Pizza Ontology Notes

Source: <http://mowl-power.cs.man.ac.uk/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf>

# 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
* E.G 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
* E.G. 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

* Link **classes** to properties
* 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
  + See [4.16](#_4.16_Cardinality_Restrictions) and [4.17](#_4.17_Qualified_Cardinality)
* hasValue Restrictions
  + See [7.2](#_7.2_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 *necessary* for it to be a MeatTopping
  + However, if something has a spiciness of medium and is a MeatTopping, we can’t say it’s a PepperoniTopping
* Defined Classes
  + If something is a CheesyPizza, it is *necessary* for it to have at least one CheeseTopping and it is also *necessary* for it to be a Pizza. If something has a CheeseTopping and is a Pizza, this knowledge is *sufficient* enough to say that it is a CheesyPizza
* Protégé 5: Necessary = **Subclass Of**, Necessary and Sufficient = **Equivalent to**

# 4.11 Automated Classification

* The reasoner can automatically classify and add subclasses to defined classes

# 4.12 Universal Restrictions

* VegetarianPizza

# 4.13 Automated Classification and Open World Reasoning

* Open World Reasoning (OWR) or Open World Assumption (OWA)
  + Cannot assume something doesn’t exist unless explicitly stated
* Unless explicitly stated, Pizzas with existential relationships (**some**) with PizzaToppings cannot be assumed to have only those PizzaToppings

## 4.13.1 Closure Axioms

* A universal restriction that acts along the property to say that it can *only* be filled by the specified fillers.
* E.G. AmericanPizza
  + hasTopping only (MozzarellaTopping or PepperoniTopping or TomatoTopping)

# 4.14 Value Partitions

* A design pattern that restricts the range of possible values to an *exhaustive list*
* Uses a *covering axiom*
* E.G. SpicinessValuePartition
  + Mild, Medium, Hot

## 4.14.1 Covering Axioms

* Specifies that a class is completely *covered* by its subclasses, i.e. every instance of the class is an instance of one of its subclasses
* Every instance of SpicinessValuePartition is either Mild, Medium, or Hot

# 4.15 Adding Spiciness to PizzaToppings

# 4.16 Cardinality Restrictions

* Specifies the number of P relationships an individual must participate in
* Does not specify class
* E.G. Interesting Pizza
  + An individual that is an InterestingPizza must have at least 3 hasTopping relationships
  + Does **not** specify PizzaTopping class

# 4.17 Qualified Cardinality Restrictions

* Specifies the number of P relationships an individual must participate in with class C
  + Does specify class
* E.G. FourCheesePizza
  + Has exactly 4 hasTopping relationships with individuals of the CheeseTopping class.

# 5.0 Datatype Properties

* Link an **individual** to data values
  + Strings, integers, etc
* Why not class-level relationships
  + Nothing intrinsic to data properties that says we can’t make them classes, but classes, in general, are too general
  + Too strong to say all MargheritaPizzas have 263 calories
* hasCalorificContentValue
  + Note: units are in name

# 6.0 More Open World Reasoning Examples

* NonVegetarianPizza defined to be all Pizzas that are not VegetarianPizzas
* UnclosedPizza does not have a closure axiom, so we can’t say whether it is a VegetarianPizza or NonVegetarianPizza

# 7.1 Creating Individuals

* Example Pizzas
* Countries

# 7.2 hasValue Restrictions

* Describes class(s) that have *at least one* P relationship to a *specific individual*
* E.G. hasCountryOfOrigin

# 7.3 Enumerated Classes

* Specify a class made up of the *enumerated* (named) individuals
* More specific than a *covered* class
* E.G. Country

# 7.4 Annotation Properties

* Used to provide information/meta-data about classes, properties, individuals, or the ontology itself
* COPIED FROM DOCUMENT:

OWL has five pre-defined annotation properties that can be used to annotate classes (including anony-

mous classes such as restrictions), properties and individuals:

1. owl:versionInfo | in general the range of this property is a string.

2. rdfs:label | has a range of a string. This property may be used to add meaningful, human readable

names to ontology elements such as classes, properties and individuals. rdfs:label can also be used

to provide multi-lingual names for ontology elements.

3. rdfs:comment | has a range of a string.

4. rdfs:seeAlso | has a range of a URI which can be used to identify related resources.

5. rdfs:isDefinedBy | has a range of a URI reference which can be used to reference an ontology that

de\_nes ontology elements such as classes, properties and individuals.

For example the annotation property rdfs:comment is used to store the comment for classes in Protege

4. The annotation property rdfs:label could be used to provide alternative names for classes, properties

etc.

There are also several annotation properties which can be used to annotate an ontology. The ontology

annotation properties (listed below) have a range of a URI reference which is used to refer to another

ontology. It is also possible to the use the owl:VersionInfo annotation property to annotate an ontology.

\_ owl:priorVersion | identi\_es prior versions of the ontology.

\_ owl:backwardsCompatibleWith | identi\_es a prior version of an ontology that the current ontology

is compatible with. This means that all of the identi\_ers from the prior version have the same

intended meaning in the current version. Hence, any ontologies or applications that reference the

prior version can safely switch to referencing the new version.

\_ owl:incompatibleWith | identi\_es a prior version of an ontology that the current ontology is not

compatible with.