OWL

Outline

- Clarification:
  - What are Ontologies?
- Revisited:
  - How we already have learned to express ontologies
- Web Ontology Language -OWL:
  - extending expressivity
- Semantics of & Reasoning with OWL:
  - using the extended expressivity

Clarification – What are Ontologies?

- "Ontology (languages)" for the Semantic Web:

  We aim at a (XML-based) language to formally describe concepts, instances, relations and axioms, i.e. data+structure in order to enable machine-processable reasoning and exchange of data.

→ Knowledge representation, exchange, combination (inference of new knowledge!)

Clarification – Knowledge Representations

- Classical split in AI knowledge systems:
  - Tbox
    - Terminology, relations, classes, properties
    - This is in OWL
  - Abox
    - Assertions, instances
    - RDF data sets or triples

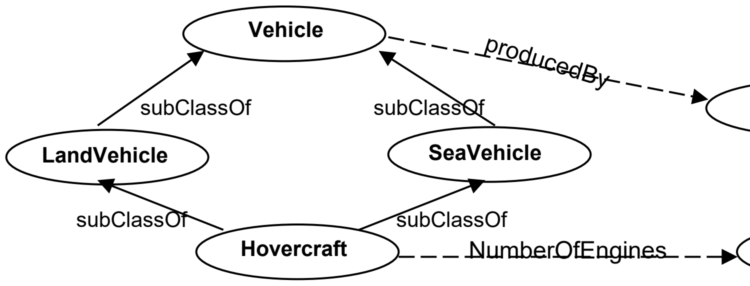
What’s inside an ontology?

- Concepts: Classes + class-hierarchy
  - instances
- Properties: often also called “Roles” or “Slots”
  - labeled instance-value-pairs
- Axioms/Relations:
  - relations between classes (disjoint, covers)
  - inheritance (multiple? single?)
  - restrictions on slots (type, cardinality)
  - Characteristics of slots (symm., trans., …)
- reasoning tasks:
  - Classification: Which classes does an instance belong to?
  - Subsumption: Does a class subsume another one?
  - Consistency checking: Is there a contradiction in my axioms/instances?

Outline

- Clarification:
  - What are Ontologies?
- Revisited:
  - How we already have learned to express ontologies
  - Why is RDF/RDFS not enough?
- Web Ontology Language -OWL:
  - extending expressivity
- Semantics of & Reasoning with OWL:
  - using the extended expressivity

RDF/RDFS



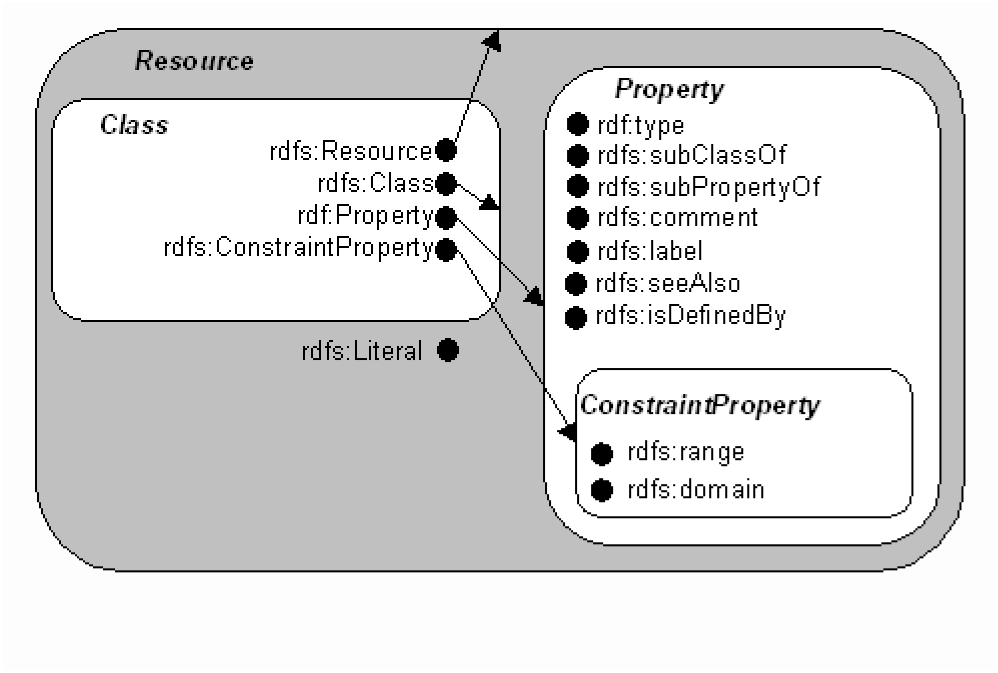
RDF + RDFS:

- RDF: triples for making assertions about resources
- RDFS extends RDF with "schema vocabulary", e.g.:
  - Class, Property
  - type, subClassOf, subPropertyOf
  - range, domain

→ representing simple assertions, taxonomy + typing

RDF Schema – Strange Things

- rdfs:Resource is the superclass of everthing, but itself is an instance of its subclass rdfs:Class
- rdfs:Class itself can be an instance of itself
- rdf:Property is an instance of rdfs:Class (all RDF properties can be rdfs:Class)



Limits of RDF and RDFS

- Local scope of properties
  - No localised range and domain constraints:
  - We cannot declare range restrictions that apply to some classes only
  - Can’t say that the range of hasChild is person when applied to persons and elephant when applied to elephants
- Disjointness of classes
  - In RDF, we cannot define classes are disjoint, we can only state the subclass relationship
    - E.g., male and female are disjoint
- Boolean combinations of classes
  - RDF does not support boolean combination of classes
    - E.g. cannot describe new classes in terms of combinations (e.g., *union* and *intersection*) of other classes
- Cardinality restriction
  - RDF does not support to place restrictions on how many distinct values a property may or must take
    - E.g. One person can only have no more than 2 parents; one course should have at least one lecturer

Limits of RDF and RDFS

- Special characteristics of properties (RDF does not support):
  - specifying that a given property (such as ex:hasAncestor) is *transitive*, e.g., that if A ex:hasAncestor B, and B ex:hasAncestor C, then A ex:hasAncestor C.
  - specifying that a given property is a unique identifier (or *key*) for instances of a particular class.
  - Inverse (e.g., eat, eat-by)
  - specifying that two different classes (having different URIrefs) actually represent the same objects (*class identity*)
  - specifying that two different instances (having different URIrefs) actually represent the same individual (*object identity*).
- Difficult to provide reasoning support

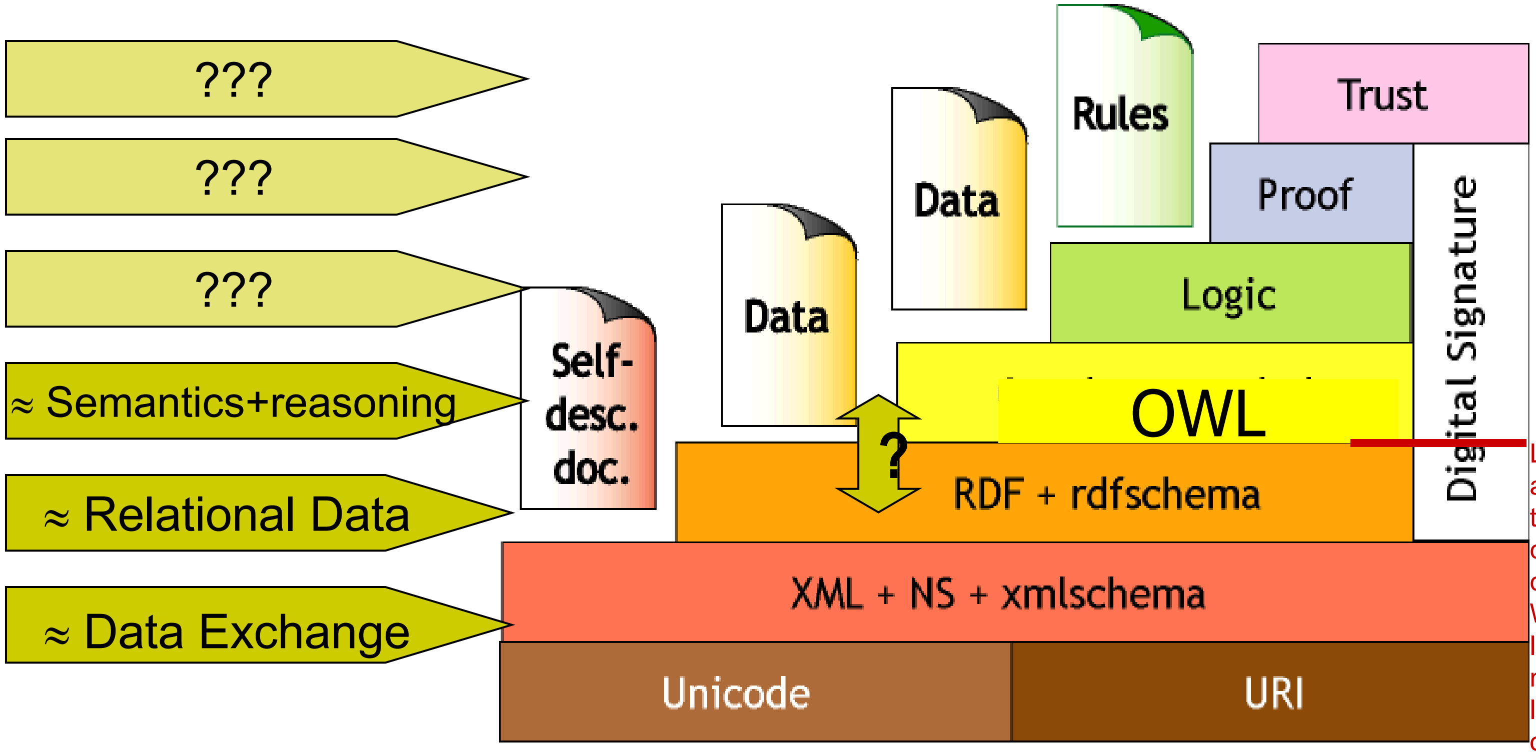
→OWL (the **W**eb **O**ntology **L**anguage)

Web Ontology Language Requirements

Desirable features identified for Web Ontology Language:

- Extends existing Web standards
  - Such as XML, RDF, RDFS
- Easy to understand and use
  - Should be based on familiar KR idioms
- Formally specified
- Of “adequate” expressive power
- Possible to provide automated reasoning support

(In)famous "Layer Cake"



Level of agreement/dispute in the research community is currently here: What kind of logic/language do we need? Where does logic start and ontology end?

We'll see:

- **Relationship between layers is blurry**
- **OWL (DL) extends "Description Logic(DL) subset" of RDF**

Description Logic

Propositional Logic

- Basics: Deals with propositions, which are statements that can be either true or false.
- Elements: Uses logical connectives like AND, OR, NOT, IMPLIES to form compound propositions.
- Scope: Concerned only with the truth values of the propositions and how they combine.
- Example: "It is raining" (P) and "It is cold" (Q). You can combine them into "It is raining AND it is cold" ($P \land Q$).

Description Logic

First-Order Predicate Logic (FOPL)

- Basics: Extends propositional logic by dealing with predicates, which are statements about objects, and quantifiers.
- Elements: Uses variables, predicates, and quantifiers (like ∀ for "for all" and ∃ for "there exists").
- Scope: More expressive, can make statements about objects and their properties.
- Example: "All humans are mortal" can be expressed as $\forall x\ (Human(x) \rightarrow Mortal(x))$.In short, propositional logic is about combining whole propositions, while first-order predicate logic delves deeper into the structure of the propositions themselves and the relationships between objects.

Description Logic (DL)

- Basics: A family of formalisms used for representing and reasoning about knowledge of the world. DL is designed to describe and reason about the concepts and relationships within a domain.
- Elements: Uses classes (or concepts), properties (or roles), and individuals.
- Focus: Primarily used in knowledge representation and ontology languages like OWL (Web Ontology Language).
- Expressiveness: More expressive than propositional logic but usually less expressive than full FOPL.
- Example: Describing a class of "Students" who are enrolled in a "Course" with properties defining the relationship.

Description Logic (DL)

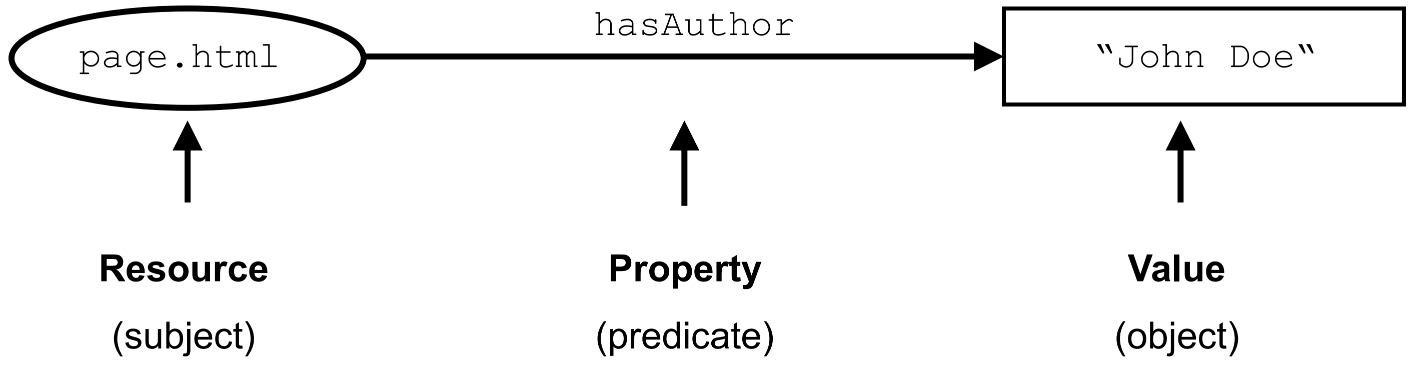
- **Comparison: DL and FOPL**
  - **Expressiveness**: FOPL is generally more expressive than DL, allowing for more complex statements.
  - **Focus**: DL is specifically tailored for structured knowledge representation, while FOPL is more general-purpose.

Outline

- Clarification:
  - What are Ontologies?
- Revisited:
  - How we already have learned to express ontologies
- Web Ontology Language -OWL:
  - extending expressivity
- Semantics of & Reasoning with OWL:
  - using the extended expressivity

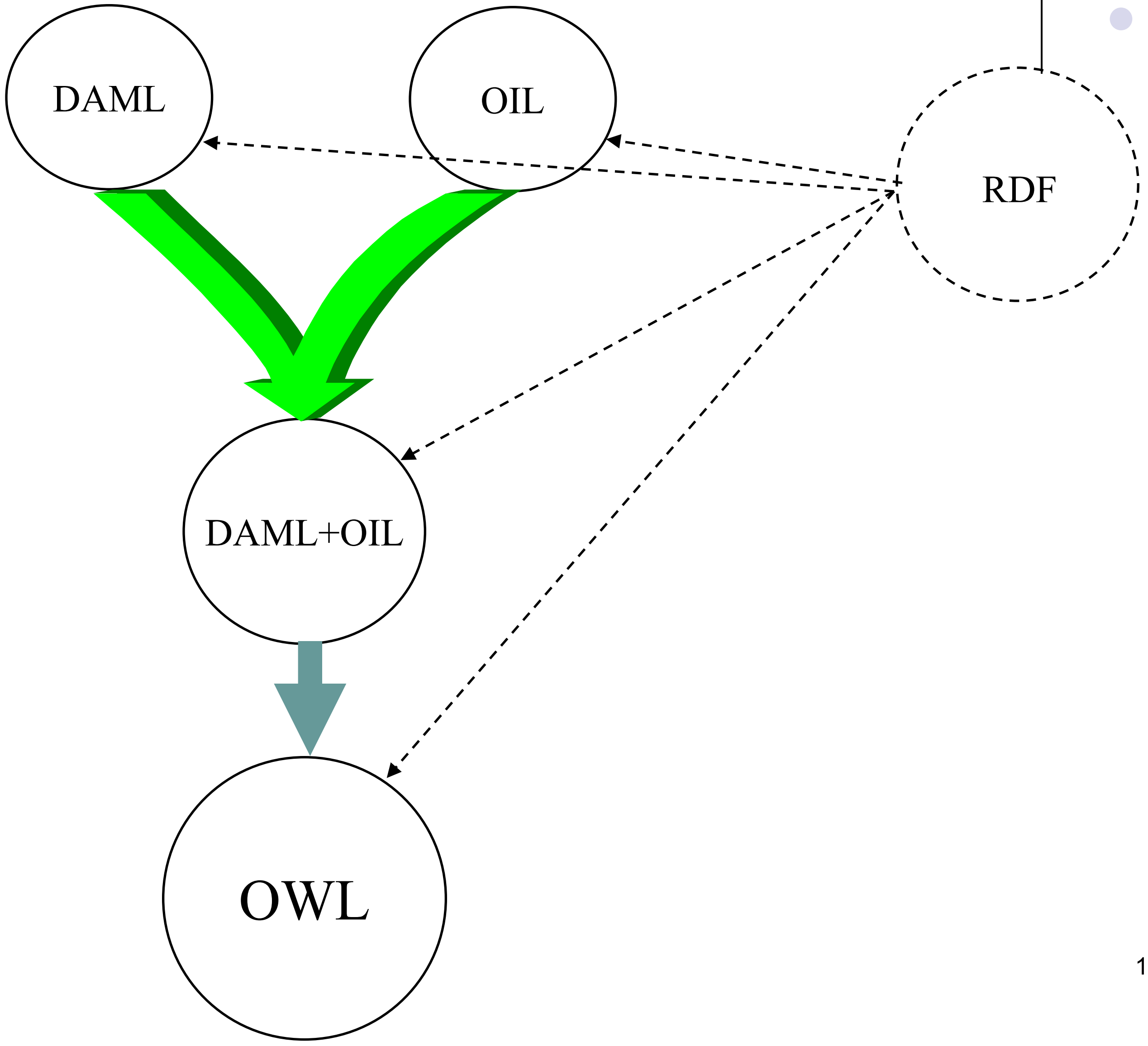
OWL-RDFS Relationship

- Both use the same data model:



- OWL extends vocabulary and adds axioms to express more complex relations of classes and properties

Origins of OWL



History

- Two languages developed to satisfy above requirements
  - OIL (Object Inference Layer): developed by group of (largely) European researchers (several from EU OntoKnowledge project)
  - DAML-ONT: developed by group of (largely) US researchers (in DARPA DAML programme)
- Efforts merged to produce DAML+OIL
  - Development was carried out by “Joint EU/US Committee on Agent Markup Languages”
  - Extends (“DL subset” of) RDF
- DAML+OIL submitted to W3C as basis for standardisation
  - Web-Ontology (WebOnt) Working Group formed
  - WebOnt group developed OWL language based on DAML+OIL
  - OWL language now a W3C Recommendation (01.02.2004)

OWL

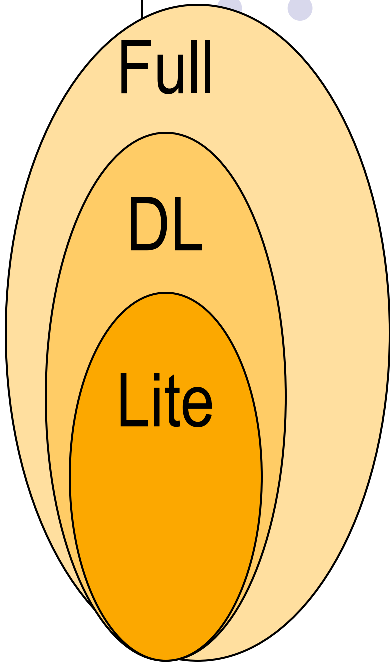
- Web Ontology Language
- Provides rich set of concepts for defining classes and properties
- Primary keys
- Cardinality restrictions
- Class equivalence, intersections and unions

OWL Language - Overview

- Three species of OWL
  - OWL DL stays in **Description Logic** fragment
  - OWL Lite is “easier to implement” subset of OWL DL
  - OWL Full is union of OWL syntax and RDF

- OWL DL benefits from many years of Description Logic
  - Well defined semantics
  - Formal properties well understood (complexity, decidability)
  - Well-developed reasoning algorithms
  - Implemented systems (highly optimised)

- OWL Full has all that and all the possibilities of RDF/RDFS which destroy decidability



Syntactic layering
Semantic layering

Three dialects of OWL

- OWL Lite
  - Sublanguage of OWL DL
  - Easier to learn and to implement
  - Only provide restricted expressivity.
- OWL DL
  - Sublanguage of OWL Full which corresponds to Description Logic
  - Permits efficient reasoning support
  - Lose full compatibility with RDF
  - Every legal OWL DL document is a legal RDF document, but not the other way around.
- OWL Full
  - OWL plus RDF and RDFS
  - Fully upward-compatible with RDF (both syntactically and semantically)
  - Any legal RDF document is also a legal OWL FULL document
  - Language is very powerful and is undecidable, impossible to provide complete (or efficient) reasoning support

Description Logic Family

- DLs are a family of logic based KR formalisms
- Particular languages mainly characterized by:
  - Set of constructors for building complex concepts and roles from simpler ones
  - Set of axioms for asserting facts about concepts, roles and individuals

- Examples:
  - "Female persons"
    - Person $\sqcap$ Female
  - "Non-female persons"
    - Person $\sqcap$ $\neg$Female
  - "Persons that have a child"
    - Person $\sqcap$ $\exists$hasChild.Person
  - "Persons all of whose children are female"
    - Person $\sqcap$ $\forall$hasChild.Female
  - "Persons that are employed or self-employed"
    - Person $\sqcap$ (Employee $\sqcup$ SelfEmployed)
  - "Persons that have at most one father"
    - Person $\sqcap$ $\leq$1.hasFather

Description Logic Family
Necessary and sufficient conditions

- *Inclusion axioms* provide **necessary** conditions:
  - concept ⊑ definition
  - Necessary condition: superclass, implies, partial
- *Equivalence axioms* provide **necessary and sufficient** conditions:
  - Necessary and sufficient condition: equal, complete

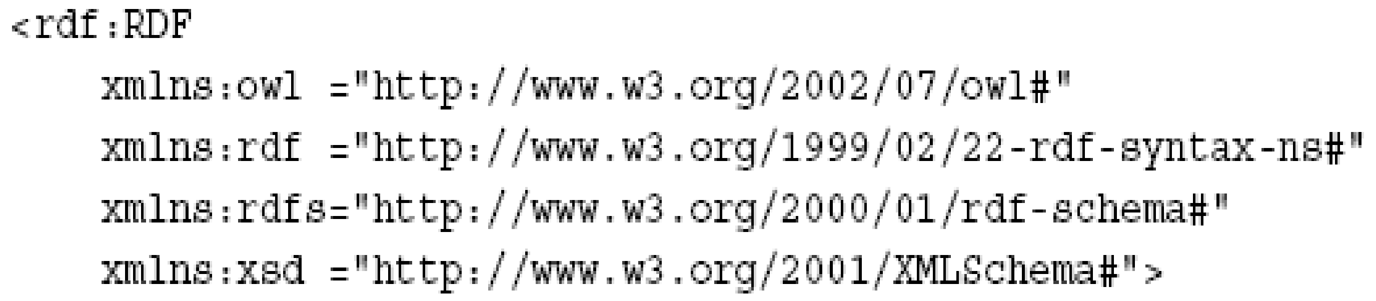
$$\text{concept} \equiv \text{definition} \begin{cases} \text{concept} \sqsubseteq \text{definition} \quad \text{and} \\ \text{definition} \sqsubseteq \text{concept} \end{cases}$$

Material

- Online OWL Tutorial
- https://protege.stanford.edu/conference/2006/submissions/slides/OWLTutorial_Part1.pdf

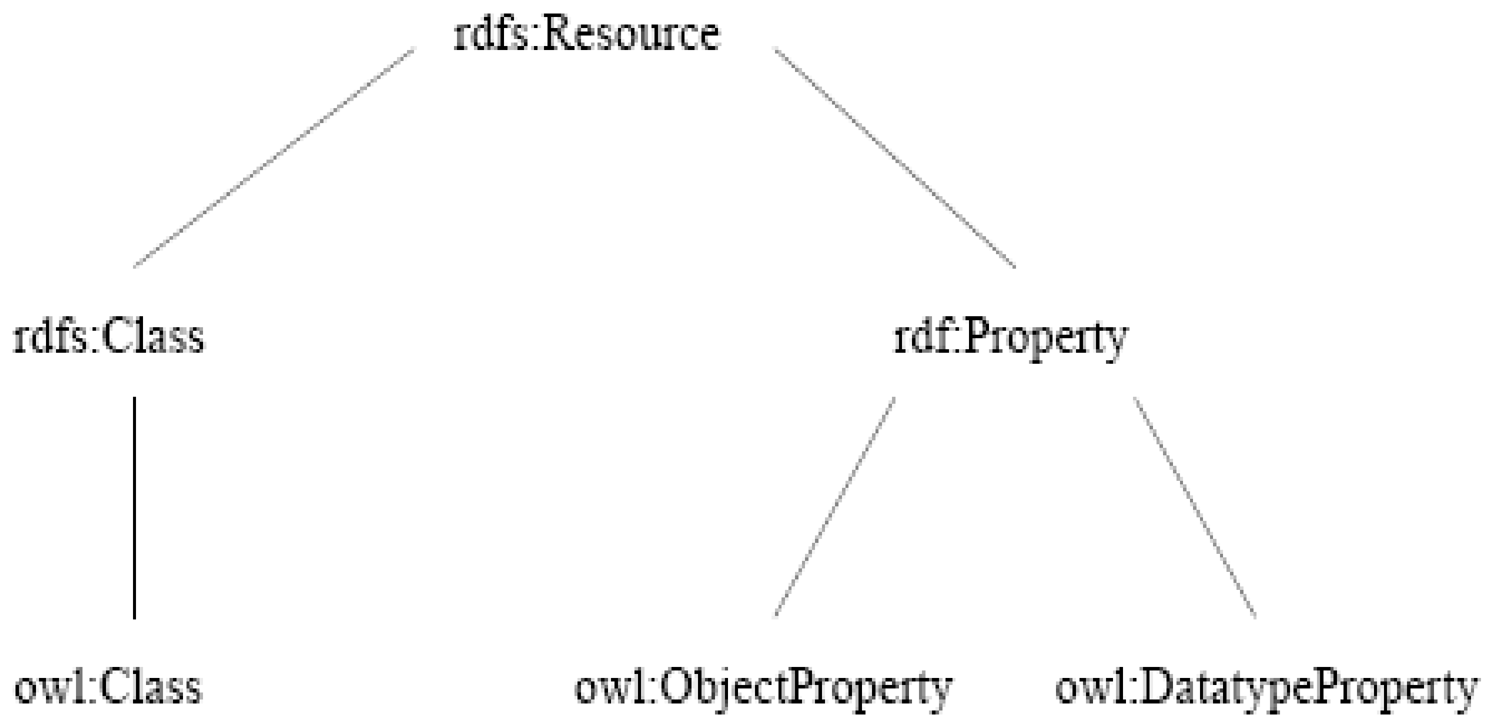
OWL Language

- Syntax
  - OWL is bulit on RDF/RDFS and uses RDF/XML-based syntax
- Header



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

OWL and RDF/RDFS



OWL Class Elements

- owl:Class

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

- owl:disjoinWith

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

- owl:equivalentClass

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

- owl:Thing and owl:Nothing

OWL Property Elements

- Object property (relate objects to other objects)

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

- Data type property (relate objects to datatype values)

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

OWL Instances

- Instances of classes are declared as in RDF
- Unlike typical database system, OWL does not adopt the unique-names assumption (two instances with different names or IDs does not imply that they are different individuals (they can be the same individuals))
  - If a course can only be taught by one teacher, then we can imply that these two instances are the same individual.

```
<course rdf:ID="CIT1111">
 <isTaughtBy rdf:resource="#949318"/>
 <isTaughtBy rdf:resource="#949352"/>
</course>
```

OWL Data types

- In fact, not even all of the built-in XML Schema data types can be used in OWL.
- The OWL reference document lists the XML Schema data types that can be used, but these includes the most frequently used types, such as string, integer, boolean, time and date.

OWL Versioning

- Can only indicating informal versioning information, none of them carry any formal meaning:
  - Owl:priorVersion- indicate earlier versions of the current ontology
  - Owl:versionInfo - a string giving information about current version
  - Owl:backwardCompatibleWith - a reference to another ontology (reuse this ontology by using owl:imports)
  - Owl:incompatibleWith- the containing ontology is the later version of the referenced ontology but is not backward-compatible with it.

Describing Classes in OWL

Classes: OWL vs. RDFS

- OWL allows greater expressiveness, but
- OWL (DL/Lite) puts certain constraints on the use of RDF
  - For instance: a class may not act as an instance of another (meta)class (the same holds for properties)

→ OWL has got it's own Class identifier

RDFS

```
<rdfs:Class rdf:ID="River">
      <rdfs:subClassOf rdf:resource="#Stream"/>
</rdfs:Class>
```

OWL

```
<owl:Class rdf:ID="River">
      <rdfs:subClassOf rdf:resource="#Stream"/>
</owl:Class>
```

Class Description

RDF constructs

What can you express in RDF/RDFS?
Not too much…

Employee ⊑ Person

- `rdfs:subClassOf` … class hierarchy, necessary conditions (also equivalence is expressible because A ⊑ B and B ⊑ A → A ≡ B)

Employee(alice)

- `rdf:type` … class membership

… OWL provides more expressive constructs to express the DL features!

Complex Classes

Union of Classes

- Instances of the Union of two Classes are either the instance of one or both classes

$\text{Person} \equiv \text{Man} \sqcup \text{Woman}$

```
<owl:Class rdf:ID="Person">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Woman" />
    <owl:Class rdf:about="#Man" />
  </owl:unionOf>
</owl:Class>
```

Complex Classes

Intersection of Classes

- Instances of the Intersection of two Classes are simultaneously instances of both class

$$Man \equiv Person \sqcap Male$$

```
<owl:Class rdf:ID="Man">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Person" />
    <owl:Class rdf:about="#Male" />
  </owl:intersectionOf>
</owl:Class>
```

one of: Enumerated Classes

- Specify a class via a direct enumeration of its members:

WhineColor ≡ {White, Rose, Red}

```
<owl:Class rdf:ID="WineColor">
<rdfs:subClassOf rdf:resource="#WineDescriptor"/>
 <owl:oneOf rdf:parseType="Collection">
   <owl:Thing rdf:about="#White"/>
   <owl:Thing rdf:about="#Rose"/>
   <owl:Thing rdf:about="#Red"/>
 </owl:oneOf>
</owl:Class>
```

Necessary vs. necessary and sufficient conditions in OWL…

- Necessary condition (⊑) via

  rdfs:subclassOf

- Necessary and sufficient (≡) either by ⊑ + ⊒, i.e. 2 subclass definitions or:

  owl:equivalentClasss

Complex Classes

Property Restrictions

- Defining a Class by restricting its possible instances via their property values
- OWL distinguishes between the following two:
  - Value constraint
    - (Mother ≡ Woman ⊓ ∃hasChild.Person)
  - Cardinality constraint
    - (MotherWithManyChildren ≡ Mother ⊓ ≥3hasChild)
- Property restrictions are local
  - remember RDFS allows only global properties

Complex Classes - Property Restrictions

someValuesFrom

- A Mother is a Woman that has a child (some Person)

$$\text{Mother} \sqsubseteq \text{Woman} \sqcap \exists \text{hasChild}.\text{Person}$$

```
<owl:Class rdf:ID="Mother">
  <rdfs:subClassOf rdf:resource="#Woman" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasChild" />
      <owl:someValuesFrom rdf:resource="#Person" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Complex Classes - Property Restrictions

allValuesFrom

- To express the set of parents that only have female children (daughters) you would write:

$$\text{ParentsWithOnlyDaughters} \sqsubseteq \text{Person} \sqcap \forall \text{hasChild.Woman}$$

```
<owl:Class rdf:ID="ParentsWithOnlyDaughters">
 <rdfs:subClassOf rdf:resource="#Person" />
   <rdfs:subClassOf>
     <owl:Restriction>
        <owl:onProperty rdf:resource="#hasChild" />
        <owl:allValuesFrom rdf:resource="#Woman" />
     </owl:Restriction>
   </rdfs:subClassOf>
    ...
</owl:Class>
```

Complex Classes - Property Restrictions

hasValue

- hasValue allows to define classes based on the existence of *particular* property values, their must be *at least one* matching property value
- The set of all children of the woman MARRY would be expressed like the following:

$MarysChildren \sqsubseteq Person \sqcap \exists hasParent.\{MARRY\}$

```
<owl:Class rdf:ID="MarysChildren">
  <rdfs:subClassOf rdf:resource="#Person" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasParent" />
        <owl:hasValue rdf:resource="#MARRY" />
      </owl:Restriction>
    </rdfs:subClassOf>
   </rdfs:subClassOf>
   ...
</owl:Class>
```

Complex Classes - Property Restrictions

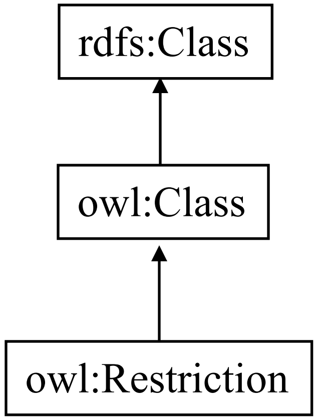
cardinality

- Definition of cardinality: the number of occurrences, either maximum (maxCardinality) or minimum (minCardinality) or exact (cardinality) based upon the context (class) in which it is used
- To define a half-orphan you would write the following:

$$\text{HalfOrphan} \sqsubseteq \text{Person} \sqcap =1\text{hasParent.Person}$$

```
<owl:Class rdf:ID="HalfOrphan">
  <rdfs:subClassOf rdf:resource="#Person" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasParent"/>
        <owl:cardinality rdf:datatype="&xsd;NonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </rdfs:subClassOf>
  …
</owl:Class>
```

Classes as Restrictions on Properties



In summary:

define a class using *LOCAL* restrictions on a specific Property

- **Properties:**
  - allValuesFrom: *rdfs:Class (lite/DL owl:Class)*
  - hasValue: *specific Individual*
  - someValuesFrom: *rdfs:Class (lite/DL owl:Class)*
  - cardinality: *xsd:nonNegativeInteger (in lite {0,1})*
  - minCardinality: *xsd:nonNegativeInteger (in lite {0,1})*
  - maxCardinality: *xsd:nonNegativeInteger (in lite {0,1})*

Describing Properties in OWL

Properties OWL vs. RDF

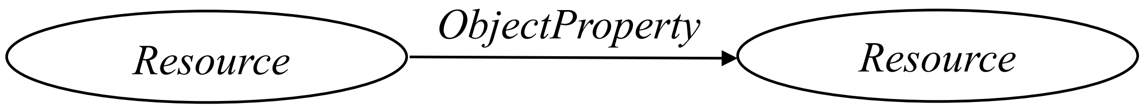
- RDF Schema provides a couple of pre-defined properties:
  - `rdfs:range` used to indicate the range of values for a property.
  - `rdfs:domain` used to associate a property with a class.
  - `rdfs:subPropertyOf` used to specialize a property.

- OWL provides additional poperty classes, which allow reasoning and inferencing, i.e.
  - `owl:functionalProperty`
  - `owl:transitiveProperty`
  - `owl:symetricProperty`

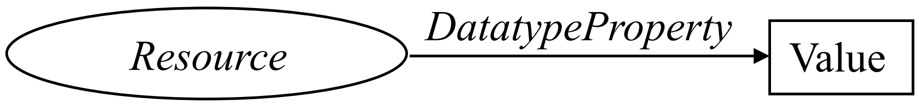
Properties OWL vs. RDF

- OWL (DL and Lite) distinguishes between data type ptoperties and object properties (RDFS does not)

An ObjectProperty relates one **Resource** to another **Resource**:



A DatatypeProperty relates a **Resource** to a **Literal** or an **XML Schema** data type:



Datatype Properties – Example:

- for instance, usage of XSD datatypes:

```
<owl:DatatypeProperty rdf:ID="yearValue">
  <rdfs:domain rdf:resource="#VintageYear" />
  <rdfs:range rdf:resource="&xsd;positiveInteger"/>
</owl:DatatypeProperty>
```

… *object properties* in OWL allow for some more sophisticated definitions than only domain and range, see following slides…

Properties in OWL

Transitive Property

- Example: If person A is an ancestor of person B and B is an ancestor of C then A is also an ancestor of C.

$\text{ancestor}^{+} \subseteq \text{ancestor}$

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

Symmetric Property

- Example: If Mary is akin to John then John is also akin to Mary

$akin^{-} \subseteq akin$ and $akin \subseteq akin^{-}$

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

Properties in OWL

Inverse Property

- Example: If Mary is a child of John then John is the Father of Mary

$\text{hasChild} \subseteq \text{hasParent}^{-}$

```
<owl:ObjectProperty rdf:ID="hasChild">
  <owl:inverseOf rdf:resource="hasParent" />
</owl:ObjectProperty>
```

Properties in OWL

Functional Properties

- A functional property states that the value of range for a certain object in the domain is always the same:
- Example: A child has always the same Father (biological)

$\text{Person} \subseteq \leq 1 \text{hasFather}$

```
<owl:ObjectProperty rdf:ID="hasFather">
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:ObjectProperty>
```

Inverse Functional Properties

- Defines a property for which two different objects cannot have the same value
- E.g., two persons cannot have the same social security number

Person has =1SocialSecurityNumber

```
<owl:ObjectProperty rdf:ID="isTheSocialSecurityNumberfor">
  <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
</owl:ObjectProperty>
```

Functional vs. inverse functional properties

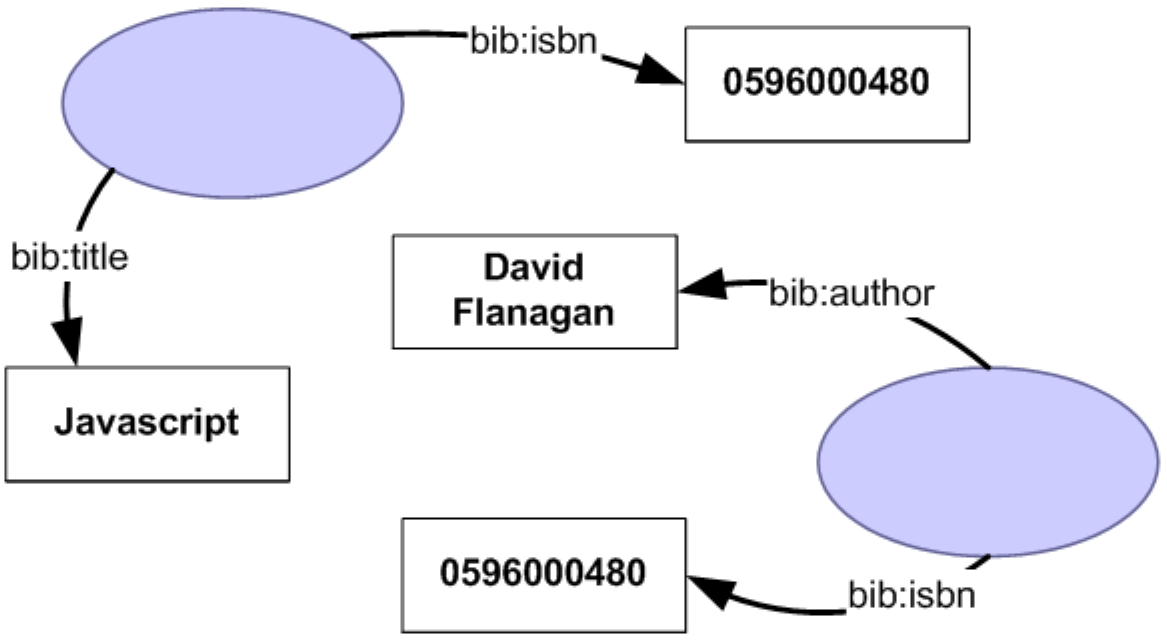
- FunctionalProperty vs InverseFunctionalProperty

	domain	range	example
Functional Property	For a given domain	Range is unique	hasFather: A hasFather B, A hasFather C →B=C
InverseFunctional Property	Domain is unique	For a given range	hasID: A hasID B, C hasID B →A=C

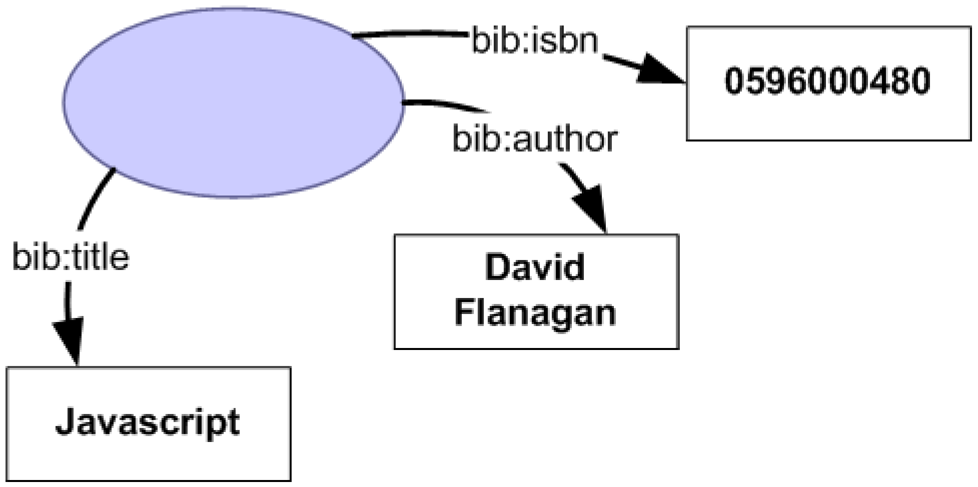
Primary keys

- owl:InverseFunctionalProperty (IFP)
- If a property is inverse functional then the value of that property identifies the subject
- e.g. the isbn relation is inverse functional
  - For any ISBN, there is only one book that has that ISBN

IFP example

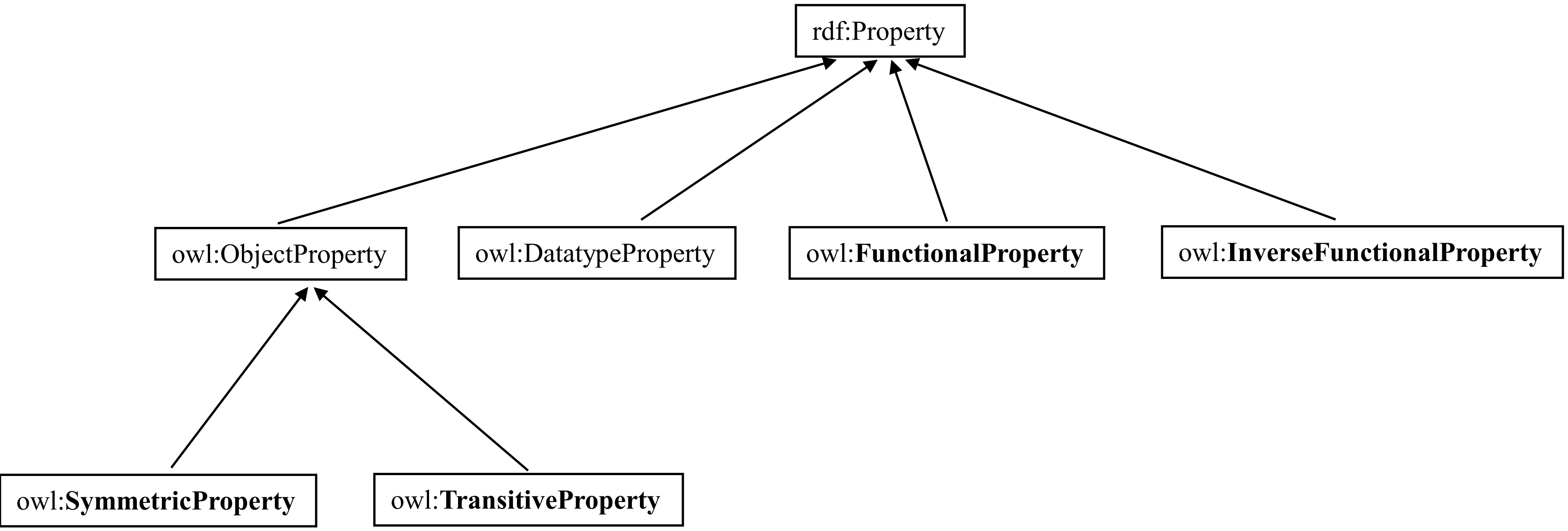

bib:isbn
0596000480
bib:title
Javascript
David Flanagan
bib:author
0596000480
bib:isbn

IFP example


bib:isbn
0596000480
bib:author
David Flanagan
bib:title
Javascript

OWL Property Classes

- The symmetric and transitive property may only used for connecting two resources:


rdf:Property
owl:ObjectProperty
owl:DatatypeProperty
owl:**FunctionalProperty**
owl:**InverseFunctionalProperty**
owl:**SymmetricProperty**
owl:**TransitiveProperty**

Outline

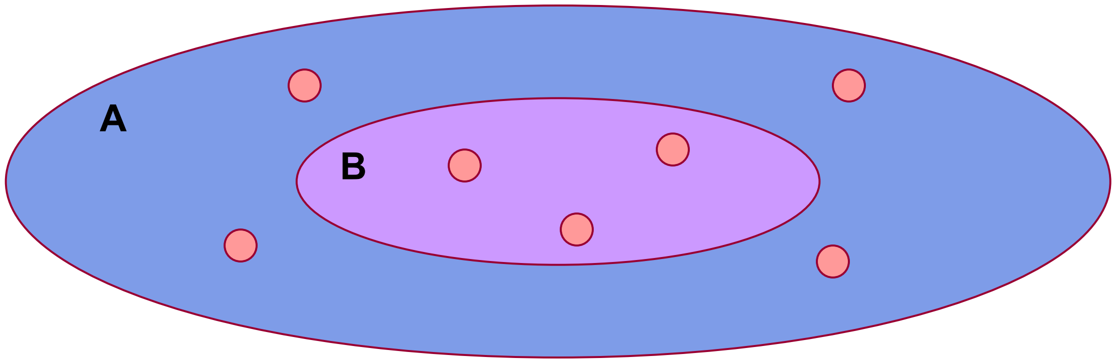
- Clarification:
  - What are Ontologies?
- Revisited:
  - How we already have learned to express ontologies
- Web Ontology Language -OWL:
  - extending expressivity
- Semantics of & Reasoning with OWL:
  - using the extended expressivity

Major kinds of reasoning

- Subsumption
- Inconsistency
- Class membership

Subsumption

- The presented examples contain a number of classes and properties intended to illustrate particular aspects of reasoning in OWL.
- We can make inferences about relationships between classes, in particular subsumption between classes
- Recall that A subsumes B when it is the case that any instance of B must necessarily be an instance of A.



Subsumption

An Example

- Woman ≡ Person ⊓ Female
- Man ≡ Person ⊓ ¬Woman
- Mother ≡ Woman ⊓ ∃hasChild.Person
- Father ≡ Man ⊓ ∃hasChild.Person
- Parent ≡ Father ⊔ Mother
- Grandmother ≡ Mother ⊓ ∃hasChild.Parent

We can further infer:

→ Grandmother ⊑ Person

Inconsistency

- You may define Classes where no individual can fulfill its definition. Via reasoning engines such a definition can be found also in big ontologies.

Inconsistency
Example

- Cow ≡ Animal ⊓ Vegetarian
- Sheep ⊑ Animal
- Vegetarian ≡ ∀eats.¬ Animal
- MadCow ≡ Cow ⊓ ∃eats.Sheep

→ From this you can infer that the class MadCow must be empty, since

1) any madcow is a cow and
2) thus any cow is a vegetarian
3) thus any madcow does not eat animals
4) any madcow eats sheep and
5) thus any madcow eats animals

CONTRADICTION!

Class membership

- You may want to infer that a certain individual *x* is a member of a certain class D.
- Can be reduced to check inconsistency
  - Simply check whether $\neg$*D(x) is inconsistent*

Example

$\text{HappyFather} \sqsubseteq \text{Male} \sqcap \exists \text{hasChild.President}$

Male(GeorgeBush)

Male(GeorgeWBush)

hasChild(GeorgeBush, GeorgeWBush)

President(GeorgeWBush)

Is GeorgeBush a HappyFather?

Can be proven by showing that adding

¬HappyFather(GeorgeBush)

and showing that this is inconsistent.

The Open World Assumption

- RDF and OWL assume an Open World
- Close World: everything we do not know is false
- Open World: everything we do not know is undefined

The Open World Assumption

- _:fred r:spouse _:wilma .
- Is _:fred married to _:betty?
- Closed world: no
- Open world: maybe

The Open World Assumption

- _:fred r:spouse _:wilma .
- _:fred r:spouse _:betty .
- Not inconsistent given our knowledge

The Open World Assumption

- Suppose cardinality of ex:spouse was one
  - i.e. things can only have one spouse
- Are our triples inconsistent now?
  - No – reasoner must assume that there is only one spouse, therefore _:wilma and _:betty are the same individual

Open world assumption (OWA):

- In Database Systems, usually the closed world assumption is made: “The fact in the databases describe completely what I know, all that is not in the database is assumed to be false”.
- This assumption is usually not valid in an open context such as the web!

Example: *resource1* says: “There is a train at 14:00”
“There is a train at 15:00”

My query: “Is there a train at 17:00?”

CWA answer: No!

OWA answer: unknown! (i.e. there could be another resource2 which has info on another train at 17:00!)

Reasoning in OWL is OWL! Assume we have an RDF file with the following triples:

T1 rdf:type Train .
T2 rdf:type Train .
T1 departureTime “15:00” .
T1 departureTime “15:00” .

→ Train ⊓ ∃departureTime.{“17:00”} is **not** inconsistent by OWA!!!

Unique Name Assumption

- If two objects have the same name, then these two objects should be the same object.
- Non Unique Name Assumption
  - If two objects have different names, they can also be the same objects.

A riddle

- “Two sons and two fathers went to pizza restaurant. They ordered three pizza. When they arrived, everyone had a whole pizza. How can that happen?
- In OWL those individuals are not assumed to be distinct – so this riddle is easily satisfied
- OWL has no Unique Name Assumption – unlike most humans

No unique name assumption:

- What does it mean? E.g. A Child has two parents, but Marry hasParents {John Doe, Lissa Doe, Lissa Minelly}??

Some facts:

family:Marry rdf:type family:person.
family:Marry ex:hasParent family:JohnDoe.
family:Marry ex:hasParent family:LissaDoe.
family:Marry ex:hasParent family:LissaMinelly.

Some cardinality axiom (in DL):

family:person $\sqsubseteq$ $\leq$2 hasParent.T

There is no unique name assumption, this means that JohnDoe, LissaDoe and LissaMinelly are not necessarily different objects.

Three possible options which can be "inferred" from the above example::

family:LissaDoe = family:LissaMinelly
family:JohnDoe = family:LissaMinelly
family:JohnDoe = family:LissaMinelly

Only the first is intuitive.
If we add facts and axioms stating that JohnDoe is male, LissaMinelly is female, LissaDoe is female and female are disjoint from male then only the first intuitive solution remains.

More Reasoning exercises

Reasoning

```
Class(pp:old+lady complete
    intersectionOf(pp:elderly pp:female pp:person))
Class(pp:old+lady partial
    intersectionOf(
        restriction(pp:has_pet allValuesFrom(pp:cat))
        restriction(pp:has_pet someValuesFrom(pp:animal))))
```

Every old lady must have a pet cat. (Because she must have some pet and all her pets must be cats.)

Reasoning

```
Class(pp:cow partial pp:vegetarian)
Class(pp:mad+cow complete
    intersectionOf(pp:cow
            restriction(pp:eats someValuesFrom(
                    intersectionOf(pp:brain
                    restriction(pp:part_of someValuesFrom pp:sheep))))))
```

There can be no mad cows (Because cows, as vegetarians, don't eat anything that is a part of an animal.)

Reasoning

```
ObjectProperty(pp:has_pet domain(pp:person) range(pp:animal))
Class(pp:old+lady complete
    intersectionOf(pp:elderly pp:female pp:person))
Class(pp:old+lady partial
    intersectionOf(
        restriction(pp:has_pet allValuesFrom(pp:cat))
        restriction(pp:has_pet someValuesFrom(pp:animal))))

Individual(pp:Minnie type(pp:elderly) type(pp:female)
value(pp:has_pet pp:Tom))
```

Minnie must be a person (because pet owners are human) and thus is an old lady. Thus Tom must be a cat (because all pets of old ladies are cats).

Reasoning

```
Class(pp:animal+lover complete
    intersectionOf(pp:person
          restriction(pp:has_pet minCardinality(3))))

Individual(pp:Walt type(pp:person)
value(pp:has_pet pp:Huey)
value(pp:has_pet pp:Louie)
value(pp:has_pet pp:Dewey))
DifferentIndividuals(pp:Huey pp:Louie pp:Dewey)
```

Walt must be an animal lover. Note that stating that Walt is a person is redundant.

Reasoning

```
Class(pp:van partial pp:vehicle)
Class(pp:driver partial pp:adult)
Class(pp:driver complete
      intersectionOf(restriction(pp:drives someValuesFrom(pp:vehicle)) pp:person))
Class(pp:white+van+man complete
      intersectionOf(pp:man restriction(pp:drives
                someValuesFrom(intersectionOf(pp:white+thing pp:van)))))
Class(pp:white+van+man partial
      restriction(pp:reads allValuesFrom pp:tabloid))

Individual(pp:Q123+ABC type(pp:white+thing) type(pp:van))
Individual(pp:Mick type(pp:male)
      value(pp:reads pp:Daily+Mirror)
      value(pp:drives pp:Q123+ABC))
```

Mick drives a white van, so he must be an adult (because all drivers are adults). As Mick is male, thus he is a white van man, so any paper he reads must be a tabloid, thus the Daily Mirror is a tabloid.

Use Case – OWL Reasoning

Acknowledgement from Talis Tutorial: Talis: http://research.talis.com/2005/rdf-intro

Simple Inferencing

- OWL enables simple inferencing
- Drawing conclusions based on the restrictions and descriptions

OWL Inferencing 1

- First of all a robbery takes place. The robber drops his gun while fleeing. A report is filed by the investigating officers:

```
<RobberyEvent>
  <date>14th June 2005</date>
  <description>Armed robbery at Kwik-e-Mart</description>
  <evidence>
    <Gun>
      <serial>983GTE-H5TF</serial>
    </Gun>
  </evidence>
  <robber>
    <Person /> <!-- an unknown person -->
  </robber>
</RobberyEvent>
```

OWL Inferencing 2

- Subsequently a car is pulled over for speeding. The traffic officer files a report

```
<SpeedingOffence>
  <date>26 October 2005</date>
  <description>Car observed driving at speed along
       M42</description>
  <speeder>
    <Person>
      <name>John Doe</name>
      <driversLicenseNumber>7431224667</driversLicenseNumber>
    </Person>
  </speeder>
</SpeedingOffence>
```

OWL Inferencing 3

- At police HQ, the computer analyses each report as it is filed. The following OWL description tells the computer that a driverLicenseNumber is unique to a Person:

```
<owl:InverseFunctionalProperty rdf:ID="driversLicenseNumber">
   <rdfs:domain rdf:resource="Person" />
   <rdfs:range rdf:resource="&rdf;Literal" />
 </owl:FunctionalProperty>
```

OWL Inferencing 4

- The computer uses this information to look up any other records it has about that person and finds a gun license:

```
<GunLicense>
  <registeredGun>
    <Gun>
      <serial>983GTE-H5TF</serial>
    </Gun>
  </registeredGun>
  <holder>
    <Person>
      <name>John Doe</name>
      <driversLicenseNumber>7431224667</driversLicenseNumber>
    </Person>
  </holder>
</GunLicense>
```

OWL Inferencing 5

- The next OWL description tells the computer that the registeredGun property is uniquely identifies a GunLicense. i.e. each gun is associated with only a single GunLicense

```
<owl:InverseFunctionalProperty rdf:ID="registeredGun">
  <rdfs:domain rdf:resource="GunLicense" />
   <rdfs:range rdf:resource="Gun" />
</owl:FunctionalProperty>
```

OWL Inferencing 6

- The computer now knows that the person stopped for speeding owns a gun. The next description tells the computer that each gun is uniquely identified by its serial

```
<owl:InverseFunctionalProperty rdf:ID="serial">
  <rdfs:domain rdf:resource="Gun" />
  <rdfs:range rdf:resource="&rdf;Literal" />
</owl:FunctionalProperty>
```

OWL Inferencing 7

- The computer uses this to determine that the gun on the license is the same gun used in the robbery. This final description, seals the speeder’s fate. It tells the computer that each GunLicense applies to only one gun and one person, so there is no doubt that the speeder is the person who owns the gun:

```
<owl:Class rdf:ID="GunLicense">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Restriction>
      <owl:onProperty rdf:resource="#registeredGun"/>
      <owl:cardinality>1</owl:cardinality>
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#holder"/>
      <owl:cardinality>1</owl:cardinality>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

OWL Inferencing 8

- The computer reports back to the traffic cop who dully arrests the speeder on suspicion of armed robbery

Summary

Summary

- Compared with RDF/RDFS, OWL provides a more formal language which allows expressing more axioms, but also adds some restrictions (not in OWL Full)
- We can do inferences such as classify instances, detect inconsistencies, etc.
- We can make use of research in reasoning engines! Strong formal background!
-

OWL Lite

- **RDF Schema Features:**
  - Class (Thing, Nothing)
  - rdfs:subClassOf
  - rdf:Property
  - rdfs:subPropertyOf
  - rdfs:domain
  - rdfs:range
  - Individual
- **(In)Equality:**
  - equivalentClass
  - equivalentProperty
  - sameAs
  - differentFrom
  - AllDifferent
  - distinctMembers
- **Property Characteristics:**
  - ObjectProperty
  - DatatypeProperty
  - inverseOf
  - TransitiveProperty
  - SymmetricProperty
  - FunctionalProperty
  - InverseFunctionalProperty
- **Property Restrictions:**
  - Restriction
  - onProperty
  - allValuesFrom
  - someValuesFrom
- **Restricted Cardinality:**
  - minCardinality (only 0 or 1)
  - maxCardinality (only 0 or 1)
  - cardinality (only 0 or 1)
- **Header Information:**
  - Ontology
  - imports
- **Class Intersection:**
  - intersectionOf
- **Versioning:**
  - versionInfo
  - priorVersion
  - backwardCompatibleWith
  - incompatibleWith
  - DeprecatedClass
  - DeprecatedProperty
- **Annotation Properties:**
  - rdfs:label
  - rdfs:comment
  - rdfs:seeAlso
  - rdfs:isDefinedBy
  - AnnotationProperty
  - OntologyProperty
- **Datatypes**
  - xsd datatypes

OWL DL and Full

- **Class Axioms:**
  - *oneOf, dataRange*
  - *disjointWith*
  - *equivalentClass* (applied to class expressions)
  - *rdfs:subClassOf* (applied to class expressions)
- **Boolean Combinations of Class Expressions:**
  - *unionOf*
  - *complementOf*
  - *intersectionOf*
- **Arbitrary Cardinality:**
  - *minCardinality*
  - *maxCardinality*
  - *cardinality*
- **Filler Information:**
  - *hasValue*

OWL on one Slide

- **Symmetric**: if P(x, y) then P(y, x)
- **Transitive**: if P(x,y) and P(y,z) then P(x, z)
- **Functional**: if P(x,y) and P(x,z) then y=z
- **InverseOf**: if P1(x,y) then P2(y,x)
- **InverseFunctional**: if P(y,x) and P(z,x) then y=z
- **allValuesFrom**: P(x,y) and y=allValuesFrom(C)
- **someValuesFrom**: P(x,y) and y=someValuesFrom(C)
- **hasValue**: P(x,y) and y=hasValue(v)
- **cardinality**: cardinality(P) = N
- **minCardinality**: minCardinality(P) = N
- **maxCardinality**: maxCardinality(P) = N
- **equivalentProperty**: P1 = P2
- **intersectionOf**: C = intersectionOf(C1, C2, …)
- **unionOf**: C = unionOf(C1, C2, …)
- **complementOf**: C = complementOf(C1)
- **oneOf**: C = one of(v1, v2, …)
- **equivalentClass**: C1 = C2
- **disjointWith**: C1 != C2
- **sameIndividualAs**: I1 = I2
- **differentFrom**: I1 != I2
- **AllDifferent**: I1 != I2, I1 != I3, I2 != I3, …
- **Thing**: I1, I2, …

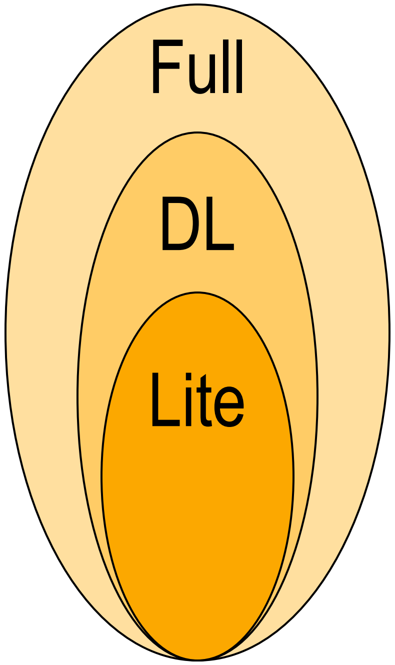
Legend:
Properties are indicated by: P, P1, P2, etc
Specific classes are indicated by: x, y, z
Generic classes are indicated by: C, C1, C2
Values are indicated by: v, v1, v2
Instance documents are indicated by: I1, I2, I3, etc.
A number is indicated by: N
P(x,y) is read as: “property P relates x to y”

Back to the OWL Layers (Lite, DL, Full)

- **OWL Lite**
  - (sub)classes, individuals
  - (sub)properties, domain, range
  - intersection

  } **RDF Schema**

  - (in)equality
  - cardinality 0/1
  - datatypes
  - inverse, transitive, symmetric
  - hasValue
  - someValuesFrom
  - allValuesFrom



- **OWL DL**
  - Negation (disjointWith, complementOf)
  - unionOf
  - arbitrary Cardinality
  - Enumerated types (oneOf)

- **OWL Full**
  - Allow meta-classes etc

* This slide taken from Frank van Harmelen

References

- W3C Documents
  - Guide
    http://www.w3.org/TR/owl-guide/
  - Reference
    http://www.w3.org/TR/owl-ref/
  - Semantics and Abstract Syntax
    http://www.w3.org/TR/owl-semantics/
- OWL Tutorials
  - Ian Horrocks, Sean Bechhofer:
    http://www.cs.man.ac.uk/~horrocks/Slides/Innsbruck-tutorial/
  - Talis: http://research.talis.com/2005/rdf-intro
  - Roger L. Costello, David B. Jacobs:
    http://www.xfront.com/owl/

W3C OWL References

- OWL Semantics and Abstract Syntax]
  - *OWL Web Ontology Language Semantics and Abstract Syntax* , Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks, Editors. W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-semantics-20040210/. Latest version available at http://www.w3.org/TR/owl-semantics/.
- [OWL Overview]
  - *OWL Web Ontology Language Overview* , Deborah L. McGuinness and Frank van Harmelen, Editors. W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-features-20040210/. Latest version available at http://www.w3.org/TR/owl-features/.
- [OWL Reference]
  - *OWL Web Ontology Language Reference* , Mike Dean and Guus Schreiber, Editors. W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-ref-20040210/. Latest version available at http://www.w3.org/TR/owl-ref/.
- [OWL Requirements]
  - *OWL Web Ontology Language Use Cases and Requirements* , Jeff Heflin, Editor. W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-webont-req-20040210/. Latest version available at http://www.w3.org/TR/webont-req/.
- [OWL Test Cases]
  - *OWL Web Ontology Language Test Cases* , Jeremy J. Carroll and Jos De Roo, Editors. W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-test-20040210/. Latest version available at http://www.w3.org/TR/owl-test/.

OWL Exercise