Web Ontology Language

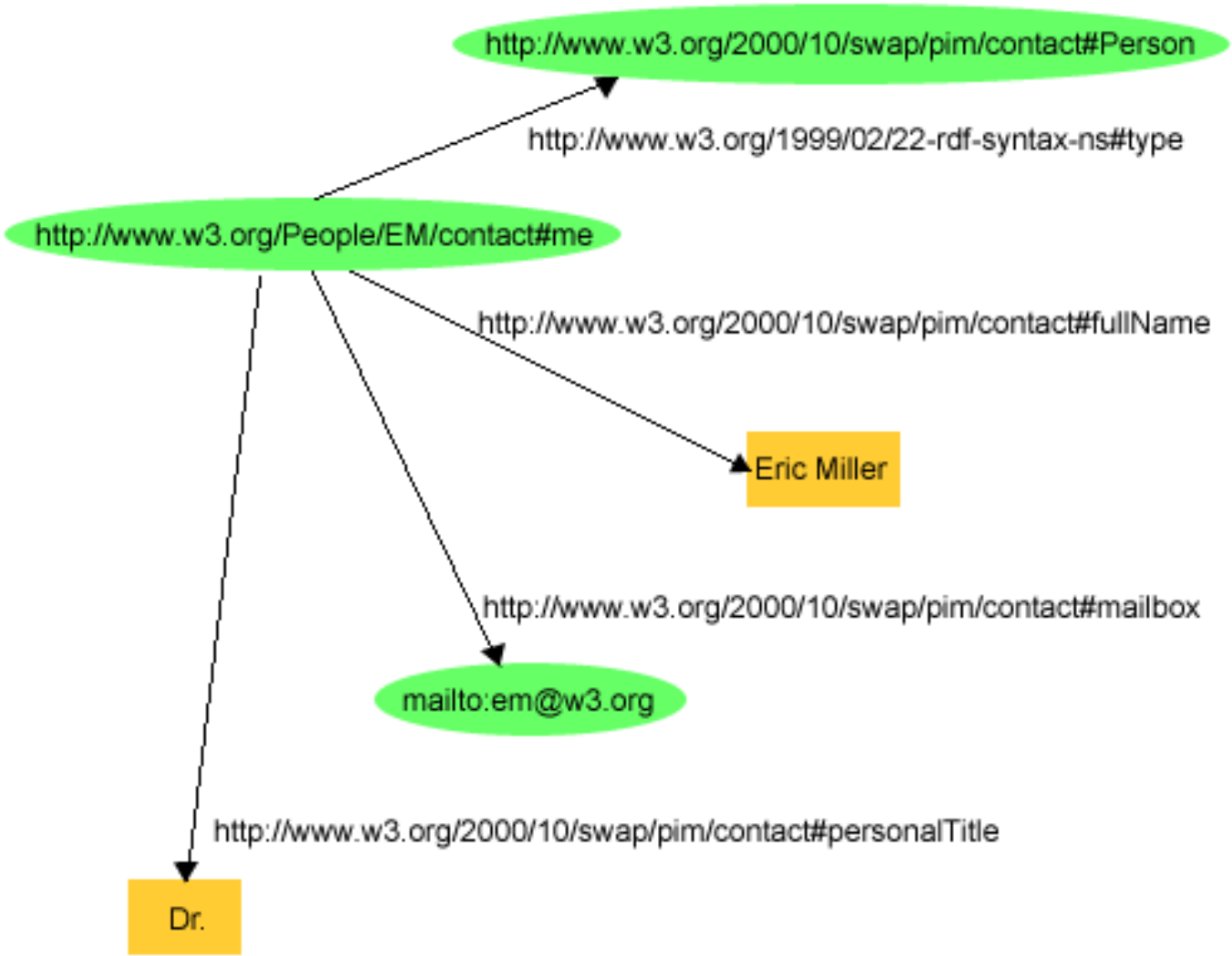
Pınar Yolum
Email: p.yolum@uu.nl

Department of Information and Computing Sciences
Utrecht University

RDF

- Describe things that have properties which have values
- Triples of <subject, predicate, object>
- Ex: <Eric Miller, hasTitle, Dr.>
- Need to uniquely identify each so that they can be processed without confusion
- RDF Graph: Show the triples with edges between two nodes

RDF graph


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/People/EM/contact#me
http://www.w3.org/2000/10/swap/pim/contact#fullName
Eric Miller
http://www.w3.org/2000/10/swap/pim/contact#mailbox
mailto:em@w3.org
http://www.w3.org/2000/10/swap/pim/contact#personalTitle
Dr.

RDF syntax

```
<?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>
```

RDF miscellaneous

- Container: Subject and object need not be a single value
  - Bag: No order, allow duplicates
  - Seq: Order important, allow duplicates
  - Alt: Alternatives
- Collection: Enumerations where the entire list is given (e.g., a collection of three students to define class)
- Reification: RDF statements about other RDF statements

Reification example

```
<rdf:Description rdf:ID="item10245">
  <exterms:weight rdf:datatype="&xsd;decimal">2.4</exterms:weight>
</rdf:Description>
<rdf:Statement rdf:about="#triple12345">
  <rdf:subject rdf:resource =
      "http://www.example.com/2002/04/products#item10245"/>
  <rdf:predicate rdf:resource=
      "http://www.example.com/terms/weight"/>
  <rdf:object rdf:datatype="&xsd;decimal">2.4</rdf:object>
  <dc:creator rdf:resource=
      "http://www.example.com/staffid/85740"/>
</rdf:Statement>
```

Staff 85740 stated that item 10245 weights 2.4

Linked Data

- Dbpedia: `http://dbpedia.org/page/The_Lord_of_the_Rings`
- Snomed CT: `http://bioportal.bioontology.org/ontologies/SNOMEDCT?p=classes&conceptid=root`
- Linked Open Data Cloud: `https://lod-cloud.net/`

RDF schema (RDFS)

- RDF represents data with graphs
- RDFS represents guidelines to use the data
  - Used to describe the vocabulary for resources
  - Main representation is sets; RDF Schema provides the types that will be used in RDF statements
- RDF talks about objects; RDF schema defines classes for objects

RDF schema classes

- Classes have a property *rdf:type* and a value *rdfs:Class*
  - ex:Person rdf:type rdfs:Class
  - ex:Book rdf:type rdfs:Class
- Resources can be defined based on new classes
  - ex:Player rdf:type ex:Person
  - ex:Instructor rdf:type ex:Person
- Classes can be specialized using *rdfs:subClassOf* property
  - ex:Footballer rdfs:subClassOf ex:Player

RDF schema properties (1)

- Classes have properties denoted by *rdf:property*
- Properties can be specialized using *rdfs:subPropertyOf*
- Example:
  - ex:driver rdf:type rdf:Property
  - ex:primaryDriver rdf:type rdf:Property
  - ex:primaryDriver rdfs:subPropertyOf ex:driver

RDF schema properties (2)

- *rdfs:domain* denotes the values for which a property applies.
  - ex:Book rdf:type rdfs:Class
  - ex:author rdf:type rdf:Property
  - ex:author rdfs:domain ex:Book
- *rdfs:range* denotes the range of values for a property
  - ex:author rdfs:range ex:Person
- A property can have 0 or more domain and range defined.

RDFS example

- MarriedWoman rdfs:subClassOf Woman.
- hasMaidenName rdfs:domain MarriedWoman.
- hasMaidenName rdfs:range Name.
- “Karen” hasMaidenName ”Stephens”
- What can we infer?

RDFS example

- MarriedWoman rdfs:subClassOf Woman.
- hasMaidenName rdfs:domain MarriedWoman.
- hasMaidenName rdfs:range Name.
- “Karen” hasMaidenName ”Stephens”
- What can we infer?
  - “Karen” rdf:type MarriedWoman
  - “Karen” rdf:type Woman
  - “Stephens” rdf:type Name

RDFS example

- MarriedWoman rdfs:subClassOf Woman.
- hasMaidenName rdfs:domain MarriedWoman.
- hasMaidenName rdfs:range Name.
- “Karen” hasMaidenName ”Stephens”
- What can we infer?
  - “Karen” rdf:type MarriedWoman
  - “Karen” rdf:type Woman
  - “Stephens” rdf:type Name
- Express above in DL.
  - MarriedWoman $\sqsubseteq$ Woman
  - $\exists$ *hasMaidenName*.*T* $\sqsubseteq$ MarriedWoman
  - $\top \sqsubseteq \forall$ hasMaidenName.Name

RDF(S) vs. Java

- Building blocks in JAVA are classes
  - JAVA: Class book has an attribute author of type person
  - RDF: There is an author property between a book and a person
  - JAVA: You cannot talk about an author attribute without a class
  - RDF: You can if you don't specify a domain
- Scope of definition
  - JAVA: If you are talking about a newspaper, you need to define a new author attribute (Local scope)
  - RDF: Define an author property once. (Global scope)
- Matching of domains and ranges based on types
  - JAVA: Class sportsarcticle has an attribute author of type male
  - JAVA: Class newsarticle has an attribute author of type female
  - RDF: Cannot match different domains with ranges

OWL Union, Intersection, Complement

```
<owl:Class rdf:ID="food#Fruit">
  <owl:unionOf rdf:parseType=Collection>
  <owl:Class rdf:resource="food#SweetFruit"/>
  <owl:Class rdf:resource="food#NonSweetFruit"/>
  </owl:unionOf>
</owl:Class>
<owl:Class rdf:ID="food#SugaryBread">
  <owl:intersectionOf rdf:parseType=Collection>
  <owl:Class rdf:about="food#Bread"/>
  <owl:Class rdf:about="food#SweetFruit"/>
  </owl:intersectionOf>
</owl:Class>
<owl:Class>
<owl:complementOf>
  <owl:Class rdf:about="food##Meat"/>
  </owl:complementOf>
</owl:Class>
```

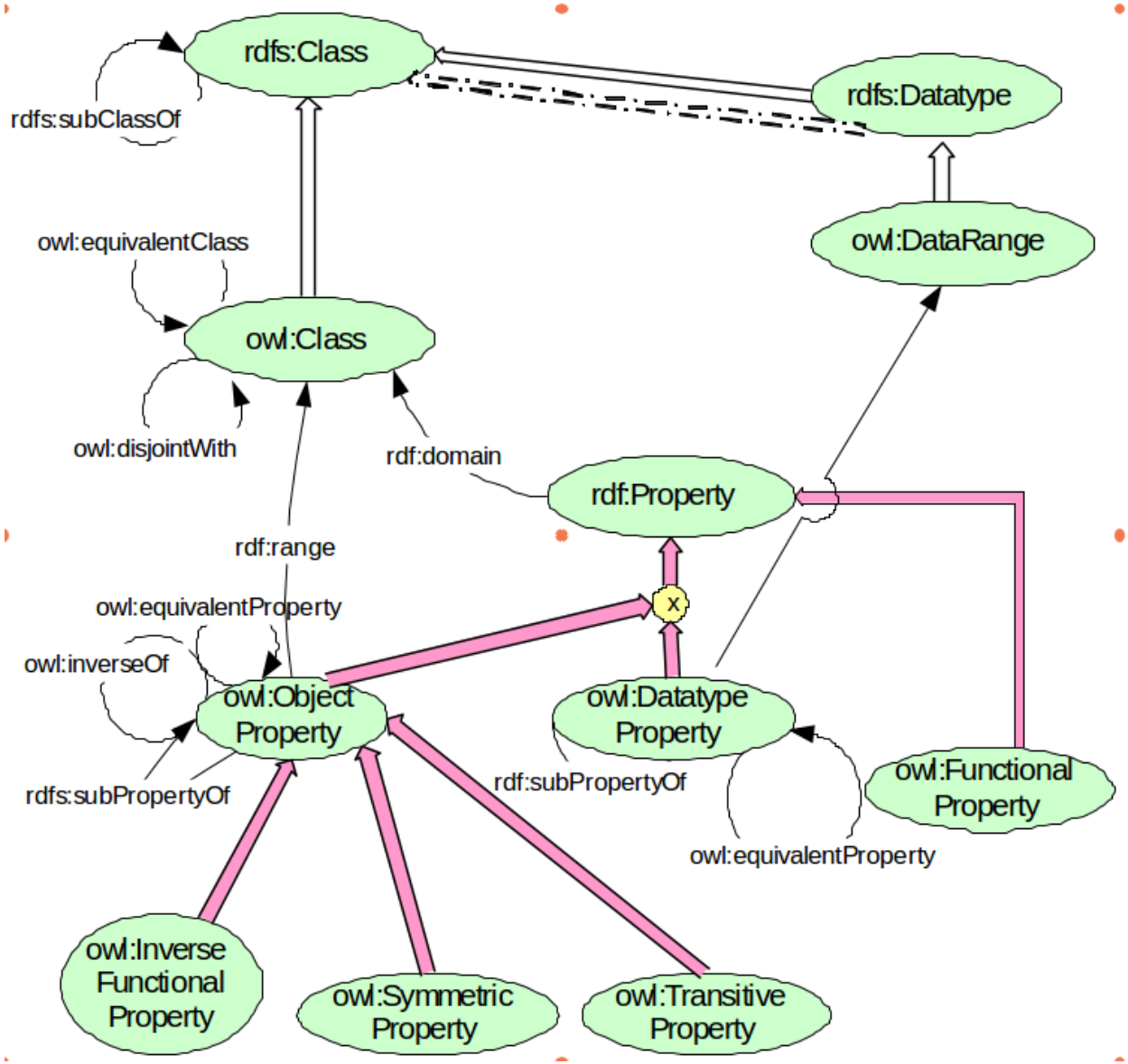
OWL equivalance and disjointness

```
<owl:Class rdf:about="food#Fruit">
 <rdfs:subClassOf rdf:resource="food#EdibleThing"/>
 <owl:disjointWith rdf:resource="food#Fowl"/>
 <owl:disjointWith rdf:resource="food#Dessert"/>
 <owl:disjointWith rdf:resource="food#Meat"/>
 <owl:disjointWith rdf:resource="food#OtherTomatoBasedFood"/>
 <owl:disjointWith rdf:resource="food#Pasta"/>
 <owl:disjointWith rdf:resource="food#Seafood"/>
 <owl:equivalentClass>
   <owl:Class>
    <owl:unionOf rdf:parseType="Collection">
    <rdf:Description rdf:about="food#NonSweetFruit"/>
    <rdf:Description rdf:about="food#SweetFruit"/>
    </owl:unionOf>
   </owl:Class>
 </owl:equivalentClass>
</owl:Class>
```

Enumeration

```
<owl:Class rdf:ID="Continent">
    <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Europe"/>
        <owl:Thing rdf:about="#Africa"/>
        <owl:Thing rdf:about="#America"/>
        <owl:Thing rdf:about="#Asia"/>
        <owl:Thing rdf:about="#Australia"/>
    </owl:oneOf>
</owl:Class>
```

OWL properties



Transitive property

P(x,y) and P(y,z) implies P(x, z)

```
<owl:ObjectProperty rdf:ID="locatedIn">
  <rdf:type rdf:resource="owl#TransitiveProperty"/>
  <rdfs:domain rdf:resource="owl#Thing"/>
  <rdfs:range rdf:resource="#Region"/>
</owl:ObjectProperty>
<Region rdf:ID="SantaCruzMountainsRegion">
   <locatedIn rdf:resource="#CaliforniaRegion"/>
</Region>
<Region rdf:ID="CaliforniaRegion">
   <locatedIn rdf:resource="#USRegion"/>
</Region>
```

Symmetric property

P(x,y) iff P(y,x)

```
<owl:ObjectProperty rdf:ID="adjacentRegion">
 <rdf:type rdf:resource="owl;SymmetricProperty"/>
 <rdfs:domain rdf:resource="#Region"/>
 <rdfs:range rdf:resource="#Region"/>
</owl:ObjectProperty>
<Region rdf:ID="MendocinoRegion">
 <locatedIn rdf:resource="#CaliforniaRegion"/>
 <adjacentRegion rdf:resource="#SonomaRegion"/>
</Region>
```

Inverse property

P1(x,y) iff P2(y,x)

```
<owl:ObjectProperty rdf:ID="hasMaker">
    <rdf:type rdf:resource="owl#FunctionalProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="producesWine">
    <owl:inverseOf rdf:resource="#hasMaker"/>
</owl:ObjectProperty>
```

Functional property

P(x, y) and P(x,z) implies y=z

```
<owl:Class rdf:ID="VintageYear"/>
<owl:ObjectProperty rdf:ID="hasVintageYear">
  <rdf:type rdf:resource="owl#FunctionalProperty"/>
  <rdfs:domain rdf:resource="#Vintage"/>
  <rdfs:range rdf:resource="#VintageYear"/>
</owl:ObjectProperty>
```

Inverse functional property

P(y,x) and P(z,x) implies y=z

```
<owl:ObjectProperty rdf:ID="hasMaker"/>
<owl:ObjectProperty rdf:ID="producesWine">
  <rdf:type rdf:resource="owl#InverseFunctionalProperty"/>
  <owl:inverseOf rdf:resource="#hasMaker"/>
</owl:ObjectProperty>
```

Restriction

- A unique feature of description logics
- Like division: define classes in terms of a restriction that they satisfy with respect to a given property
- Key primitives
  - someValuesFrom a specified class
  - allValuesFrom a specified class
  - hasValue equal to a specified individual or data type
  - minCardinality
  - maxCardinality

Restriction

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="food#PotableLiquid"/>
  <rdfs:subClassOf>
      <owl:Restriction>
          <owl:onProperty rdf:resource="food#hasMaker"/>
          <owl:allValuesFrom rdf:resource="food#Winery"/>
      </owl:Restriction>
  </rdfs:subClassOf>
  </owl:Class>
```

Restriction

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="food#PotableLiquid"/>
  <rdfs:subClassOf>
      <owl:Restriction>
          <owl:onProperty rdf:resource="food#hasMaker"/>
          <owl:allValuesFrom rdf:resource="food#Winery"/>
      </owl:Restriction>
  </rdfs:subClassOf>
  </owl:Class>
```

- $\text{Wine} \sqsubseteq \text{PotableLiquid} \sqcap \forall\ \text{hasMaker.Winery}$

Restriction

```
<owl:Class rdf:ID="Wine">
    <rdfs:subClassOf rdf:resource="food#PotableLiquid"/>
    <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#locatedIn"/>
          <owl:someValuesFrom rdf:resource="#Region"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Restriction

```
<owl:Class rdf:ID="Wine">
    <rdfs:subClassOf rdf:resource="food#PotableLiquid"/>
    <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#locatedIn"/>
          <owl:someValuesFrom rdf:resource="#Region"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

- $\text{Wine} \sqsubseteq \text{PotableLiquid} \sqcap \exists\ \text{locatedIn.Region}$

Inference

- OWL is about content, not the syntax
- Statements from different documents about the same URI are automatically conjoined
- OWL declarations may seem conflicting
  - Declare that no one can have more than one mother
  - Declare Mary is John's mother
  - Declare Jane is John's mother
- What would DBMS say? What would OWL reasoning say?

Inference

- OWL is about content, not the syntax
- Statements from different documents about the same URI are automatically conjoined
- OWL declarations may seem conflicting
  - Declare that no one can have more than one mother
  - Declare Mary is John's mother
  - Declare Jane is John's mother
- What would DBMS say? What would OWL reasoning say?
- A DBMS would declare an integrity violation
- An OWL reasoner would say Mary = Jane

Axioms for instances

To define instances, whether they are the same or different

```
<ex:Country rdf:ID=''Iran''/>
<ex:Country rdf:ID=''Persia''>
  <owl:sameIndividualAs rdf:resource=''#Iran''/>
</ex:Country>

<owl:AllDifferent>
  <owl:distinctMembers rdf:parseType=''Collection''>
    <ex:Country rdf:ID=''Russia''/>
    <ex:Country rdf:ID=''India''/>
    <ex:Country rdf:ID=''USA''/>
  <owl:distinctMembers/>
</owl:AllDifferent>
```

Example: Wine color

Wine color can only be White, Rose, or 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>
```

Example: White wine

White Wine are wine that have color white.

```
<owl:Class rdf:ID="WhiteWine">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Wine" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasColor" />
      <owl:hasValue rdf:resource="#White" />
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Example: All wines

All wines have color white.

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#hasColor"/>
   <owl:hasValue rdf:resource="#White"/>
  </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Example: Non-sweet white wine

WhiteNonSweetWine is WhiteWine with Dry or OffDry sugar level.

```
<owl:Class rdf:ID="WhiteNonSweetWine">
  <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#WhiteWine"/>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#hasSugar"/>
    <owl:allValuesFrom>
      <owl:Class>
        <owl:oneOf rdf:parseType="Collection">
         <owl:Thing rdf:about="#Dry"/>
         <owl:Thing rdf:about="#OffDry"/>
        </owl:oneOf>
      </owl:Class>
    </owl:allValuesFrom>
  </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Example: Zinfandel

Zinfandel is Wine that is made of at most 1 ZinfandelGrape.

```
<owl:Class rdf:ID="Zinfandel">
  <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Wine" />
  <owl:Restriction>
    <owl:onProperty rdf:resource="#madeFromGrape" />
    <owl:hasValue rdf:resource="#ZinfandelGrape" />
  </owl:Restriction>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#madeFromGrape" />
    <owl:maxCardinality rdf:datatype="#xsd:nonNegativeInteger">1</owl:ma
  </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Example: Zinfandel instance

SaucelitoCanyonZinfandel is an instance of Zinfandel that is located in ArroyoGrandeRegion, has a maker named SaucelitoCanyon, is dry, has moderate flavor and has body medium.

```
<Zinfandel rdf:ID="SaucelitoCanyonZinfandel">
  <locatedIn rdf:resource="#ArroyoGrandeRegion" />
  <hasMaker rdf:resource="#SaucelitoCanyon" />
  <hasSugar rdf:resource="#Dry" />
  <hasFlavor rdf:resource="#Moderate" />
  <hasBody rdf:resource="#Medium" />
</Zinfandel>
```

Expressiveness Limitations (1)

OWL DL cannot express the following:

- Non-tree models
  - Instance variables are implicit in OWL restrictions, OWL cannot express conditions that require that two variables be identified
  - Example: Siblings (two people who have the same parents)
- Specialized properties
  - Cannot state that the child of a mammal must be a mammal and so on without
  - Defining new child properties for each class
  - Similar to impossibility of domain-range matching

Expressiveness Limitations (2)

- Constraints among individuals
  - Cannot define tall person: class of persons whose height is above a certain threshold (current average)
  - Can define ETHusband: class of persons who have been married to Elizabeth Taylor (constant)
- Cannot capture defeasibility (also known as nonmonotonicity)
  - Birds fly
  - Penguins are birds
  - Penguins don't fly

Resources

- Wine ontology: `http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf`
- Food ontology: `http://www.w3.org/TR/2004/REC-owl-guide-20040210/food.rdf`
- W3C: `http://www.w3.org/TR/owl2-overview/`
- Editor: Protégé `http://protege.stanford.edu/`
- Reasoners:
  - Pellet: `http://www.mindswap.org/2003/pellet/index.shtml`
  - Jena: `http://jena.sourceforge.net/`
  - OWLAPI: `https://github.com/owlcs/owlapi/wiki`

Reasoner

- Check ontologies for consistency, infer subsumption relations, answer DL queries
- Examples: Hermit, Fact++, Pellet
- OWL provides conformance tests for reasoners
- The reasoners can be used inside Protégé or as APIs inside programs

DL Query

- Check consistency of an ontology
- Individual queries: Returns the set of individuals that satisfy a given description
- Class queries: Returns the class that is either a superclass, subclass, ancestor, or descendant based on what is asked for
- Frame the query in OWL DL

Manchaster Syntax

- A conjunction of descriptions
- Class name
- DataProperty
  - value individual
  - some range
  - min/max/exactly YYY
- ObjectProperty
  - some/only restriction
  - value individual

Example DL Queries

1. Wine
   - Return: BancroftChardonnay, ChateauChevalBlancStEmilion
2. Wine and hasColor value Red
   - Return: ChateauChevalBlancStEmilion, ChiantiClassico, ...
3. Wine and locatedIn value FrenchRegion
   - Return: ChateauChevalBlancStEmilion...

Example DL Queries

4. Wine and locatedIn some (Region and adjacentRegion value MendocinoRegion)
   - Wine from Regions that are adjacent to MendocinoRegion
   - Return: CotturiZinfandel, GaryFarrelMerlot, ...
5. CaliforniaWine and not (hasBody value Medium)
   - Open world assumption
   - Return: CotturiZinfandel, ElyseZinfandel,...
   - Wine and hasColor value Red

Finding similarity

- Mostly among concepts; different than inference
- When you want to refer to things approximately
  - Ex: Things that are similar to chair
  - Might find table because both are furniture
  - Might find couch because both are used to sit on
- Need to evaluate how similar a concept is to a given concept
- Various metrics with various biases

Tversky's metric

- Designed for comparing similarity of two vectors
- Can be adopted for ontology concepts through measuring property similarity
- Two concepts are identical if their properties are the same
- Can be adapted for individuals by comparing property values

$$S = \frac{\alpha(common)}{\alpha(common) + \alpha(different)}$$

Wu and Palmer's metric

- The similarity between c_1 and c_2 is estimated in relation to c_0, which is the most specific concept subsuming them.
- N_1 is the number of edges between c_1 and c_0.
- N_2 is the number of edges between c_2 and c_0.
- N_0 is the number of IS-A links of c_0 from the root of the taxonomy.

$$S = \frac{2 * N_0}{N_1 + N_2 + 2 * N_0}$$