Ontology tools

Oscar Corcho
ocorcho@fi.upm.es
http://www.oeg-upm.net/

Ontological Engineering Group
Laboratorio de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo sn,
28660 Boadilla del Monte, Madrid, Spain

Acknowledgements

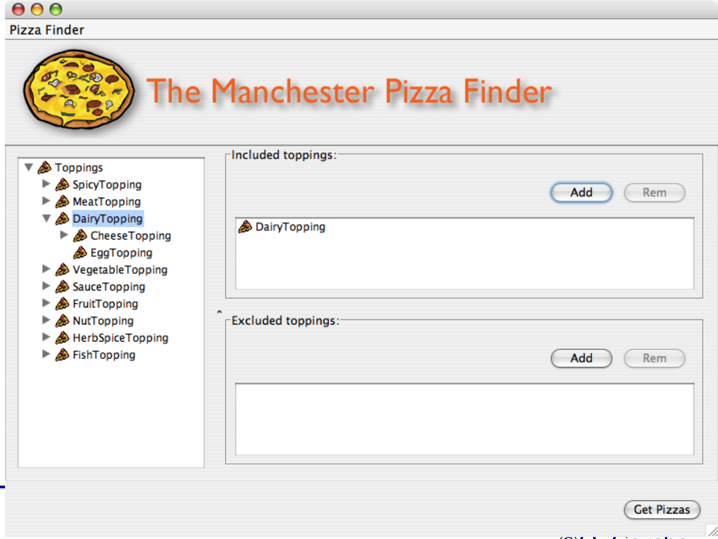
- **Asunción Gómez-Pérez and Mariano Fernández-López**
  - **Most of the slides have been done jointly with them**
- **Nick Drummond and Matthew Horridge (University of Manchester)**
  - **Reasoning with OWL ontologies**

Table of contents

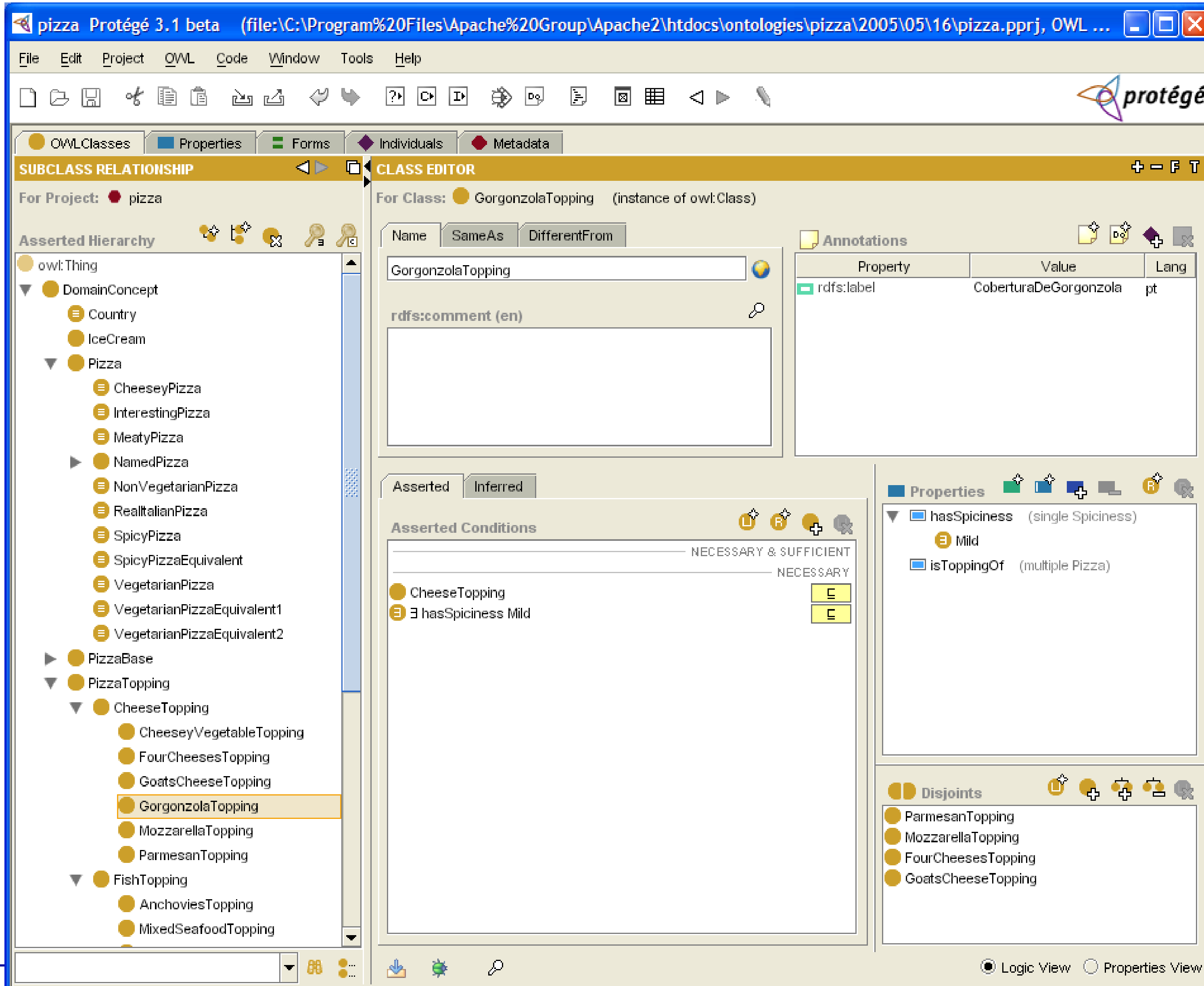
Our Domain and Our Application

- **Pizzas selected as a domain for several reasons:**
  - **They are fun and fairly neutral**
  - **They are internationally known**
  - **They are highly compositional**
  - **They have a natural limit to their scope**

- **Application**
  - **The PizzaFinder**
    - www.co-ode.org/downloads/pizzafinder/



Starting with a Pizza Ontology...

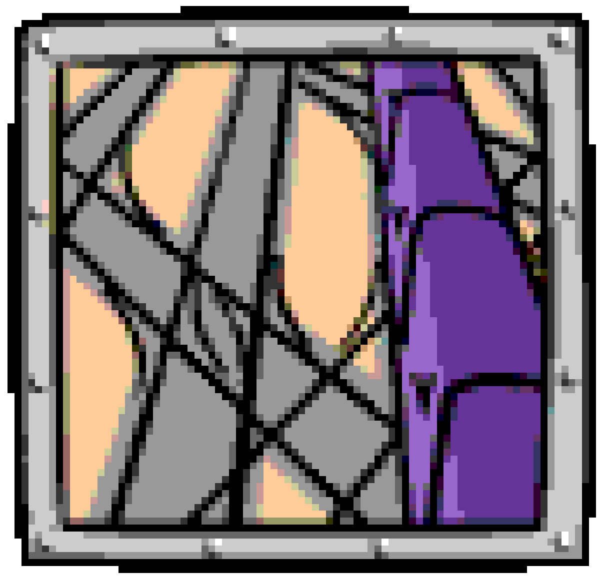


Consistency Checking

- **We've just created a class that doesn't really make sense**
  - **What is a MeatyVegetableTopping?** ***What is a MadCow?***
- **We'd like to be able to check the logical consistency of our model**
  - **This is one of the tasks that can be done by a Reasoner/Classifier**

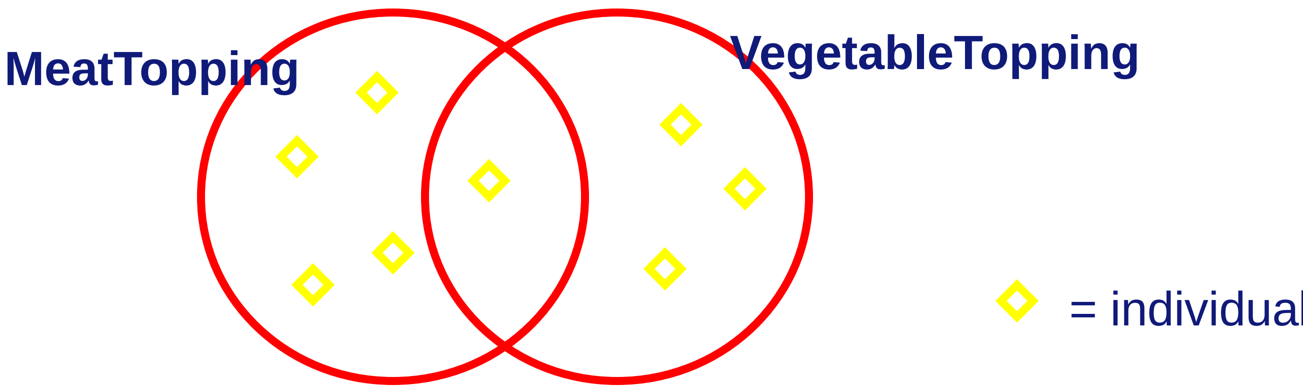
- **Protégé-OWL supports the use of reasoners implementing the DIG interface**
  - **The reasoner is independent of the ontology editor**
  - **We can choose an implementation depending on our needs (eg some may be more optimised for speed/memory, others may have more features)**
- **These reasoners typically set up a service running locally or on a remote server**
  - **Protégé-OWL can only connect to reasoners over an http:// connection**

Check consistency



Disjointness

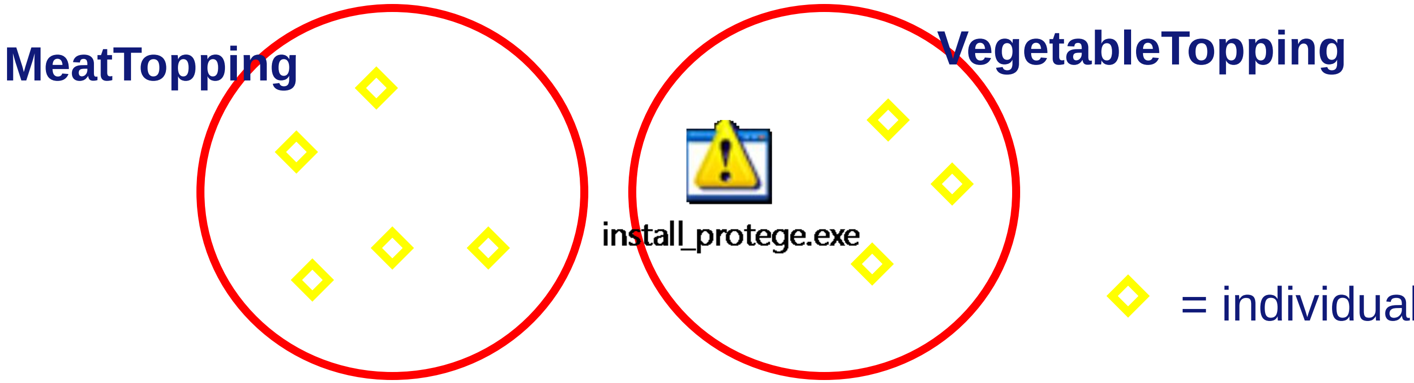
- **OWL assumes that classes overlap**



  - ▶ This means an individual could be both a **MeatTopping** and a **VegetableTopping** at the same time
  - ▶ We want to state this is not the case

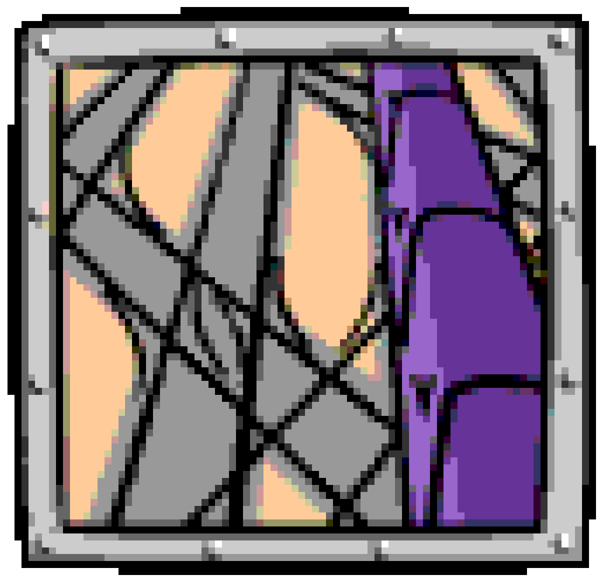
Disjointness

- **If we state that classes are disjoint**



  - ► This means an individual cannot be both a **MeatTopping** and a **VegetableTopping** at the same time
  - ► We must do this explicitly in the interface

Check consistency



Why is MeatyVegetableTopping Inconsistent?

- **We have asserted that a** MeatyVegetableTopping **is a subclass of two classes we have stated are disjoint**

- **The disjoint means nothing can be a** MeatTopping **and a** VegetableTopping **at the same time**
- **This means that** MeatyVegetableTopping **can never contain any individuals**
  - **The class is therefore inconsistent**
  - **This is what we expect!**

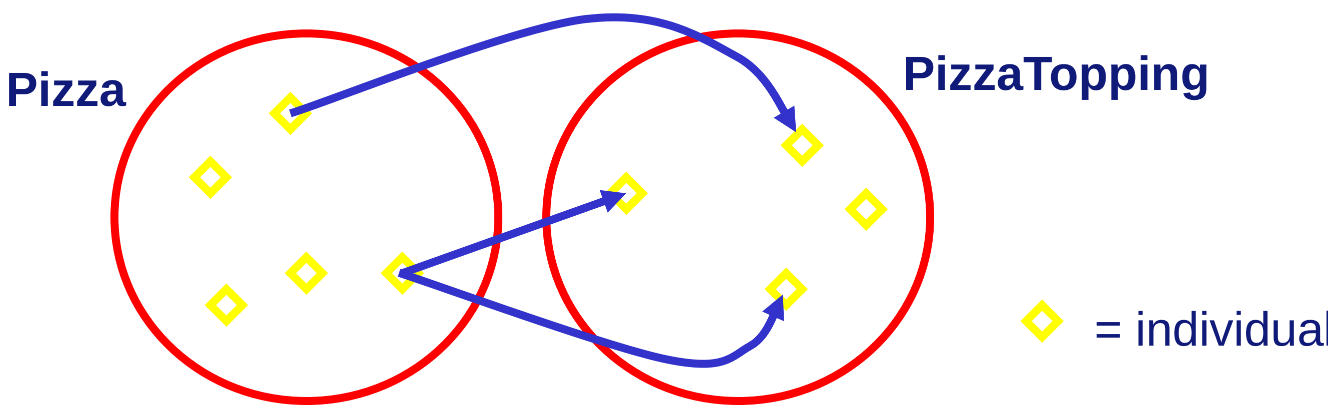
- **It can be useful to create classes we expect to be inconsistent to "test" your model – often we refer to these classes as "probes" – generally it is a good idea to document them as such to avoid later confusion**

Table of contents

- **Reasoning with OWL ontologies**
  - **Consistency checking**
    - Disjointness
    - Restrictions
  - **Primitive and Defined classes**
  - **Alternative definitions for a class (Vegetarian Pizzas: only vegetarian toppings, no meat or fish toppings or not a MeatyPizza?)**
    - Union classes and covering axioms
      - The Open World Assumption (closure)
    - Negation in OWL

What are we missing?

- **This is not a semantically rich model**
- **Apart from "is kind of" (subsumption) and "is not kind of" (disjoint), we currently don't have any other information of interest**
- **We want to say more about Pizza Individuals, such as their relationship with other Individuals**



Creating Properties. Naming conventions

- **Use camelNotation**
  - **Lowercase letter to begin**

- **Create properties using 2 standard naming patterns:**
  - **has… (eg hasColour)**
  - **is…Of (eg isTeacherOf) or other suffixes (eg …In …To)**

- **Advantages:**
  - **It is easier to find properties**
  - **It is easier for tools to generate a more readable form (see tooltips on the classes in the hierarchy later)**
  - **Inverses properties typically follow this pattern eg hasPart, isPartOf**

Class Restrictions: Associating Properties with Classes

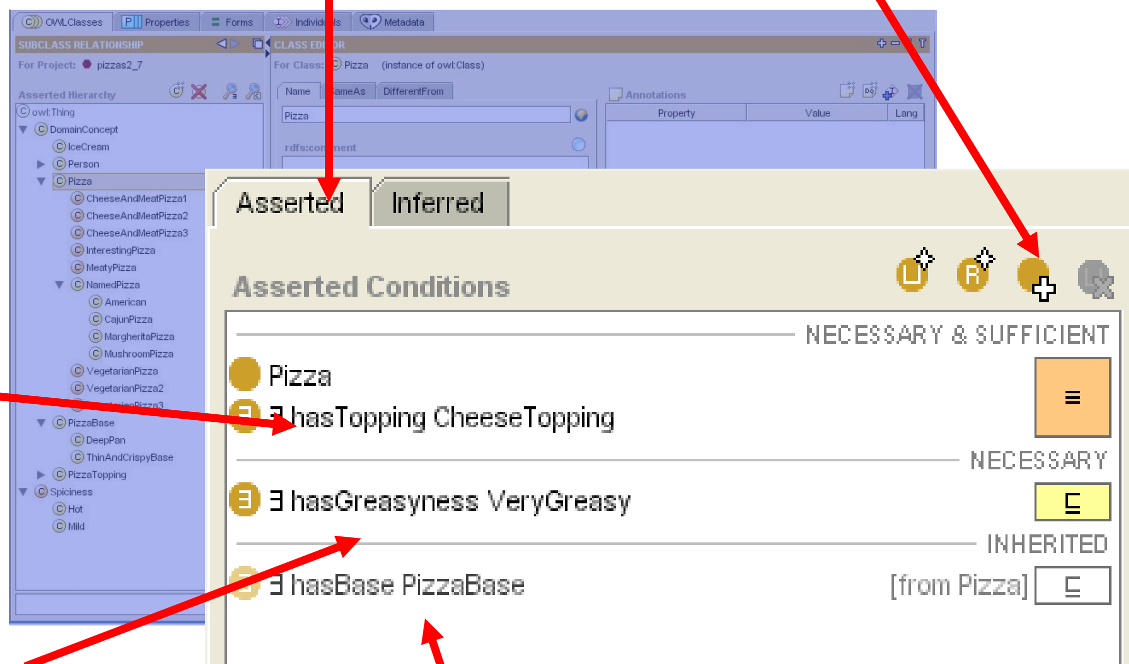
- **Property that we want to use to describe Pizza individuals**
  - **hasTopping**

- **Steps**
  - **Go back to the Pizza class and add some further information**
  - **Use the Conditions widget**
  - **Conditions can be any kind of Class**
    - Named superclasses (already added)
    - Class restrictions of type "Anonymous Class"

Conditions Widget

Conditions asserted by the ontology engineer

Add different types of condition



Definition

of the class

(later)

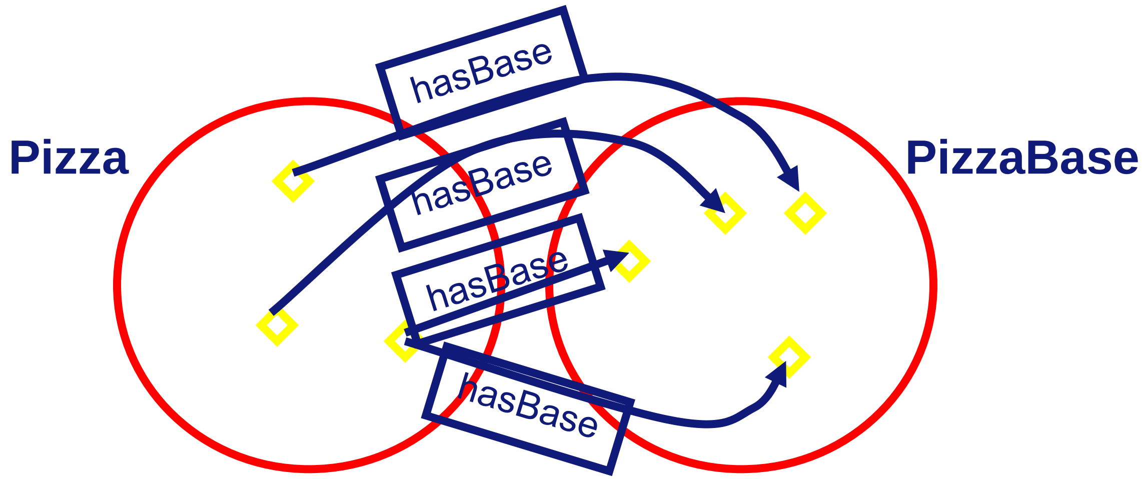
Description

of the class

Conditions inherited from superclasses

What does this mean?

- **We have created a restriction: ∃ hasBase** PizzaBase **on Class** Pizza **as a necessary condition**

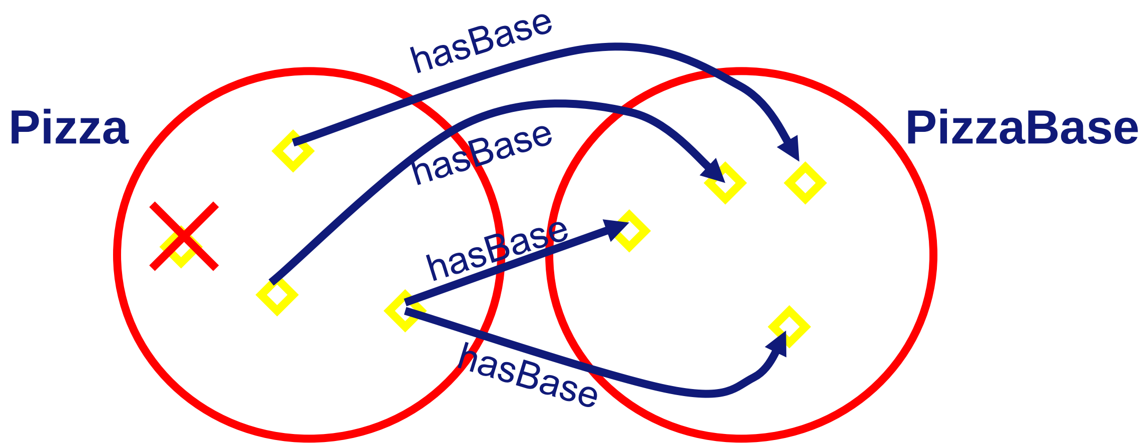


"If an individual is a member of this class, it is necessary that it has at least one hasBase relationship with an individual from the class **PizzaBase**"

"Every individual of the **Pizza** class must have at least one base from the class **PizzaBase**"

What does this mean?

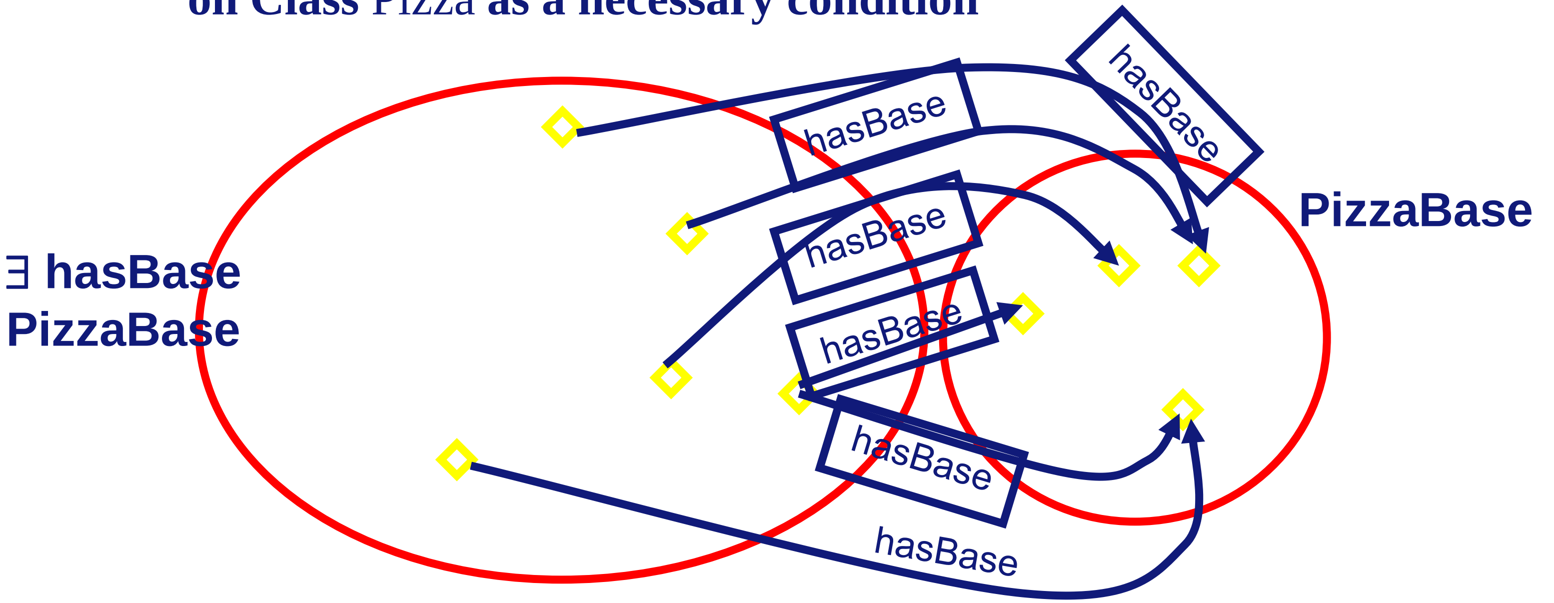
- **We have created a restriction: ∃ hasBase** PizzaBase **on Class** Pizza **as a necessary condition**



► "There can be no individual, that is a member of this class, that does not have at least one hasBase relationship with an individual from the class **PizzaBase**"

Why?

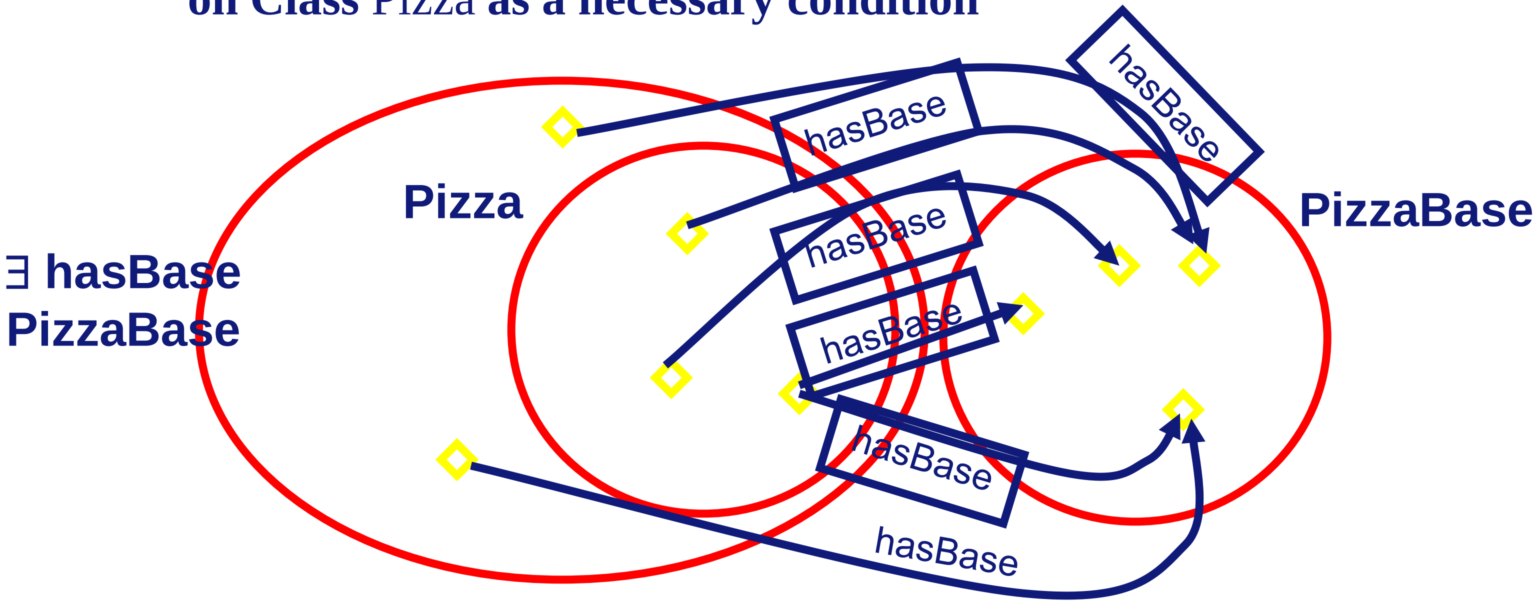
- **We have created a restriction: ∃ hasBase** PizzaBase **on Class** Pizza **as a necessary condition**



Each Restriction or Class Expression describes the set of all individuals that satisfy the condition

Why? Necessary conditions

- **We have created a restriction: ∃ hasBase** PizzaBase **on Class** Pizza **as a necessary condition**

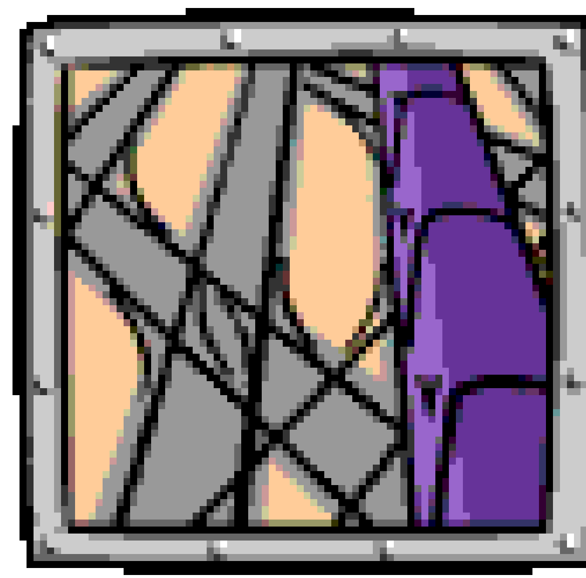


- ► Each necessary condition on a class is a superclass of that class
- ► ie The restriction ∃ hasBase **PizzaBase** is a superclass of **Pizza**
- ► As **Pizza** is a subclass of the restriction, all **Pizza**s must satisfy the restriction that they have at least one base from **PizzaBase**

Define Cheesey Pizza and Classify

Define a Cheesey Pizza, as a Pizza that has some cheese on it

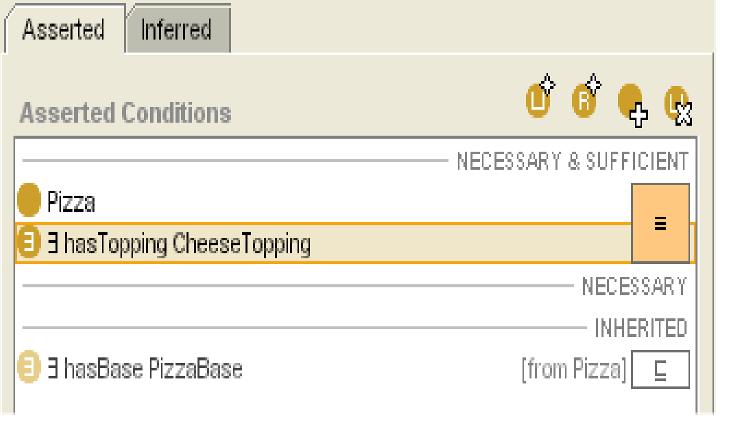
- **Usual steps**
  - Create primitive classes and then migrate them to defined classes
  - All the defined pizzas will be direct subclasses of Pizza
  - So, we create a CheesyPizza Class (do not make it disjoint) and add a restriction:
    "Every CheeseyPizza must have at least one CheeseTopping"



Use the reasoner to help us produce a polyhierarchy without having to assert multiple parents

Creating a CheeseyPizza

- Classifying shows that we currently don't have enough information to do any classification
- We then move the conditions from the Necessary block to the Necessary & Sufficient block which changes the meaning



- And classify again…

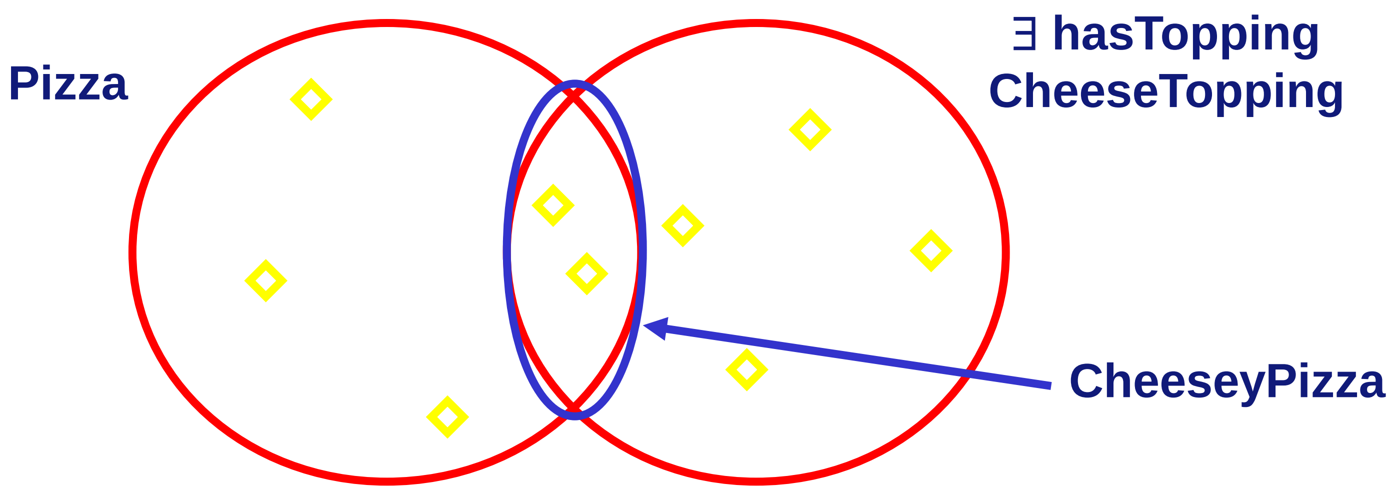
Reasoner Classification

- **The reasoner has been able to infer that anything that is a** Pizza **that has at least one topping from** CheeseTopping **is a** CheeseyPizza

The inferred hierarchy is updated to reflect this and moved classes are highlighted in blue

Why?
Necessary & Sufficient Conditions

- **Each set of necessary & sufficient conditions is an Equivalent Class**



CheeseyPizza is equivalent to the intersection of **Pizza** and **∃ hasTopping CheeseTopping**

Classes, all of whose individuals fit this definition are found to be subclasses of **CheeseyPizza**, or are subsumed by **CheeseyPizza**

Primitive Classes

- **All classes in our ontology so far are Primitive**
- **We describe primitive pizzas**
- **Primitive Class = only Necessary Conditions**
- **They are marked as plain orange circles in the class hierarchy**

We condone building a disjoint tree of primitive classes

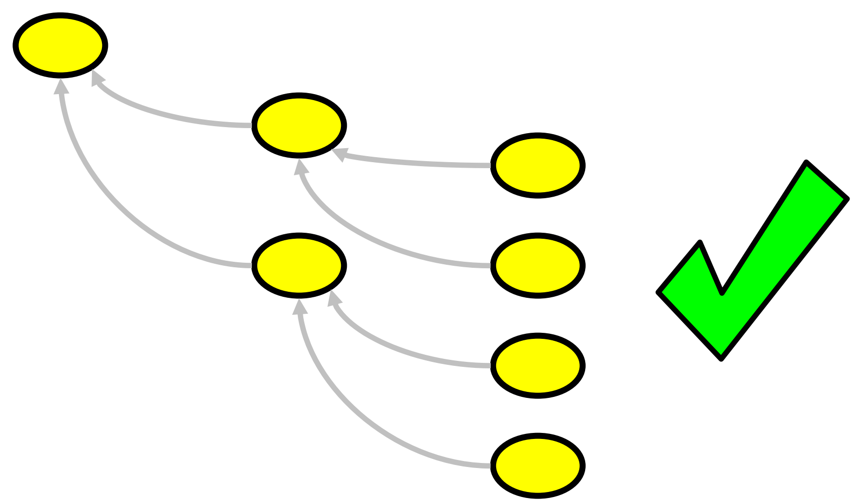


Table of contents

- **Reasoning with OWL ontologies**
  - **Consistency checking**
    - Disjointness
    - Restrictions
  - **Primitive and Defined classes**
  - **Alternative definitions for a class (Vegetarian Pizzas: only vegetarian toppings, no meat or fish toppings or not a MeatyPizza?)**
    - Union classes and covering axioms
      - The Open World Assumption (closure)
    - Negation in OWL
  - **Elephant Traps – Common modelling errors**
    - Functional properties
    - Intersection classes
    - Universal restrictions
- **Using an ontology API to deal with OWL ontologies**

Polyhierarchies

- **By the end of this tutorial we intent to create a** VegetarianPizza
- **Some of our existing Pizzas should be types of** VegetarianPizza
- **However, they could also be types of** SpicyPizza **or** CheeseyPizza

- **We need to be able to give them multiple parents in a principled way**
- **We could just assert multiple parents like we did with** MeatyVegetableTopping **(without disjoints)**

BUT…

Defined Classes

- **We've created a Defined Class,** CheeseyPizza
  - **It has a definition. That is *at least one* Necessary and Sufficient condition**
  - **Classes, all of whose individuals satisfy this definition, can be inferred to be subclasses**
  - **Therefore, we can use it like a query to "collect" subclasses that satisfy its conditions**
  - **Reasoners can be used to organise the complexity of our hierarchy**
- **It's marked with an equivalence symbol in the interface**
- **Defined classes are rarely disjoint**

Table of contents

- **Reasoning with OWL ontologies**
  - **Consistency checking**
    - Disjointness
    - Restrictions
  - **Primitive and Defined classes**
  - **Alternative definitions for a class (Vegetarian Pizzas: only vegetarian toppings, no meat or fish toppings or not a MeatyPizza?)**
    - Union classes and covering axioms
      - The Open World Assumption (closure)
    - Negation in OWL

Define a Vegetarian Pizza

- **Not as easy as it looks…**
- **Define in words?**
  - **"a pizza with only vegetarian toppings"?**
  - **"a pizza with no meat (or fish) toppings"?**
  - **"a pizza that is not a MeatyPizza"?**
- **More than one way to model this**

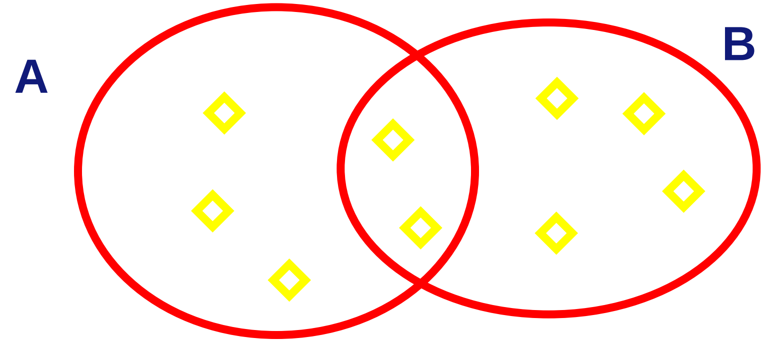
We'll start with the first example

Vegetarian Pizza = Pizza with only vegetarian toppings

- **Requirements**
  - Create a vegetarian topping → Union Class (aka disjunction)
  - "Only" → Universal Restriction

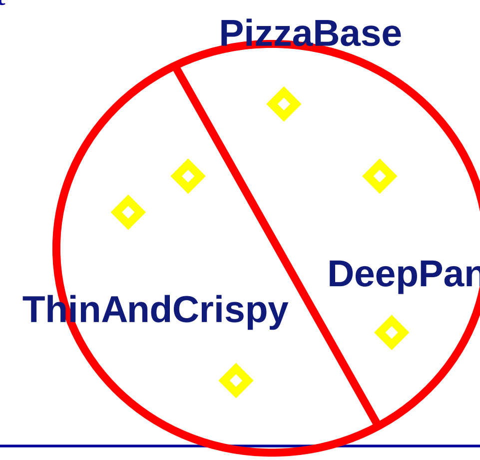
Vegetarian Topping: Union Classes and Covering Axioms

- A U B includes
  all individuals of class A and
  all individuals from class B and
  all individuals in the overlap
  (if A and B are not disjoint)



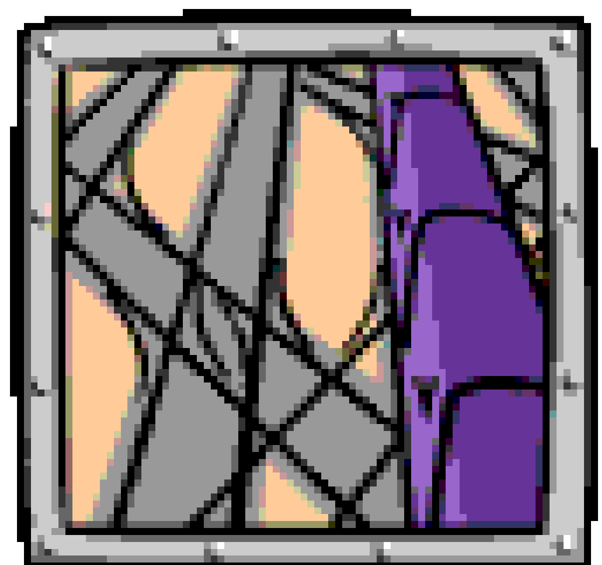
- Covering axiom
  - Union expression containing several covering classes
  - A covering axiom in the Necessary & Sufficient Conditions of a class means:
    the class cannot contain any instances other than those from the covering classes
  - Note: If the covering classes are subclasses of the covered class, the covering axiom only needs to be a Necessary condition
    - It doesn't harm to make it Necessary & Sufficient though – its just redundant

- Example: PizzaBase ≡ ThinAndCrispy U DeepPan
  - The class PizzaBase is covered by ThinAndCrispy or DeepPan
  - All PizzaBases must be ThinAndCrispy or DeepPan
  - "There are no other types of PizzaBase"



Define Vegetarian Pizza and Classify

Define a Vegetarian topping and define Vegetarian Pizza



VegetarianPizza Classification

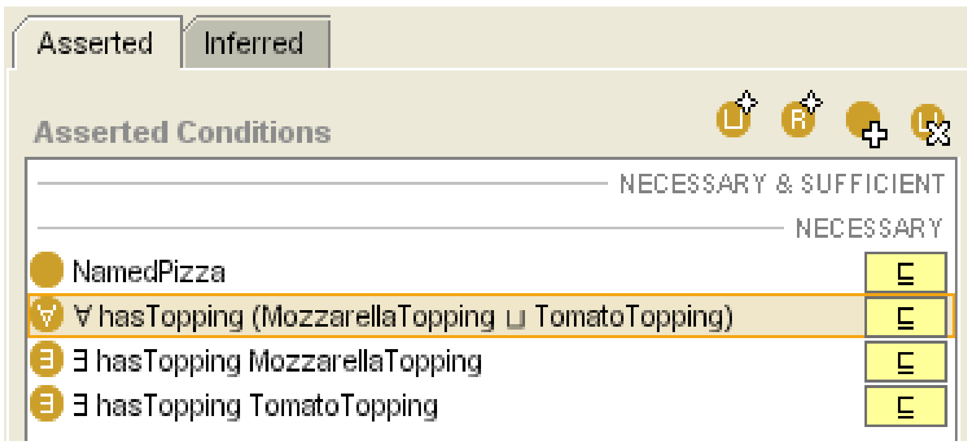
- **Nothing classifies under VegetarianPizza**
  - **Actually, there is nothing wrong with our definition of VegetarianPizza**
  - **It is actually the descriptions of our Pizzas that are incomplete**
- **The reasoner has not got enough information to infer that any Pizza is subsumed by VegetarianPizza**
- **This is because OWL makes the Open World Assumption**
  - **In a closed world (like DBs), the information we have is everything**
    - A database, for example, returns a negative if it cannot find some data.
  - **In an open world, we assume there is always more information than is stated**
    - The reasoner makes no assumption about the completeness of the information it is given
    - The reasoner cannot determine something does not hold unless it is explicitly stated in the model

Open World Assumption

- **Typical pattern**
  - **Several existential restrictions on a single property with different fillers**
    - Example: primitive pizzas on hasTopping
- **Must state whether a description is complete or not**
  - **Incomplete:**
    - Existential restrictions should be paraphrased by “amongst other things…”
  - **Complete:**
    - Existential restrictions should be paraphrased by “and no other XXX”
- **In our example:**
  - **We need closure for the property hasToppings**
    - In the form of a Universal Restriction with a filler that is the Union of the other fillers for that property
    - Closure works along a single property

Closure example: MargheritaPizza

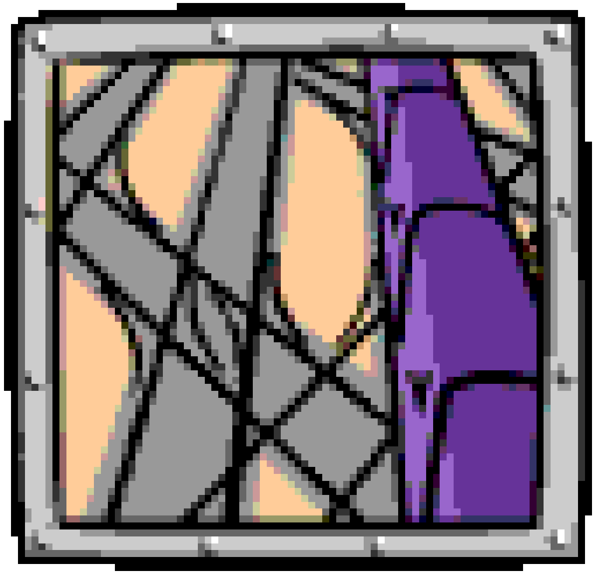
- **All MargheritaPizzas must have:**
  at least 1 topping from MozzarellaTopping and
  at least 1 topping from TomatoTopping and
  only toppings from MozzarellaTopping or TomatoTopping



- **The last part is paraphrased into “no other toppings”**
- **The union closes the hasTopping property on MargheritaPizza**

Define Margherita Pizza and Classify

Define a Margherita pizza



Ontology tools

Oscar Corcho
ocorcho@fi.upm.es
http://www.oeg-upm.net/

Ontological Engineering Group
Laboratorio de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo sn,
28660 Boadilla del Monte, Madrid, Spain