQL processor, the processor answers whether or not a solution exists.

**SPARQL ASK query that checks whether a resource has latitude and longitude**

a)	<pre>1. PREFIX geo: &lt;http://www.w3.org/2003/01/geo/wgs84_pos#&gt; 2. ASK 3. FROM &lt;http://dbpedia.org/data/Milan_Cathedral.rdf&gt; 4. WHERE { ?s geo:lat ?lat . 5.           ?s geo:long ?long . }</pre>
b)	<pre>1. PREFIX geo: &lt;http://www.w3.org/2003/01/geo/wgs84_pos#&gt; 2. ASK 3. FROM &lt;http://dbpedia.org/data/ Basilica_of_Sant%27Ambrogio.rdf&gt; 4. WHERE { ?s geo:lat ?lat . 5.           ?s geo:long ?long . }</pre>

## DESCRIBE Form

- ◆ A DESCRIBE SPARQL query returns an RDF graph determined by the SPARQL endpoint.
- ◆ The query can have two forms: the very simple or the slightly more complex

**Two SPARQL DESCRIBE queries returning a description of Sant'Ambrogio church**

a)	1. DESCRIBE < <a href="http://dbpedia.org/resource/Basilica_of_Sant%27Ambrogio">http://dbpedia.org/resource/Basilica_of_Sant%27Ambrogio</a> >
b)	1. PREFIX geo: < <a href="http://www.w3.org/2003/01/geo/wgs84_pos#">http://www.w3.org/2003/01/geo/wgs84_pos#</a> > 2. PREFIX xsd: < <a href="http://www.w3.org/2001/XMLSchema#">http://www.w3.org/2001/XMLSchema#</a> > 3. DESCRIBE ?s 4. WHERE { ?s geo:lat "45.4624"^^xsd:float . 5. ?s geo:long "9.1758"^^xsd:float . }

# SPARQL Semantics

---

- ◆ The core of the SPARQL semantics is the so-called SPARQL algebra, a system of clearly defined computational operations which can be used to calculate the result of a query.
- ◆ Some of the operations used in SPARQL are very similar to SQL.
- ◆ The main focus here is to translate query in SPARQL algebra.

# Translating Queries to SPARQL Algebra

---

- ◆ Operators associated to BGP (Basic Graph Pattern): Join (Conjunctions), LeftJoin (operational conditions), Filter, and Union.
- ◆ The **operators** returns the result of a sub-query.
- ◆ ASPARQL engine has to map all the variables in the BGP (the set  $V$ ) into the terms in the RDF graph (the set  $T$ ). A term can be an IRI, a blank node, or a literal. A solution  $m$  is a partial function  $\mu: V \rightarrow T$ .
- ◆ A solution is valid if  $\mu$  maps  $V$  in a subgraph of the RDF graph against which the query is evaluated

## Translating Queries to SPARQL Algebra [2]

---

- ◆ For instance, the solution  $m$  that maps the variable  $x$  in the term  $t$ , the variable  $y$  in the term  $s$ , and the variable  $z$  in the term  $r$ , is:

$$\mu(?x \rightarrow t, ?y \rightarrow s, ?z \rightarrow r) \equiv \{?x, t\}, \{?y, s\}, \{?z, r\}$$

- ◆ The solution of a BGP is a bag of mapping, an unordered set of solutions that admits duplicates.

## four algebraic operators of BGP

---

- ◆ Filter(expr, pattern): this operator filters the pattern in input. It gives formal semantics to the following block in a WHERE clause:

```
WHERE { pattern  
        FILTER (expr)}
```

- ◆ The operator evaluates the expression expr for the solution of pattern and it gives as a result the solution of the pattern if expr evaluates true, otherwise it gives no results.
- ◆ Formally, if the solution of the pattern is  $\Omega$ , the filter operation returns:

$$\text{Filter(expr, } \Omega) = \{\mu | \mu \in \Omega \wedge \exp(\mu) = \text{true}\}.$$

## four algebraic operators of BGP

---

- ◆ Filter(expr, pattern): this operator filters the pattern in input. It gives formal semantics to the following block in a WHERE clause:

```
WHERE { pattern  
        FILTER (expr)}
```

- ◆ The operator evaluates the expression expr for the solution of pattern and it gives as a result the solution of the pattern if expr evaluates true, otherwise it gives no results.
- ◆ Formally, if the solution of the pattern is  $\Omega$ , the filter operation returns:

$$\text{Filter(expr, } \Omega) = \{\mu | \mu \in \Omega \wedge \exp(\mu) = \text{true}\}.$$

## four algebraic operators of BGP

- ◆  $\text{Join}(\text{pattern1}, \text{pattern2})$ : this operator computes the join between the two patterns in input. It gives formal semantics to the following block in a WHERE clause:

```
WHERE { {pattern1}  
        {pattern2}}
```

- ◆ To give the semantics of the join operator, some terminology must be introduced. Denoting with  $\text{dom}(\mu)$  the domain of the solution  $\mu$  (i.e., the set of the variable  $V$  in  $\mu$ ), two solutions  $\mu_1$  and  $\mu_2$  are compatible if:

$$\forall v \in (\text{domain}(\mu_1) \cap \text{domain}(\mu_2)): \mu_1(v) = \mu_2(v)$$

- ◆  $\text{merge}(\mu_1, \mu_2) = \mu_1 \text{ set} - \text{union } \mu_2$

## four algebraic operators of BGP

- ◆ if the solutions of pattern1 and pattern2 are  $\Omega_1, \Omega_2$ , the join operator returns:  
$$\text{Join}(\Omega_1, \Omega_2) = \{\text{merge}(\mu_1, \mu_2) | \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \wedge \mu_1, \mu_2 \text{ compatible}\}$$
- ◆ Union(pattern1, pattern2): this operator computes the union of the two patterns in input. It gives formal semantics to the following block in a WHERE clause:  

```
WHERE { {pattern1}
        UNION {pattern2} }
```
- ◆ If the solutions of pattern1 and pattern2 are  $\Omega_1$  and  $\Omega_2$ , the union operator returns:  
$$\text{Union}(\Omega_1, \Omega_2) = \{\mu | \mu \in \Omega_1 \vee \mu \in \Omega_2\}$$

Duplicates in  $\Omega_1$  or  $\Omega_2$  are not removed; if  $\mu$  appears  $n_1$  times as solution of and it appears  $n_2$  times as solution of, then appears  $n_1+n_2$  times in the union.

## four algebraic operators of BGP

- ◆ if the solutions of pattern1 and pattern2 are  $\Omega_1, \Omega_2$ , the join operator returns:  
$$Join(\Omega_1, \Omega_2) = \{merge(\mu_1, \mu_2) | \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \wedge \mu_1, \mu_2 \text{ compatible}\}$$
- ◆ Union(pattern1, pattern2): this operator computes the union of the two patterns in input. It gives formal semantics to the following block in a WHERE clause:  
WHERE { {pattern1}  
          UNION {pattern2} }
- ◆ If the solutions of pattern1 and pattern2 are  $\Omega_1$  and  $\Omega_2$ , the union operator returns:  
$$\text{Union}(\Omega_1, \Omega_2) = \{\mu | \mu \in \Omega_1 \vee \mu \in \Omega_2\}$$
- ◆ Duplicates in  $\Omega_1$  or  $\Omega_2$  are not removed; if  $\mu$  appears  $n_1$  times as solution of and it appears  $n_2$  times as solution of, then appears  $n_1+n_2$  times in the union.

## four algebraic operators of BGP

- ◆ LeftJoin (pattern1, pattern2, expr): this operator computes the left-join (or semijoin) between the two patterns in input evaluating the expression expr. It gives formal semantics to the following block in a WHERE clause:

```
WHERE { {pattern1}
        OPTIONAL
        {pattern2
            FILTER(expr)} }
```

- ◆ Diff(pattern1, pattern2, expr) : computes the insiemistic difference between the two patterns while evaluating the expression expr. If the solutions of pattern1 and pattern2 are  $\Omega_1$  and  $\Omega_2$ , this operator returns:

$$\text{Diff}(\Omega_1, \Omega_2, \text{expr}) = \{\mu | \mu \in \Omega_1 : \forall \mu^1 \in \Omega_2 : \mu, \mu' \text{ are not compatible} \vee (\mu, \mu' \text{ compatible} \wedge \text{exp}(\text{merge}(\mu, \mu')) = \text{false}\}$$

## four algebraic operators of BGP

---

- ◆ LeftJoin (pattern1, pattern2, expr):

$$\begin{aligned} \text{LeftJoin}(\Omega_1, \Omega_2, \text{expr}) = & \text{Filter(expr, Join}(\Omega_1, \Omega_2) \\ & \cup \text{Diff}(\Omega_1, \Omega_2, \text{expr})) \end{aligned}$$

- ◆ Notice that the bag of solutions returned by the Filter and Diff operators are disjoined, thus they cannot generate duplicates.