Computational Semantics with Haskell

Yulia Zinova

Winter 2016/2017

We follow Van Eijck and Unger 2010, electronic access from the library

Architecture

- Predicate logic $\rightarrow$ semantic representation language
- Models of predicate logic $\rightarrow$ Haskell data types
- Interpreting predicate logic languages in appropriate models:
    1. construct a logical form from a natural language expression
    2. evaluate the logical form with respect to a model

Linguistic form to logic

- Funny properties:
  - *Alice walked on the road* implies that someone walked on the road
  - *No one walked on the road* does not imply that someone walked on the road
- So the structure of the two sentences must differ $\rightarrow$ first-order predicate logic
- Logical translation for
  Every dwarf loved Goldilocks.

Linguistic form to logic

- Funny properties:
  - *Alice walked on the road* implies that someone walked on the road
  - *No one walked on the road* does not imply that someone walked on the road
- So the structure of the two sentences must differ → first-order predicate logic
- Logical translation for *Every dwarf loved Goldilocks.*
- $\forall x (Dwarf\ x \rightarrow Love\ x\ g)$
- What is strange? What does the logical form say?

Linguistic form to logic

- Funny properties:
  - *Alice walked on the road* implies that someone walked on the road
  - *No one walked on the road* does not imply that someone walked on the road
- So the structure of the two sentences must differ $\rightarrow$ first-order predicate logic
- Logical translation for
  Every dwarf loved Goldilocks.
- $\forall x (Dwarf\ x \rightarrow Love\ x\ g)$
- What is strange? What does the logical form say?
- All objects in the domain of discourse have the property of either not being dwarfs or being objects who loved Goldilocks.

Linguistic form to logic

- Funny properties:
  - *Alice walked on the road* implies that someone walked on the road
  - *No one walked on the road* does not imply that someone walked on the road
- So the structure of the two sentences must differ $\rightarrow$ first-order predicate logic
- Logical translation for
  Every dwarf loved Goldilocks.
- $\forall x (Dwarf\ x \rightarrow Love\ x\ g)$
- What is strange? What does the logical form say?
- All objects in the domain of discourse have the property of either not being dwarfs or being objects who loved Goldilocks.
- The constituent *every dwarf* disappeared!

Believes

- Proper names and quantified noun phrases combine with a predicate in different ways
- Therefore, linguistic form of natural language is misleading
- But: if we use lambda calculus where natural language constituents correspond to typed expressions that combine with one another as functions and arguments
- As a result, fully unreduced expressions directly correspond to language elements and account for the observed differences

Representations with predicate logic

- Type of entities is represented by terms
- Type of truth values is represented by formulas
- `type LF = Formula Term`
- Our fragment: declarative sentences with meaning that can be represented with predicate logic

Representing rules

- Recall our English grammar fragment in BNF
- First rule **S** $\rightarrow$ **NP VP**
- Should we represent NP as a function that takes a VP representation as argument, or vice versa?
- VP representations must have a functional type, as VPs denote properties
- VP type: `Term` $\rightarrow$ `LF`
- Types for *Goldilocks* and *every boy*?

Representing rules

- Recall our English grammar fragment in BNF
- First rule **S** → **NP VP**
- Should we represent NP as a function that takes a VP representation as argument, or vice versa?
- VP representations must have a functional type, as VPs denote properties
- VP type: `Term` → `LF`
- Types for *Goldilocks* and *every boy*?
- Let us explore the representations...

Representing a model for predicate logic

- We need a domain of entities and suitable interpretations of names and predicates
- Domain: individuals `A ... Z` and `Unspec`
- Simple names are interpreted as entities
- Common nouns and intransitive verbs are interpreted as properties of entities

Predicates

- Transitive verbs are interpreted s relations between entities
- Define one-, two-, and three-place predicates
- *Currying* is the conversion of a function of type `((a,b) → c)` to one of type `a → b → c`
- Uncurrying is the converse operation.
- *curry* and *uncurry* are predefined in Prelude
- Passivization: the agent of the action is dropped

Exercises

- Consider the verbs *help* and *defeat* and the noun phrases *Alice*, *Snow White*, *every wizard*, *a dwarf*. For every sentence of the form NP (V NP) with these items check whether it is true of false in the given model.
- Check how `passivize` works by applying it to the predicates `admire` and `help`.
- Define another passivization function that works for three-place predicates.

Evaluating formulas in models

- Up to now we specified how to represent models for predicate logic.
- The next thing is to evaluate formulas with respect to these models.
- We need interpretation functions and variable assignments
- One interpretation function for relation of different arities
- An interpretation function is a function from relation names to appropriate relations in the model

Variable assignments

- Now we need to implement variable assignments (variable lookup)
- Example of variable assignment: ass0 - map every variable to object A
- ass1 - take ass0 but map y to B
- Can be modified further

Domain and the evaluation function

- Two assumptions: allows tests for equality, can be enumerated
- To check an infinite domain: as Haskell only evaluates something when it is needed, an open list can be an argument, but “forall" is not possible

References:

Van Eijck, J. and Unger, C. (2010). *Computational semantics with functional programming*. Cambridge University Press.