CPSC 312 — Functional and Logic Programming

- Project #2 - should be underway....
- Talk to a TA if you want to change your project, or it has drifted from what was originally proposed.

"Pascal [Java] is for building pyramids – imposing, breathtaking, static structures built by armies pushing heavy blocks into place. Lisp [Haskell/Prolog] is for building organisms – imposing, breathtaking, dynamic structures built by squads fitting fluctuating myriads of simpler organisms into place.

...

the pyramid must stand unchanged for a millennium; the organism must evolve or perish."

– Alan J. Perlis, Foreword to "Structure and Interpretation of Computer Programs", 1985, 1996

Plan

Last time

- difference lists
- definite clause grammars
- natural language interfaces to databases
- computer algebra and calculus
- Knowledge graphs, triples, reification, URI, RDF, triple store

Today

- Semantic web
- Ontologies

Ontologies and Knowledge Sharing

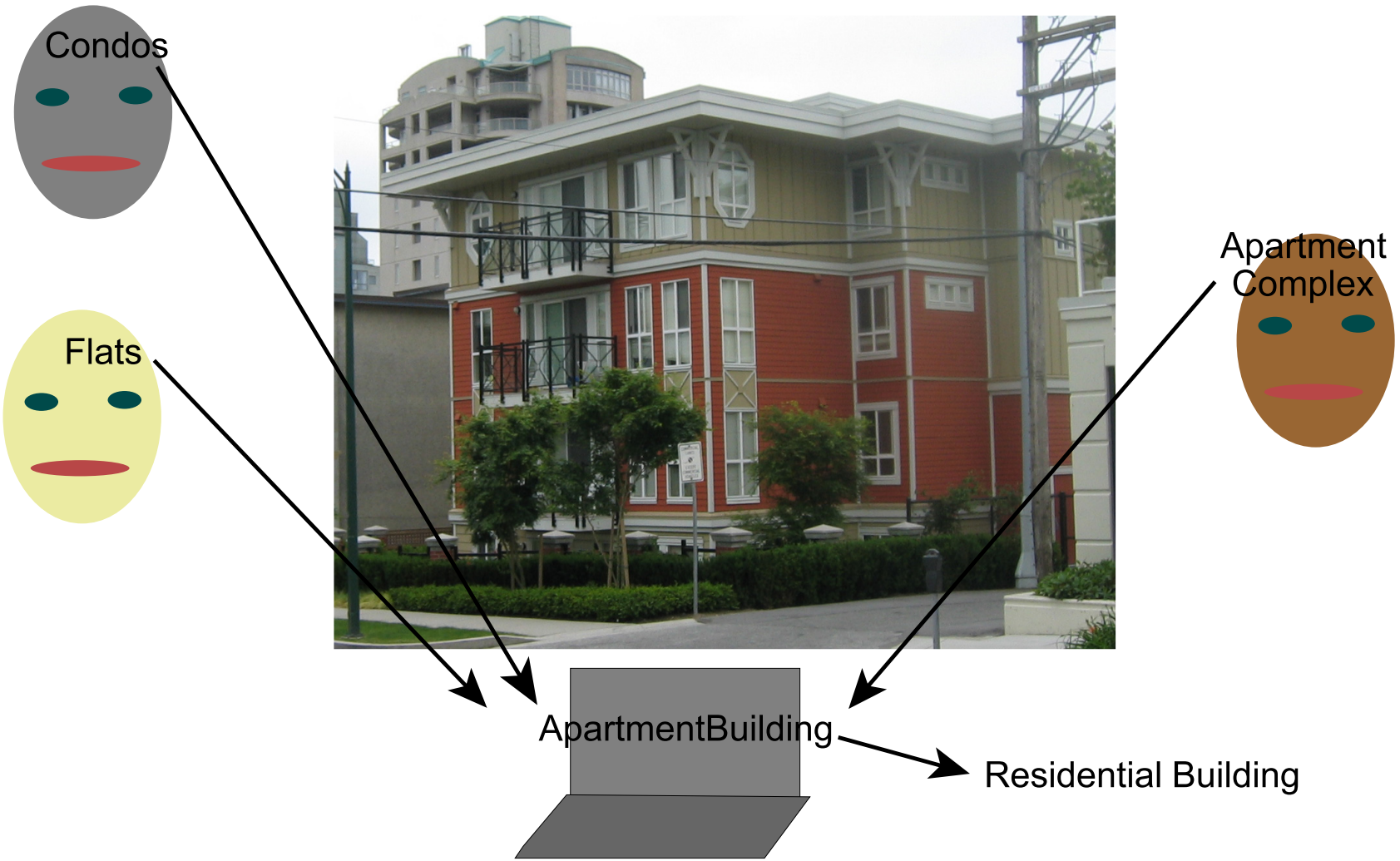
Building large knowledge respoitories:

- Knowledge often comes from multiple sources.
- Fields have their own terminology and division of the world.
- Systems evolve over time and it is difficult to anticipate all future distinctions that should be made.
- Designers must agree on what indiviuals, classes and relationships to represent. The world is not divided into individuals.
- It is often difficult to remember what notation means:
  - Given a symbol used in the computer, what does it mean?
  - Given a concept in someone's mind, what symbol to use?
    - Has the concept already been defined?
    - If already defined, what symbol has been used for it?
    - If not already defined, what can it be defined in terms of?

Knowledge Sharing

- Idea: Let's better represent an intended interpretation, so that computers as well as people can understand it.
- A conceptualization is a map from the problem domain into the representation. A conceptualization specifies:
  - What sorts of individuals are being modeled
  - The vocabulary for specifying individuals, relations and properties
  - The meaning or intention of the vocabulary
- If more than one person is building a knowledge base, they must be able to share the conceptualization.
  $\longrightarrow$ challenge: inter-operability of separately designed knowledge bases.
- An ontology is a specification of a conceptualization.
  An ontology specifies the meanings of the symbols in an information system.

Mapping from a conceptualization to a symbol



Semantic Web

- Ontologies are published on the web in machine readable form.
- Builders of knowledge bases or web sites adhere to and refer to a published ontology:
  - A symbol defined by an ontology means the same thing across web sites that obey the ontology.
  - If someone wants to refer to something not defined, they publish an ontology defining the terminology.
    Others adopt the terminology by referring to the new ontology.
    In this way, ontologies evolve.
  - Separately developed ontologies can have mappings between them published.

Challenges of building ontologies

- They can be huge: finding the appropriate terminology for a concept may be difficult.
- How one divides the world can depend on the application. Different ontologies describe the world in different ways.
- People can fundamentally disagree about an appropriate structure.
- Different knowledge bases can use different ontologies.
- To allow KBs based on different ontologies to inter-operate, there must be mapping between ontologies.
- It has to be in user's interests to use an ontology.
- The computer doesn't understand the meaning of the symbols. The formalism can constrain the meaning, but can't define it.

Semantic Web Technologies Revisited

- RDF the Resource Description Framework is a language of triples, including the property `rdf:type` and containers (bags, lists, etc)
- RDF-S RDF Schema is RDF plus the class: `rdfs:Class`, and properties: `rdfs:domain`, `rdfs:range`, `rdfs:subClassOf`, `rdfs:subPropertyOf`, ...
- Lots of alternative syntaxes: XML, Turtle, N-Triples, Json ...
- OWL the Web Ontology Language, defines some primitive properties that can be used to define terminology. (Uses multiple alternative syntaxes).
- SPARQL Query Language for RDF
- SWRL Semantic web rule language

Main Components of an Ontology

- Individuals the things / objects in the world (not usually specified as part of the ontology)
- Classes sets of individuals
- Properties between individuals and their values

Individuals

- Individuals are things that can be named.
- Unique names assumption (UNA): different names refer to different individuals.
- The UNA is not an assumption you can universally make: "Lewis Carroll", "Charles Lutwidge Dodgson", "the author of Alice's Adventures in Wonderland" etc.
- Without the determining equality, we can't count!
  Joe's mother is in the room. Sam's cousin is there. Chris's football coach is there. How many people are in the room?
- Using OWL:

  $(i_1, \text{'owl:SameIndividual'}, i_2)$

  $(i_1, \text{'owl:DifferentIndividuals'}, i_3)$

Classes

- A class is a set of individuals. E.g., house, officeBuilding
- One class can be a subclass of another

  (*house*, 'rdfs:SubClassOf', *building*)

  (*officeBuilding*, 'rdfs:SubClassOf', *building*)

  (or 'owl:subClassOf')
- The most general class is 'owl:Thing'.
- Classes can be declared to be the same or to be disjoint:

  (*house*, 'owl:EquivalentClasses', *singleFamilyDwelling*)

  (*house*, 'owl:DisjointClasses', *officeBuilding*)
- Different classes are not necessarily disjoint.
  E.g., a building can be both a commercial building and a residential building.

see `http://www.cs.ubc.ca/~poole/cs312/2024/prolog/sem_web_schema.pl`

Example Concepts in an Ontology

The following are some of the concepts in an ontology for documents.
`http://www.cs.umd.edu/projects/plus/DAML/onts/docmnt1.0.daml`

homepage	correspondence	publication
letter	periodical	article
book	email	magazine
journal	document	communication
workshopPaper	journalPaper	discussion
newspaper	PersonalHomepage	speech

Properties

- A property is between an individual and a value.
- A property has a domain and a range.

  rdfs:domain(*livesIn*, *person*)

  rdfs:range(*livesIn*, *placeOfResidence*)
- An *ObjectProperty* is a property whose range is an individual.
- A *DatatypeProperty* is one whose range isn't an individual, e.g., is a number or string.
- There can also be property hierarchies:

  rdfs:subPropertyOf(*livesIn*, *enclosure*)

  rdfs:subPropertyOf(*principalResidence*, *livesIn*)

Clicker Question

Suppose we are given the following triple as true:

```
years_eligibility 'rdfs:domain' student.
sam  years_eligibility  3.
```

Which is the following can we infer

A Sam is a student

B Sam could a student (but maybe isn't)

C All students have value 3 for `years_eligibility`

D We can infer nothing about whether Sam is a student

Clicker Question

Suppose we are given the following triples as true:

```
years_eligibility  'rdfs:domain'  student.
years_eligibility 'rdfs:domain' athlete.
sam years_eligibility 3.
```

Which is the following is true

A Sam is both a student and an athlete.

B Sam could be either student or an athlete.

C We can infer nothing about whether Sam is an athlete or a student

D There are no student athletes.

E The facts are inconsistent, and couldn't possible all be true

Clicker Question

RDF-schema provides a vocabulary for classes and properties. RDF-schema has a syntax for *domain* and *range* of a property. schema.org does not use `rdfs:domain` and `rdfs:range`. Why?

A The scheme.org designers didn't know about it even though they used other terminology from RDF-schema

B The scheme.org designers didn't care about domains and ranges because they just wanted to define a vocabulary.

C schema.org does not define anything, and so does not need domain and ranges

D The scheme.org designers did not want the meaning associated with RDF-schema's domain and range.

Properties (Cont.)

- One property can be inverse of another

  owl:InverseObjectProperties(*livesIn*, *hasResident*)

- Properties can be declared to be transitive, symmetric, functional, or inverse-functional.
  (Which of these are only applicable to object properties?)
- We can also state the minimum and maximal cardinality of a property.

  owl:minCardinality(*principalResidence*, 1)

  owl:maxCardinality(*principalResidence*, 1)

Property and Class Restrictions

- We can define complex descriptions of classes in terms of restrictions of other classes and properties.
  E.g., A homeowner is a person who owns a house.

$$homeOwner \subseteq person \cap \{x : \exists h \in house \text{ such that } x \text{ owns } h\}$$

owl:subClassOf(homeOwner,person)

owl:subClassOf(*homeOwner*,
owl:ObjectSomeValuesFrom(*owns*, *house*))

OWL Class Constructors

owl:Thing $\equiv$ all individuals

owl:Nothing $\equiv$ no individuals

owl:ObjectIntersectionOf$(C_1, \ldots, C_k) \equiv C_1 \cap \cdots \cap C_k$

owl:ObjectUnionOf$(C_1, \ldots, C_k) \equiv C_1 \cup \cdots \cup C_k$

owl:ObjectComplementOf$(C) \equiv \mathit{Thing} \setminus C$

owl:ObjectOneOf$(I_1, \ldots, I_k) \equiv \{I_1, \ldots, I_k\}$

owl:ObjectHasValue$(P, I) \equiv \{x : x\ P\ I\}$

owl:ObjectAllValuesFrom$(P, C) \equiv \{x : x\ P\ y \rightarrow y \in C\}$

owl:ObjectSomeValuesFrom$(P, C) \equiv$

$\{x : \exists y \in C$ such that $x\ P\ y\}$

owl:ObjectMinCardinality$(n, P, C) \equiv$

$\{x : \#\{y | xPy$ and $y \in C\} \geq n\}$

owl:ObjectMaxCardinality$(n, P, C) \equiv$

$\{x : \#\{y | xPy$ and $y \in C\} \leq n\}$

OWL Predicates

owl:EquivalentClasses$(C_1, C_2) \equiv C_1 \equiv C_2$
owl:DisjointClasses$(C_1, C_2) \equiv C_1 \cap C_2 = \{\}$
owl:EquivalentObjectProperties$(P_1, P_2) \equiv xP_1y$ if and only if xP_2y
owl:DisjointObjectProperties$(P_1, P_2) \equiv xP_1y$ implies not xP_2y
owl:InverseObjectProperties$(P_1, P_2) \equiv xP_1y$ if and only if yP_2x
owl:SameIndividual$(I_1, \ldots, I_n) \equiv \forall j \forall k \; I_j = I_k$
owl:DifferentIndividuals$(I_1, \ldots, I_n) \equiv \forall j \forall k \; j \neq k$ implies $I_j \neq I_k$
owl:FunctionalObjectProperty$(P) \equiv$ if xPy_1 and xPy_2 then $y_1 = y_2$
owl:InverseFunctionalObjectProperty$(P) \equiv$
if x_1Py and x_2Py then $x_1 = x_2$
owl:TransitiveObjectProperty$(P) \equiv$ if xPy and yPz then xPz
owl:SymmetricObjectProperty $\equiv$ if xPy then yPx