Basic composition, the typed λ-calculus

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

  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 $D_{\langle \tau, \sigma \rangle}$

(2) Some examples:

a.	$[\![\text{Tama}]\!]$	type e	d.	$[\![\text{green dog}]\!]$	type $\langle e, t \rangle$
b.	$[\![\text{sleep}]\!]$	type $\langle e, t \rangle$	e.	$[\![\text{saw}]\!]$	type $\langle e, \langle e, t \rangle \rangle$
c.	$[\![\text{dog}]\!]$	type $\langle e, t \rangle$	f.	$[\![\text{saw Dana}]\!]$	type $\langle e, t \rangle$

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.

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:

(3) $f = \lambda x \,.\, x + 1$

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 in D_n:

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

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

Exercise:

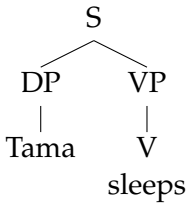
(5) $g = [\lambda x \;.\; \lambda y \;.\; y \times (x-1)]$. Compute $(g(3))\,(4)$.

3 Composition

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

(7) ⟦Tama sleeps⟧ = Sleep(Tama)

(8)



(I refer to noun phrases as DP (Determiner Phrase) as H&K does.)

A recursive definition for the interpretation function ⟦...⟧ (first step):

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

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

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.

(12) **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.

4 Notes on notation

Denotations in Heim and Kratzer 1998:

Heim and Kratzer 1998 — which we will generally follow from hereon out — writes denotations in terms of truth conditions written in English, rather than in predicate logic:

	Heim and Kratzer	Predicate logic style
⟦Tama sleeps⟧	1 iff Tama sleeps	Sleep(Tama)
⟦sleeps⟧	λx . 1 iff x sleeps	λx . Sleep(x)
	(or: λx . x sleeps[1])	

We will write denotations in predicate logic style in this class.

Three shortcuts people take with λ notation:

1. If the domain condition is of the form $x \in$..., then just add it to the argument variable: ⟦sleep⟧ = $\lambda x \in D_e$. Sleep(x)
2. If the domain condition is of the form $x \in D_e$, just leave it off. The default type for arguments is type e: ⟦sleep⟧ = λx . Sleep(x)
3. If the domain condition is of the form $x \in D_\tau$, then just add the type as a subscript $_\tau$ to the variable: ⟦sleep⟧ = λx_e . Sleep(x)

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

In other words, all three variants above are equivalent to saying ⟦sleep⟧ = $\lambda x : x \in D_e$. Sleep(x)

[1] H&K uses a convention where, for functions that return a truth value, they just write "[condition]" to mean "1 iff [condition]." But sentences (type t) themselves always are written with "1 iff...". This part can be confusing — it's discussed in H&K pages 36–37. But for the purposes of our class, because we will write denotations in predicate logic, we can largely avoid this complication.

5 More examples

(13) Brie loves Cara.

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

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

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

b. Tama does not sleep.

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

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

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

(17) Brie introduced Cara to Dana.

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

References

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.