Model-Theoretic Semantics

Natalie Parde
UIC CS 421

Model-Theoretic Semantics

What do most meaning representation schemes share in common?

- An ability to represent objects, properties of objects, and relations among objects (symbols)

A **model** is a formal construct that stands for a particular state of affairs in the world that we're trying to represent

Expressions (words or phrases) in the meaning representation language can be mapped to elements of the model

Relevant Terminology

- Vocabulary
  - **Non-Logical Vocabulary:** Open-ended sets of names for objects, properties, and relations in the world we're representing
  - **Logical Vocabulary:** Closed set of symbols, operators, quantifiers, links, etc. that provide the formal means for composing expressions in the language
- **Domain:** The set of objects that are part of the state of affairs being represented in the model
- **Each object in the non-logical vocabulary corresponds to a unique element in the domain**; however, each element in the domain does not need to be mentioned in a meaning representation

Additional Terminology

- For a given domain, **objects** are elements
  - grapes, violets, plums, CS421, Mina, Mohammad
- **Properties** are sets of elements corresponding to a specific characteristic
  - purple = {grapes, violets, plums}
- **Relations** are sets of tuples, each of which contain domain elements that take part in a specific relation
  - StudentIn = {(CS421, Ankit), (CS421, Nimeesha)}

How do we create mappings from non-logical vocabulary to formal **denotations**?

We create functions (interpretations)!

Example Application

- Assume that we have:
  - A collection of restaurant patrons and restaurants
  - Various facts regarding the likes and dislikes of patrons
  - Various facts about the restaurants
- In our current state of affairs (our **model**) we're concerned with four patrons designated by the non-logical symbols (**elements**) *Natalie*, *Usman*, *Nikolaos*, and *Mina*
- We'll use the constants *a*, *b*, *c*, and *d* to refer to those respective elements

Example Application

patron = {Natalie, Usman, Nikolaos, Mina} = {a, b, c, d}

- We’re also concerned with three restaurants designated by the non-logical symbols *Giordano’s*, *IDOF*, and *Artopolis*
- We’ll use the constants *e*, *f*, and *g* to refer to those respective elements

Example Application

patron = {Natalie, Usman, Nikolaos, Mina} = {a, b, c, d}

restaurants = {Giordano's, IDOF, Artopolis} = {e, f, g}

- Finally, we'll assume that our model deals with three cuisines in general, designated by the non-logical symbols *Italian*, *Mediterranean*, and *Greek*
- We'll use the constants *i*, *j*, and *k* to refer to those elements

Example Application

patron = {Natalie, Usman, Nikolaos, Mina} = {a, b, c, d}

restaurants = {Giordano's, IDOF, Artopolis} = {e, f, g}

cuisines = {Italian, Mediterranean, Greek} = {i, j, k}

- Now, let's assume we need to represent a few properties of restaurants:
  - *Fast* denotes the subset of restaurants that are known to make food quickly
  - *TableService* denotes the subset of restaurants for which a waiter will come to your table to take your order
- We also need to represent a few relations:
  - *Like* denotes the tuples indicating which restaurants individual patrons like
  - *Serve* denotes the tuples indicating which restaurants serve specific cuisines

Example Application

patron = {Natalie, Usman, Nikolaos, Mina} = {a, b, c, d}

restaurants = {Giordano's, IDOF, Artopolis} = {e, f, g}

cuisines = {Italian, Mediterranean, Greek} = {i, j, k}

Fast = {f}
TableService = {e, g}
Likes = {(a, e), (a, f), (a, g), (b, g), (c, e), (d, f)}
Serve = {(e, i), (f, j), (g, k)}

- This means that we have created the domain D = {a, b, c, d, e, f, g, i, j, k}
- We can evaluate representations like *Natalie likes IDOF* or *Giordano's serves Greek* by mapping the objects in the meaning representations to their corresponding domain elements, and any links to the appropriate relations in the model
  - Natalie likes IDOF → a likes f → Like(a, f) 🙂
  - Giordano's serves Greek → e serves k, Serve(e, k) 🤨

Example Application

patron = {Natalie, Usman, Nikolaos, Mina} = {a, b, c, d}

restaurants = {Giordano’s, IDOF, Artopolis} = {e, f, g}

cuisines = {Italian, Mediterranean, Greek} = {i, j, k}

Fast = {f}
TableService = {e, g}
Likes = {(a, e), (a, f), (a, g), (b, g), (c, e), (d, f)}
Serve = {(e, i), (f, j), (g, k)}

- Thus, we’re just using sets and operations on sets to ground the expressions in our meaning representations
- What about more complex sentences?
  - Nikolaos likes Giordano’s and Usman likes Artopolis.
  - Mina likes fast restaurants.
  - Not everybody likes IDOF.

Example Application

patron = {Natalie, Usman, Nikolaos, Mina} = {a, b, c, d}

restaurants = {Giordano’s, IDOF, Artopolis} = {e, f, g}

cuisines = {Italian, Mediterranean, Greek} = {i, j, k}

Fast = {f}
TableService = {e, g}
Likes = {(a, e), (a, f), (a, g), (b, g), (c, e), (d, f)}
Serve = {(e, i), (f, j), (g, k)}

- Plausible meaning representations for the previous examples will not map directly to individual entities, properties, or relations!
- They involve:
  - Conjunctions
  - Equality
  - Variables
  - Negations
- What we need are **truth-conditional semantics**
- This is where **first-order logic** comes in handy