Semantic Web Schema and Ontologies

CS 431 – April 3, 2006

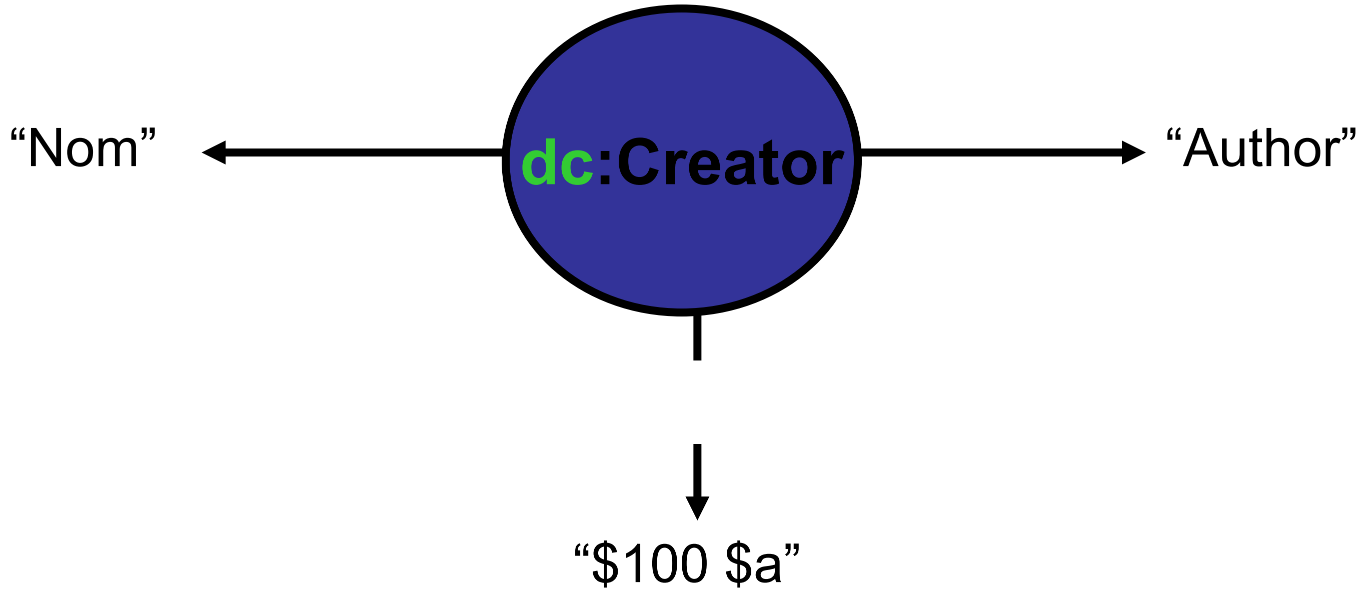
Carl Lagoze – Cornell University

Acknowledgements for various slides and ideas

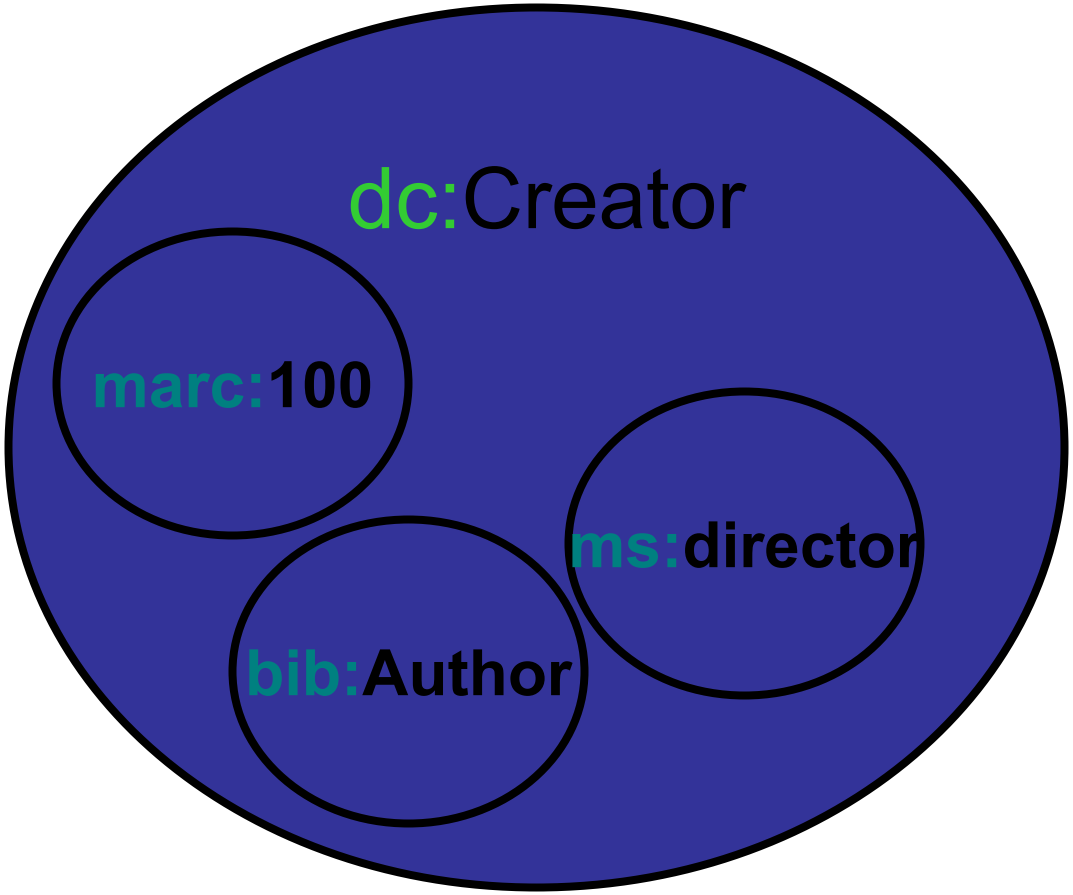
- Ian Horrocks (Manchester U.K.)
- Eric Miller (W3C)
- Dieter Fensel (Berlin)
- Volker Haarslev (Montreal)

Why Schema (1)?

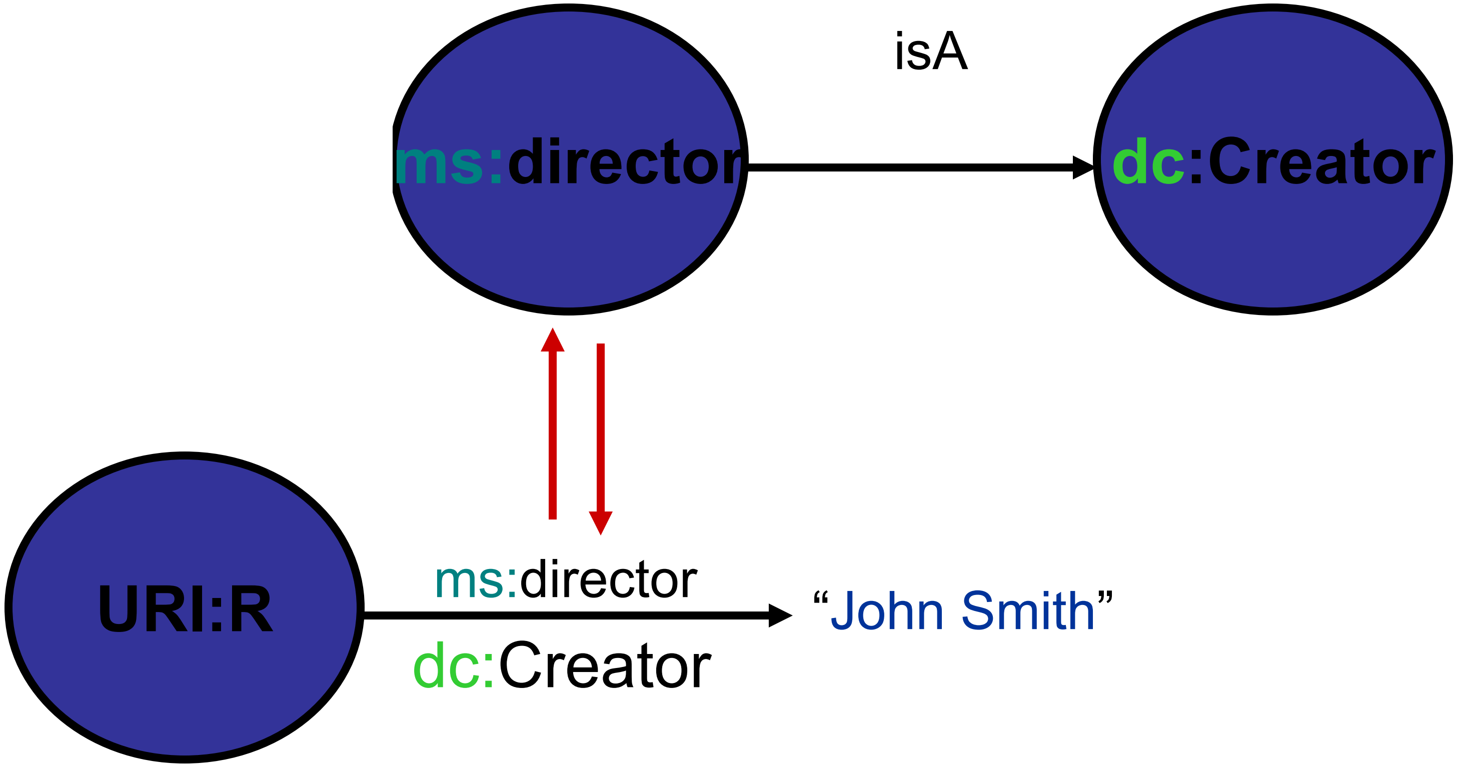
- Enables communities to share machine readable tokens and locally define human readable labels.



Why Schema (2)?
Relationships among vocabularies



Why Schema(3)? Relationships among vocabulary elements



Jena Toolkit

- Robust tools for building and manipulating RDF models
  - HP Labs Bristol
  - Capabilities
    - Model construction
    - XML and N3 parsing
    - Model persistence (DB foundation)
    - Model querying
    - Ontology building
    - Inferencing
- http://www.hpl.hp.com/semweb/jena2.htm

IsaViz

- Visualizing and constructing RDF models
- http://www.w3.org/2001/11/IsaViz/

RDQLPlus

- Simple RDF and OWL experimentation application
- http://rdqlplus.sourceforge.net/
- Chris Wilper - Cornell

RDF Schemas

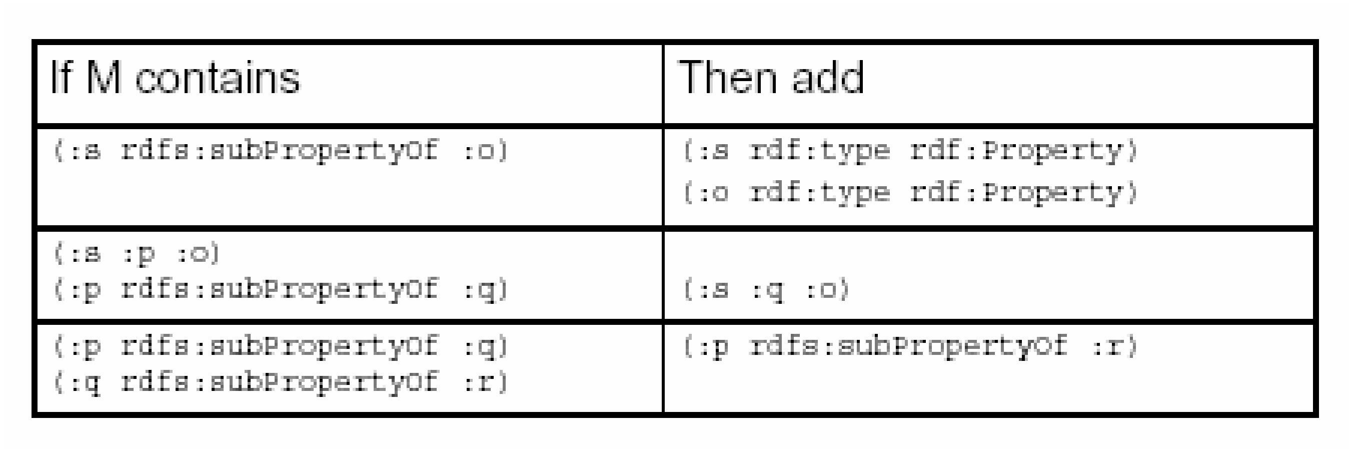
- Declaration of vocabularies
  - classes, properties, and structures defined by a particular community
  - relationship of properties to classes
- Provides substructure for inferences based on existing triples
- NOT prescriptive, but descriptive
- Schema language is an expression of basic RDF model
  - uses meta-model constructs
  - schema are “legal” rdf graphs and can be expressed in RDF/XML syntax

RDFs Namespace

- Class-related
  - rdfs:Class, rdfs:subClassOf
- Property-related
  - rdfs:subPropertyOf, rdfs:domain, rdfs:range

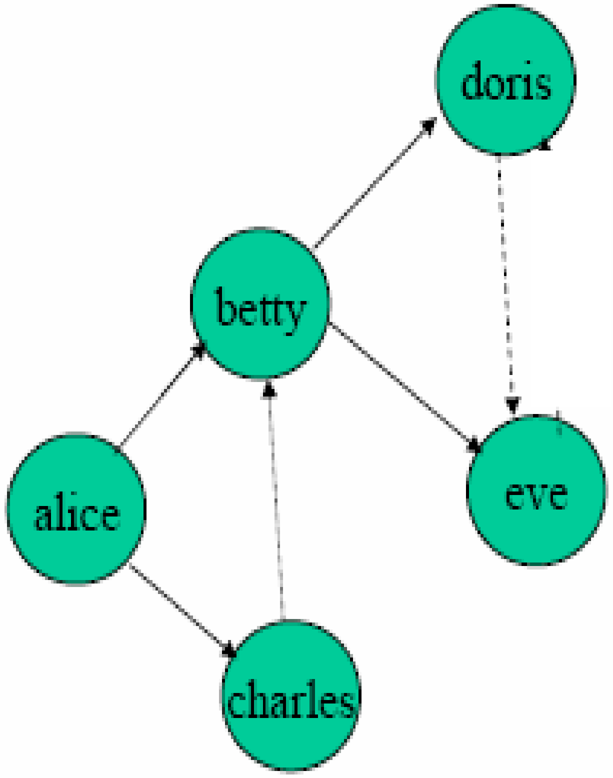
RDF Schema: Specializing Properties

- `rdfs:subPropertyOf` – allows specialization of relations
  - E.g., the property “father” is a subPropertyOf the property parent
- subProperty semantics



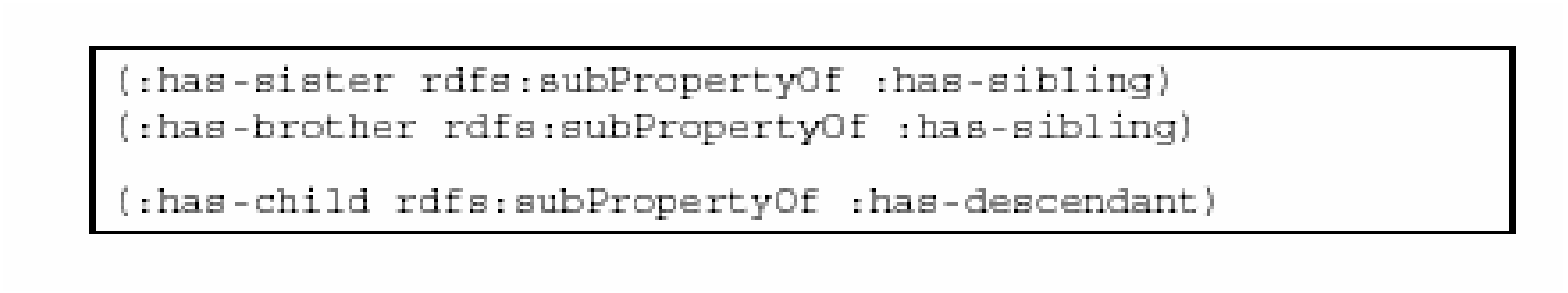
If M contains	Then add
`(:s rdfs:subPropertyOf :o)`	`(:s rdf:type rdf:Property)` `(:o rdf:type rdf:Property)`
`(:s :p :o)` `(:p rdfs:subPropertyOf :q)`	`(:s :q :o)`
`(:p rdfs:subPropertyOf :q)` `(:q rdfs:subPropertyOf :r)`	`(:p rdfs:subPropertyOf :r)`

Inferences from Property Relationships



```
(:alice  :has-child  :betty)
(:alice  :has-child  :charles)

(:betty  :has-child  :doris)
(:betty  :has-child  :eve)

(:charles  : has-sibling  :betty)

(:doris  :has-sister  :eve)
```

Sub-Property Semantics



```
(:has-sister rdfs:subPropertyOf :has-sibling)
(:has-brother rdfs:subPropertyOf :has-sibling)

(:has-child rdfs:subPropertyOf :has-descendant)
```

implies

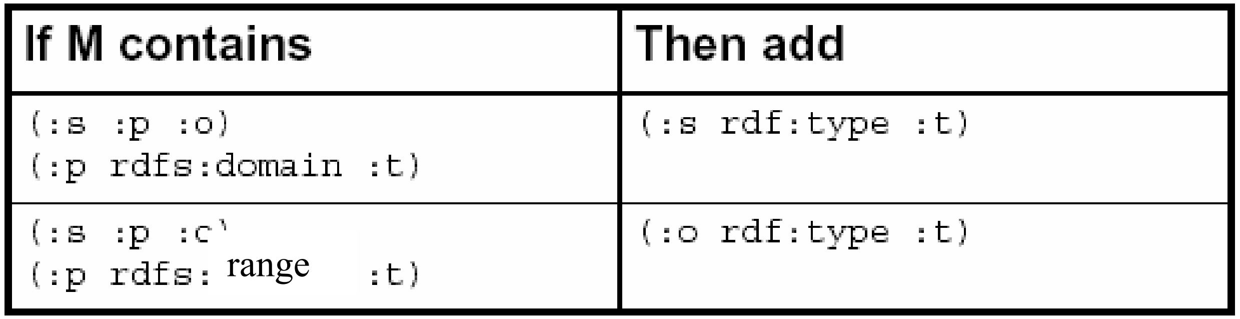
(:alice :has-descendent :betty)
(:betty :has-descendent :doris)
(:doris :has-sibling :eve)

??

(:alice :has-descendent :doris)
(:eve :has-sibling :doris)

Property-based semantics

- Provide basis for type inference from properties
- Not restrictive like xml schema constraints
- rdfs:domain
  - classes of resources that have a specific property
- rdfs:range
  - classes of resources that may be the value of a specific property



If M contains	Then add
(:s :p :o) (:p rdfs:domain :t)	(:s rdf:type :t)
(:s :p :c' (:p rdfs: range :t)	(:o rdf:type :t)

Inferences from Constraints

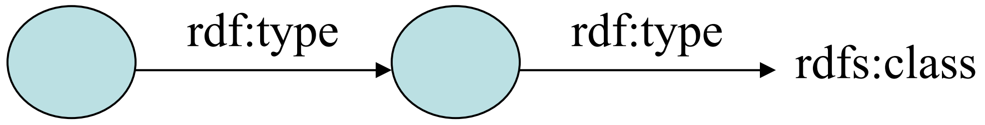
```
(:has-child rdfs:domain parent)
(:has-child rdfs:range person)

(:has-sibling rdfs:domain person)

(:has-brother rdfs:range :male-person)
(:has-sister rdfs:range :female-person)
```

- Using the intended semantics, we can infer:

```
(:alice rdf:type parent)
(:betty rdf:type parent)

(: :eve    rdf:type femal-person)

(:charles rdf:type :person)
```

Class Declaration

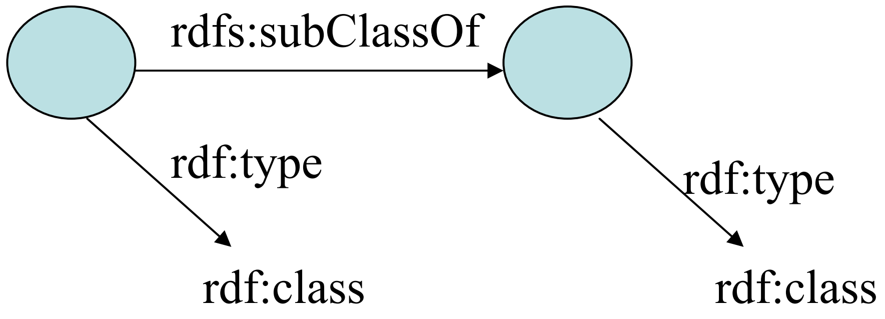
- `rdfs:Class`
  - Resources denoting a set of resources; range of `rdf:type`



ex:MotorVehicle rdf:type rdfs:Class
exthings:companyCar rdf:type ex:MotorVehicle

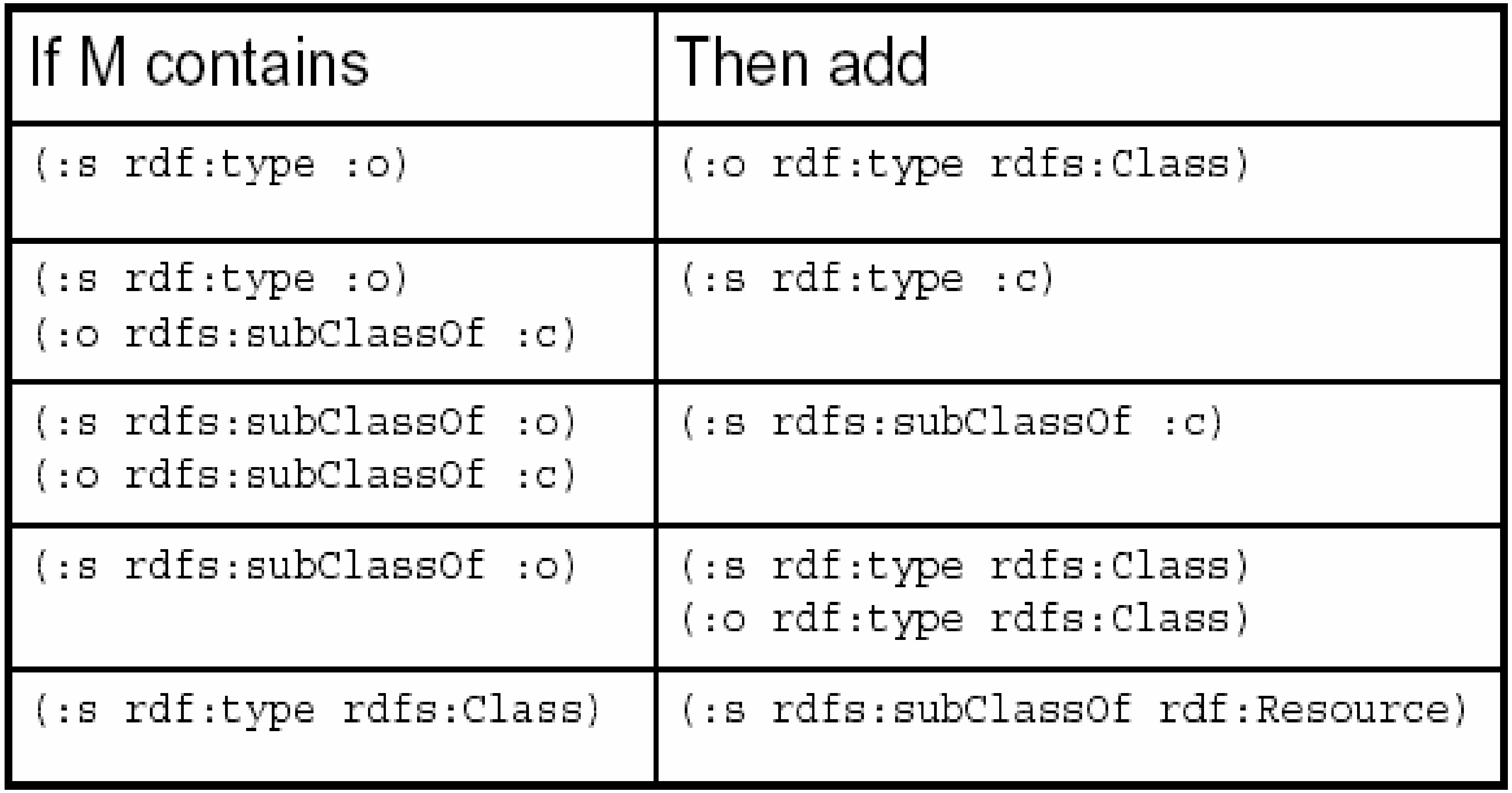
Class Hierarchy

- `rdfs:subClassOf`
  - Create class hierarchy



ex:MotorVehicle rdf:type rdfs:Class
ex:SUV rdf:type rdfs:Class
ex:SUV rdf:subClassOf ex:MotorVehicle
exthings:companyCar rdf:type ex:SUV

Sub-Class Inferencing



If M contains	Then add
`(:s rdf:type :o)`	`(:o rdf:type rdfs:Class)`

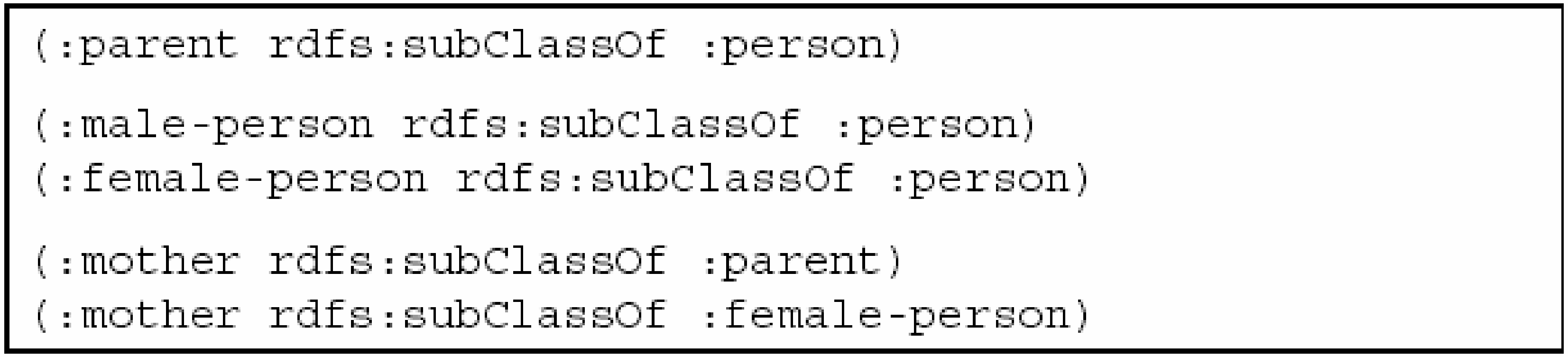
`(:s rdf:type :o)`
`(:o rdfs:subClassOf :c)`
	`(:s rdf:type :c)`

`(:s rdfs:subClassOf :o)`
`(:o rdfs:subClassOf :c)`
	`(:s rdfs:subClassOf :c)`

`(:s rdfs:subClassOf :o)`	`(:s rdf:type rdfs:Class)`
	`(:o rdf:type rdfs:Class)`

`(:s rdf:type rdfs:Class)`	`(:s rdfs:subClassOf rdf:Resource)`

Sub-class Inferencing Example

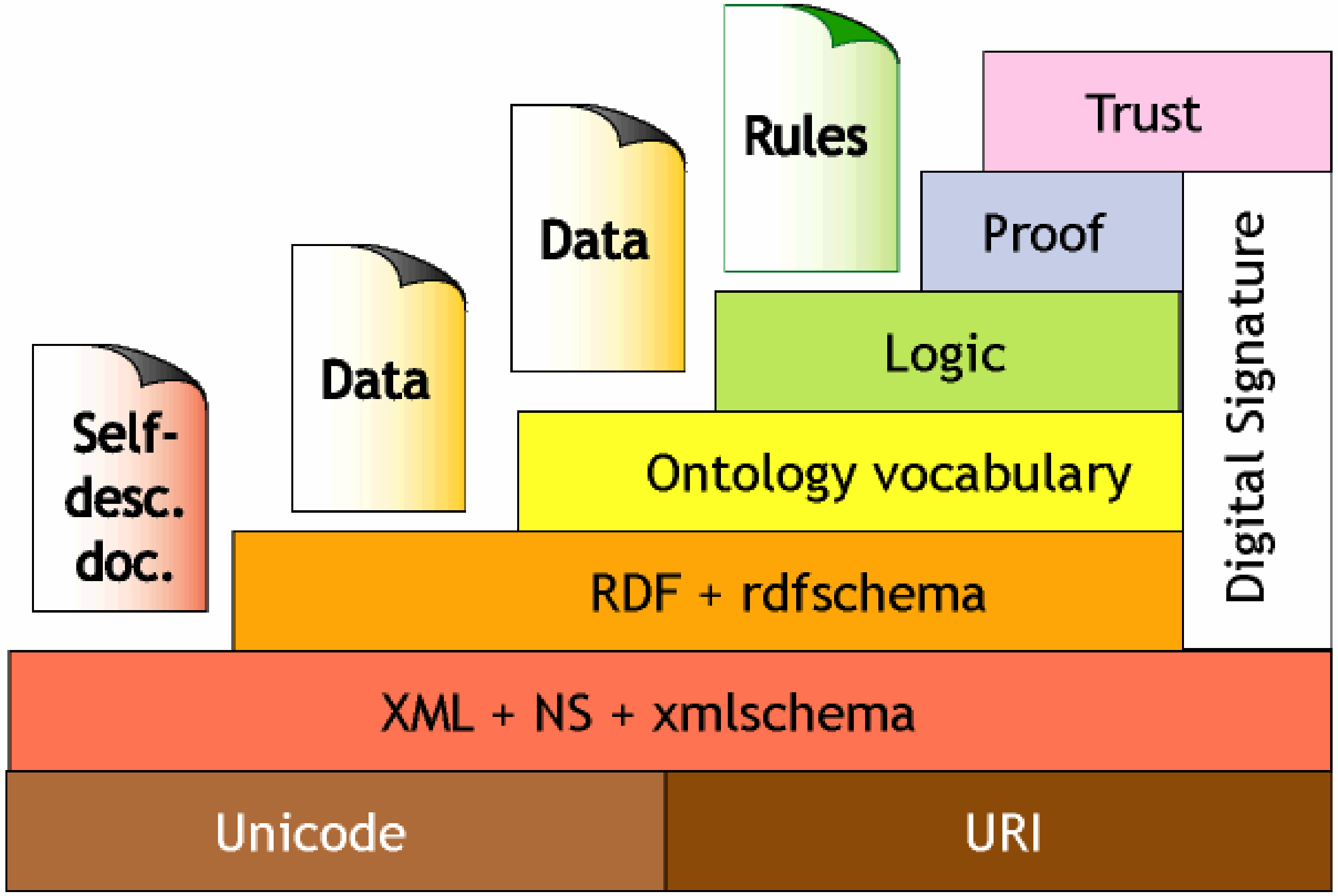


```
(:parent rdfs:subClassOf :person)

(:male-person rdfs:subClassOf :person)
(:female-person rdfs:subClassOf :person)

(:mother rdfs:subClassOf :parent)
(:mother rdfs:subClassOf :female-person)
```

implies

(:alice rdf:type :person)
(:betty rdf:type :person)
(:doris rdf:type :person)

Components of the Semantic Web



Problems with RDF/RDFs Non-standard, overly “liberal” semantics

- No distinction between class and instances
  - <Species, type, Class>
  - <Lion, type, Species>
  - <Leo, type, Lion>
- Properties themselves can have properties
  - <hasDaughter, subPropertyOf, hasChild>
  - <hasDaugnter, type, Property>
- No distinction between language constructors and ontology vocabulay, so constructors can be applied to themselves/each other
  - <type, range, Class>
  - <Property, type, Class>
  - <type, subPropertyOf, subClassOf>
- No known reasoners for these non-standard semantics

Problems with RDF/RDFs Weaknesses in expressivity

- No localized domain and range constraints
  - Can’t say the range of hasChild is person in context of persons and elephants in context of elephants
- No existence/cardinality constraints
  - Can’t say that all instances of persons have a mother that is also a person
  - Can’t say that persons have exactly two biological parents
- No transitive, inverse or symmetric properties
  - Can’t say isPartOf is a transitive property
  - Can’t say isPartOf is inverse of hasPart
  - Can’t say touches is symmetric

So, we need a more expressive and well-grounded ontology language….

What is an *Ontology*?

- A formal specification of conceptualization shared in a community
- Vocabulary for defining a set of things that exist in a world view
- Formalization allows communication across application systems and extension
- Parallel concepts in other areas:
  - *Domains:* database theory
  - *Types*: AI
  - *Classes*: OO systems
  - *Types/Sorts*: Logic
- Global vs. Domain-specific

XML and RDF are *ontologically neutral*

- No standard vocabulary just primitives
  - Resource, Class, Property, Statement, etc.
- Compare to classic first order logic
  - Conjunction, disjunction, implication, existential, universal quantifier

Components of an Ontology

- Vocabulary (concepts)
- Structure (attributes of concepts and hierarchy)
- Relationships between concepts
- Logical characteristics of relationships
  - Domain and range restrictions
  - Properties of relations (symmetry, transitivity)
  - Cardinality of relations
  - etc.

Wordnet

- On-line lexical reference system, domain-independent
- >100,000 word meanings organized in a taxonomy with semantic relationships
  - Synonymy, meronymy, hyponymy, hypernymy
- Useful for text retrieval, etc.
- http://www.cogsci.princeton.edu/~wn/online/

CYC

- Effort in AI community to accommodate all of human knowledge!!!
- Formalizes concepts with logical axioms specifying constraints on objects and classes
- Associated reasoning tools
- Contents are proprietary but there is OpenCyc
  - http://www.opencyc.org/

So why re-invent ontologies for the Web

- Not re-invention
  - Same underlying formalisms (frames, slots, description logic)
- But new factors
  - Massive scale
    - Tractability
    - Knowledge expressiveness must be limited or reasoning must be incomplete
  - Lack of central control
    - Need for federation
    - Inconsistency, lies, re-interpretations, duplications
    - New facts appear and modify constantly
  - Open world vs. Close world assumptions
    - Contrast to most reasoning systems that assume anything absent from knowledge base is not true
    - Need to maintain monotonicity with tolerance for contradictions
  - Need to build on existing standards
    - URI, XML, RDF

Web Ontology Language (OWL)

- W3C Web Ontology Working Group (WebOnt)
- Follow on to DAML, OIL efforts
- W3C Recommendation
- Vocabulary extension of RDF

Species of OWL

- *OWL Lite*
  - Good for classification hierarchies with simple constraints (e.g., thesauri)
  - Reasoning is computational simple and efficient
- *OWL DL*
  - Computationally complete and decidable (computation in finite time)
  - Correspondence to *description logics* (decidable fragment of first-order logic)
- *OWL Full*
  - Maximum expressiveness
  - No computational guarantees (probably never will be)

- Each language is extension of simpler predecessor

Description Logics

- Fragment of first-order logic designed for logical representation of object-oriented formalisms
  - frames/classes/concepts
    - sets of objects
  - roles/properties
    - binary relations on objects
  - individuals
- Representation as a collection of statements, with unary and binary predicates that stand for concepts and roles, from which deductions can be made
- High expressivity with decidability and completeness
  - Decidable fragment of FOL

Description Logics Primitives

- Atomic Concept
  - Human
- Atomic Role
  - likes
- Conjunction
  - human *intersection* male
- Disjunction
  - nice *union* rich
- Negation
  - *not* rich
- Existential Restriction
  - *exists* has-child.Human
- Value Restriction
  - *for-all* has-child.Blond
- Number Restriction
  - ≥ 2 has-wheels
- Inverse Role
  - has-child, has-parent
- Transitive role
  - has-child

Description Logic - Tboxes

- Terminological knowledge
- Concept Definitions
  - Father is conjunction of Man and has-child.Human
- Axioms
  - motorcycle *subset-of* vehicle
  - has-favorite.Brewery *subrelation-of* drinks.Beer

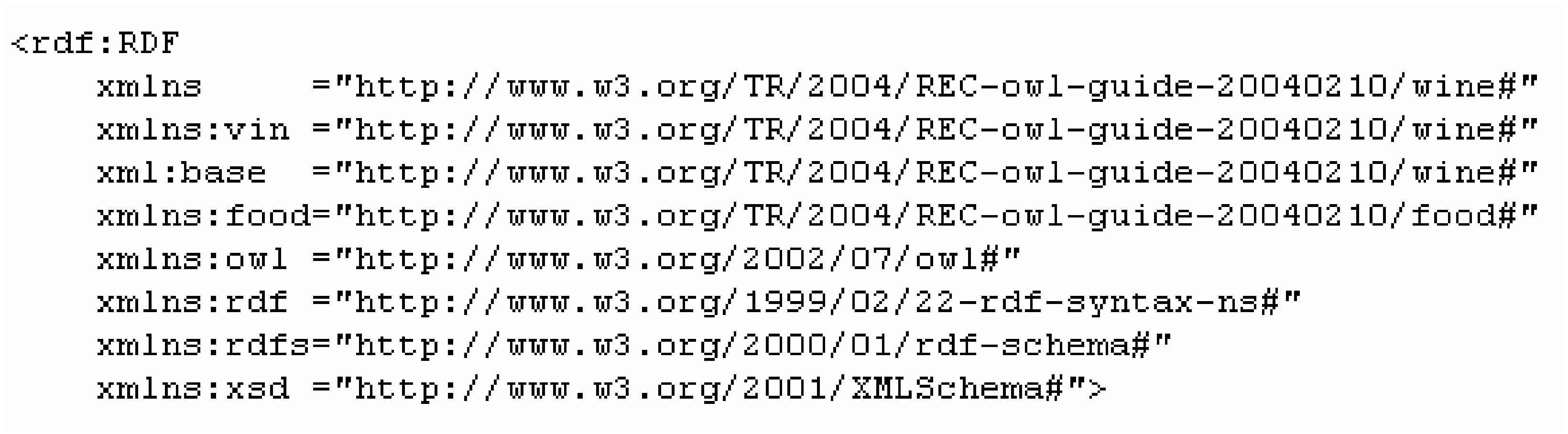
Description Logics: Aboxes

- Assertional knowledge
- Concept assertions
  - John is-a Man
- Role assertions
  - has-child(John, Bill)

Description Logics: Basic Inferencing

- Subsumption
  - Is C1 subclass-of C2
  - Compute taxonomy
- Consistency
  - Can C have any individuals

Namespaces and OWL



```
<rdf:RDF
    xmlns     ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
    xmlns:vin ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
    xml:base  ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
    xmlns:food="http://www.w3.org/TR/2004/REC-owl-guide-20040210/food#"
    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 Class Definition

```
<owl:Class rdf:ID="Winery"/>
<owl:Class rdf:ID="Region"/>
<owl:Class rdf:ID="ConsumableThing"/>
```

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
  <rdfs:label xml:lang="en">wine</rdfs:label>
  <rdfs:label xml:lang="fr">vin</rdfs:label>
  ...
</owl:Class>
```

Why owl:class vs. rdfs:class

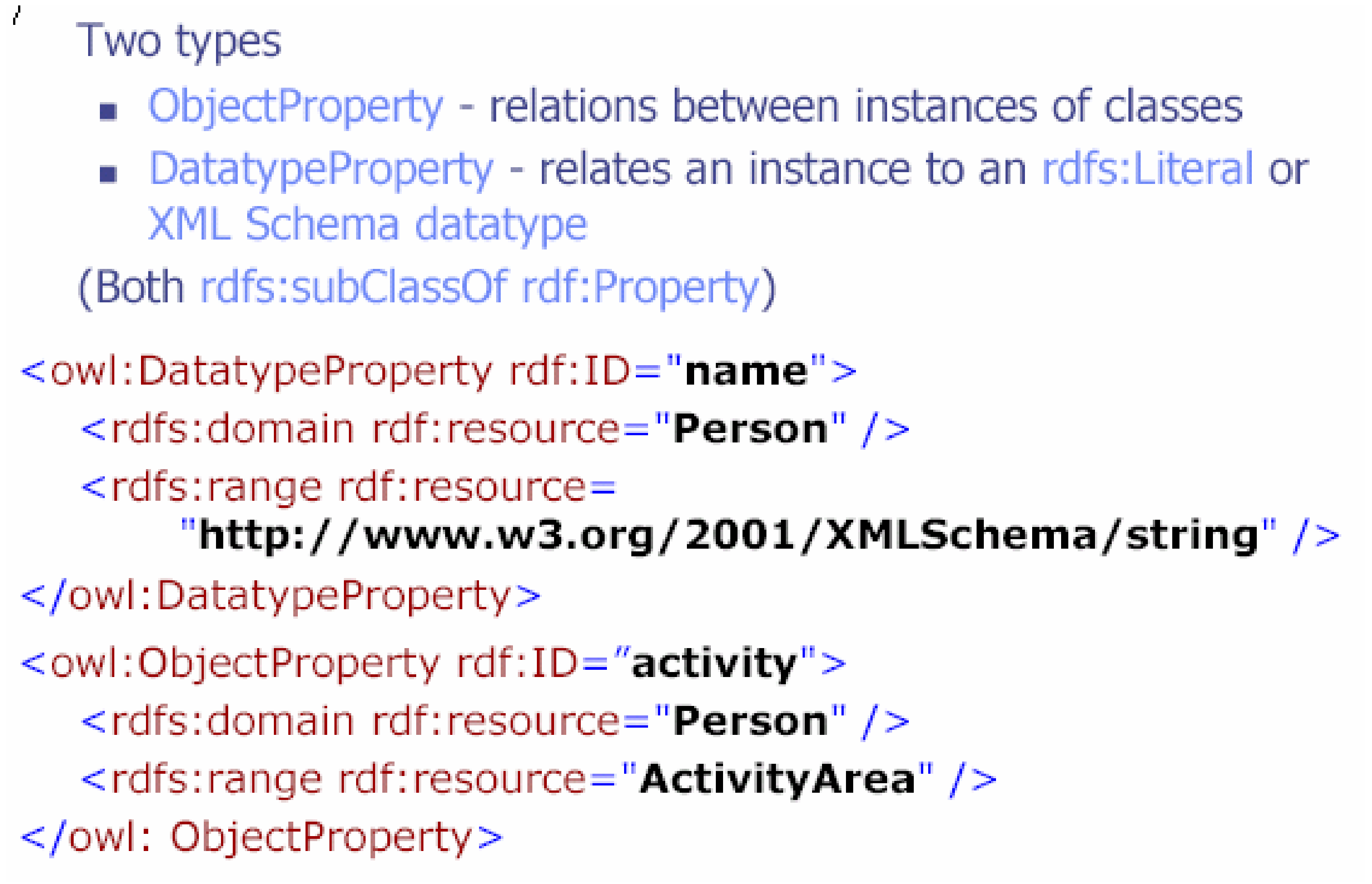
- Rdfs:class is “class of all classes”
- In DL class can not be treated as individuals (undecidable)
- Thus owl:class, which is expressed as rdfs:subclass of rdfs:class
  - No problem for standard rdf processors since an owl:class “is a” rdfs:class

- Note: there are other times you want to treat class of individuals
  - Class drinkable liquids has instances wine, beer, ….
  - Class wine has instances merlot, chardonnay, zinfandel, …

OWL class building operations

- disjointWith
  - No vegetarians are carnivores
- sameClassAs (equivalence)
- Enumerations (on instances)
  - The Ivy League is Cornell, Harvard, Yale, ….
- Boolean set semantics (on classes)
  - Union (logical disjunction)
    - Class *parent* is union of *mother, father*
  - Intersection (logical conjunction of class with properties)
    - Class *WhiteWine* is conjunction of things of class *wine* and have property *white*
  - complimentOf (logical negation)
    - Class *vegetarian* is disjunct of class *carnivore*

OWL Properties



Two types

- ObjectProperty - relations between instances of classes
- DatatypeProperty - relates an instance to an rdfs:Literal or XML Schema datatype

(Both rdfs:subClassOf rdf:Property)

```
<owl:DatatypeProperty rdf:ID="name">
   <rdfs:domain rdf:resource="Person" />
   <rdfs:range rdf:resource=
        "http://www.w3.org/2001/XMLSchema/string" />
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="activity">
   <rdfs:domain rdf:resource="Person" />
   <rdfs:range rdf:resource="ActivityArea" />
</owl: ObjectProperty>
```

OWL property building operations & restrictions

- Transitive Property
  - P(x,y) and P(y,z) -> P(x,z)
- SymmetricProperty
  - P(x,y) iff P(y,x)
- Functional Property
  - P(x,y) and P(x,z) -> y=z
- inverseOf
  - P1(x,y) iff P2(y,x)
- InverseFunctional Property
  - P(y,x) and P(z,x) -> y=z
- Cardinality
  - Only 0 or 1 in lite and full

OWL DataTypes

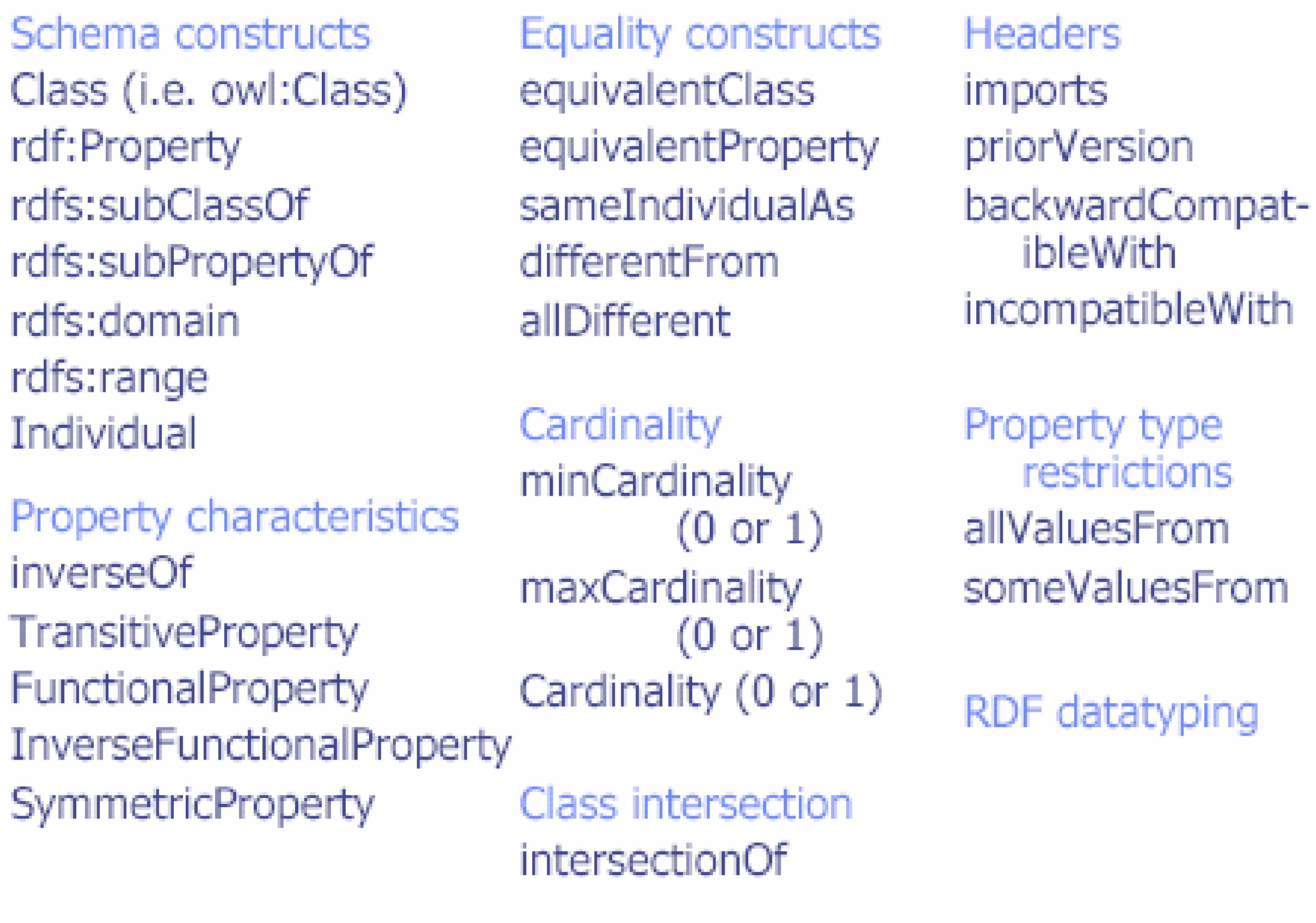
- Full use of XML schema data type definitions
- Examples
  - Define a type age that must be a non-negative integer
  - Define a type clothing size that is an enumeration “small” “medium” “large”

OWL Instance Creation

- Create individual objects filling in slot/attribute/property definitions

```
<Person ref:ID=“William Arms”>
        <rdfs:label>Bill</rdfs:label>
        <age><xsd:integer rdf:value=“57”/></age>
        <shoesize><xsd:decimal rdf:value=“10.5”/></shoesize>
</Person>
```

OWL Lite Summary



Schema constructs
Class (i.e. owl:Class)
rdf:Property
rdfs:subClassOf
rdfs:subPropertyOf
rdfs:domain
rdfs:range
Individual

Property characteristics
inverseOf
TransitiveProperty
FunctionalProperty
InverseFunctionalProperty
SymmetricProperty

Equality constructs
equivalentClass
equivalentProperty
sameIndividualAs
differentFrom
allDifferent

Cardinality
minCardinality (0 or 1)
maxCardinality (0 or 1)
Cardinality (0 or 1)

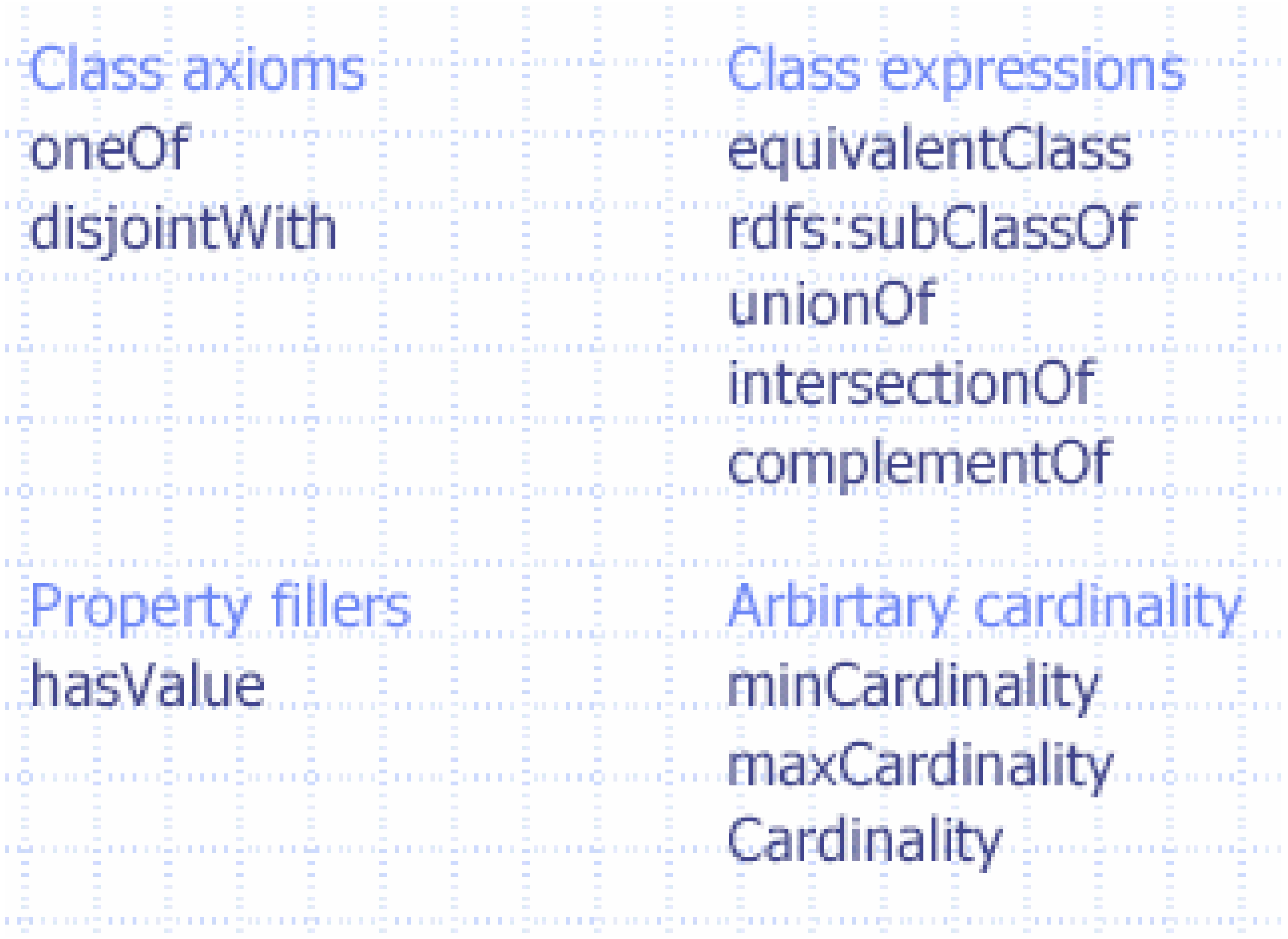
Class intersection
intersectionOf

Headers
imports
priorVersion
backwardCompat-ibleWith
incompatibleWith

Property type restrictions
allValuesFrom
someValuesFrom

RDF datatyping

OWL DL and Full Summary



Class axioms	Class expressions
oneOf	equivalentClass
disjointWith	rdfs:subClassOf
	unionOf
	intersectionOf
	complementOf
Property fillers	Arbirtary cardinality
hasValue	minCardinality
	maxCardinality
	Cardinality

OWL DL vs. OWL-Full

- Same vocabulary
- OWL DL restrictions
  - Type separation
    - Class can not also be an individual or property
    - Property can not also be an individual or class
  - Separation of ObjectProperties and DatatypeProperties

Language Comparison

	DTD	XSD	RDF(S)	OWL
Bounded lists ("X is known to have exactly 5 children")				X
Cardinality constraints (Kleene operators)	X	X		X
Class expressions (unionOf, complementOf)				X
Data types		X		X
Enumerations	X	X		X
Equivalence (properties, classes, instances)				X
Formal semantics (model-theoretic & axiomatic)				X
Inheritance			X	X
Inference (transitivity, inverse)				X
Qualified contraints ("all children are of type person"				X
Reification			X	X

Protégé and RACER – tools for building, manipulating and reasoning over ontologies

- Protégé - http://protege.stanford.edu/
  - Use the 3.x version
  - Multiple plug-ins are available
- Protégé OWL plug-in
  - http://protege.stanford.edu/plugins/owl/
- Other semantic web related plug-ins
  - http://protege.cim3.net/cgi-bin/wiki.pl?ProtegePluginsLibraryByTopic#nid349
- Racer
  - Description Logic based reasoning engine
  - Server-based
  - Integrates with Protégé-OWL