Basic composition, the typed λ-calculus

1 Review

- Meaning as truth conditions
  - Sentences are either true (1) or false (0), *given a particular model/world/situation*.
- D_e is the domain of individuals
- $[\![...]\!]$ is the *interpretation function*: it takes a linguistic expression and returns its denotation (meaning)
- We can think of predicates like *sleep* in two ways:
  1. as a function from $D_e \rightarrow \{0, 1\}$:

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

  2. as the set of individuals that satisfy the predicate

  Today we'll concentrate on their meaning as functions.
- Nouns like *cat* and *student* are also predicates like *sleep*: the set of cats or the set of students or a corresponding function from $D_e \rightarrow \{0, 1\}$.

2 Semantic types

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?

Contingent sentences have denotations of the form "1 iff [condition]." I call that a *conditional truth value* and we will treat it as type t, in D_t.

3 λ notation

Functions in math classes are often defined by saying, for example, $f(x) = x + 1$ and x must be a real number (in $\mathbb{R}$). 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 $\mathbb{R}$:

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

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

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

$[\![\text{sleep}]\!]$ 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 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 iff $\forall x$. if $P(x)$ true, then $Q(x)$ true)]?

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

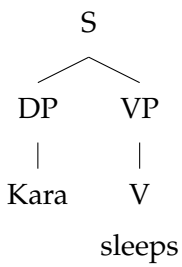
4 Composition

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

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

(11) **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 $[\![\alpha]\!] = [\![\beta]\!]([\![\gamma]\!])$.

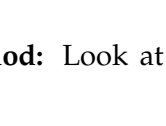
(12)



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  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 Four shortcuts people take with λ notation:

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

   But important: $[\![\text{Kara sleeps}]\!] = [\lambda x : x \in D_e \,.\, x \text{ sleeps}](\text{Kara}) = (1 \text{ iff Kara sleeps})$
   In other words, the "1 iff" *reappears* when describing a conditional truth value of type t.
   This part can be confusing — it's discussed in H&K pages 36–27.

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

(14) John loves Mary.

(15) 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

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

b. Kara does not sleep.

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

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

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

(18) John introduced Mary to Bill.

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