Basic composition, the typed λ-calculus

Announcement: Please post textbook issues/questions in the Luminus forum.

Review

- Meaning as truth conditions
  - Sentences are either true 1 or false 0, *given a particular model*.
  - p "entails" q if and only if, for any model M, if $[\![p]\!]^M = 1$ then $[\![q]\!]^M = 1$
- D_e is the domain of individuals
- We can think of predicates...

  1. as a function from $D_e \rightarrow \{0, 1\}$:

     (1) $[\![\text{sleep}]\!]^M(x) = \begin{cases} 1 & \text{if } x \text{ sleeps in } M \\ 0 & \text{otherwise} \end{cases}$ (with some abuse of notation, we will clarify below)

  2. as the set of individuals that satisfy the predicate

  Nouns like *cat* and *student* are also predicates like *sleep*: functions from $D_e \rightarrow \{0, 1\}$.

► Today we'll concentrate on their meaning as functions.

1 Functions and semantic types

Every linguistic expression has a *semantic type*:

- Terms (which are individuals) are type e and in D_e e.g. proper names
- Wffs (which have truth values) are type t and in $D_t = \{0, 1\}$ e.g. sentences
- A function from type τ to σ is type $\langle\tau, \sigma\rangle$ and in $D_{\langle\tau,\sigma\rangle}$

Semantic type does not simply map to syntactic category. Where do they come apart?

2 λ notation

Functions in math classes are often defined by saying, for example, $f(x) = x + 1$ and x must be a number (in D_n). We will use **λ notation** for defining functions:

(2) $f = \lambda x \;.\; x + 1$

For example, $f(5) = [\lambda x \;.\; x + 1](5) = 5 + 1 = 6$. Applying a function to an argument means "replacing" instances of the *outermost* λ variable with the argument (5) in the value description.

We can be more specific and clarify that arguments of f need to be of type n:

(3) $\lambda x_{\boldsymbol{n}} \,.\, x + 1$

If the function's argument is not of the right type, the result is undefined. For example, f(John) is undefined because John $\notin D_n$.

Exercises

(4) $g = [\lambda x \,.\, \lambda y \,.\, y \times (x-1)]$. Compute $\big(g(3)\big)\,(4)$.

(5) What type is $\lambda x_e \,.\, \lambda y_e \,.\, x = y$?

(6) What type is $\lambda P_{\langle e,t\rangle} \,.\, \lambda Q_{\langle e,t\rangle} \,.\, \forall x[P(x) \rightarrow Q(x)]$?

(7) Compute: $[\lambda f_{\langle e,t\rangle} \,.\, [\lambda x_e \,.\, f(x) \wedge \text{Gray}(x)]]\,\big([\lambda y_e \,.\, \text{Cat}(y)]\big)$

Note: Where there are two λ terms next to each other, some authors leave out the period . : "$\lambda x \lambda y \,.\, x = y$" is equivalent to "$\lambda x \,.\, \lambda y \,.\, x = y$" .

3 A note on the our mapping(s) and *IFS* notation

Our goal is to develop a systematic mapping between *linguistic form* and *meanings*, which we can think of as truth conditions for declarative sentences. We do this in two steps:

linguistic expression	predicate logic expression	example meaning in model/world
love	$\lambda y \,.\, \lambda x \,.\, \text{Love}(x,y)$	$\{\langle\text{Fido},\text{Mary}\rangle, \langle\text{Pochi},\text{Taro}\rangle\}$
Pochi is a dog	Dog(Pochi)	1
Agneta (the word)	Agneta / ag (the symbol)	Agneta (the person)

The textbook *IFS* uses $\rightsquigarrow$ for the first step and $[\![...]\!]^M$ for the second step:

linguistic expression		predicate logic expression		example meaning in M
love	$\rightsquigarrow$	$\lambda y \,.\, \lambda x \,.\, \text{Love}(x,y)$		
		$[\![\text{Love}]\!]^M$	$=$	$\{\langle\text{Fido},\text{Mary}\rangle, \langle\text{Pochi},\text{Taro}\rangle\}$

In this class we will use the notation $[\![...]\!]$ for both steps!

linguistic expression		predicate logic expression		example meaning in M
$[\![\text{love}]\!]$	$=$	$\lambda y \,.\, \lambda x \,.\, \text{Love}(x,y)$		
		$[\![\text{Love}]\!]^M$	$=$	$\{\langle\text{Fido},\text{Mary}\rangle, \langle\text{Pochi},\text{Taro}\rangle\}$

Practically, in this class we will often just be doing the first step — translating the linguistic expression to its corresponding predicate logic expression.

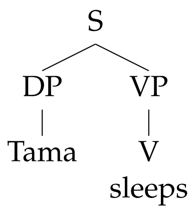
Practical advice:

- When reading *IFS*: if you see "expression $\rightsquigarrow$ blah", it's always the first step. You can think of it as "$\llbracket$expression$\rrbracket$ = blah".[1]
- When reading something I write, or some other semanticist writes: if you see $\llbracket ... \rrbracket$, look at what's inside. If it's a linguistic expression, then it's the first step and the output should be a predicate logic expression (or equivalent); if it's a predicate logic expression, and a model M is specified, the result should be its meaning in the model M.

4 Composition

(8) **The Principle of Compositionality:** The meaning of a linguistic expression is built of the meaning of its constituent parts, in a systematic fashion.

(9)



A recursive definition for the interpretation function $\llbracket ... \rrbracket$ (first step):

(10) **Terminal Nodes (TN):**
If α is a terminal node, $\llbracket \alpha \rrbracket$ is specified in the lexicon.

(11) **Non-branching Nodes (NN):**
If α is a non-branching node, and β is its daughter node, then $\llbracket \alpha \rrbracket = \llbracket \beta \rrbracket$.

(12) **Functional Application (FA):**
If α is a branching node, $\{\beta, \gamma\}$ is the set of α's daughters, and $\llbracket \beta \rrbracket$ is a function whose domain contains $\llbracket \gamma \rrbracket$, then $\llbracket \alpha \rrbracket = \llbracket \beta \rrbracket(\llbracket \gamma \rrbracket)$.

[1] Later they switch to $\langle\langle ... \rangle\rangle$ for the first step; again, you can think of $\langle\langle$expression$\rangle\rangle$ as $\llbracket$expression$\rrbracket$ (first step).

Some hints for computing the denotation of complex structures:

1. Start with a tree. (We ignore DP-internal structure for names and, for now, model the sentence as S, not TP/IP.)
2. Annotate each node with its semantic type.

   (13) **The Triangle Method:**

   Look at projections like $\begin{array}{c}\alpha\\ \frown\\ \beta\ \ \gamma\end{array}$ as triangles. Like the angles on a triangle, if you know two of the three, you should be able to determine the type of the third "corner." Sometimes there will be two or three options, but at least you will know what the limited set of possibilities are.

3. Compute the denotation of each node in the tree. You can work bottom-up or top-down. For each node, give the rule that is being used (TN, NN, FA, etc.) to obtain the denotation.

5 More examples

(14) John loves Mary.

(15) Are these the same?

a. $[\![\text{love}]\!] = \lambda x\ .\ \lambda y\ .\ \text{Love}(y, x)$

b. $[\![\text{love}]\!] = \lambda y\ .\ \lambda x\ .\ \text{Love}(x, y)$

c. $[\![\text{love}]\!] = \lambda x\ .\ \lambda y\ .\ \text{Love}(x, y)$

Following *IFS*, read two-place functions in subject–object order: in $R(x, y)$, x is the subject and y is the object.

(16) a. It-is-not-the-case-that Tama sleeps.

b. Tama does not sleep.

(17) $[\![\text{does}]\!]$ = Id (the identity function)

$= \lambda P\ .\ P$ for any type of argument

I use "Id" as a shorthand: Id $\in D_{\langle\tau,\tau\rangle}$ for any type τ

(18) John introduced Mary to Bill.

(19) **Binary branching:** Every branching node will have exactly two daughters.