Building OWL Ontologies with Protégé (2)

CS 431 – April 14, 2008

Carl Lagoze – Cornell University

MANCHESTER
1824

The University of Manchester

Parts extracted with permission from:

A Practical Introduction to Ontologies & OWL

Session 1: Primitive Classes in OWL

Nick Drummond & Matthew Horridge

MANCHESTER 1824

The University of Manchester

Parts extracted with permission from:

A Practical Introduction to Ontologies & OWL

Session 2: Defined Classes and Additional Modelling Constructs in OWL

Nick Drummond & Matthew Horridge

CO-ODE cooperative ontologies programme

protégé

AKT

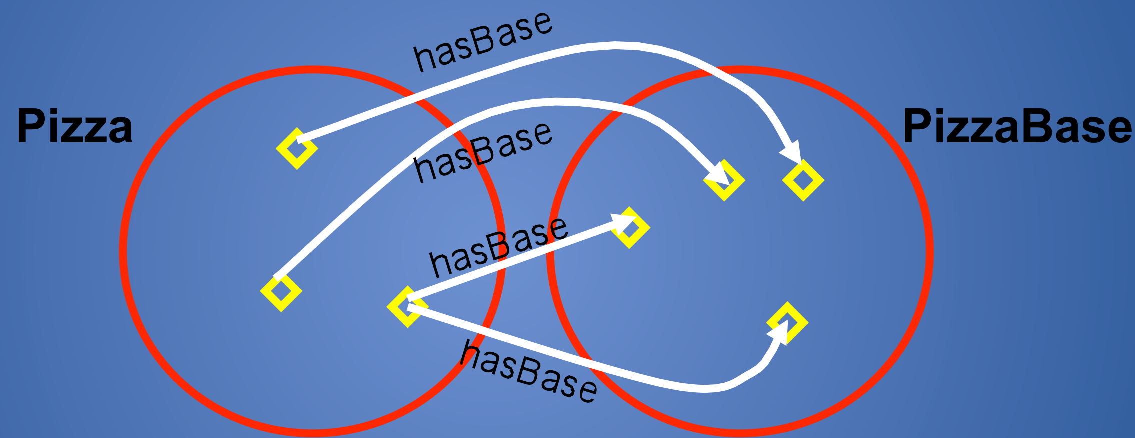
EPISTEMICS LTD

OpenGALEN

Bio Health Informatics Group

Restrictions

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

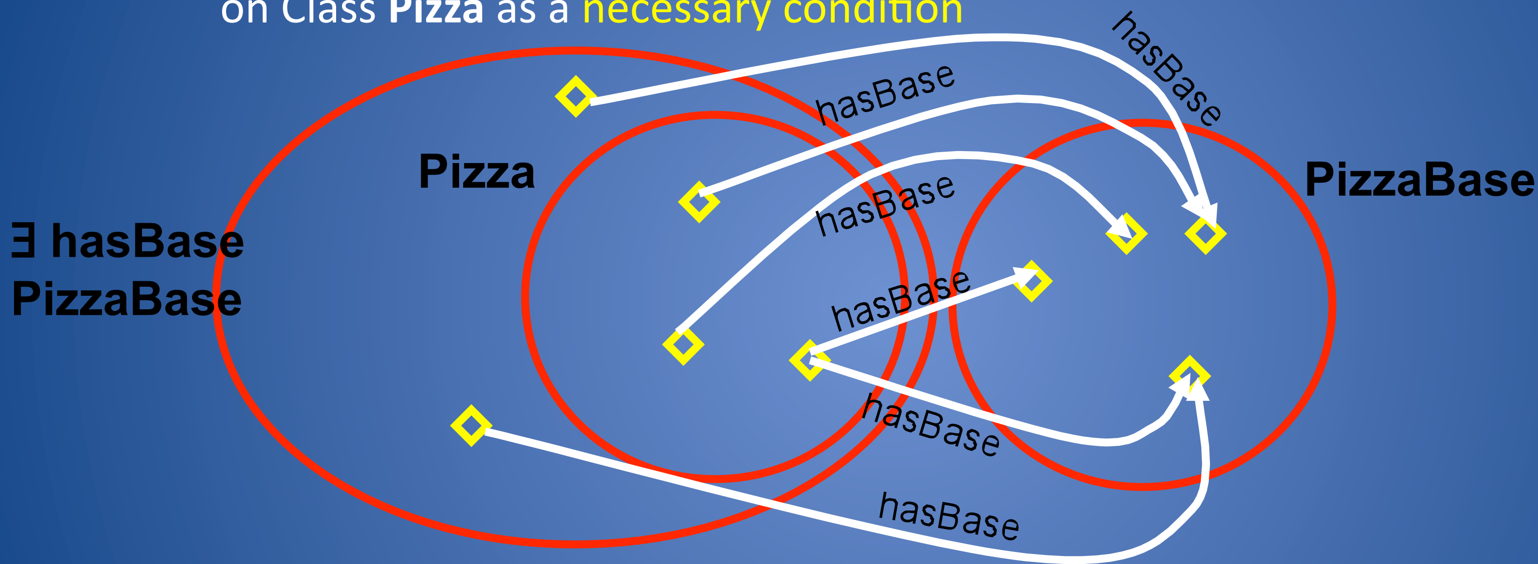


► "If an individual is a member of this class, it is necessary that it has at least one hasBase relationship with an individual from the class **PizzaBase**"

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

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**

Consistency Checking

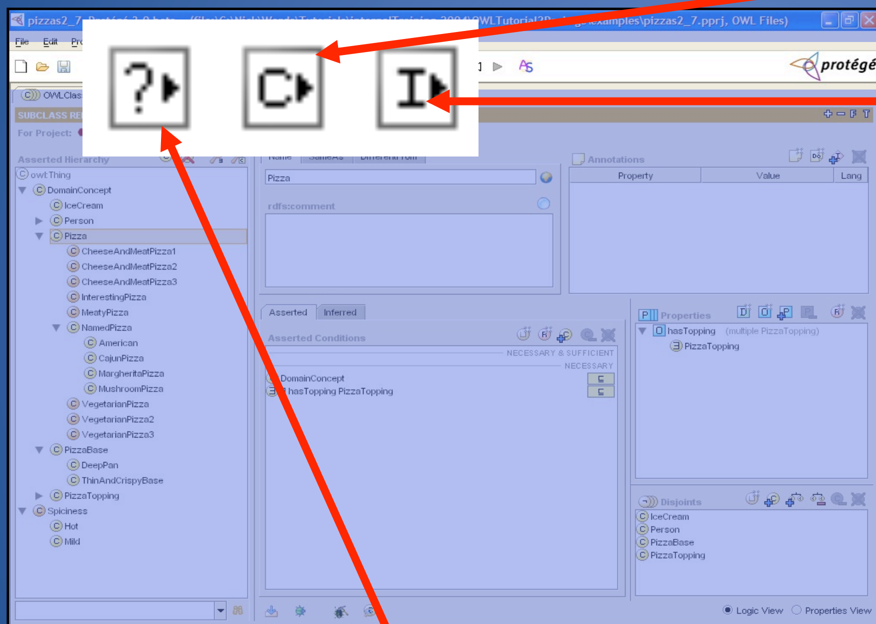
- Create a class that doesn't really make sense
  - What is a MeatyVegetableTopping?
- We'd like to be able to check the logical consistency of our model
- This is one of the tasks that can be done automatically by software known as a **Reasoner**
- Being able to use a reasoner is one of the main advantages of using a logic-based formalism such as OWL (and why we are using OWL-DL)

- We will use Pellet (server-based DIG reasoner)

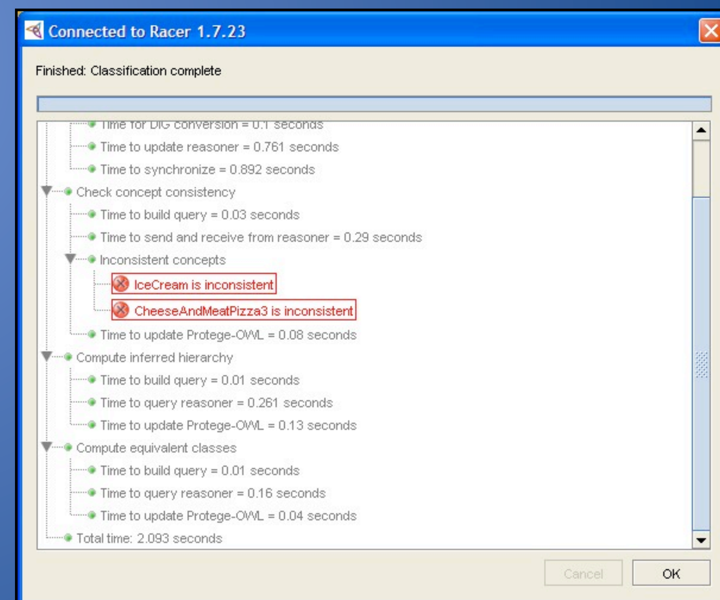
Accessing the Reasoner

Classify taxonomy
(and check consistency)

Compute inferred types
(for individuals)



Just check consistency
(for efficiency)



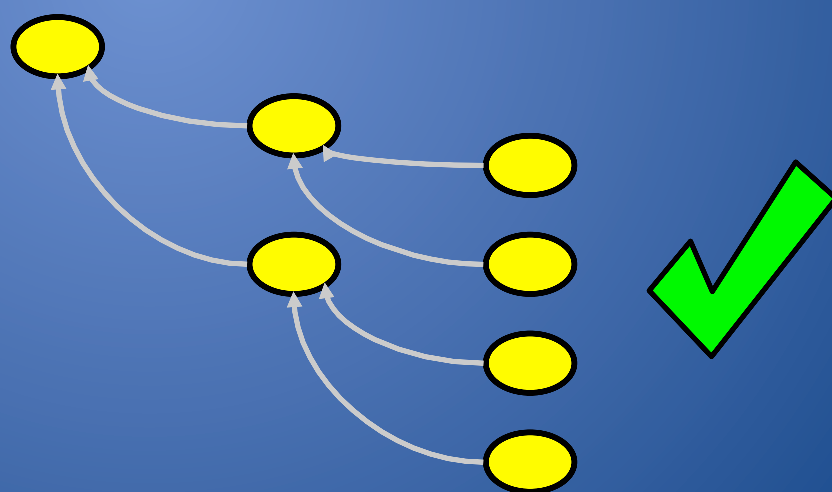
Reasoning about our Pizzas

- *When we classify an ontology we could just use the "Check Consistency" button but we'll get into the habit of doing a full classification as we'll be doing this later*
- *The reasoner dialog will pop up while the reasoner works*
- *When the reasoner has finished, you will see an inferred hierarchy appear, which will show any movement of classes in the hierarchy*
- *If the reasoner has inferred anything about our model, this is reported in the reasoner dialog and in a separate results window*
- *inconsistent classes turn red*
- *moved classes turn blue*

Primitive Classes

- Primitive Class = only Necessary Conditions
- Can not yet judge an individual based on primitive classes – why?

Start with building a disjoint tree of primitive classes

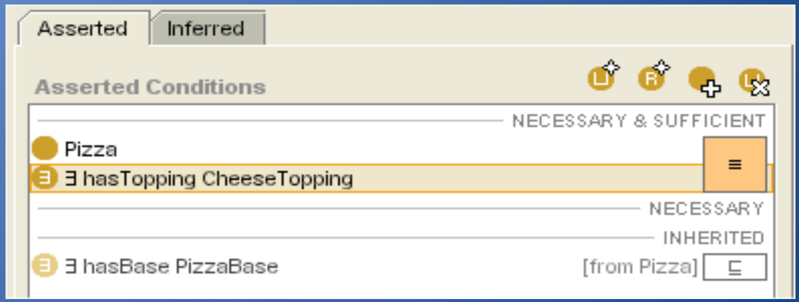


Defined Classes

- We want to be able to definitively type some thing
  - E.g., “I know it’s a Cheesy Pizza because it has cheese on it”
    - Note that this is different from “A Cheesy Pizza must have cheese on it”

Creating a CheeseyPizza

- So, we create a CheesyPizza Class (do not make it disjoint) and add a restriction:
  "Every **CheeseyPizza** must have at least one **CheeseTopping**"
- 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

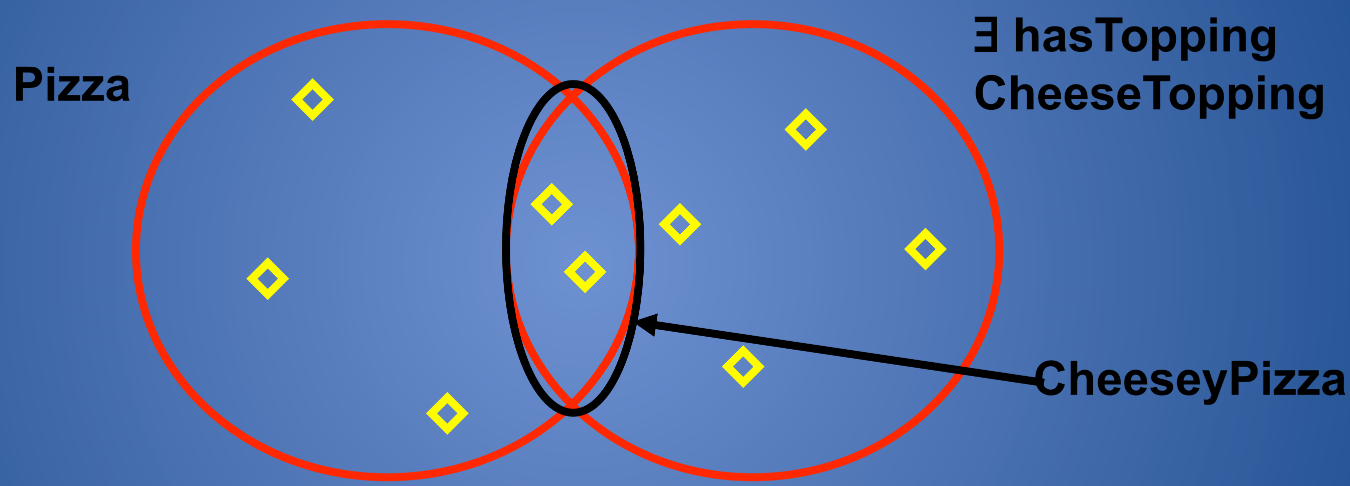
- The reasoner has been able to infer that anything that is a **Pizza** that has at least one topping from **CheeseTopping** is a **CheeseyPizza**

► 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**

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

Polyhierarchies

- Note that just because a Pizza is a CheesyPizza it can be another type of Pizza
- **Take a look at InterestingPizza**

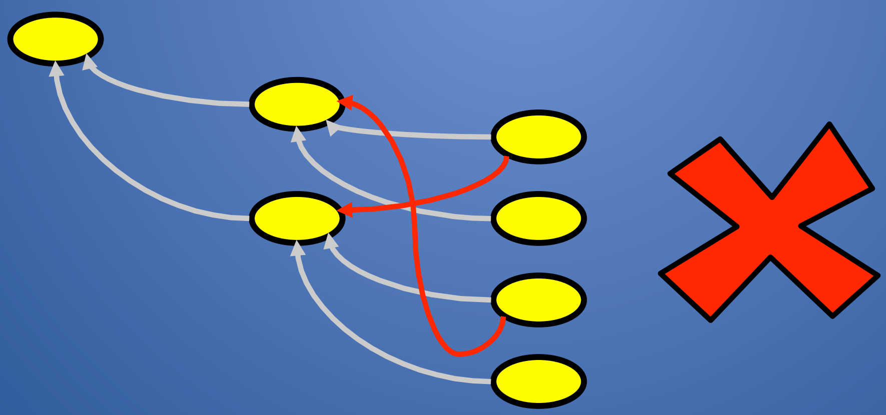
- We need to be able to give them multiple parents in a principled way
- We could just assert multiple parents

BUT…

Asserted Polyhierarchies

In most cases asserting polyhierarchies is bad

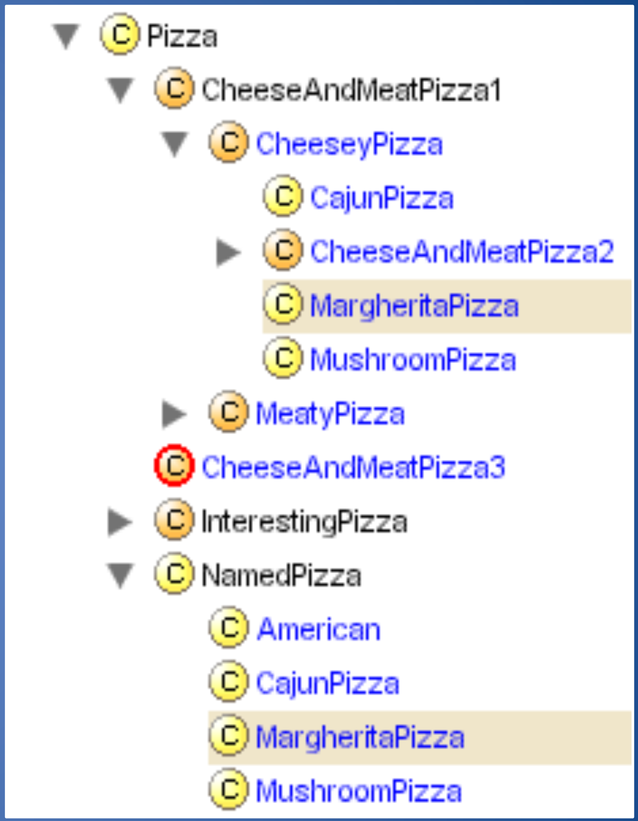
- We lose some encapsulation of knowledge
  - Why is this class a subclass of that one?
- Difficult to maintain
  - Adding new classes becomes difficult because all subclasses may need to be updated
  - Extracting from a graph is harder than from a tree



let the reasoner do it!

Untangling

- We can see that certain Pizzas are now classified under multiple parents
- **MargheritaPizza** can be found under both **NamedPizza** and **CheeseyPizza** in the inferred hierarchy



Mission Successful!

Untangling

- However, our unclassified version of the ontology is a simple tree, which is much easier to maintain
- We've now got a polyhierarchy without asserting multiple superclass relationships
- Plus, we also know why certain pizzas have been classified as CheeseyPizzas

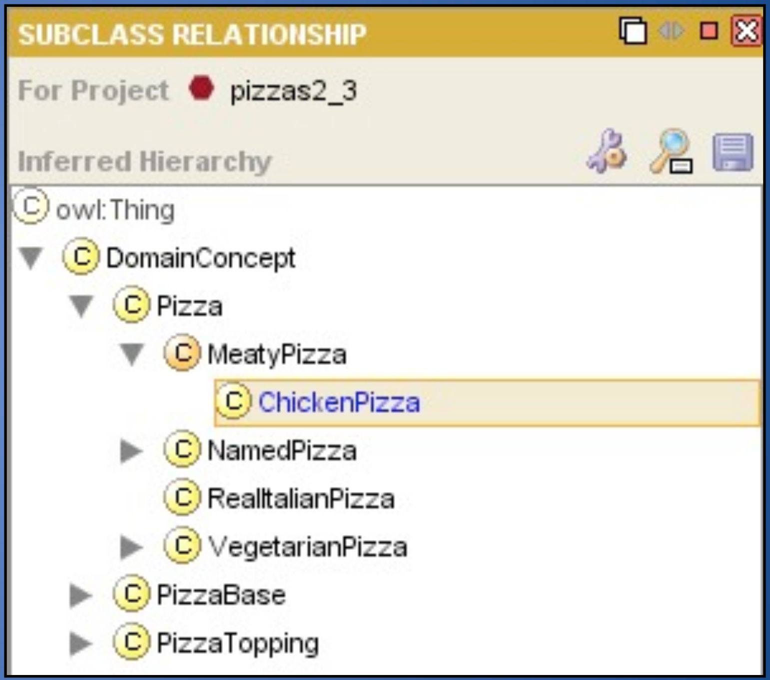
Untangling

- We don't currently have many kinds of primitive pizza but its easy to see that if we had, it would have been a substantial task to assert **CheeseyPizza** as a parent of lots, if not all, of them

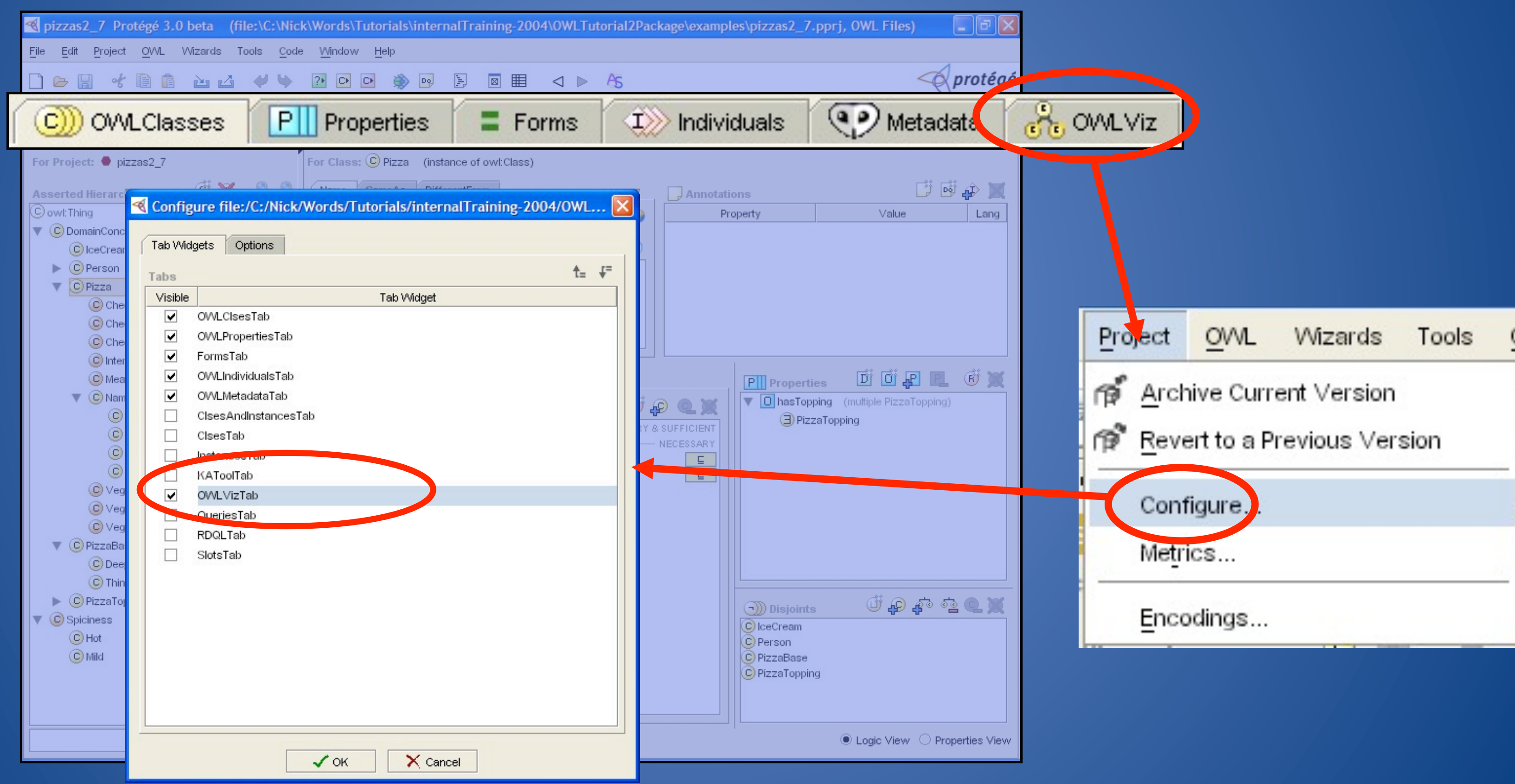
- And then do it all over again for other defined classes like **MeatyPizza** or whatever

Viewing polyhierarchies

- As we now have multiple inheritance, the tree view is less than helpful in viewing our "hierarchy"



Viewing our Hierarchy Graphically

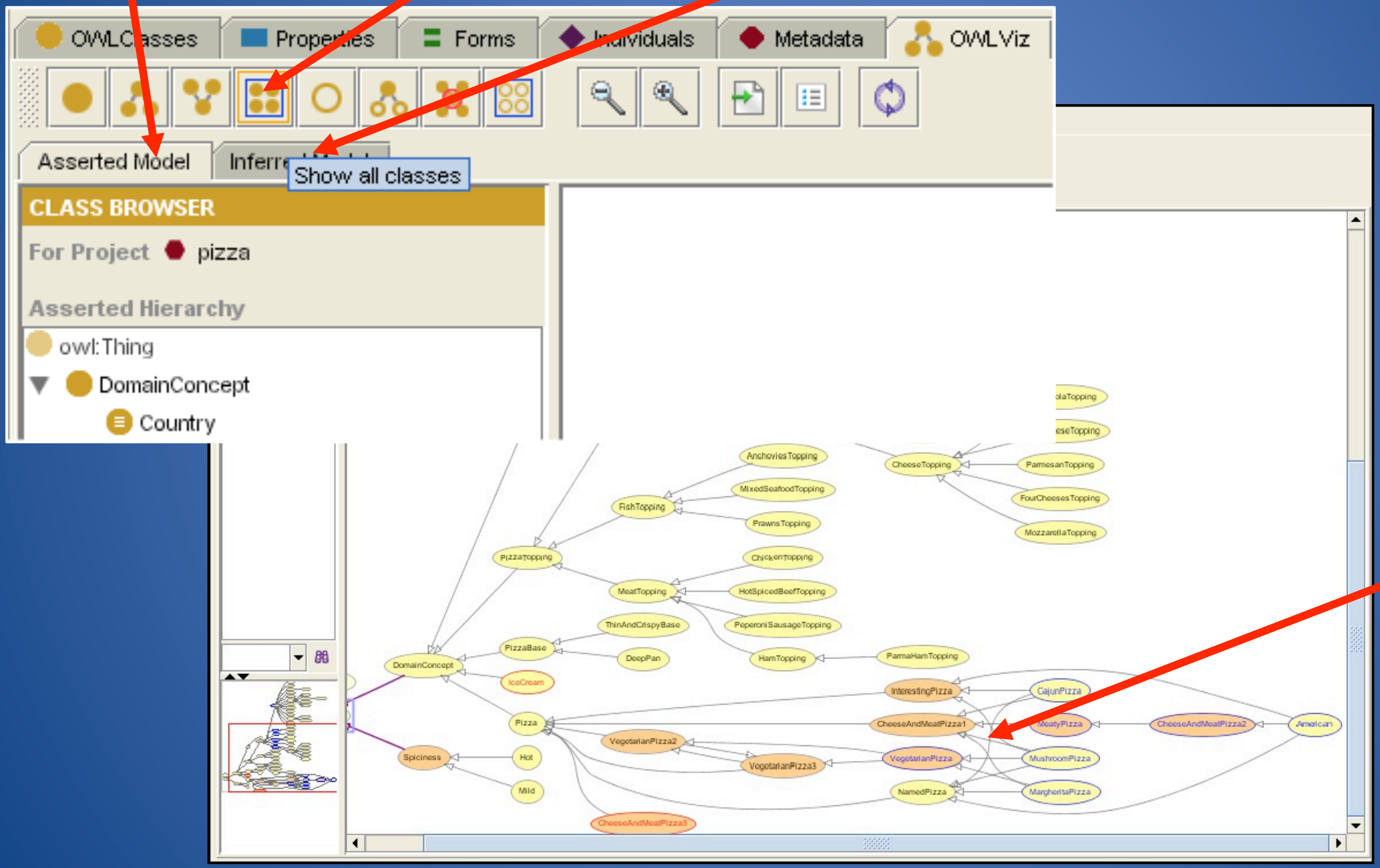


OWLViz Tab

Show All Classes

View Asserted Model

View Inferred Model



Polyhierarchy tangle

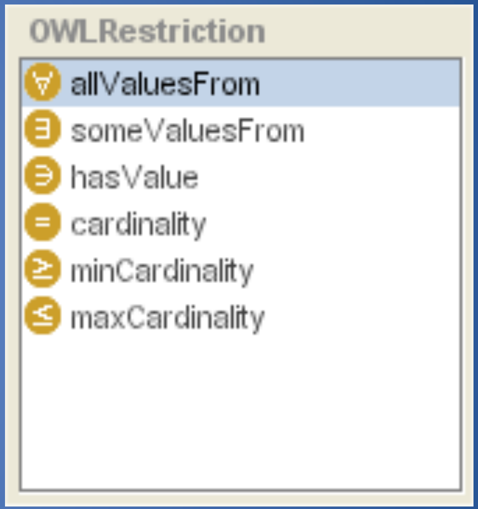
Using OWLViz to untangle

- The asserted hierarchy should, ideally, be a tidy tree of disjoint primitives
- The inferred hierarchy will be tangled
- By switching from the asserted to the inferred hierarchy, it is easy to see the changes made by the reasoner
- OWLViz can be used to spot tangles in the primitive tree and also disjoints (including inherited ones) are marked (with a ¬)

Universal Restriction

Universal Restrictions

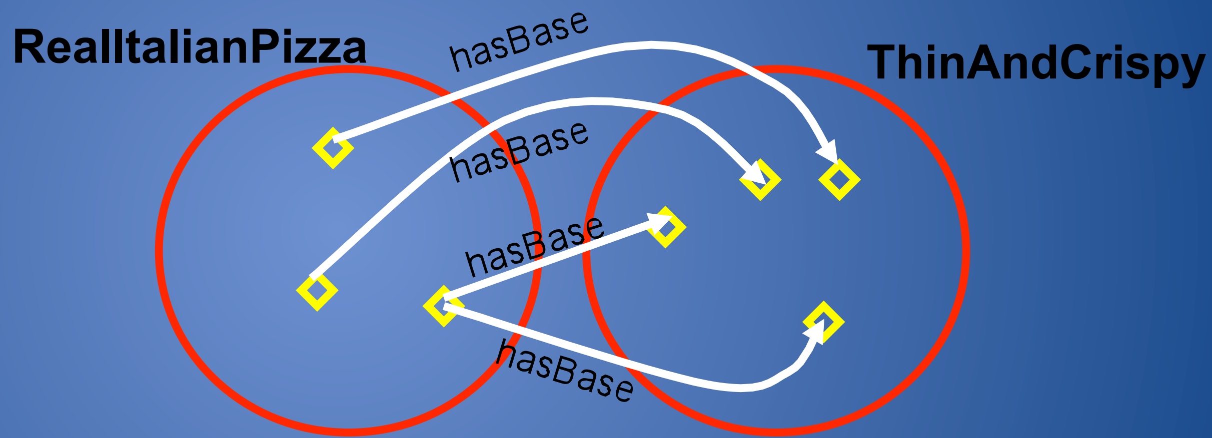
- "RealItalianPizzas only have bases that are ThinAndCrispy"



- A Universal Restriction is added just like an Existential one, but the restriction type is different

What does this mean?

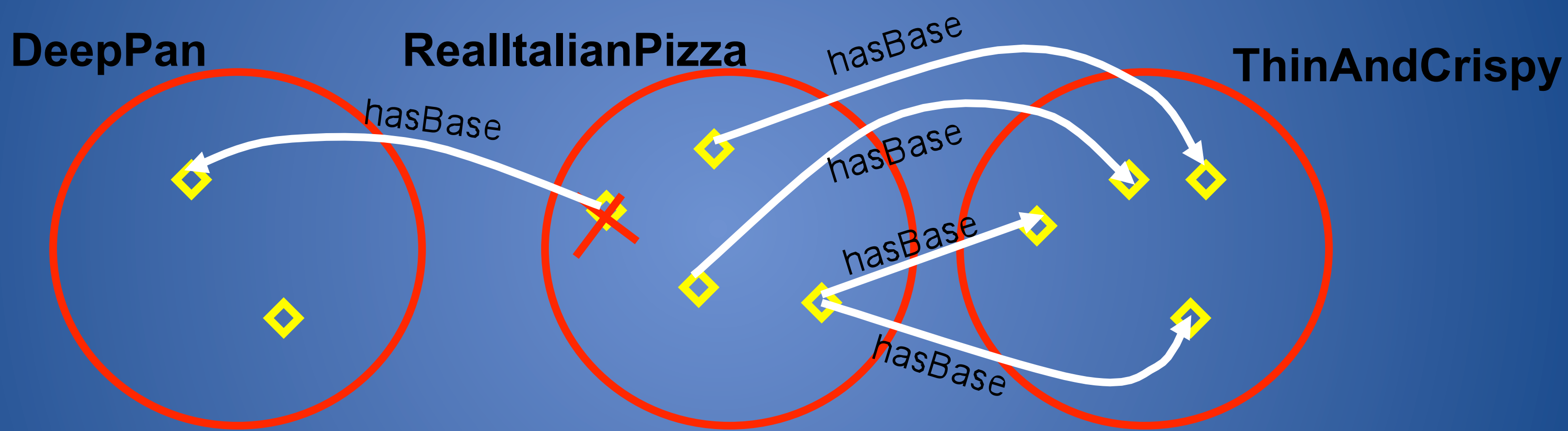
- We have created a restriction: ∀ hasBase **ThinAndCrispy** on Class **RealItalianPizza** as a necessary condition



- "If an individual is a member of this class, it is necessary that it must only have a hasBase relationship with an individual from the class **ThinAndCrispy**"

What does this mean?

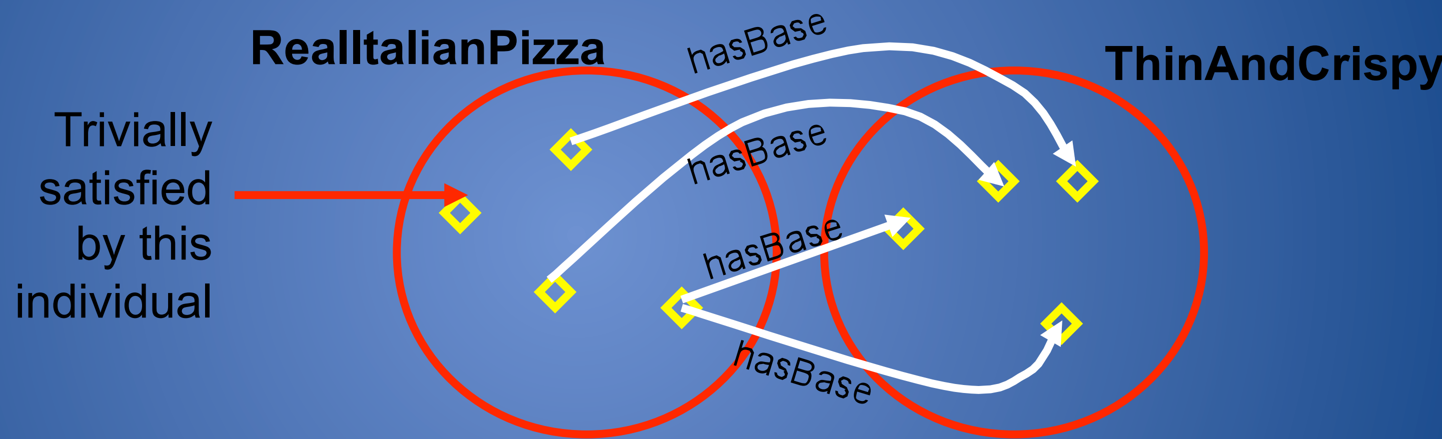
- We have created a restriction: ∀ hasBase **ThinAndCrispy** on Class **RealItalianPizza** as a necessary condition



- "No individual of the **RealItalianPizza** class can have a base from a class other than **ThinAndCrispy**"
- NB. DeepPan and ThinAndCrispy are disjoint

Warning: Trivial Satisfaction

- If we had not already inherited: ∃ hasBase **PizzaBase** from Class **Pizza** the following could hold



- "If an individual is a member of this class, it is necessary that it must only have a hasBase relationship with an individual from the class **ThinAndCrispy,** or no hasBase relationship at all"
- Universal Restrictions by themselves do not state "at least one"

Extending universal restrictions with union classes and covering axioms

Define a Vegetarian Pizza

To be able to define a vegetarian pizza as
a Pizza with only Vegetarian Toppings

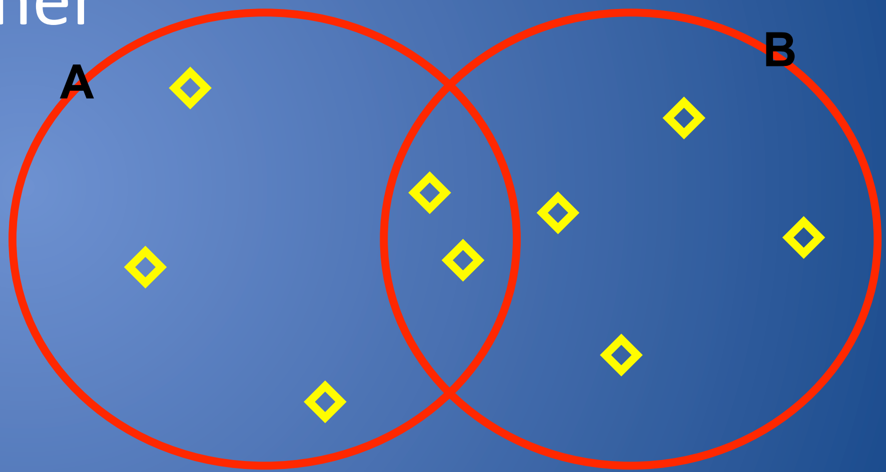
we need:

1. To be able to say "only"
   This requires a Universal Restriction
2. To be able to create a vegetarian topping
   This requires a Union Class

Union Classes

- aka "disjunction"
- This OR That OR TheOther
- This ⊔ That⊔ TheOther

A⊔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)

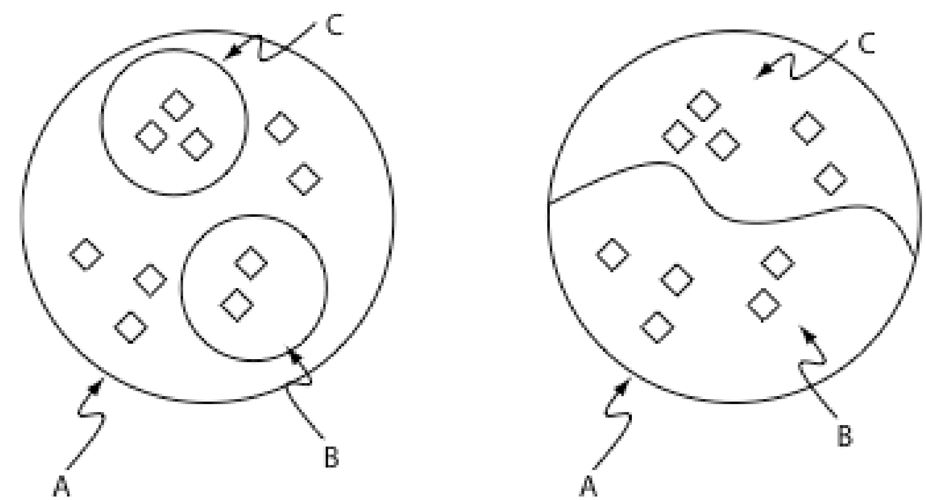


- Commonly used for:
  - Covering axioms
  - Closure

Covering Axioms

- Covering axiom – a 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



Without a covering axiom
(B and C are subclasses of A)

With a covering axiom
(B and C are subclasses of A
and A is a subclass of B union C)

VegetarianPizza Classification

- How come a Margherita pizza is not classified under **VegetarianPizza**

- Actually, there is nothing wrong with our definition of **VegetarianPizza**
- It is actually the description of Margherita that is incomplete

- The reasoner has not got enough information to infer that Margherita is subsumed by **VegetarianPizza. Why?**

- This is because OWL makes the Open World Assumption

Open World Assumption

- In a closed world (like DBs), the information we have is everything
- In an open world, we assume there is always more information than is stated

- Where a database, for example, returns a negative if it cannot find some data, 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

- Typically we have a pattern of several Existential restrictions on a single property with different fillers – like primitive pizzas on hasTopping
- Existential restrictions should be paraphrased by "amongst other things…"
- Must state that a description is complete
- We need closure for the given property

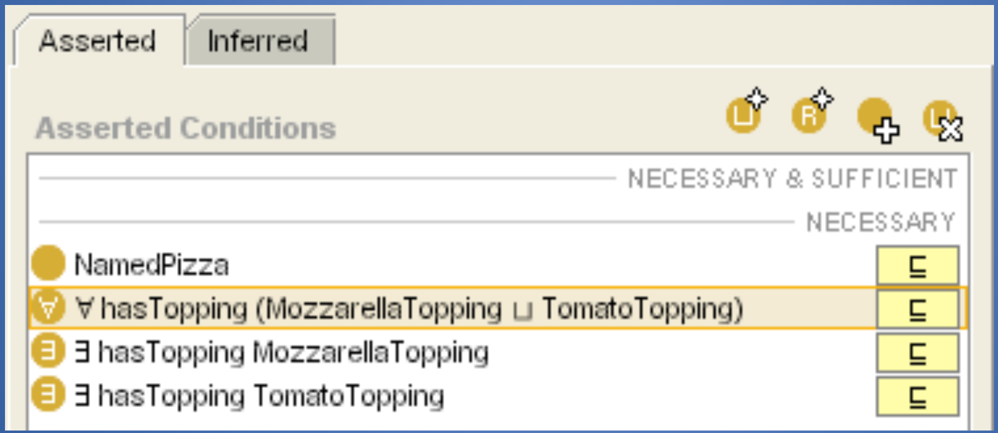
Closing the Open World
Closure

- This is in the form of a Universal Restriction with a filler that is the Union of the other fillers for that 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**

Cardinality Constraints
Interesting Pizza