Knowledge management

OWL Web Ontology Language

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

Limitations of RDFS

- 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

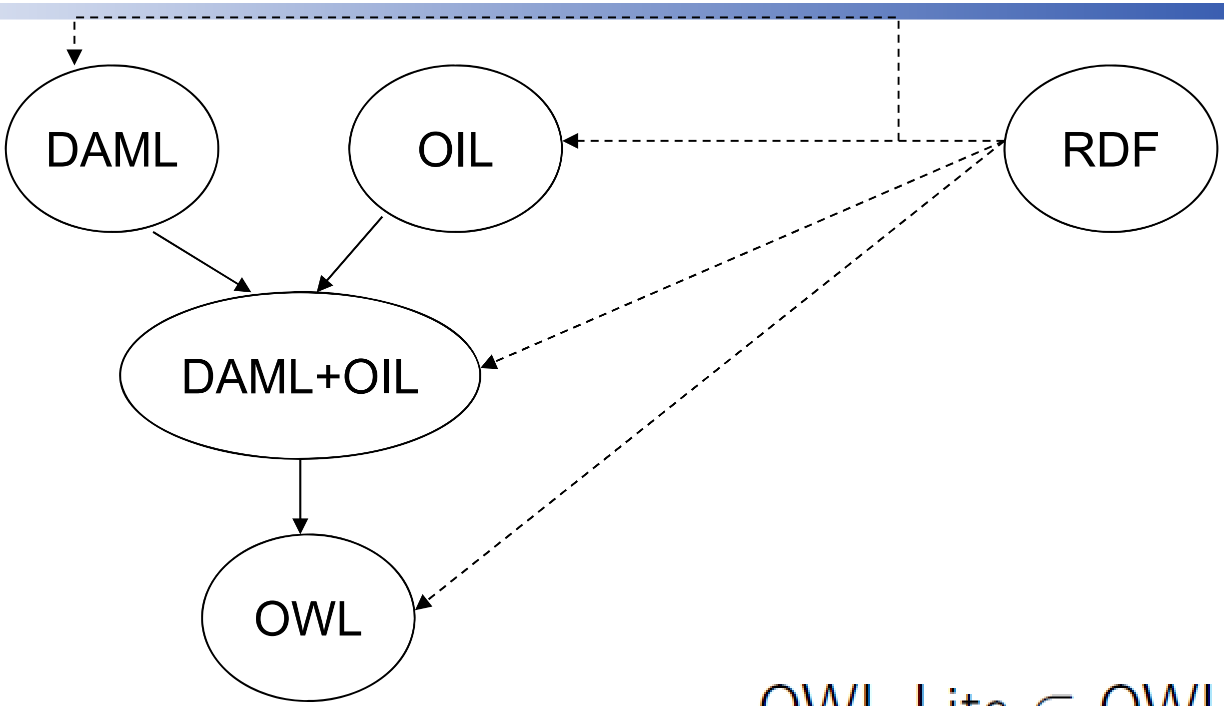
Limitations of RDFS

- 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

- No in/equality
  - Can’t say that a class/instance is the same as some other class/instance, can’t say that the classes/instances are definitely disjoint/different.

- No boolean algebra
  - Can’t say that that one class is the union, intersection, etc. of other classes

Ontology Web Language -- OWL



$$\text{OWL Lite} \subset \text{OWL DL} \subset \text{OWL Full}$$

- Three species of OWL
  - OWL Lite is the simplest language (+easy to implement/-less expressive)
  - OWL DL (+more expressive)
  - OWL Full is union of OWL syntax and RDF
- OWL allows greater expressiveness than RDF-S

THE STRUCTURE OF OWL ONTOLOGIES

OWL: Ontology Namespaces

- Standard namespaces in an OWL ontology:

```
<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: Ontology Namespaces

- Example:

```
<rdf:RDF
        xmlns:sm ="http://www.example.org/superMarket#"
        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#" >
```

- Can be also written

```
<!DOCTYPE rdf:RDF [
  <!ENTITY sm "http://www.example.org/superMarket#"> ]
>

<rdf:RDF
        xmlns:sm = "&sm;"
        …>
```

OWL: Ontology Headers

- Collection of assertions about the ontology grouped under an **owl:Ontology** tag

```
<owl:Ontology rdf:about="">
        <rdfs:comment>An example OWL ontology</rdfs:comment>
        <owl:priorVersion
rdf:resource="http://www.example.org/old/superMarket"/>
        <owl:imports rdf:resource="http://www.example.org/person"/>
        <rdfs:label>Super Market Ontology</rdfs:label>
        …
</owl:Ontology>
```

- *priorVersion* provides a link to the previous version → Ontology versioning
- *imports* provides an include-style mechanism

BASIC ELEMENTS OF OWL ONTOLOGIES

Classes: Declaration

- Every *class* in the OWL world is a member of the class *owl:Thing*
- Example of classes in the super Market Ontology

```
<owl:Class rdf:ID="Shelf"/>
<owl:Class rdf:ID="Product"/>
<owl:Class rdf:ID="Customer"/>
```

- rdf:ID=“Shelf“ introduces the name of the resource
  - Inside the ontology: the Shelf class can be referred to using #Shelf (e.g. rdf:resource="#Shelf)".
  - Outside the ontology: the Shelf class can be referred to using its complete URI (e.g. http://www.example.org/superMarket#Shelf)".

Classes: Definition

- A class definition has two parts: a name introduction or reference and a list of restrictions.

```
<owl:Class rdf:ID="Customer">
        <rdfs:subClassOf rdf:resource="cl:Person"/>
        <rdfs:label xml:lang="en">customer</rdfs:label>
        <rdfs:label xml:lang="fr">client</rdfs:label>
        ...
</owl:Class>
```

rdfs:SubClassOf defines a restriction

Individuals

- Individuals are the members of a class

```
<Product rdf:ID= "Apple" />
```

Equivalent to

```
<owl:Thing rdf:ID="Apple" />

<owl:Thing rdf:about="#Apple">
        <rdf:type rdf:resource="#Product"/>
</owl:Thing>
```

Properties

- Two types of properties:
  - Object property: `resource property resource`

    owl:ObjectProperty
  - Datatype property: `resource property literal`

    owl:DatatypeProperty

- A property has the same “properties” used in RDF-S:
  - rdfs:subPropertyOf, rdfs:domain and rdfs:range
- Example (Wine Ontology)

```
<owl:ObjectProperty rdf:ID="madeFromGrape">
        <rdfs:domain rdf:resource="#Wine"/>
        <rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>
```

Properties Hierarchy

- Example (Wine Ontology)

```
<owl:Class rdf:ID="WineDescriptor" />

<owl:Class rdf:ID="WineColor">
        <rdfs:subClassOf rdf:resource="#WineDescriptor" />
        ...
</owl:Class>

<owl:ObjectProperty rdf:ID="hasWineDescriptor">
        <rdfs:domain rdf:resource="#Wine" />
        <rdfs:range rdf:resource="#WineDescriptor" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasColor">
        <rdfs:subPropertyOf rdf:resource="#hasWineDescriptor" />
        <rdfs:range rdf:resource="#WineColor" />
        ...
</owl:ObjectProperty>
```

Properties Characteristics (1/5)

- Transitive property
  - **P(x,y) and P(y,z) ➔ P(x,z)**
  - Wine Ontology Example:

```
<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>
```

Properties Characteristics (2 / 5)

- Symmetric property
  - **P(x,y) if and only if P(y,x)**
  - Wine Ontology Example:

```
<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>
```

Properties Characteristics (3 / 5)

- Functional property
  - **P(x,y) and P(x,z) implies y = z**
  - A functional property states that the value of range for a certain object in the domain is always the same.
  - Wine Ontology Example:

```
<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>
```

Properties Characteristics (4/5)

- InverseOf property
  - **`P1(x,y) iff P2(y,x)`**
  - Wine Ontology Example:

```
<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>
```

Properties Characteristics (5 / 5)

- Inverse Functional property
  - **P(y,x) and P(z,x) implies y = z**
  - A functional property states that the value of range for a certain object in the domain is always the same.
  - Wine Ontology Example:

```
<owl:ObjectProperty rdf:ID="hasMaker" />

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

Exercise

- Represent the following Object Properties:
  - ancestor such as If a person A is an ancestor of person B and B of C then A is also an ancestor of C.
  - akin such as if a Person A is akin to a Person B then B is also akin to A.
  - hasFather such as a child has always the same (biological) Father
  - hasChild such as If a Person A hasChild a Person B then B hasFather A

<rdf:type rdf:resource="&owl;TransitiveProperty" />

<rdf:type rdf:resource="&owl;FunctionalProperty" />

<owl:inverseOf rdf:resource=« propertyName" />

<rdf:type rdf:resource="&owl;SymmetricProperty" />

Property Restrictions

- Defining a Class by restricting its possible instances via their property values

- OWL distinguishes between the following two:
  - Value constraint
    - ($\text{Mother} \equiv \text{Woman} \sqcap \exists \text{hasChild.Person}$)
  - Cardinality constraint
    - ($\text{MotherWithManyChildren} \equiv \text{Mother} \sqcap \geq 3\text{hasChild}$)

Property Restrictions: allValuesFrom

- Wine Ontology example:

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

  - The maker of a Wine must be a Winery.
  - The restriction is on the hasMaker property of this Wine class ***only***.

Property Restrictions: someValuesFrom

- Wine Ontology example:

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

  – At least one of the makers of a Wine must be a Winery.

allValuesFrom vs. someValuesFrom

- The difference between the two formulations is the difference between a **universal** and **existential** quantification:
  - allValuesFrom: Universal quantification

    e.g. For all wines, if they have makers, all the makers are wineries

    Does not require a wine to have a maker
  - someValuesFrom: Existential quantification

    e.g. For all wines, they have at least one maker that is a winery

    A wine must have a maker

Property Restrictions: hasValue

- Allows to define classes based on the existence of particular property values
- Wine Ontology example:

```
<owl:Class rdf:ID="Burgundy">
        ...
        <rdfs:subClassOf>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasSugar" />
                        <owl:hasValue rdf:resource="#Dry" />
                </owl:Restriction>
        </rdfs:subClassOf>
</owl:Class>
```

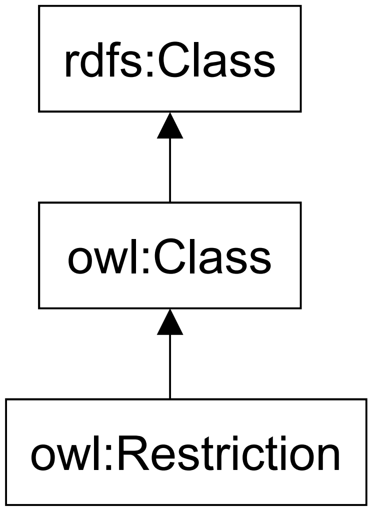
Property Restrictions: cardinality

- 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
- Wine Ontology example:

```
<owl:Class rdf:ID="Vintage">
        <rdfs:subClassOf>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasVintageYear"/>
                        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
                                1
                        </owl:cardinality>
                </owl:Restriction>
        </rdfs:subClassOf>
</owl:Class>
```

Property Restrictions: Summary

- define a class using LOCAL restrictions on a specific Property



- **Property restrictions:**
  - 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})

Exercises on Property Restrictions

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

  $Mother \sqsubseteq Woman \sqcap \exists\ hasChild.Person$

- The set of parents that only have daughters (female children)

  $ParentsWithOnlyDaughters \sqsubseteq Person \sqcap \forall hasChild.Woman$

- The set of all child of the woman MARRY

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

- A half Orphan (i.e. a person that has only one Parent)

  $HalfOrphan \sqsubseteq Person \sqcap =1 hasParent.Person$

COMPLEX CLASSES IN OWL ONTOLOGIES

Complex Classes

- (OWL DL) provide constructors with which we can form classes based on basic set operations:
  - Intersection
  - Union
  - Complement
- Enumerated classes
- Disjoint classes

Complex Classes: Intersection of Classes

- Instances/Individuals of the Intersection of two Classes are simultaneously instances of both class
- Wine Ontology example:

```
<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>
```

→ Defines white wine

Complex Classes: Union of Classes

- Instances/Individuals of the Union of two Classes are either the instance of one or both classes
- Wine Ontology example:

```
<owl:Class rdf:ID="Fruit">
        <owl:unionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#SweetFruit" />
                <owl:Class rdf:about="#NonSweetFruit" />
        </owl:unionOf>
</owl:Class>
```

Complex Classes: Complement

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

```
<owl:Class rdf:ID="NonConsumableThing">
        <owl:complementOf rdf:resource="#ConsumableThing" />
</owl:Class>
```

- Question: What is the meaning of

```
<owl:Class rdf:ID="NonFrenchWine">
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Wine"/>
        <owl:Class>
                <owl:complementOf>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#locatedIn" />
                        <owl:hasValue rdf:resource="#FrenchRegion" />
                    </owl:Restriction>
                </owl:complementOf>
         </owl:Class>
    </owl:intersectionOf>
</owl:Class>
```

Complex Classes: Enumerated Classes

- OWL provides the means to specify a class via a direct enumeration of its members
  - the **`owl:oneOf`** construct.
- Completely specifies the class extension, and no other individuals can be declared to belong to the class.
- Wine Ontology example:

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

Complex Classes: Disjoint Classes

- The disjointness of a set of classes can be expressed using the **`owl:disjointWith`** constructor
- An individual that is a member of one class cannot simultaneously be an instance of another one.
- Example:

```
<owl:Class rdf:ID="Pasta">
        <rdfs:subClassOf rdf:resource="#EdibleThing"/>
        <owl:disjointWith rdf:resource="#Meat"/>
        <owl:disjointWith rdf:resource="#Fowl"/>
        <owl:disjointWith rdf:resource="#Seafood"/>
        <owl:disjointWith rdf:resource="#Dessert"/>
        <owl:disjointWith rdf:resource="#Fruit"/>
</owl:Class>
```

Exercises complex classes

- $Person \equiv Man \sqcup Woman$
- $Man \equiv Person \sqcap Male$

ONTOLOGY MAPPING AND REUSE

Ontology Reuse

- To create a knowledge base or a semantic Web application we can create a new ontology
  - ☹ Designing a large ontology is difficult
  - ☺ Better reuse, compose, extend existing ontologies to define a new one.
- Blending existing ontologies is difficult, but OWL provides constructs facilitating ontology reuse

Ontology Reuse: equivalence (1/3)

- When several ontologies are used as part of another ontology, it's useful to be able to indicate that a particular class (or property) in one ontology is equivalent to a class (or property) in a second ontology.
  - **`owl:equivalentClass`**
  - **`owl:equivalentProperty`**
- Example: SuperMarket ontology linking to Wine Ontology

```
<rdf:RDF
        xmlns:vin ="http://www.w3.org/REC-owl-guide-20040210/wine#
        xmlns:owl ="http://www.w3.org/2002/07/owl#"
 ...>
<owl:Class rdf:ID="Wine">
        <owl:equivalentClass rdf:resource=“vin:Wine"/>
</owl:Class>
```

Ontology Reuse: equivalence (2/3)

- Equivalence can be used over a restriction
- Exemple:

```
<owl:Class rdf:ID="TexasThings">
        <owl:equivalentClass>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#locatedIn" />
                        <owl:someValuesFrom rdf:resource="#TexasRegion" />
                </owl:Restriction>
        </owl:equivalentClass>
</owl:Class>
```

  – TexasThings contains exactly the objects located in the TexasRegion

Ontology Reuse: equivalence (3/3)

- What is the difference between using:

```
<owl:Class rdf:ID="TexasThings">
        <owl:equivalentClass>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#locatedIn" />
                        <owl:someValuesFrom rdf:resource="#TexasRegion" />
                </owl:Restriction>
        </owl:equivalentClass>
</owl:Class>
```

- AND

```
<owl:Class rdf:ID="TexasThings">
        <owl:subClassOf>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#locatedIn" />
                        <owl:someValuesFrom rdf:resource="#TexasRegion" />
                </owl:Restriction>
        </owl:subClassOf>
</owl:Class>
```

Ontology Reuse: equivalence (3/3)

- What is the difference between using:
  - `owl:subClassOf`
    - things that are located in Texas are not necessarily TexasThings

      ➔ Expresses Necessary condition
  - `owl:equivalentClass`
    - if something is located in Texas, then it must be in the class of TexasThings

    ➔ Expresses Necessary and Sufficient condition

Ontology Reuse: Property equivalence

SAME AS Classes equivalence

Using `owl:equivalentProperty`

Ontology Reuse: Identity between Individuals

- To explicitly state that two individuals are identical
  - owl:sameIndividualAs / owl:sameAS
- Wine Ontology example:

```
<Wine rdf:ID="MikesFavoriteWine">
        <owl:sameAs rdf:resource="#StGenevieveTexasWhite" />
</Wine>
```

- Another example:

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

<owl:Thing rdf:about="#BancroftChardonnay">
        <hasMaker rdf:resource="#Bancroft" />
        <hasMaker rdf:resource="#Beringer" />
</owl:Thing>
```

#Bancroft is the same as #Beringer?

Ontology Reuse: Different Individuals (1/2)

- `owl:differentFrom` provides the opposite effect from `owl:sameAS`
- Wine Ontology example:

```
<WineSugar rdf:ID="Dry" />

<WineSugar rdf:ID="Sweet">
        <owl:differentFrom rdf:resource="#Dry"/>
</WineSugar>
```

- In some cases it’s important to ensure such distinctions. Example:
  - We have not asserted that **Dry** and **Sweet** are different
  - WineSugar is functional
  - If we describe a wine as both Dry and Sweet
    ➔ this would imply that Dry and Sweet are identical

Ontology Reuse: Different Individuals (2/2)

- To define a set of mutually distinct individuals :

```
<owl:AllDifferent>
        <owl:distinctMembers rdf:parseType="Collection">
                <vin:WineColor rdf:about="#Red" />
                <vin:WineColor rdf:about="#White" />
                <vin:WineColor rdf:about="#Rose" />
        </owl:distinctMembers>
</owl:AllDifferent>
```

➔ Red, White, and Rose are pairwise distinct

- **Note**: `owl:distinctMembers` can only be used in combination with `owl:AllDifferent`

OWL on 2 slides

- **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

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

OWL on 2 slides

- **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

Exercise

Create an OWL ontology that models the following concepts:

1. There should be three classes: Customer, Shop and Product.
2. Customer and Shop should be equipped with properties name (xsd:string) and email (xsd:string), which are equivalent to foaf:name and foaf:mbox.
3. Each Product should have an order number (xsd:int). An order number can be unambiguously assigned to a Product.
4. A Shop should have a property sells (range: Product) and a Product should have a property soldBy (range: Shop) respectively.
5. Instances of class Shop that sell more than 100 products should belong to a new class BigShop.
6. A Product must not be a Customer.
7. Instances that are both, Shop and Customer should belong to a class PurchaseAndSale.

References

- Slides based on:
  - OWL guide: http://www.w3.org/TR/owl-guide/
- OWL page: http://www.w3.org/2004/OWL/
- OWL reference: http://www.w3.org/TR/owl-ref/