Basic composition, the typed λ-calculus

1 Review

- Meaning as truth conditions
  - Sentences are either true 1 or false 0, *given a particular model*.
  - S_1 entails S_2 if and only if $[\![S_1]\!]^M \leq [\![S_2]\!]^M$ for any model M
- D_e is the domain of individuals
- $[\![...]\!]$ is the *interpretation function*: it takes a linguistic expression and returns its denotation
- We can think of predicates as sets: $[\![\text{sleep}]\!]^M$ = the set of individuals that sleep in M
- Nouns like *cat* and *student* are also predicates like *sleep*: the set of cats or the set of students or a corresponding function from $D_e \to \{0, 1\}$.
- Quantificational determiners (*no*, *every*, *some*,...) can be thought of as a function taking two sets and returning a truth value.
- From the problem set: It appears that many quantificational determiners are *conservative*. Barwise and Cooper (1981) claim that all quantificational determiners in natural language are conservative; i.e. no language has a determiner like *allnon*. (The analysis of words like *only* then become a problem...)

2 Functions and semantic types

We can also think about predicates as functions from individuals to truth values:

(1) $$[\![\text{sleep}]\!]^M(x) = \begin{cases} 1 & \text{if } x \text{ sleeps in } M \\ 0 & \text{otherwise} \end{cases}$$

$[\![\text{sleep}]\!]^M$ is ambiguous between a *set denotation* and a *function denotation*. We will think mostly about function denotations today.

Every linguistic expression has a *semantic type*:

- Individuals are type e and in D_e — e.g. proper names
- 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?

Types in Winter (2016) and Heim and Kratzer (1998) :

Winter (2016) uses $\tau\sigma$ or $(\tau\sigma)$ as the type of functions in $D_{\langle\tau,\sigma\rangle}$.

Here are some types and their equivalents:

Winter	Heim and Kratzer
(et)t	$\langle\langle e,t\rangle,t\rangle$
(et)(et)	$\langle\langle e,t\rangle,\langle e,t\rangle\rangle$
(et)((et)t)	$\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle$

I will use the Heim and Kratzer (1998) style, which is more common in the literature.

3 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 should be more specific and clarify that arguments of f need to be in D_n:

(3) $f = \lambda x : x \in D_n \,.\, x+1$ $\qquad f = \lambda \underbrace{x}_{\text{argument variable}} : \underbrace{x \in D_n}_{\text{domain condition}} \,.\, \underbrace{x+1}_{\text{value description}}$

If the domain condition is not met, the result is undefined. For example, f(John) is undefined because John $\notin D_n$. We can also use this notation for functions like (1):

(4) $[\![\text{sleep}]\!]^M = \lambda x : x \in D_e \,.\, (1 \text{ iff } x \text{ sleeps in } M)$

$[\![\text{sleep}]\!]^M$ takes an argument of type e and returns a value of type t, so $[\![\text{sleep}]\!]$ is type $\langle e,t\rangle$.

Exercises

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

(6) What type is $[\lambda x : x \in D_e \,.\, \lambda y : y \in D_e \,.\, (1 \text{ iff } x = y)]$?

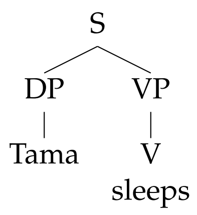
(7) What type is $[\lambda P : P \in D_{\langle e,t\rangle} \,.\, \lambda Q : Q \in D_{\langle e,t\rangle} \,.\, (1 \text{ iff } \forall x. \text{ if } P(x) \text{ true, then } Q(x) \text{ true})]$?

(8) Compute:
$[\lambda f : f \in D_{\langle e,t\rangle} \,.\, [\lambda x : x \in D_e \,.\, 1 \text{ iff } f(x) = 1 \text{ and } x \text{ is gray}]]\ ([\lambda y : y \in D_e \,.\, 1 \text{ iff } y \text{ is a cat}])$

4 Composition

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

(10)



A recursive definition for the interpretation function $[\![...]\!]$:

(11) **Terminal Nodes (TN):**
If α is a terminal node, $[\![\alpha]\!]$ is specified in the lexicon.

(12) **Non-branching Nodes (NN):**
If α is a non-branching node, and β is its daughter node, then $[\![\alpha]\!] = [\![\beta]\!]$.

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

Note: Winter (2016) does not use Non-branching Nodes (NN).

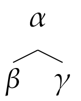
What kind of words have denotations that are model-sensitive — i.e. there are models M_1 and M_2 such that $[\![\alpha]\!]^{M_1} \neq [\![\alpha]\!]^{M_2}$? What kinds of words have denotations that are model-insensitive?

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.

   (14) **The Triangle Method:**

   Look at projections like 

 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. (See discussion of *type equations* in Winter.)

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 Four shortcuts people take with λ notation:

1. If the function returns a truth value, instead of writing "1 iff [condition]," just write "[condition]": $[\![\text{sleep}]\!]^M = \lambda x : x \in D_e \,.\, x$ sleeps in M

   But important: $[\![\text{Tama sleeps}]\!]^M = [\lambda x : x \in D_e \,.\, x \text{ sleeps in } M](\text{Tama}) = 1$

   iff Tama sleeps in M

   In other words, the "1 iff" *reappears* when describing the resulting truth value of type t. This part can be confusing — it's discussed in H&K pages 36–37.
2. If the domain condition is of the form $x \in$..., then just add it to the argument variable: $[\![\text{sleep}]\!] = \lambda x \in D_e \,.\, x$ sleeps
3. If the domain condition is of the form $x \in D_e$, just leave it off. The default type for arguments is type e: $[\![\text{sleep}]\!] = \lambda x \,.\, x$ sleeps
4. If the domain condition is of the form $x \in D_\tau$, then just add the type as a subscript $_\tau$ to the variable: $[\![\text{sleep}]\!] = \lambda x_e \,.\, x$ sleeps

   H&K does *not* use this last shortcut, but you see it in the literature.

6 More examples

(15) John loves Mary.

(16) Are these the same?

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

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

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

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

b. Tama does not sleep.

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

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

Id $\in D_{\langle \tau,\tau \rangle}$ for any type τ

(19) John introduced Mary to Bill.

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

References

Barwise, Jon, and Robin Cooper. 1981. Generalized quantifiers and natural language. *Linguistics and Philosophy* 4:159–219.

Heim, Irene, and Angelika Kratzer. 1998. *Semantics in generative grammar*. Malden, Massachusetts: Blackwell.

Winter, Yoad. 2016. *Elements of formal semantics: An introduction to the mathematical theory of meaning in natural language*. Edinburgh University Press.