Artificial Intelligence Programming

Ontologies and Protege

Chris Brooks

Department of Computer Science

University of San Francisco

17-2: Graphical Representations

- Logic is a very powerful representation language, but it can be difficult for humans to work with.
- In addition, the lack of structure between sentences in a KB can make inference difficult.
- A *semantic network* is a way to graphically represent and reason about some logical concepts.

17-3: Semantic Net example

- $SisterOf(Mary, John)$
- $Female(Mary)$
- $\forall x Female(x) \rightarrow Person(x)$
- $Male(John)$
- $\forall x Male(x) \rightarrow Person(x)$
- $\forall x Person(x) \rightarrow Mammal(x)$
- $\forall x Person(x) \rightarrow DefaultNumberOfLegs(x, 2)$
- $legs(John, 1)$

17-4: Inference in a Semantic Net

- Inference becomes easy in a semantic net; to answer questions about John, we follow the labeled edges emanating from the *John* node.
  - This is the same sort of inference done by modern OO languages to resolve inheritance.
- Strengths: Knowledge is easily visualized, relationships between objects are clearer.
- Weaknesses: only binary relations, no negation, disjunction, or existential quantifiers.
  - We can extend semantic nets to include these features, but we lose the transparency.
  - Instead, use a semantic net to model class/property relations, and FOL (or something similar) to model rules.

17-5: Knowledge Engineering

- Logic provides one answer to the question of *how* to say things.
- It does not tell us *what* to say.
- Typically, this is the hard part.
- We want to give our agent enough knowledge to solve all of the problems we are interested in.
- The process of building a knowledge base is referred to as *knowledge engineering*
    - Many of the same principles as software engineering.

17-6: Knowledge Enginering

- The knowledge engineering process typically consists of the following steps.
  - Determine the queries and types of facts that are available/allowed
    - For example, will the agent need to select actions or make conclusions, or just answer questions from a human user?
    - Will we ask existential queries, or just universal queries?
  - Gather the relevant knowledge
    - Interview experts and find out how the domain works.

17-7: Knowledge Enginering

- Select a vocabulary of classes, functions, predicates and constants
  - This vocabulary is called an *ontology*
- Encode general knowledge about the domain
  - Formally represent the knowledge gathered in step 2.
  - This may require revisiting step 3.
- Encode specific queries or problems to be solved.
  - This may require returning to steps 3 and 4.

17-8: Ontologies

- An ontology is a vocabulary that describes all of the objects of interest in the domain and the relations between them.
  - All of the relevant knowledge about a problem we are interested in.
- Ontolgies allow knowledge about a domain to be shared between agents (including humans).
- This can allow knowledge to be reused more easily.
- Allows an agent to perform inference with its current knowledge.

17-9: Vocabulary

- An ontology consists of:
  - A set of *concepts* or *classes*
  - $Professor(Brooks), \forall x \; Professor(x) \rightarrow USFEmployee(x)$
- Instances of these classes.
- Properties that hold between classes of objects, sometimes called *slots* or *relations* or *roles*.
  - $Salary(Brooks, \$500), Name(Brooks, ``Chris'')$
- Restrictions on slots (sometimes called *facets*)
  - $\forall x, y \; Professor(x) \wedge Salary(x, y) \rightarrow y < \$1000$

17-10: Ontologies vs OO design

- Classes are a primary focus of ontology design.
- In many ways, this looks like object-oriented design.
- We have classes and subclasses, and properties of classes that look like data members.
- However, properties have richer semantics than data members.
- A property may attach to several classes at once.
- Properties can have subproperties
- We can specify constraints on the values in a property's range.
- Slots can exist without being assigned to a class.

17-11: Protege

- Protege is a Java-based graphical tool for constructing ontologies.
- Has a rich set of plugins for performing inference and visualizing data.
- Can export data in RDF and OWL for use with the Semantic Web.

17-12: OWL

- Web Ontology Language (OWL) is an XML-based language for representing ontologies.
- Built on top of RDF
- Encodes a *description logic*
  - Decidable subset of first-order logic.
- We won't be working with OWL directly in here.

17-13: The Pizza Tutorial

- The pizza tutorial provides a nice tour of the issues involved in creating an ontology in Protege with OWL.
- We begin by asking *competency questions* - these are questions we'd like our KB to be able to answer.
  - What toppings are on a Margherita pizza?
  - Is the Americana pizza vegetarian?
  - What can I put on my pizza to make it spicy?
  - What pizzas have Tomato on them?

17-14: Classes

- As with OO design, we can work top-down, bottom-up, or in a combination of the two.
- Let’s start by making a Pizza class.
- We then make PizzaBase and PizzaTopping.
  - Make these disjoint.

17-15: Classes

- Use the Wizard to add subclasses of PizzaBase: DeepDishBase, ThinAndCrispyBase
- Use the Wizard to subclass PizzaTopping: MeatTopping, VeggieTopping, CheeseTopping, SeafoodTopping
- Subclass these.

17-16: Properties

- Now we can start to create Properties.
- Add a ’hasIngredient’ property.
  - In Protege, properties can have subProperties.
  - Add subproperties hasTopping, hasBase.
- We can also add inverse properties.
- Also functional, inverse functional, symmetric, and transitive.
- make hasBase functional.

17-17: Domains and ranges

- We can specify that properties can only apply to certain types of objects.
- set the range of hasTopping to be PizzaTopping
- Set the domain of hasTopping to be Pizza.
- Same for hasBase

17-18: Property Restrictions

- The primary way that we write rules in Protege is through the use of property restrictions.
- We can use an existential restriction to specify that a Pizza must have a PizzaBase.

17-19: Types of pizzas

- Add a subclass of pizza called NamedPizza.
- Add a subclass of this: MargheritaPizza.
- existential restriction: has Tomato and Mozzarella toppings.
- Add Americana Pizza: clone Margherita, add Pepperoni
- Add AmericanaHotPizza: clone Americana, add jalapenos.
- Add SohoPizza: Margherita plus Olives, Parmesan.

17-20: Reasoners

- As your ontology gets large, it can be difficult to maintain:
  - Do you have the hierarchy right?
  - Are there contradictions?
  - Are there other relations that are entailed?
- A big advantage of using this sort of tool is the ability to perform this inference automatically.
- Tools for doing this are known as reasoners.
- We'll be using a particular reasoner known as RACER.

17-21: Reasoners

- We can use the reasoner to:
  - Check consistency
  - Classify the taxonomy
  - Infer class membership
- Let's add an inconsistent class: InconsistentTopping
  - Subclass of Cheese
  - Restriction: must be a Veggie.
- Run the reasoner.

17-22: Necessary and Sufficient Conditions

- So far, we've only specificed *necessary* conditions.
- This is not enough to show that something must be a member of a particular class.
- Create a CheesePizza class
- Add a necessary and Sufficient restriction - has a Cheese topping.
    - This is now a *definition* - a CheesyPizza is a Pizza with a Cheese topping.
- Run RACER again.

17-23: Universal restrictions

- We specificed that a Margherita pizza must have Tomato and Mozzarella, but not that those were the only things it could have.
- To do this, we use universal restrictions.
- Create a VeggiePizza.
- Under necessary and sufficient, indicate that all toppings must be veggie or cheese.
- Run RACER again.

17-24: Open World Reasoning

- Why is Margherita (and Soho) not suclassed as Veggie?
- Protege and OWL use open-world reasoning
  - Can’t assume something is false just because it’s not stated to be true.
- Margherita could have other toppings.
- Add a closure axiom to MargheritaPizza.
- Use the widget to add closure axioms for Soho and Americana.

17-25: Value Partitions

- Use the Wizard to create a SpicynessValuePartition
- Subclass this to get Hot, Mild, Medium
- Add hasSpiciness property
- Use the PropertyRestrictions wizard.

17-26: Cardinality

- We can also specify the number of toppings a pizza must have.
- Create a CrowdedPizza
- Necessary and Sufficient: minimum cardinality: 3.

17-27: Individuals

- We can then add individuals using the individuals tab.
- the Forms tab lets us specify how individuals should be displayed.
- Add a NamedPizza that has tomato and Mozzarella
- Run the reasoner again.

17-28: Querying the KB

- The query pane lets us create, save and retrieve queries
- We can specify either AND or OR.
- Try finding all pizzas that have a cheese topping.