OWL: abstract syntax

For details see http://www.w3.org/TR/owl-semantics/syntax.html#2.3.2.1
Classes: primitive vs. defined

**descriptions**

```
Class(name partial ...)  
'all name ...'
```

**primitive concepts**

```
Class(MargheritaPizza partial Pizza  
   restriction(hasTopping  
      someValuesFrom(Mozzarella))  
   restriction(hasTopping  
      someValuesFrom(Tomato)))
```

Example:

‘All Margherita pizzas have, amongst other things, some mozzarella topping and also some tomato topping’

**definitions**

```
Class(name complete ...)
'a name is anything that ...'
```

**defined concepts**

```
Class(CheesyPizza complete Pizza  
   restriction(hasTopping  
      someValuesFrom(Cheese)))
```

‘A cheesy pizza is any pizza that has, amongst other things, some cheese topping’
Classes: disjointness

“What does such a hierarchy actually mean?”

In OWL, classes are overlapping until **disjointness axiom** is entered:

\[
\text{DisjointClasses}(\text{class}_1 \ldots \text{class}_n)
\]

Example:

\[
\text{DisjointClasses(}
\text{Vegetable Meat Seafood Cheese})
\]

PizzaTopping
- Vegetable
  - Tomato
  - Pepper
  - Mushroom
- Meat
  - SpicyBeef
  - Pepperoni
- Seafood
  - Tuna
  - Prawn
  - Anchovy
- Cheese
  - Mozzarella
  - Parmesan
Property restrictions

existential

\[ \text{restriction}(\text{prop someValuesFrom class})]  

'\textit{some}', 'at least one'

universal

\[ \text{restriction}(\text{prop allValuesFrom class})]  

'\textit{only}', 'no value except'

Example:

Class(DogOwner complete Person  
  restriction(hasPet someValuesFrom Dog)))

'A dog owner is any person who has as a pet some dog'

Class(FirstClassLounge complete Lounge  
  restriction(hasOccupants allValuesFrom FirstCPassenger)))

'A first class lounge is any lounge where the occupants are only first class passengers'

'A first class lounge is any lounge where there are no occupants except first class passengers'
Existential

\[ \exists \]

Example:

Class(DogOwner partial Person
  restriction(hasPet someValuesFrom(Dog)))

‘Dog owners are people and have as a pet some dog’

Universal

\[ \forall \]

Class(FirstClassLounge partial Lounge
  restriction(hasOccupants allValuesFrom(FirstCPassenger)))

‘All first class lounges have only occupants who are first class passengers’

‘All first class lounges have no occupants except first class passengers’

‘All first class lounges have no occupants who are not first class passengers’
Boolean combinations

**union** (disjunction)

\[
\text{unionOf}(\text{class}_1 \ldots \text{class}_n)
\]

‘\text{class}_1 \text{ and/or class}_2’

**intersection** (conjunction)

\[
\text{intersectionOf}(\text{class}_1 \ldots \text{class}_n)
\]

‘\text{both class}_1 \text{ and also class}_2’

**Example:**

Class(VegetarianPizza complete Pizza restriction(hasTopping allValuesFrom(unionOf(Vegetable Cheese))))

‘A vegetarian pizza is any pizza which, amongst other things, has only vegetable and/or cheese toppings’

Class(ProteinLoversPizza complete Pizza restriction(hasTopping allValuesFrom(intersectionOf(Meat Seafood))))

‘A protein lover’s pizza is any pizza that, amongst other things, has only toppings that are both meat and also seafood’

**NO** topping is both meat and also seafood!

(therefore, the intersection is empty)
Boolean combinations (cont.)

- **complementOf**(class)
- **complementOf**(intersectionOf(class$_1$ class$_2$))
  - ‘not all of’ / ‘not both class$_1$ and also class$_2$’
- **complementOf**(unionOf(class$_1$ class$_2$))
  - ‘neither class$_1$ nor class$_2$’
- **restriction**(prop someValuesFrom(complementOf(class)))
  - ‘has some prop that are not class’
- **complementOf**(restriction(prop someValuesFrom(class)))
  - ‘does not have any prop that are class’
- **restriction**(prop allValuesFrom(complementOf(class)))
  - ‘has prop no class’ / ‘has only prop that are not class’
- **complementOf**(restriction(prop allValuesFrom(class)))
  - ‘does not have only prop that are class’
Cardinality constraints

**restriction**(prop \(\text{minCardinality}(n)\))

‘at least \(n\) (distinct) \(prop\)’

\(\geq\)

**restriction**(prop \(\text{maxCardinality}(n)\))

‘at most \(n\) (distinct) \(prop\)’

\(\leq\)

**Example:**

Class(InterestingPizza complete Pizza restriction(hasTopping \(\text{minCardinality}(3)\)))

‘An interesting pizza is any pizza that, amongst other things, has at least 3 (distinct) toppings’

Class(Pizza partial restriction(hasBase \(\text{maxCardinality}(1)\)))

‘Any pizza, amongst other things, has at most 1 pizza base’
Object properties

**ObjectProperty** \((name \ldots \text{ domain(classD) range(classR)})\)

Domain and range constraints are actually **axioms**:

**range**

\[
\text{Class(owl:Thing partial restriction(name allValuesFrom(classR)))}
\]

‘All things have no name except classR’

**domain**

\[
\text{SubClassOf(restriction(name someValuesFrom(owl:Thing)) classD)}
\]

‘Having a name implies being classD’
Object properties: domain constraints

ObjectProperty(hasTopping domain(Pizza))

‘Having a topping implies being pizza’

Consider now ice-cream cones:

Class(IceCreamCone partial restriction(hasTopping someValuesFrom(IceCream)))

‘All ice-cream cones, amongst other things, have some ice-cream topping’

NB: if ice-cream cone is disjoint from pizza then the definition of ice-cream cone is inconsistent

otherwise ice-cream cone will be classified as a kind of pizza
Examples:
Bus Drivers are Drivers

Class(Driver complete Person restriction(drives someValuesFrom(Vehicle))))

‘A driver is any person that drives a vehicle’

Class(Bus partial Vehicle)

‘All buses are vehicles’

Class(BusDriver complete Person restriction(drives someValuesFrom(Bus))))

‘A bus driver is any person that drives a bus’

So, a bus driver must be a driver:

BusDriver ⊑ Driver

(the subclass is inferred due to subclasses being used in existential quantification)
Drivers are Grown-ups

Class(Driver complete
    Person
    restriction(drives
        someValuesFrom(Vehicle)))

‘A driver is any person that
    drives a vehicle’

Class(Driver partial Adult)

‘Drivers are adults’

Class(GrownUp complete
    Person Adult)

‘A grown up is any person that is an adult’

So, all drivers must be adult persons (grown-ups):

\[
\text{Driver} \subseteq \text{GrownUp}
\]

(an example of axioms being used to assert additional necessary information about a class; we do not need to know that a driver is an adult in order to recognise one, but once we have recognised a driver, we know that they must be adult)
Cat Owners like Cats

Class(CatOwner complete Person restriction(hasPet someValuesFrom(Cat)))

'A cat owner is any person that has a cat as a pet'

SubPropertyOf(hasPet likes)

'Anything that has a pet must like that pet'

Class(CatLover complete Person restriction(likes someValuesFrom(Cat)))

'A cat-lover is any person that likes a cat'

So, a cat owner must like a cat:

CatOwner ⊑ CatLover

(the subclass is inferred due to a subproperty assertion)