Pizza Ontology Notes

# 4.1 Named Classes

* Classes set by the user
* Different from anonymous classes
* E.G.
  + Country, Pizza, PizzaBase, PizzaTopping, Spiciness

# 4.2 Disjoint Classes

* Specifies that an individual cannot be an instance of more than one of these classes
* E.G.
  + Country, Pizza, PizzaBase, PizzaTopping, Spiciness

# 4.3 Class Hierarchy

* Subclasses of a class = *necessary implication*
  + If VegetableTopping is a *subclass* of PizzaTopping, then all instances of VegetableTopping are instances of PizzaTopping

# 4.4 OWL Properties

* Represent relationships
  + Object properties
  + Datatype properties
  + Annotation properties

# 4.5 Inverse Properties

* Each object property can have a corresponding inverse property
  + hasParent <-> hasChild
  + hasIngredient <-> isIngredientOf

# 4.6 OWL Object Property Characteristics

## 4.6.1 Functional Properties

* Given functional property P, (x P y1), and (x P y2), y1 and y2 must be the same, i.e. **uniquely declares the object**
* E.G.
  + hasBirthMother
  + Jean hasBirthMother Peggy AND Jean hasBirthMother Margaret
    - Peggy and Margaret must be the same person
  + hasBase

## 4.6.2 Inverse Functional Properties

* Given inverse functional property P, (x1 InvP y), and (x2 InvP y), x1 and x2 must be the same, i.e. **the inverse uniquely declares the subject**
* E.G.
  + hasBirthMother
  + Peggy isBirthMotherOf Jean AND Margaret isBirthMotherOf Jean
    - Peggy and Margaret must be the same person
  + **DOES NOT MEAN:** Peggy isBirthMotherOf George AND Peggy isBirthMotherOf Jack
    - George and Jack must be the same person (?)

## 4.6.3 Transitive Properties

* If P is transitive, then given (x P y) and (y P z), we can infer (x P z)
* E.G.
  + has Ancestor
  + Mathew hasAncestor Peter AND Peter hasAncestor William
    - Mathew hasAncestor William
  + isIngredientOf, hasIngredient

## 4.6.4 Symmetric Properties

* If P is symmetric, then if (x P y), we can infer (y P x)
* E.G.
  + hasSibling
  + Mathew hasSibling Gemma
    - Gemma hasSibling Mathew

## 4.6.5 Asymmetric Properties

* If P is asymmetric, then if (x P y), we can infer that (y P x) is **not possible**
* E.G.
  + isChildOf
  + Jean hasChild Mathew
    - Mathew cannot be related to Jean with hasChild

## 4.6.6 Reflexive Properties

* If P is reflexive, P must relate x to itself
* E.G.
  + Knows
  + George knows George

## 4.6.7 Irreflexive Properties

* Opposite of Reflexive Properties, P can’t relate x to itself
* E.G.
  + motherOf

# 4.7 Property Domains and Ranges

# 4.8 Describing Classes

## 4.8.1 Property Restrictions

* Quantifier Restrictions
  + Existential
    - Protégé 5: keyword **some**
    - Specifies all individuals that have *at least* *one* relationship P with members of a specified class
    - Most common type of restriction in OWL ontologies
  + Universal
    - Protégé 5: keyword **only**
    - Specifies all individuals that have only relationship P with members of a specified class
* Cardinality Restrictions
* hasValue Restrictions

# 4.9 Protégé Reasoner

## 4.9.1 Running the reasoner

* Select HermiT 1.3.8.x
* CTRL + r

## 4.9.2 Inconsistent Classes

* Add *Probe Classes* to test integrity of the ontology
* E.G. ProbeInconsistentTopping is subclass of both CheeseTopping and VegetableTopping

# 4.10 Necessary and Sufficient Conditions (Primitive and Defined Classes)

* Primitive Classes
  + If something is a PepperoniTopping, it is *necessary* for it to have a spiciness of medium and it is also inferred to be a MeatTopping