

Advanced Reasoning With OWL

Kaustubh Supekar, Olivier Dameron
SMI - Stanford University

Credits:

- Matthew Horridge, Holger Knublauch et al.
A Practical guide to building OWL ontologies using the Protégé-OWL plugin and CO-ODE tools
- Natasha Noy, Alan Rector
W3C "Semantic Web Best Practice" Working Group

Warning:

- This file is a lightweight version of its pdf counterpart, that contains additional screenshots before and after each step. Please use the pdf file if you want to go through this tutorial on your own.
- This tutorial covers (far) more than we would have time to do during the allocated timeslot. It is intended to do so, as it is easier to adjust to the audience needs during the "live" session, while providing a comprehensive and autonomous set of information.

Reasoning : how to

- Run the reasoner as a separate process
 - How to call a remote reasoner ?
- Configure Protégé (if you have a weird config.)
- Use the “classify” button
 - Classes
 - Instances

RacerPro

<http://www.racer-systems.com/index.html>

Outline

- OWL semantics
- Reasoning
- Good practice
- Tips and tricks

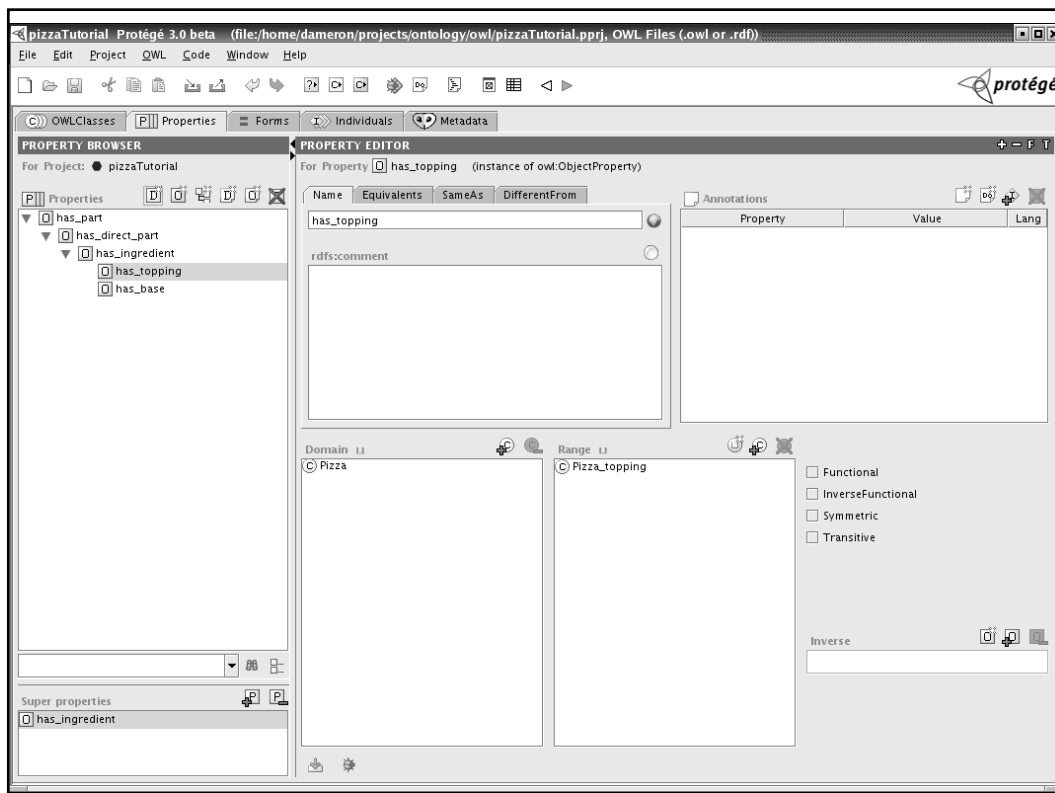
OWL Semantics *(the theoretical part)*

Individuals

- Atoms
- Unique Name Assumption
 - Not in OWL
 - in Protégé
 - in reasoners (Racer...)

Properties

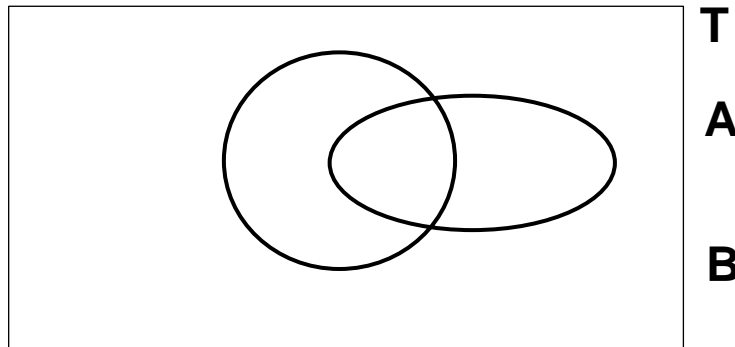
- Binary relationships on individuals
- ~ Slots
- Domain, Range
 - Used as axioms (e.g. *hasTopping* and ice creams)
- Subproperties
- Characteristics
 - Transitive: e.g. *hasPart*, *hasAncestor*...
 - Functional: e.g. *hasSSN*, *hasMother*...
 - Symmetric: e.g. *isSiblingOf*...



Classes

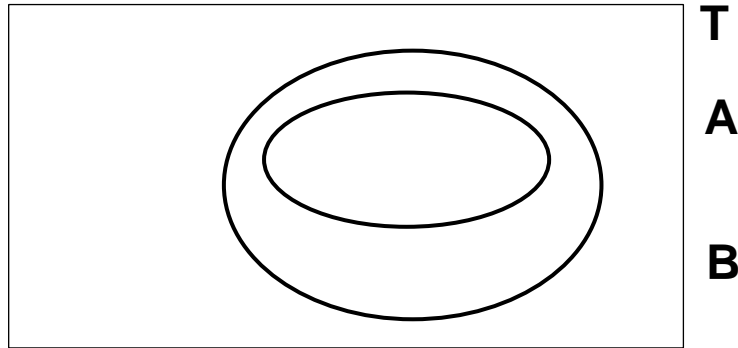
- Sets of individuals
 - Special classes:
 - top (T) = owl:Thing i.e. set of all the individuals
 - bottom (\perp) = empty set
 - Can be combined using set operators
 - subset (subsumption)
 - disjoint sets
 - union
 - intersection
 - complement

Classes: Disjointness



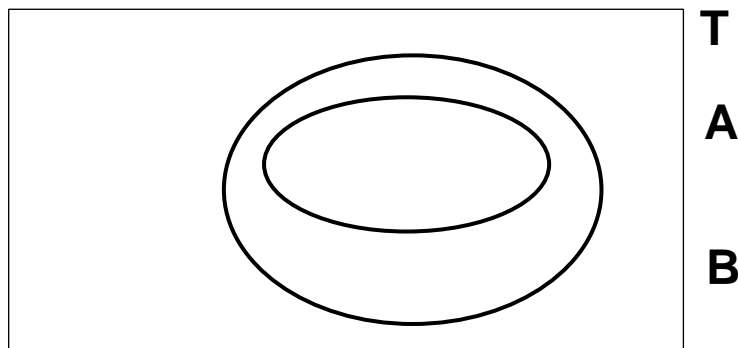
By default, any individual MAY be an instance of any classes => partial overlap of classes is assumed

Classes: subsumption



$A \sqsubset B$: all the instances of A
are instances of B

Classes: subsumption



- It is necessary to be a B in order to be a A
- It is sufficient to be a A in order to be a B

Classes

- Cumulative approach: combine classes
 - using set operators (union, intersection, complt)
 - express constraints
 - define complex concepts
- Intensional approach: describe the characteristics of a class and the system will automatically:
 - recognize that an individual is an instance of it
 - recognize that it is a subclass or a superclass of another class

Combining Concepts

Prerequisite

- Pizza
 - Vegetarian_pizza
 - Spicy_pizza
 - Named_pizza

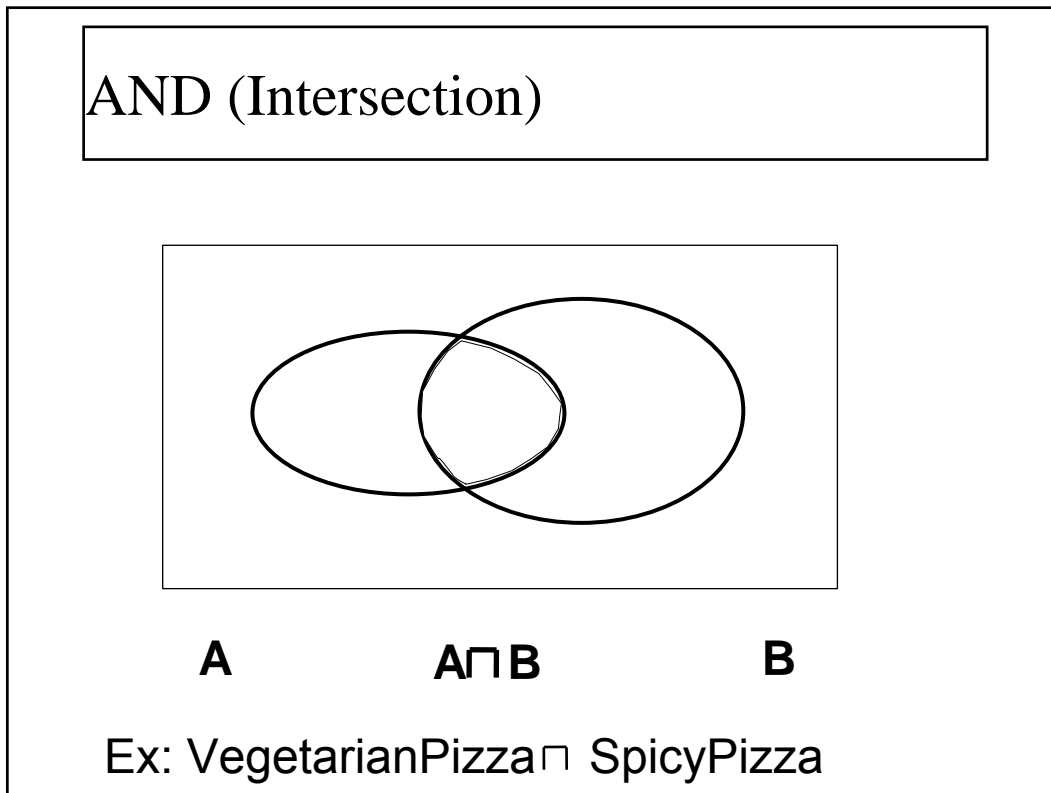
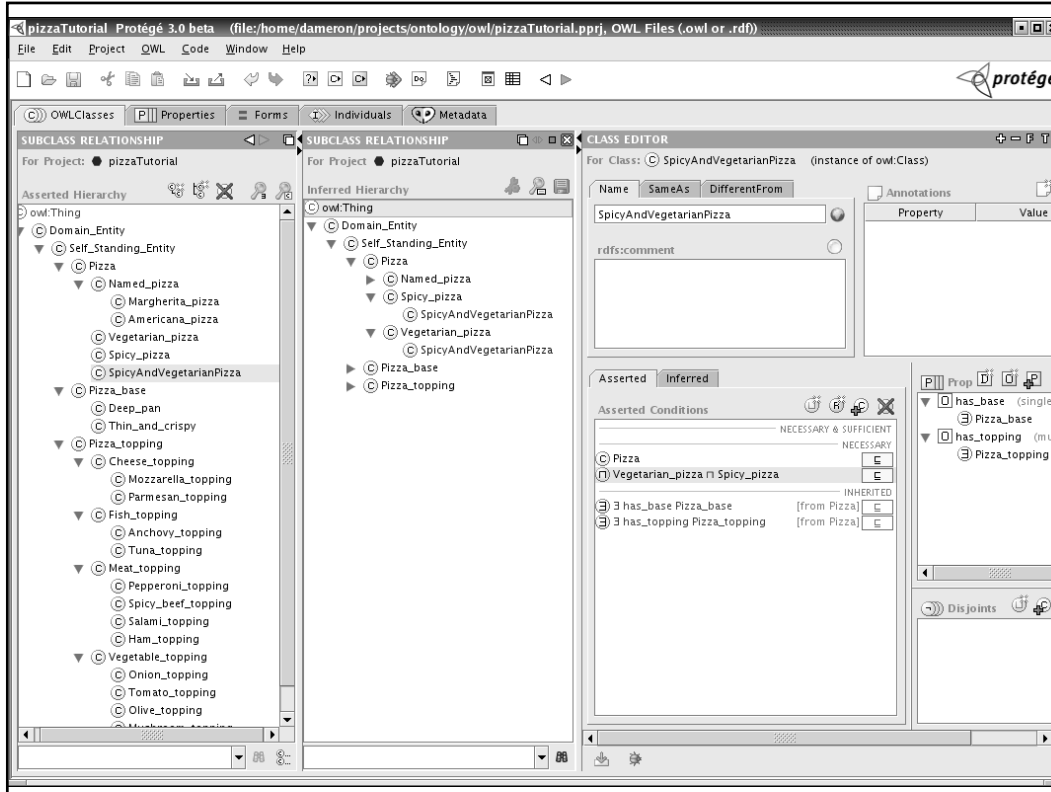
- Margherita_pizza
- Americana_pizza
- Jalapeno_pizza

DISJOINTS

Don't worry about the toppings, this is the next step!

AND (Intersection)

- Create SpicyAndVegetarian_pizza as a subclass of Pizza
- Add the necessary condition:
Vegetarian_pizza \cap Spicy_pizza
- Classify

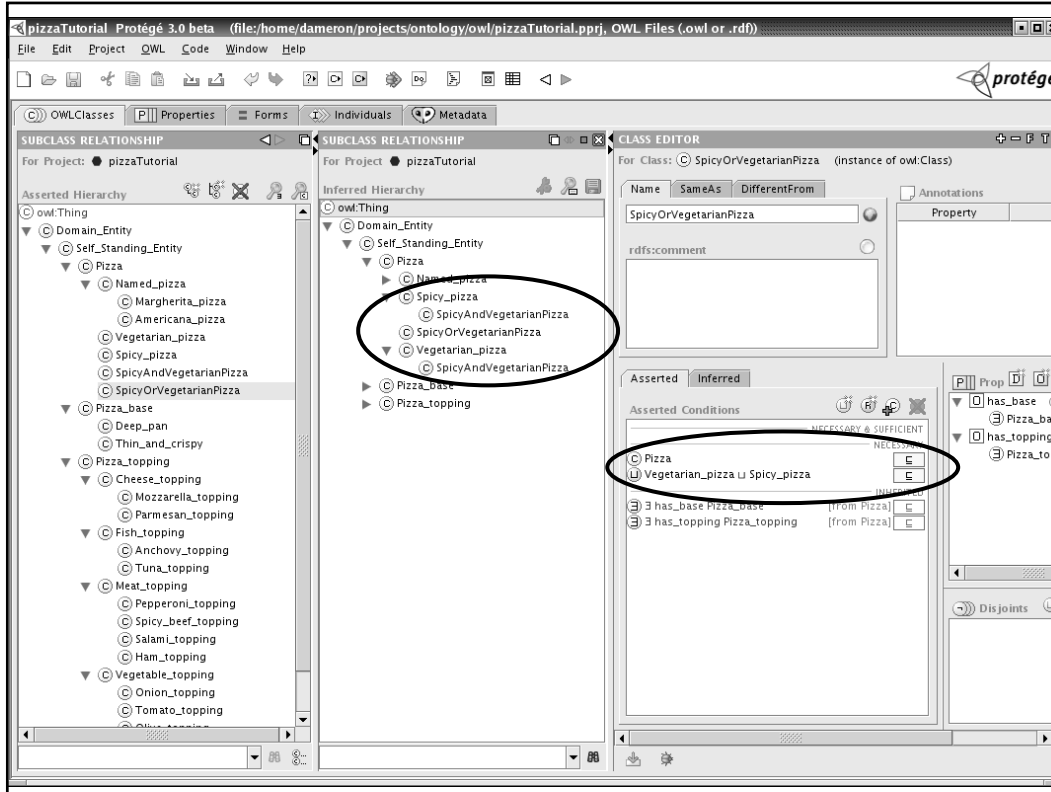


OR (Union)

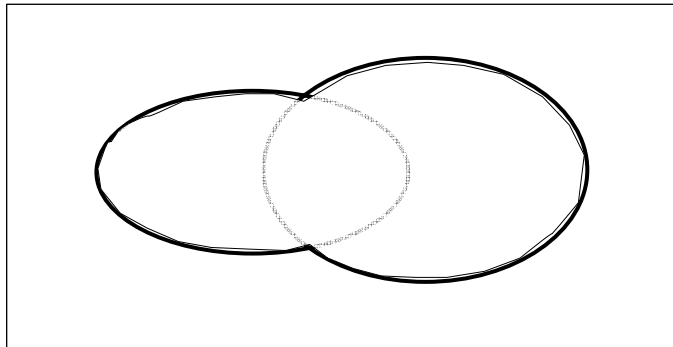
1. Create SpicyOrVegetarian_pizza as a subclass of Pizza
2. Add the necessary condition:
Vegetarian_pizza \sqcup Spicy_pizza
3. Classify :-()

The screenshot displays the Protégé 3.0 beta interface for editing an ontology. The main window is titled 'pizzaTutorial Protégé 3.0 beta (file:/home/dameron/projects/ontology/owl/pizzaTutorial.pprj, OWL Files (.owl or .rdf))'. The interface is divided into several panes:

- Left Pane (Asserted Hierarchy):** Shows a tree view of classes. Under 'Pizza', there are subclasses like 'Named_pizza', 'Vegetarian_pizza', 'Spicy_pizza', and 'SpicyOrVegetarianPizza'.
- Center Pane (CLASS EDITOR):** Shows the editor for the class 'SpicyOrVegetarianPizza'. The 'Name' field is set to 'SpicyOrVegetarianPizza'. Below it, there is an 'rdfs:comment' field. The 'Asserted Conditions' section shows a list of conditions, including 'Vegetarian_pizza \sqcup Spicy_pizza' under the 'NECESSARY & SUFFICIENT' category.
- Right Pane (Properties):** Shows the 'Properties' section for the class. It lists 'has_base' (single Pizza_base) and 'has_topping' (multiple Pizza_topping).



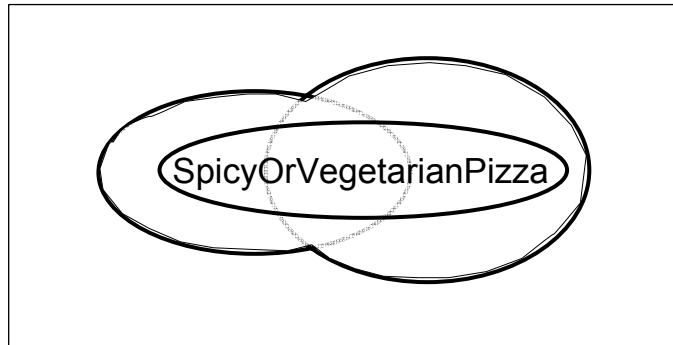
OR (Union)



A **$A \cup B$** **B**

Ex: VegetarianPizza \sqcup SpicyPizza

OR (Union)



A

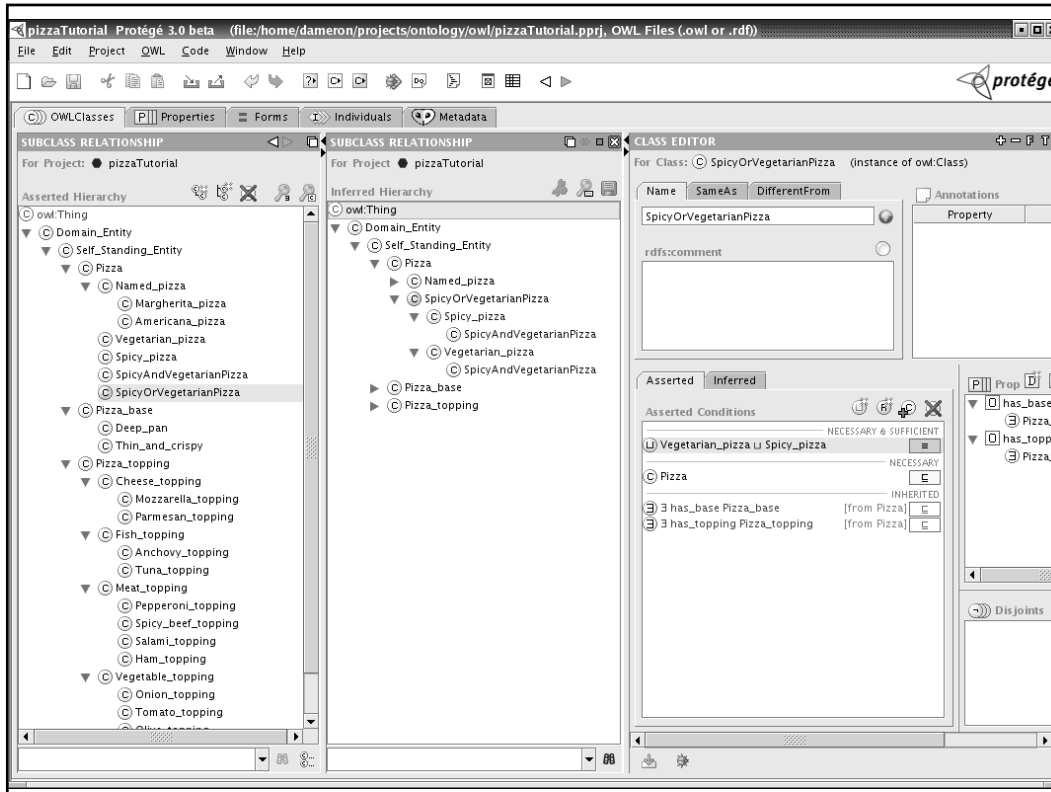
$A \cup B$

B

Ex: VegetarianPizza \cup SpicyPizza

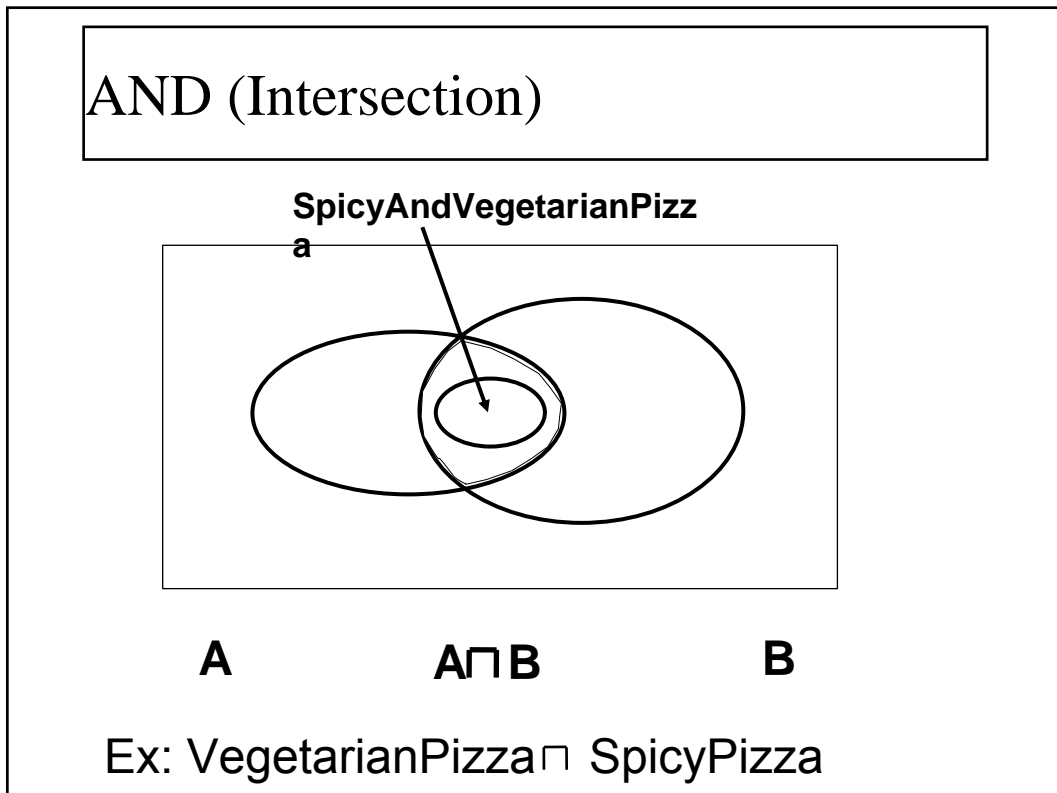
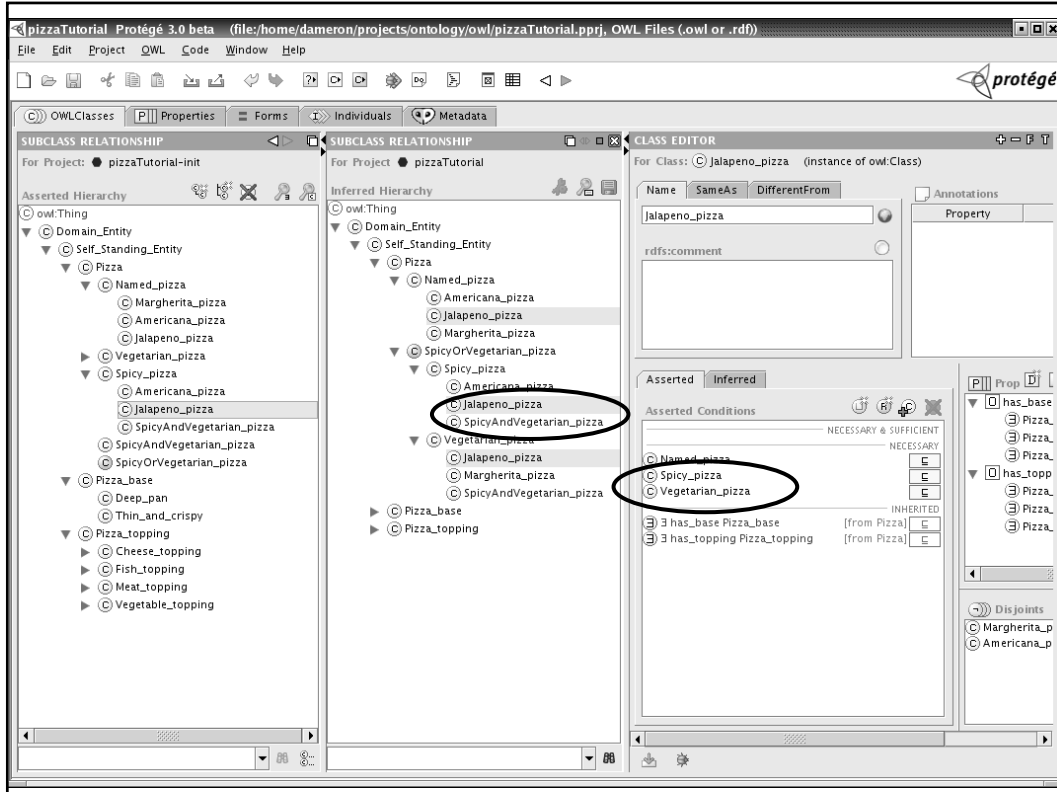
OR (Union)

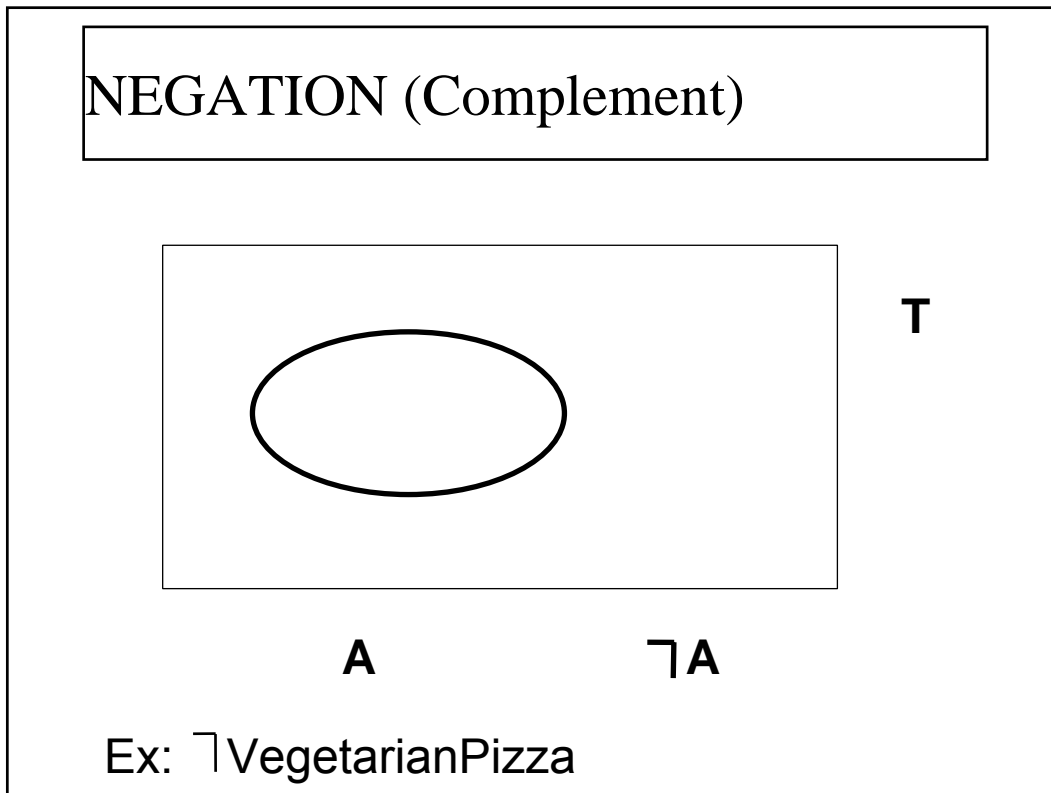
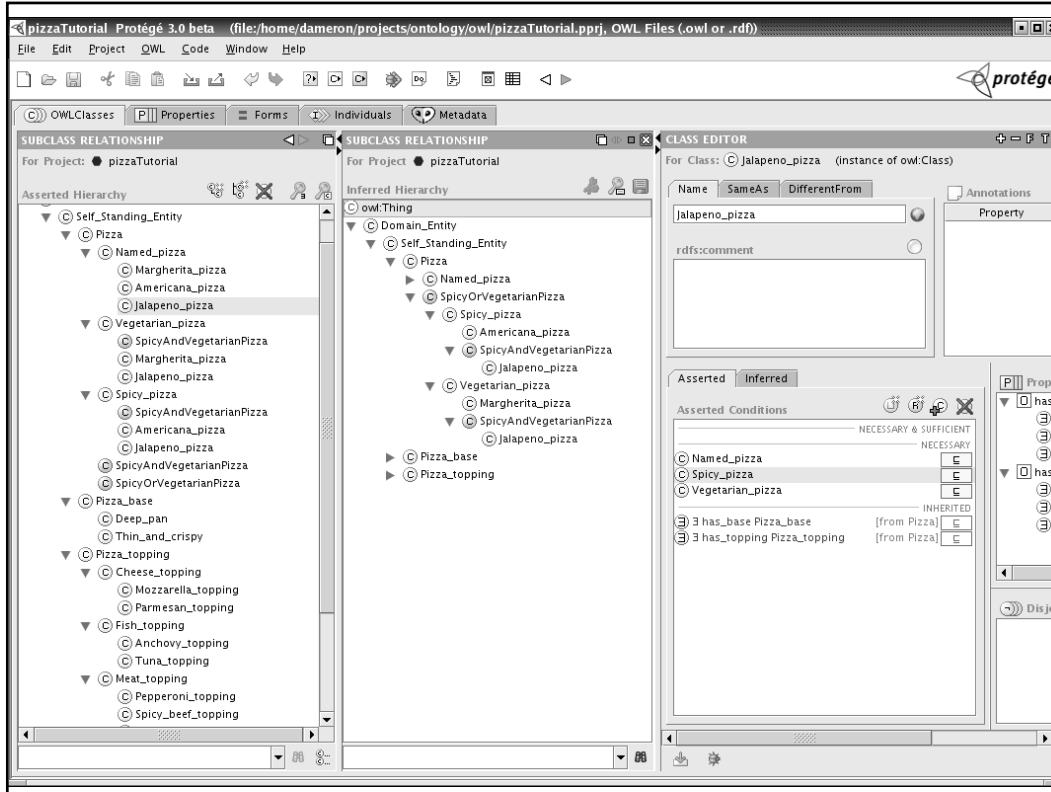
1. Now, use a definition for SpicyOrVegetarian_pizza (tip: right-click is your friend)



Examples

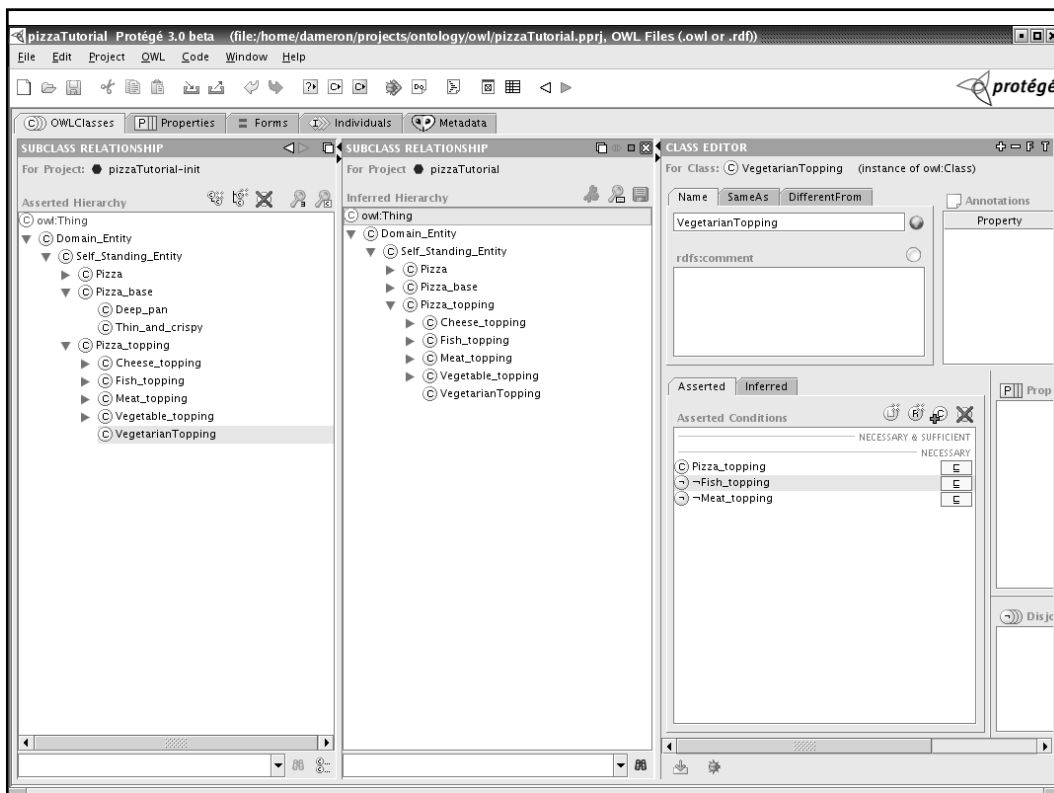
- Declare Margherita_pizza to be a Vegetarian_pizza
- Declare Americana_pizza to be a Spicy_pizza
- Declare Jalapeno_pizza to be both Spicy_pizza and Vegetarian_pizza
- Classify
 - :-)
 - why isn't Jalapeno_pizza classified as expected ?





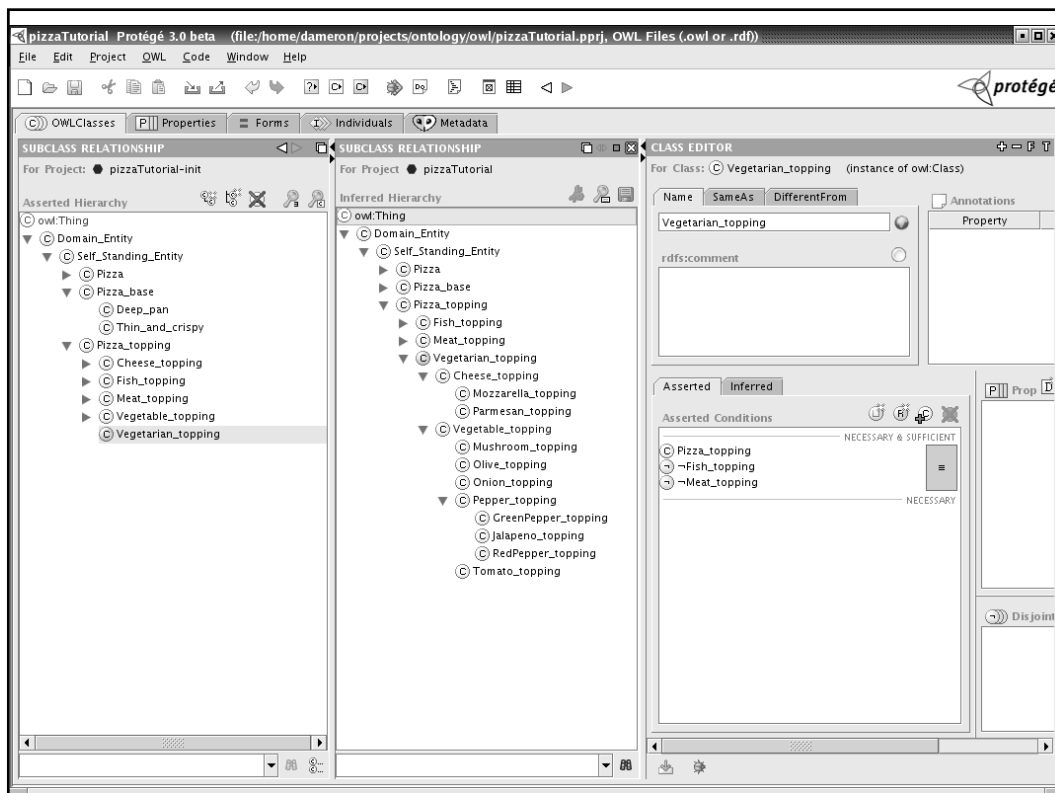
NEGATION (Complement)

- Create Vegetarian_topping as a subclass of Pizza_topping
- A Vegetarian topping is
 - neither a meat topping, nor a fish topping
 - a topping



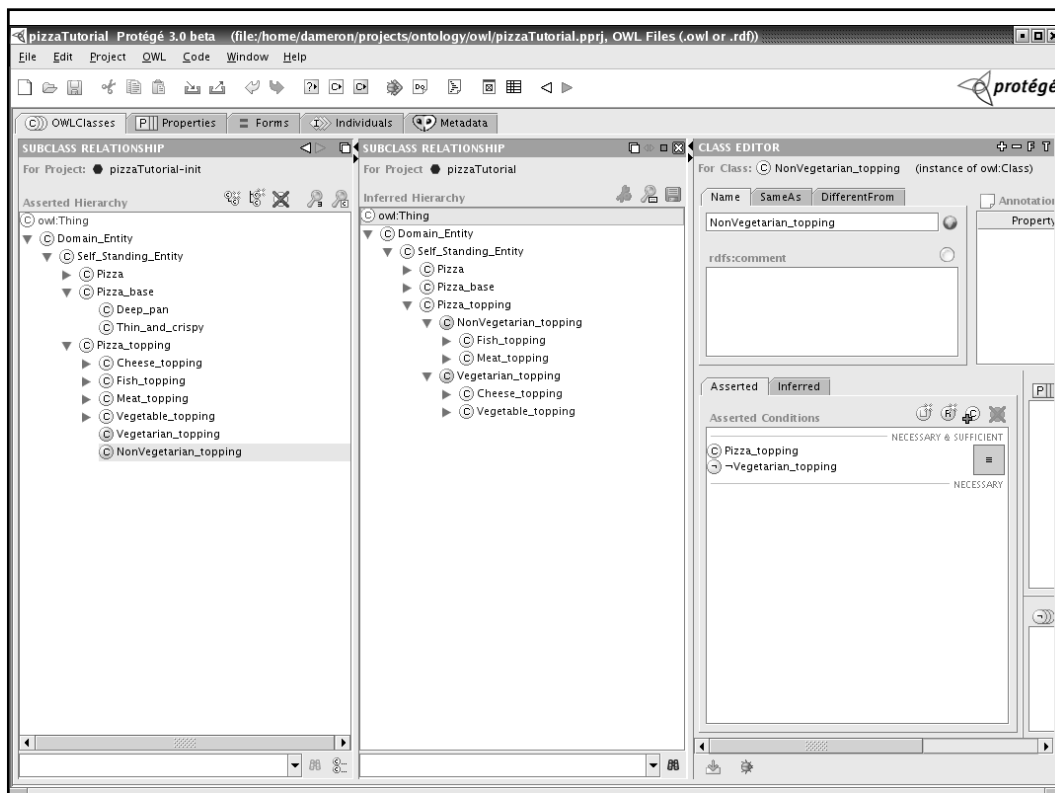
NEGATION (Complement)

- Create Vegetarian_topping as a subclass of Pizza_topping
- A Vegetarian topping is neither a meat topping, nor a fish topping
- Classify
- Why do we have to provide a Necessary and Sufficient definition ?



NEGATION (Complement)

- Create Vegetarian_topping as a subclass of Pizza_topping
- A Vegetarian topping is neither a meat topping, nor a fish topping
- Classify
- Why do we have to provide a N&S definition ?
- Create NonVegetarian_topping
- Classify



Expressing constraints

Constraints

- Quantifier restriction
 - How to represent the fact that every pizza must have a base ?
 - How to represent the fact that all the ingredients of a vegetarian pizza must be vegetarians ?
- Cardinality restrictions
 - How to represent that a Hand must have 5 fingers as parts ?
- hasValue restrictions
 - How to define the value of a relation for a class ?

Principles

- A restriction describes an anonymous class composed of all the individuals that satisfy the restriction
 - e.g. all the individuals that have (amongst other things) mozzarella as topping
- This anonymous class is used as a superclass of the (named) class we want to express a constraint on
 - e.g. MargheritaPizza

Existential restriction

- $(\exists \text{ hasTopping Mozzarella})$: set of the individuals being linked to at least one instance of Mozzarella through the *hasTopping* property
 - They can be linked to multiple instances of Mozzarella
 - They can also be linked to instances of other classes (provided domain and range integrity)
- Margherita $\sqsubset (\exists \text{ hasTopping Mozzarella})$

Existential restriction

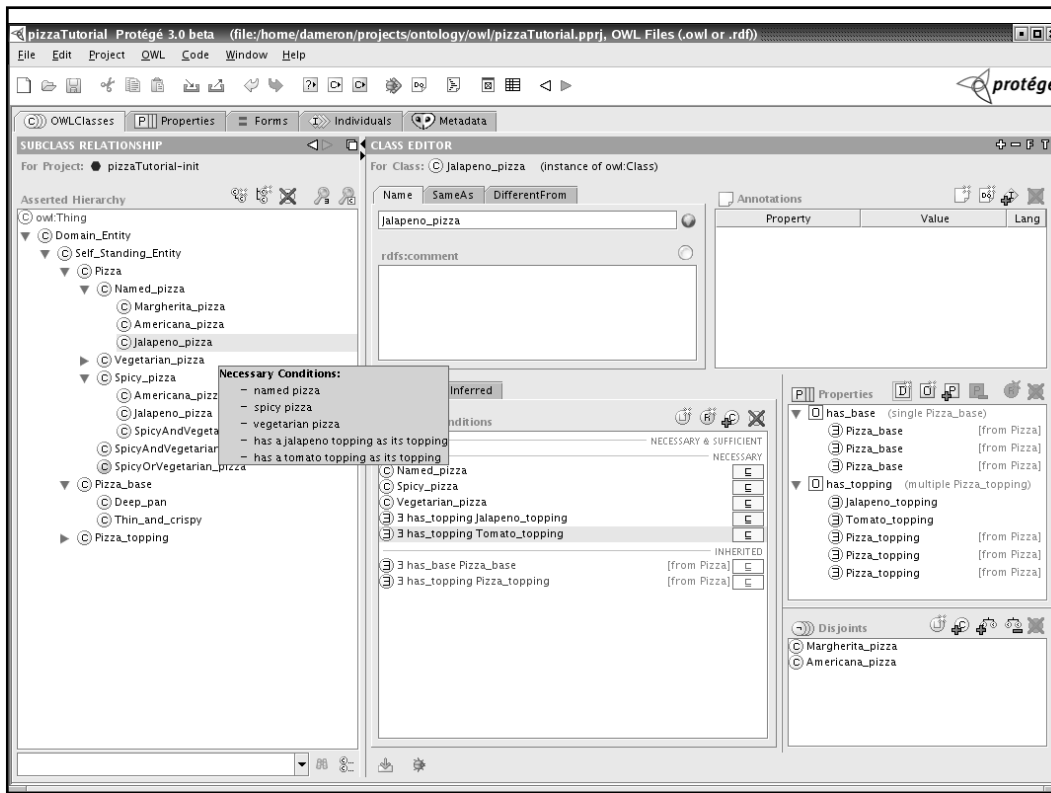
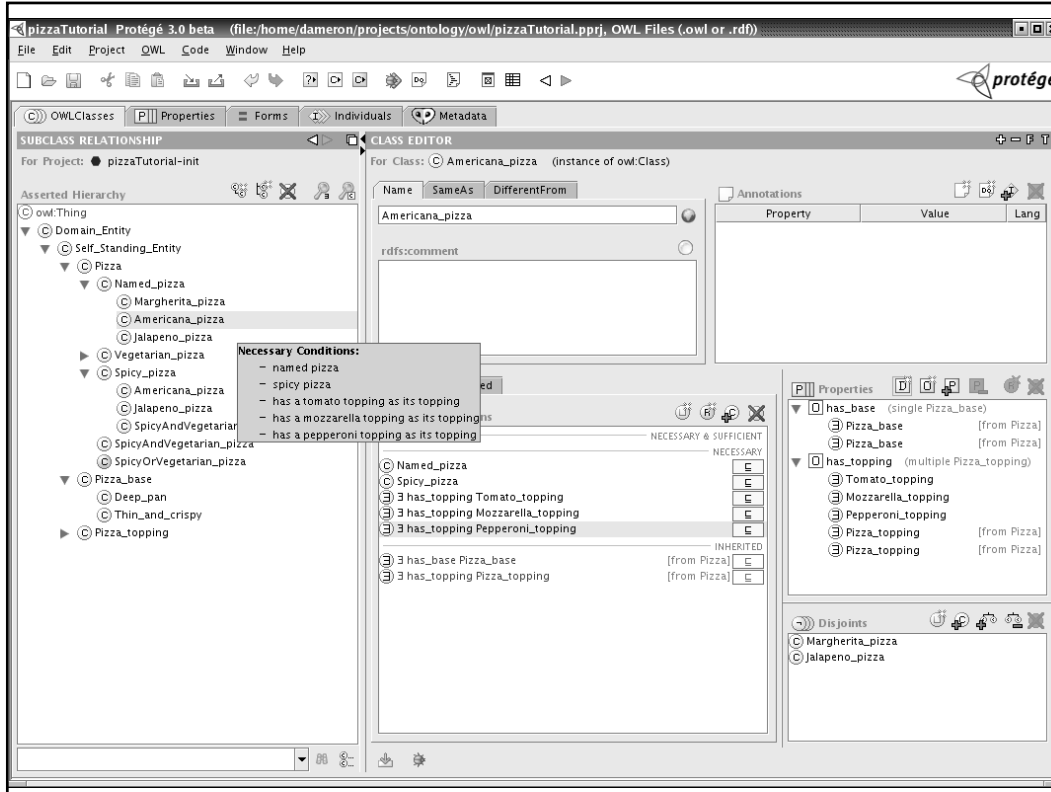
- Margherita pizzas' toppings:
 - Tomato
 - Mozzarella
- Americana pizzas' toppings:
 - Tomato
 - Mozzarella
 - Pepperoni
- Jalapeno pizzas' toppings:
 - Jalapeno
 - Tomato

The screenshot displays the Protégé 3.0 beta interface for editing an ontology. The main window is titled 'pizzaTutorial Protégé 3.0 beta'. The interface is divided into several panes:

- Left Pane (Asserted Hierarchy):** Shows a tree view of classes. The hierarchy is:
 - owl:Thing
 - Domain_Entity
 - Self_Standing_Entity
 - Pizza
 - Named_pizza
 - Margherita_pizza
 - Americana_pizza
 - Jalapeno_pizza
 - Vegetarian_pizza
 - Spicy_pizza
 - Americana_pizza
 - Jalapeno_pizza
 - SpicyAndVegetarian_pizza
 - SpicyOrVegetarian_pizza
 - SpicyAndVegetarian_pizza
 - Pizza_base
 - Deep_pan
 - Thin_and_crispy
 - Pizza_topping

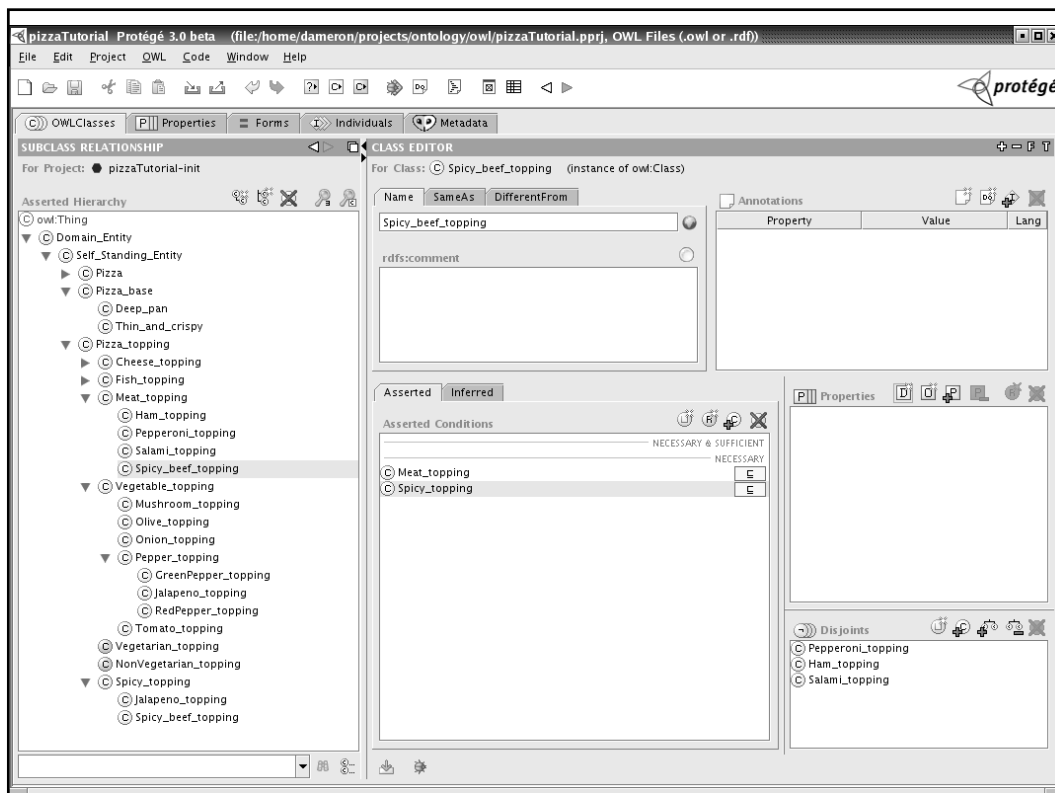
- Center Pane (CLASS EDITOR):** Shows the editor for the class 'Margherita_pizza'. It includes a table for defining relationships:

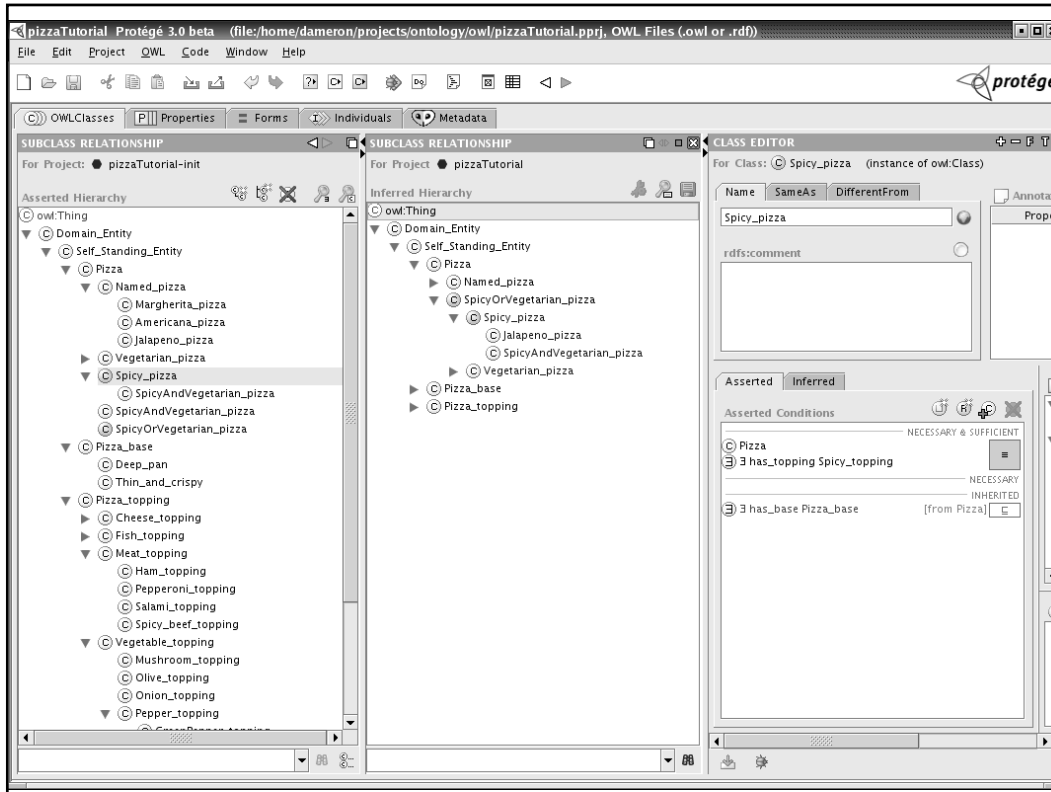
Name	SameAs	DifferentFrom
Margherita_pizza		
rdfs:comment		
- Right Pane (Properties):** Shows the properties of the class. The 'has_topping' property is expanded, showing its domain and range:
- has_base (single Pizza_base)
 - Pizza_base [from Pizza]
- has_topping (multiple Pizza_topping)
 - Mozzarella_topping
 - Tomato_topping
 - Pizza_topping [from Pizza]
 - Pizza_topping [from Pizza]



Complete the ontology

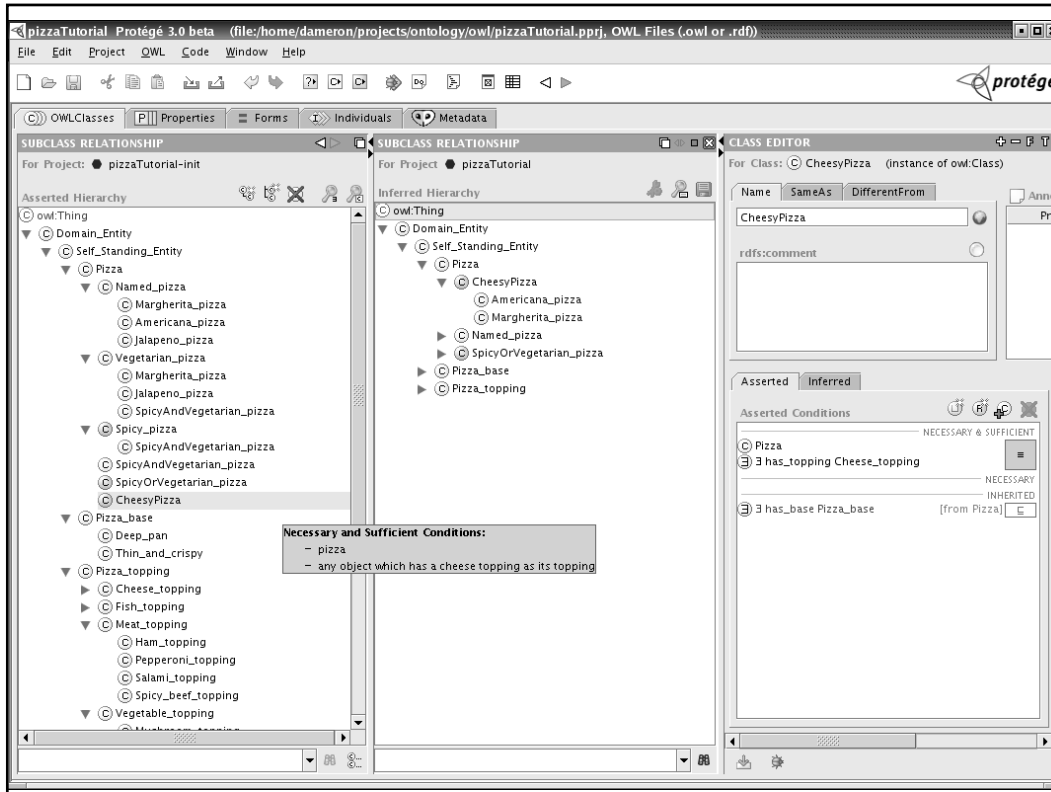
- Create Spicy_topping as a subclass of Pizza_topping
- Use Spicy_topping as a superclass of:
 - Jalapeno_topping
 - SpicyBeef_topping
- Define Spicy_pizza as a pizza having at least one spicy topping
- Remove the fact that Americana_pizza and Jalapeno_pizza are subclasses of Spicy_pizza





Complete the ontology

- Exercise: define Cheesy_pizza as a pizza having at least one cheese topping



Universal restriction

- $(\forall \textit{hasTopping} \textit{VegetarianTopping})$: set of all the individuals only linked to instances of *VegetarianTopping* through the *hasTopping* property
- Warning: also includes all the individuals linked to nothing through the *hasTopping* property

Universal restriction

- (\forall *hasTopping* VegetarianTopping)
- Remove the fact that Margherita_pizza and Jalapeno_pizza are subclasses of Vegetarian_pizza
- Define Vegetarian_pizza as any pizza for which all the toppings are vegetarian toppings
- Classify :-()

The screenshot displays the Protégé 3.0 beta interface. The main window shows the 'SUBCLASS RELATIONSHIP' view for the 'pizzaTutorial-init' project. The 'Asserted Hierarchy' pane on the left shows a tree structure starting from 'owl:Thing', with 'Domain_Entity' containing 'Self_Standing_Entity', which in turn contains 'Pizza'. Under 'Pizza', there are several subclasses including 'Vegetarian_pizza'. The 'CLASS EDITOR' on the right is set to edit the 'Vegetarian_pizza' class. It shows the 'Name' field as 'Vegetarian_pizza' and the 'rdfs:comment' field as empty. Below the editor, the 'Asserted Conditions' section is visible, showing a table of conditions for the 'Pizza' class:

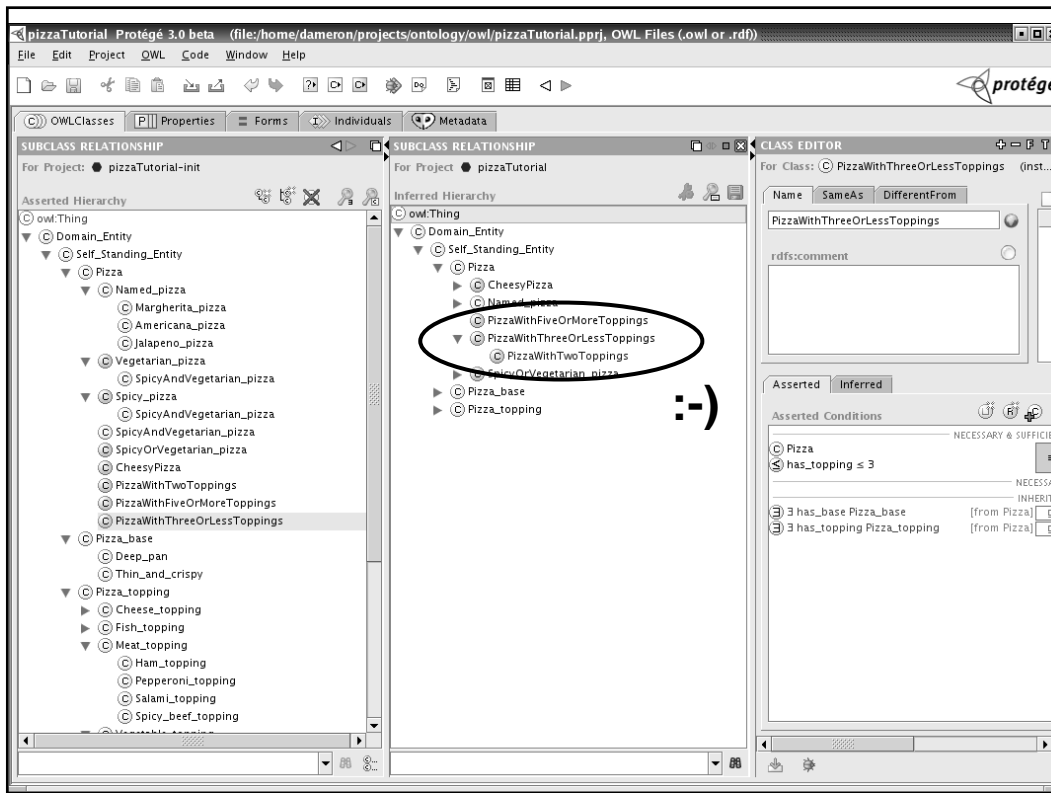
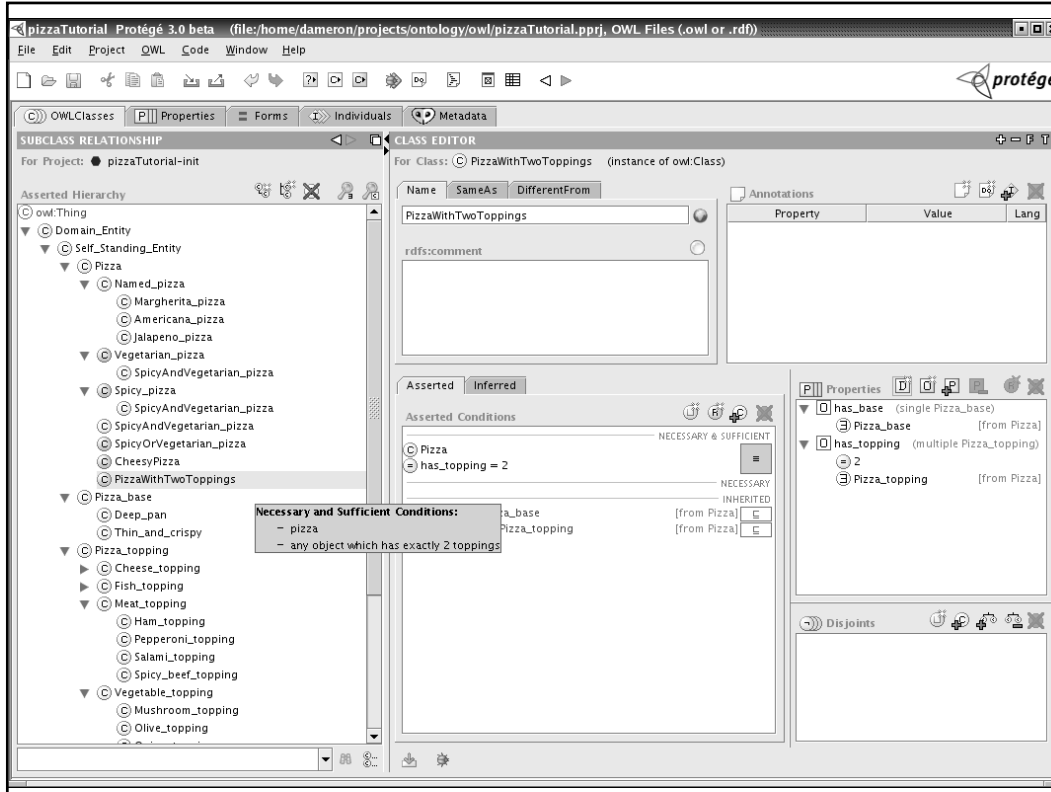
Asserted Conditions	NECESSARY & SUFFICIENT
<input checked="" type="checkbox"/> Pizza	
<input checked="" type="checkbox"/> \forall has_topping Vegetarian_topping	
<input checked="" type="checkbox"/> \exists has_base Pizza_base	INHERITED [from Pizza]
<input checked="" type="checkbox"/> \exists has_topping Pizza_topping	INHERITED [from Pizza]

Universal restriction

- Why Margherita and Jalapeno pizze were not recognised as vegetarian pizze ?
- ... Find out in a few slides

Cardinality restriction

- PizzaWithTwoTopping
 - Pizza \sqcap (hasTopping = 2)
- PizzaWithFiveOrMoreToppings
 - Pizza \sqcap (hasTopping \geq 5)
- PizzaWithThreeOrLessToppings
 - Pizza \sqcap (hasTopping \leq 3)
- Warning: This is NOT qualified cardinality restrictions



hasValue restriction

- So far, we have been narrowing the range of relationship
 - create the class Person
 - create the relation *hasPizzaMaker*: Pizza -> Person
 - create ItalianPerson as a subclass of Person
 - define GenuinePizza = ($\exists hasPizzaMaker$ ItalianPers.)
- We may also want to restrict it to a precise value (and not to a set of values)
 - create olivier as an instance of Person
 - define OliviersPizza = (*hasPizzaMaker* \ni olivier)

hasValue restriction

- Create luigi as an instance of ItalianPerson
- Create LuigisPizza
- Classify
- Why isn't LuigisPizza a subclass of GenuinePizza ?
... because it is a Racer (reasoner) bug !
IT SHOULD !!!

Reasoning

Reasoning usage

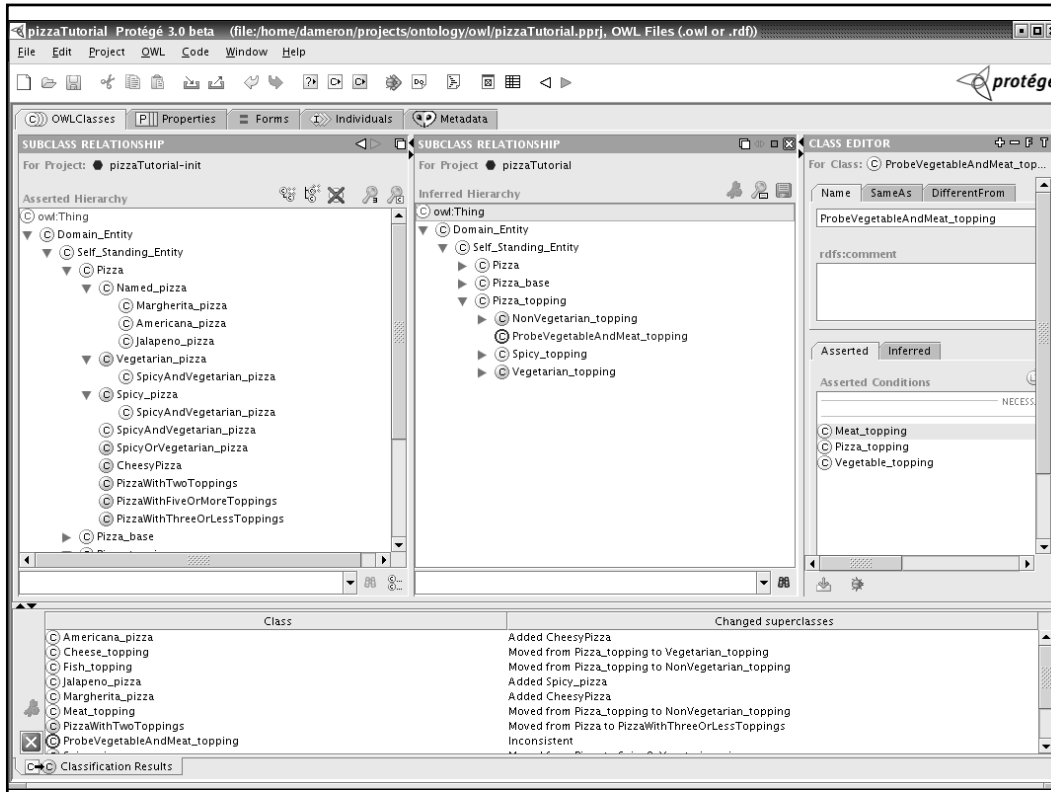
- Detecting inconsistencies (i.e. insatisfiability)
- Finding subclasses
- Detecting trivially satisfiable classes

Finding inconsistencies

Inconsistent class: class that cannot have any instance (insatisfiable)

- Create ProbeVegetableAndMeatTopping as a subclass of:
 - VegetableTopping
 - MeatTopping
- Classify
- Why is it inconsistent ?

The screenshot shows the Protégé 3.0 beta interface. The main window displays the 'CLASS EDITOR' for the class 'ProbeVegetableAndMeat_topping'. The 'Necessary Conditions' section is expanded, showing a list of conditions: 'pizza topping', 'vegetable topping', and 'meat topping'. The class hierarchy on the left shows 'ProbeVegetableAndMeat_topping' as a subclass of 'Vegetable_topping' and 'Meat_topping'. The interface also shows the 'SUBCLASS RELATIONSHIP' view and the 'CLASS EDITOR' tabs.



Finding inconsistencies

- ProbePizzaWithNoBase
- ProbePizzaWithNoTopping
- ProbePizzaWithTwoBases

Reasoning

Reasoning can be used to build and maintain an ontology: give intentional definitions for your concepts and let the computer do the work

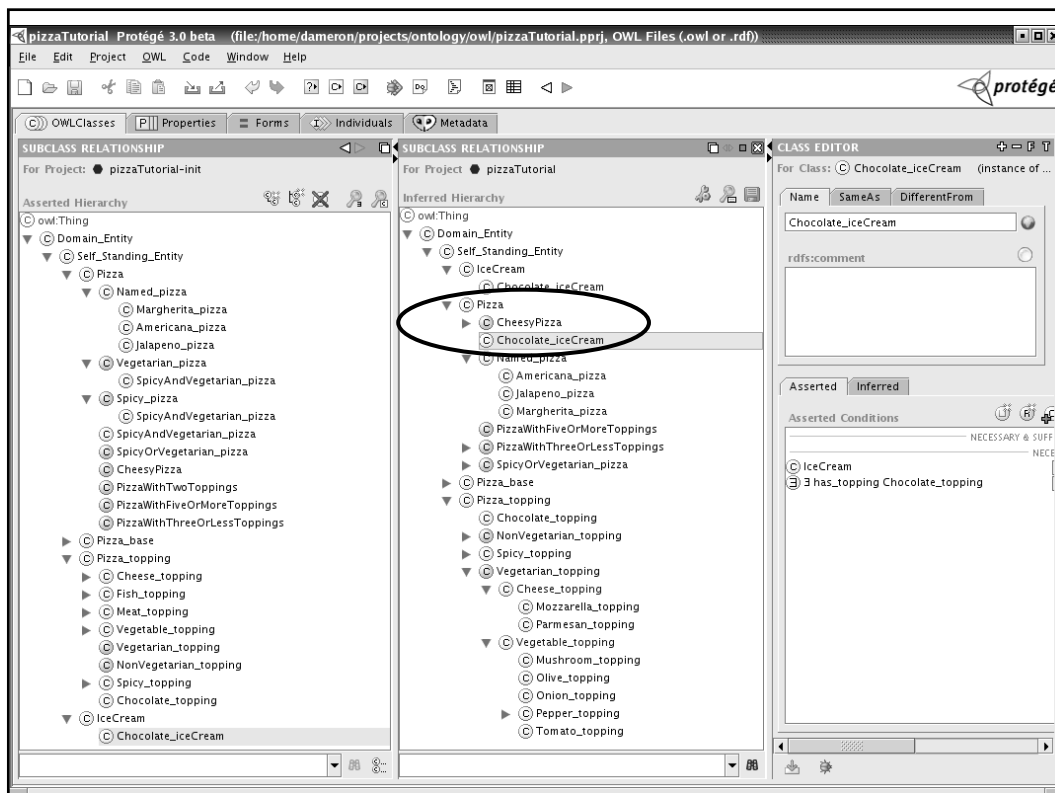
- This is exactly what we have been doing when we described VegetarianPizza and SpicyPizza
 - Create CreamTopping and make it disjoint from MeatTopping
 - After classification, CreamTopping should be a subclass of VegetarianTopping

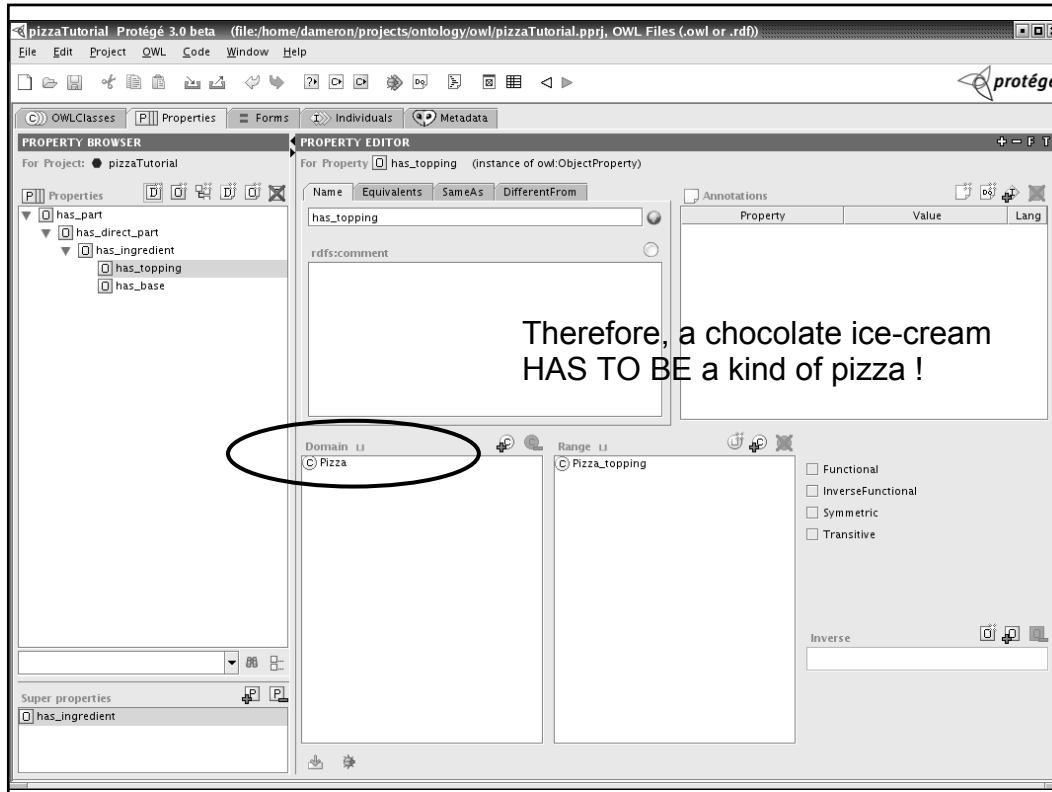
Reasoning

- Why we need defined concepts ?
 - Try to classify an example with a primitive concept
 - Find out / explain why we didn't get the expected result

Open VS Closed World Reasoning

- Create IceCream as a sibling of Pizza
- Create Chocolate_iceCream as a subclass of IceCream
- Create Chocolate_topping as a subclass of Pizza_topping (nevermind if it is weird)
- Express that all chocolate ice creams have Chocolate_topping as topping
- Classify
- How come Chocolate_iceCream is a Pizza ?





Open VS Closed World Reasoning

- Remember a few slides ago ???
- MargheritaPizza $\sqsubset (\exists \textit{hasTopping}$
Mozzarella) $\sqcap (\exists \textit{hasTopping}$ Tomato)
- Vegetarian_pizza = Pizza $\sqcap (\forall \textit{hasTopping}$
Vegetarian_topping)
- Tomato and Mozzarella ARE Vegetarian toppings
- **So, why isn't Margherita classified under Vegetarian_pizza ?**

Open VS Closed World Reasoning

- Remember a few slides ago ???
- MargheritaPizza $\sqsubset (\exists \textit{hasTopping}$ Mozzarella) $\sqcap (\exists \textit{hasTopping}$ Tomato)
- Vegetarian_pizza = Pizza $\sqcap (\forall \textit{hasTopping}$ Vegetarian_topping)
- **So, why isn't Margherita classified under Vegetarian_pizza ?**
- **Because some Margheritas may have other toppings (e.g. SpicyBeef_topping) !**

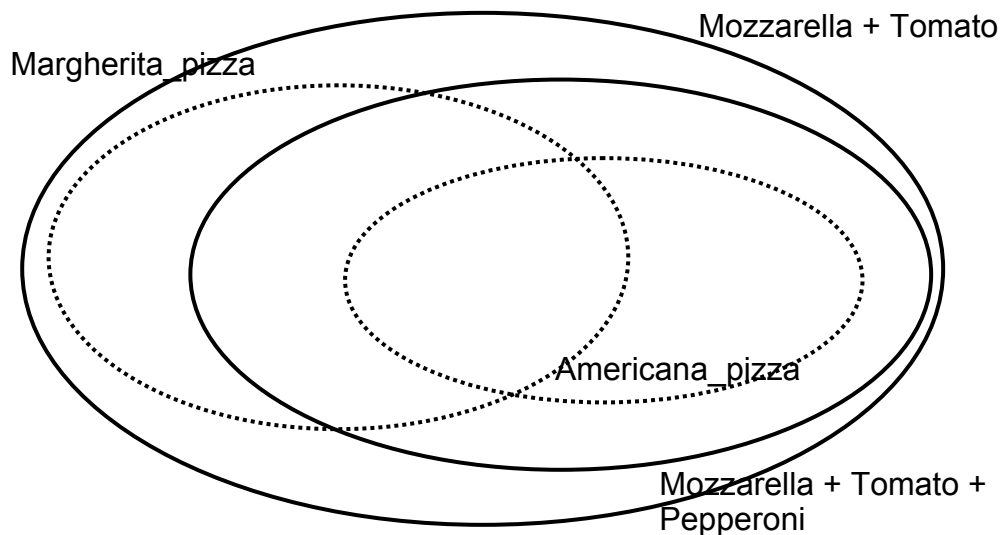
Open VS Closed World Reasoning

- Closed-World reasoning
 - Negation as failure
 - Anything that cannot be found is false
 - Reasoning about this world
- Open-World reasoning
 - Negation as contradiction
 - Anything might be true unless it can be proven false
 - Reasoning about any world consistent with the model

Need for closure

- MargheritaPizza $\sqsubset (\exists \text{ hasTopping Mozzarella}) \sqcap (\exists \text{ hasTopping Tomato})$
- AmericanaPizza $\sqsubset (\exists \text{ hasTopping Mozzarella}) \sqcap (\exists \text{ hasTopping Tomato}) \sqcap (\exists \text{ hasTopping Pepperoni})$
- Classify :-)

Need for closure



Need for closure

- MargheritaPizza = $(\exists hasTopping\ Mozzarella) \sqcap (\exists hasTopping\ Tomato)$
- AmericanaPizza = $(\exists hasTopping\ Mozzarella) \sqcap (\exists hasTopping\ Tomato) \sqcap (\exists hasTopping\ Pepperoni)$
- Classify :-)

Need for closure

Margherita pizzas only have Tomato and Mozzarella for topping

- MargheritaPizza = $(\exists hasTopping\ Mozzarella) \sqcap (\exists hasTopping\ Tomato) \sqcap$
?????

Need for closure

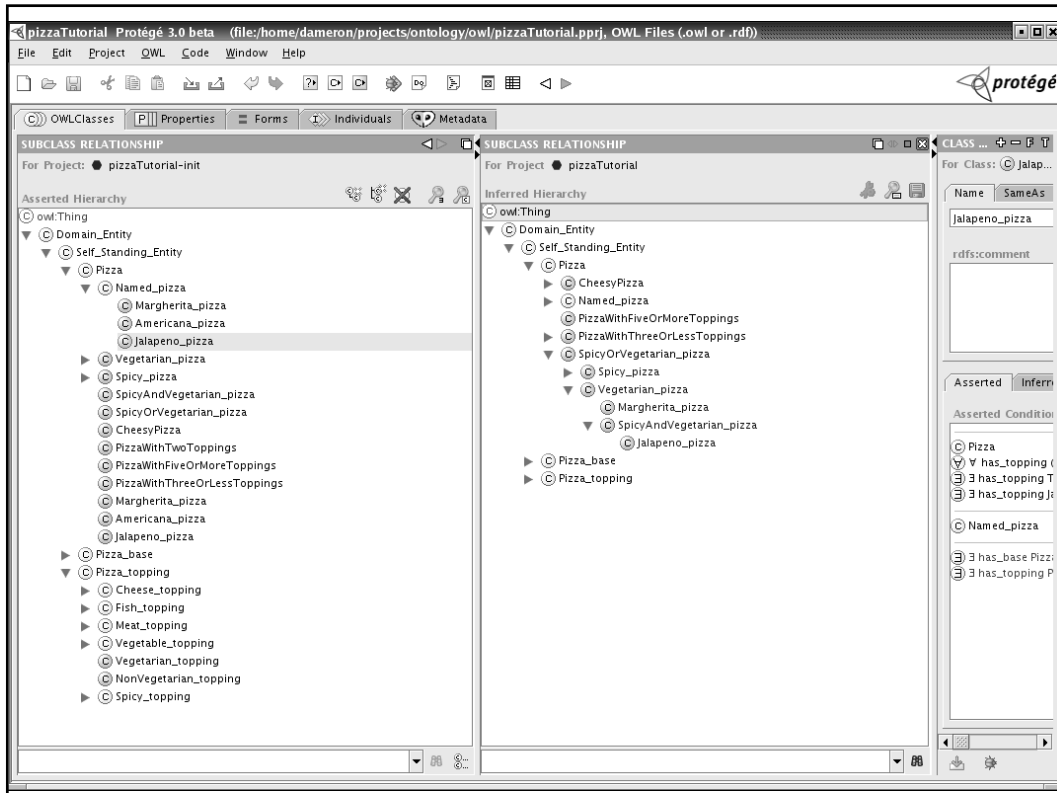
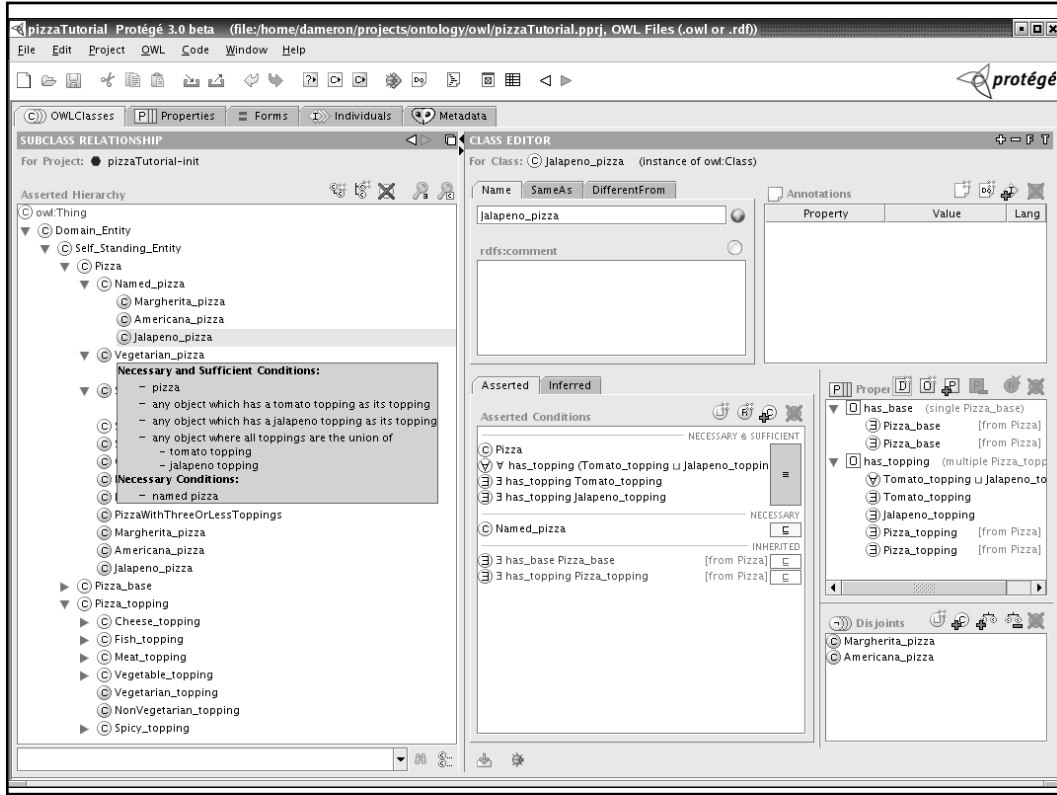
Margherita pizzas only have Tomato and Mozzarella for topping

- MargheritaPizza = $(\exists \textit{hasTopping Mozzarella}) \sqcap (\exists \textit{hasTopping Tomato}) \sqcap (\forall \textit{hasTopping (Mozzarella} \sqcup \textit{Tomato)})$
- The universal constraint (\forall) alone is not enough ! We need both \exists and \forall constraints

Need for closure

Margherita pizzas only have Tomato and Mozzarella for topping

- MargheritaPizza = $(\exists \textit{hasTopping Mozzarella}) \sqcap (\exists \textit{hasTopping Tomato}) \sqcap (\forall \textit{hasTopping (Mozzarella} \sqcup \textit{Tomato)})$
- AmericanaPizza = $(\exists \textit{hasTopping Mozzarella}) \sqcap (\exists \textit{hasTopping Tomato}) \sqcap (\exists \textit{hasTopping Pepperoni}) \sqcap (\forall \textit{hasTopping...})$
- Classify :-)



Defined concepts

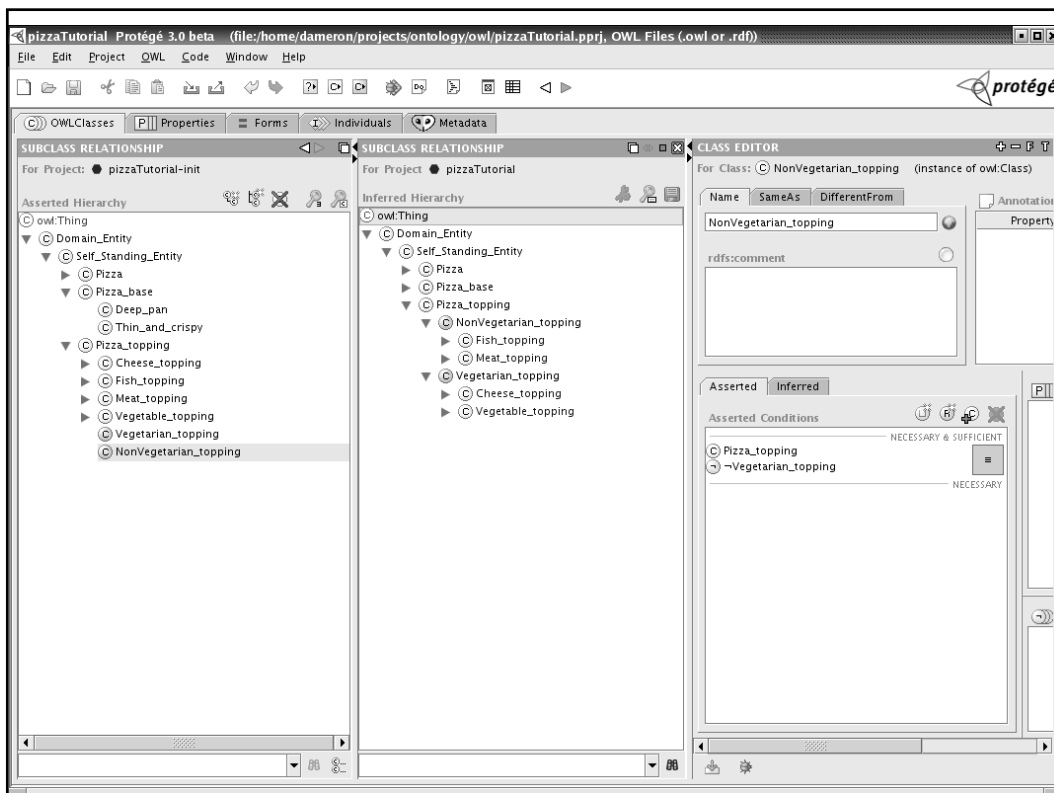
- Exhaustive decomposition
 - PepperTopping = RedPepper OR GreenPepper OR JalapenoPepper
 - RedPepper, GreenPepper and JalapenoPepper are disjoint

Defined concepts

- Provide intensional definitions
 - SpicyPizza (exists)
 - VegetarianPizza (for all)
- Exercice: add a closure to JalapenoPizza definition, and check that it is actually classified as a SpicyAndVegetarianPizza

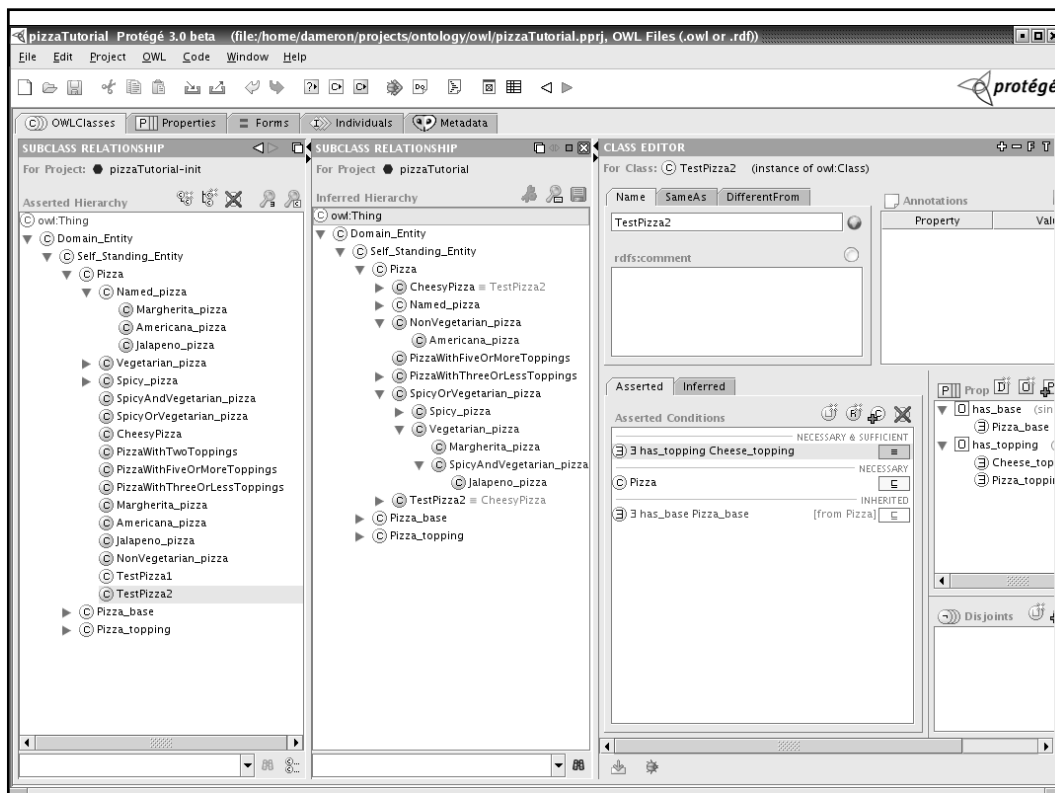
More fun with negation

- Define VegetarianPizza
- Define NonVegetarianPizza = \neg VegetarianPizza
- Classify
- Adapt the definition of NonVegetarianPizza



More fun with negation

- TestPizza1 \square (\exists hasTopping Cheese)
- Classify
 - Why isn't it a VegetarianPizza ?
 - Why isn't it a NonVegetarianPizza ?
- TestPizza2 = (\exists hasTopping Cheese)
- Classify
 - Same question...



More fun with negation

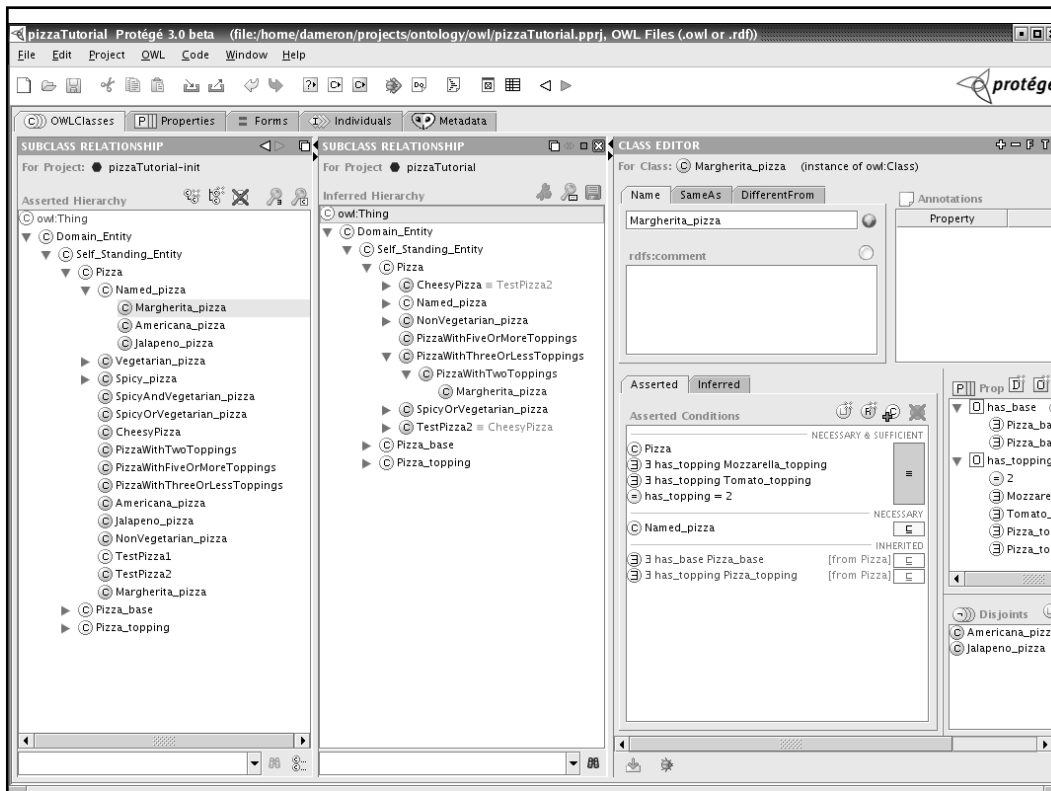
- TestPizza1 \sqsubset (\exists hasTopping Cheese)
- Classify
 - Why isn't it a VegetarianPizza ?
 - Why isn't it a NonVegetarianPizza ?
- TestPizza2 = (\exists hasTopping Cheese)
- Classify
 - **Still the open-world assumption:
there may be some instances with only
vegetarian toppings, and some other with
also non-vegetarian toppings !!!**

More fun with cardinality

- Why isn't MargheritaPizza classified under PizzaWithTwoToppings ?
- Hints:
 - Why isn't it classified under PizzaWithThreeOrLessToppings ?
 - Why isn't it even classified under PizzaWithFiveOrMoreToppings ?

More fun with cardinality

- Why isn't MargheritaPizza classified under PizzaWithTwoToppings ?
- Still... the open-world assumption: imagine one instance of Margherita_pizza having as topping:
 - one instance of Mozzarella_topping
 - one other instance of Mozzarella_topping
 - one instance of Tomato_topping



Reasoning about spiciness

- Create the SpicinessValue class
- Create the hasSpiciness relation
- Define SpicyPizza
- Define TooSpicyPizza (pizza with only spicy toppings)
- Describe toppings' spiciness
 - by hand
 - with wizard

Reasoning makes life easier :-)

- Allow to define classes by intension, rather than extension
 - If you add a new named pizza in your ontology, you don't have to worry about “is it a cheesy pizza?”, “is it a spicy pizza?” or “is it a vegetarian pizza?”...
 - Just describe your new pizza and let the classifier do the work for you
- Explain the Finger example

Reasoning makes life easier :-)

- Supports queries such as:
 - What are the vegetarian pizzas ?
 - What are the spicy pizzas ?
 - What are the non-spicy pizzas ?
 - What are the spicy vegetarian pizzas ?
- ... it allows you to take advantage of the knowledge you put into your ontology

Suspicious case: Trivially satisfiable classes

- Trivially satisfiable class: with an universal restriction which filler is unsatisfiable
 - There is probably something wrong going on
 - These classes are not inconsistent
- Ex: pizza AND (forall hasTopping (Mozzarella AND Tomato))
 - This is the set of pizzas with NO topping!
 - Why ???
- Can be detected by Protégé OWL tests

Reasoning tips

- Proceed by small incremental steps !
 - be cautious
 - classify often
- Use probe classes
- Use Protégé tests
 - to detect potential problems
 - to fix what can be fixed
 - ... even develop your own tests!

Reasoning tips

- Use probe classes
 - Positive probes: make sure that they are classified at the right place
 - Negative probes: make sure that impossible or abnormal situations are prevented

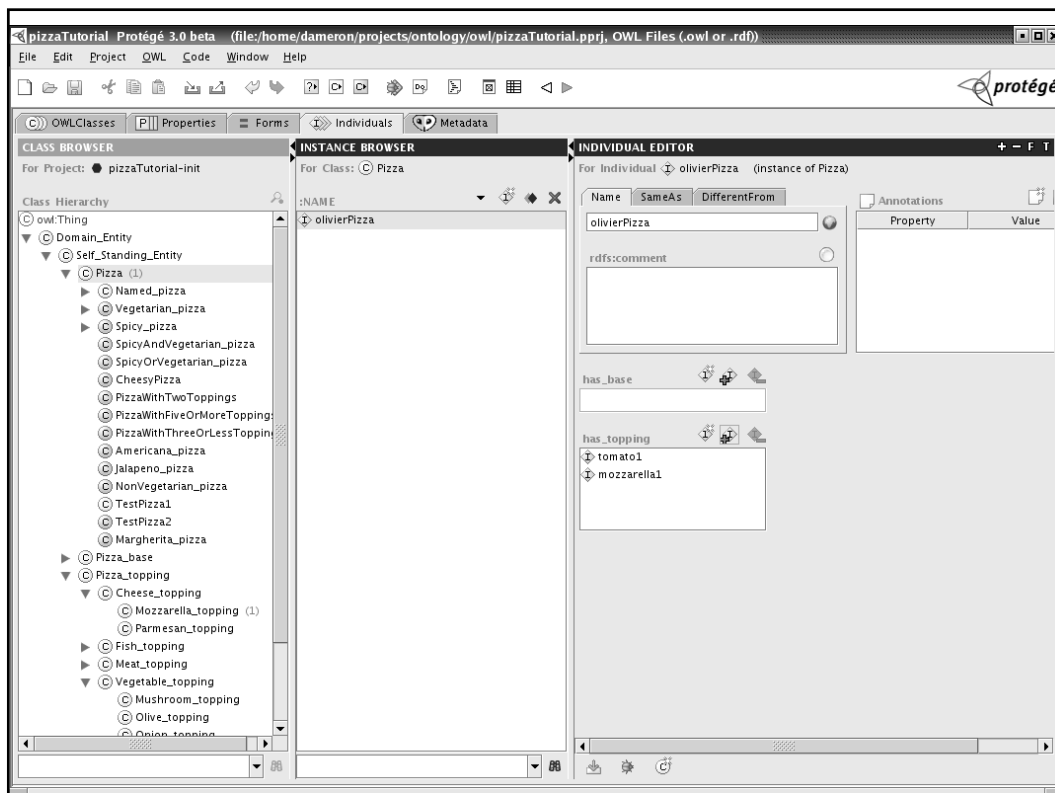
Reasoning Good Practice (2)

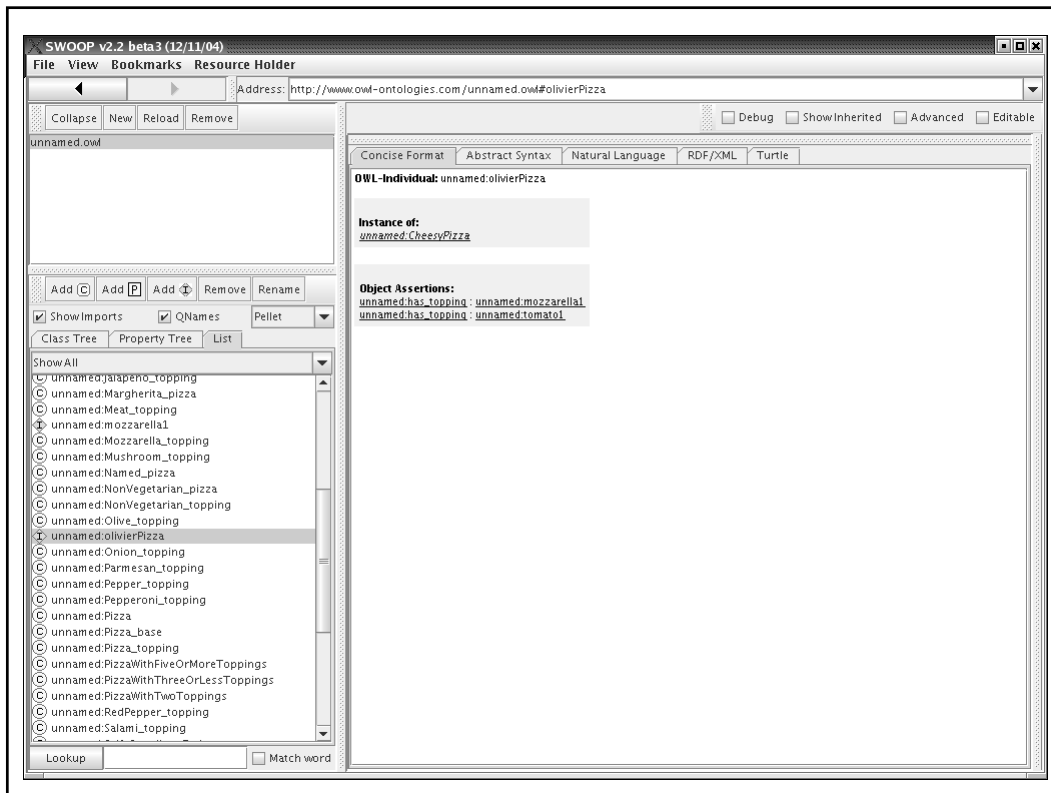
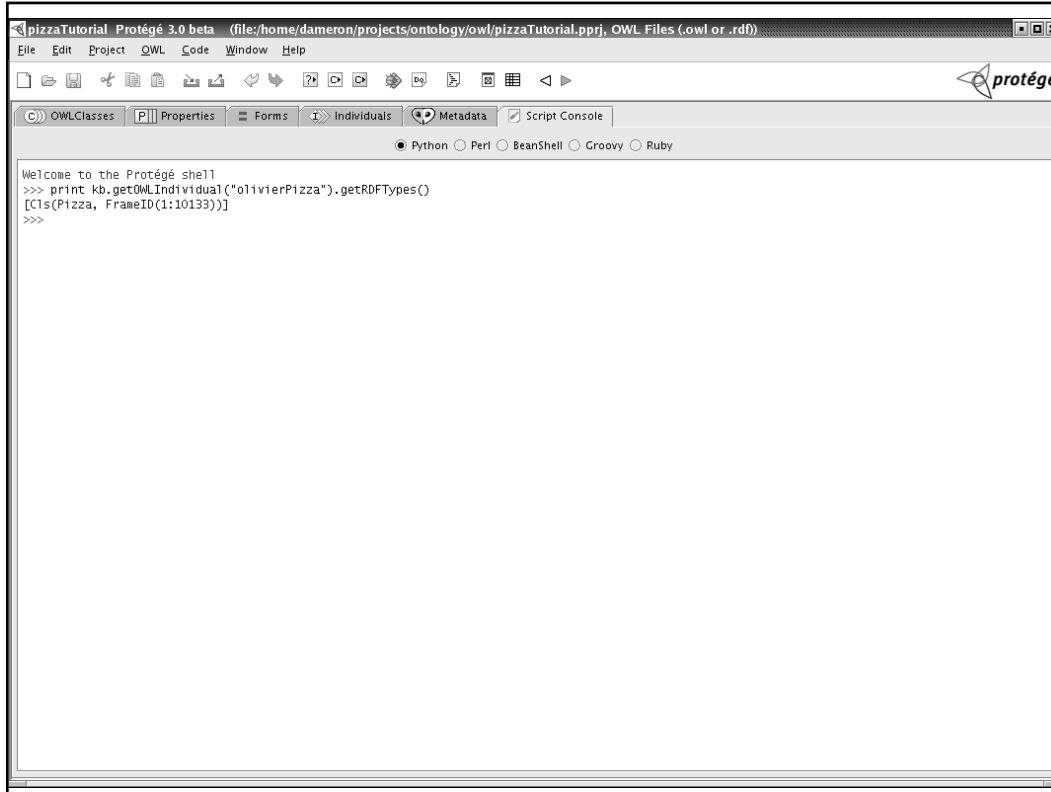
- Asserted Vs. Inferred hierarchies: represents what the ontology should look like, not how it was built
 - maintain a version with probes and export a “clean” version without probes
 - maintain the asserted version; export the classified one

Individuals

Reasoning with individuals

- Goal
- Either use “compute individuals belonging to class” (right-click on the class name) or use the API
- Can be tricky when you need to use closure
- Still incompletely supported by reasoners





Best Practice
(or at least not-so-bad)

Common mistakes

- Confuse AND and OR (e.g. pizza with cheese and pepper)
- Forget disjointness
- Forget closure
- Interference from relationships' domain and range
- Confusion between $(\exists r \Gamma A)$ and $\Gamma(\exists r A)$
 - e.g. $(\exists \text{ hasChild } \Gamma \text{ Male})$ VS $\Gamma(\exists \text{ hasChild Male})$

Best practice

One type of restriction by granularity level:

- ~~Meat Topping~~
 - ~~Spicy Beef Topping~~

- Meat Topping
 - Beef Topping
 - Spicy Beef

Best practice

All siblings at the same granularity level:

- Pizza Topping
 - Vegetable Topping

 - Fish Topping
 - Anchovy Topping
 - Salmon Topping
 - Cheese Topping
 - Mozzarella
 - ...

Best practice: transitive properties

- Always make a non-transitive subproperty of a transitive property.
- Use the latter
 - has Part (+)
 - has Direct Part
 - has Ingredient
 - ...

Covering

Beware of the open-world assumption!

- When your taxonomic decomposition is exhaustive, define (=) the more general as the union of the specifics
- If the specifics are mutually exclusive, flag them as disjoint

Value partition

The preferred approach is by a partition of classes (cf SpicinessValue)

- Philosophically sound
- Extensible
- You could also use enumerations of individuals (e.g. Sex = {male; female})
 - Not extensible
 - Doesn't support multiple partitions of the domain

Inverse of relationships

- Creation
- Modeling
 - part necessary to the whole
 - Body subclassOf (\mathcal{E} hasDirectPart Brain)
 - part optional to the whole
 - Body subclassOf (\mathcal{E} hasDirectPart (Hand U ...))
 - whole necessary to the part
 - Brain subclassOf (\mathcal{E} isDirectPartOf Body)
 - whole optional to the part
 - Cell

Composing ontologies

- Putting it back in the Semantic Web context
 - modularity
 - semantic interoperability
- Whenever possible, refer to an upper-level ontology
 - SUMO
 - BFO
 - DOLCE
- Avoiding ambiguity: namespaces

Summary

Summary

- Compositional approach
- Intensional description
- Reasoning
 - classification
 - open-world assumption
 - inconsistency