Lambda-notation for functions

1 Basics

In formal semantics, it is customary to use **lambda-notation** for functions, instead of lengthy descriptions in English like 'the function that takes an individual x and returns 1 iff x smokes in M.' Everything we will do in this course can as well be done without lambda-notation, but it would be too cumbersome to always use English to talk about functions, so let's get used to lambda-notation at this point.

In the standard lambda-notation used in formal semantics,[1] a function generally looks like:

$$\lambda v\colon\ \phi.\ \alpha$$

- The symbol λ (the Greek letter 'lambda') has no meaning on its own. It's there to indicate that this whole formal expression names a function.
- v is a variable, representing the input of the function. It's a variable because its value varies depending on what the input is. You are allowed to pick any variable name, but we will use x, y, f, P, etc. in this couse.
- ϕ is a statement describing what kind of input the function admits (typically in terms of v). This means that ϕ defines the domain of the function. Keep in mind that ϕ needs to be a statement that can be true or false. The idea is that if ϕ is true, the input is an appropriate kind of object for the function to operate on, and if ϕ is false, it is not and the function will not output anything.
- α describes the output, often in terms of v.

It's easier to understand this by looking at concrete examples. Here is one:

(1) $\lambda x\colon\ x$ is a natural number. $x + 5$

This is a function that takes a natural number and adds 5 to it. You are probably more familiar with the notation $f(x) = x + 5$. (1) is exactly the same function. You might wonder why we need another way of writing the same thing. As you will see later in this module, lambda-notation becomes particularly convenient in the context of formal semantics, because it makes it clear that you are referring to the function itself, rather than the output value of f applied to x (which we will consistently denote by '$f(x)$'). Also, in lambda-notation the domain of the function can be explicitly stated. In (1), it's the part that says 'x is a natural number'. As we will see, this will be very useful for analyzing natural language expressions.

You can also write the same function more compactly as (2). Recall $\mathbb{N}$ is the set of natural numbers.

(2) $\lambda x\colon\ x \in \mathbb{N}.\ x + 5$

Functions like (2) whose domain is expressed as the variable x being a member of some set S are often abbreviated to '$\lambda x \in S.\ \alpha$'. For instance, (2) can be written as:

[1] Linguists borrowed this notation from lambda calculus, a formal system for functions developed by Alonzo Church.

(3) $\lambda x \in \mathbb{N}.\ x + 5$

We will adopt this abbreviation in this module. In this shorthand, you no longer see ':', but don't be tricked by it. Every well-formed description of a function in lambda notation is made up of the same aforementioned three components, a variable, a description of the domain, and a description of the output (plus the λ at the beginning).

The choice of the variable x above is not important, because it is a variable. We could have chosen, say, y instead, as in (4), and meant the same thing.

(4) $\lambda y \in \mathbb{N}.\ y + 5$

Usually, we use x, y, z as variable names, especially when they stand for arbitrary entities, but in principle they can be any letter. It is, for example, legitimate to use a completely arbitrary symbol as in:

$$\lambda \sqrt{\dagger_{\Sigma}^{\bullet}} \in \mathbb{N}.\ \sqrt{\dagger_{\Sigma}^{\bullet}} + 5$$

but if there's no reason to complicate the representation, you'd better use simple symbols for variables.

Functions need not be about mathematical operations. For instance, (5) is a legitimate function (assuming that each dog has a unique name).

(5) $\lambda y\colon$ y is a dog in London. y's name

This is a function that takes any dog in London and returns its name.

Using lambda-notation, we can re-write the denotations of the intransitive verbs we analysed last week as follows.

(6) For any model M,
  a. $[\![\texttt{smokes}]\!]^M = \lambda x\colon$ x is an entity in M. 1 iff x smokes in M
  b. $[\![\texttt{cried}]\!]^M = \lambda x\colon$ x is an entity in M. 1 iff x cried in M

The set of all entities in a given model (called the domain of the model) is often written as D. So, these functions are more compactly written as:

(7) For any model M,
  a. $[\![\texttt{smokes}]\!]^M = \lambda x \in D.\ 1$ iff x smokes in M
  b. $[\![\texttt{left}]\!]^M = \lambda x \in D.\ 1$ iff x left in M

From now on, we will use these representations.

(If you read published papers in formal semantics, '1 iff' is often also omitted, but we will keep it here, because it seems to take some time to get use to this convention.)

2 Functions that return functions

Things seem to get a bit more complicated, when you consider functions that return functions as values, but the basic principles are the same. For example, we can define a function A that takes a natural number n, and returns a function, which in turn takes a natural

number m and returns the value $n + m$. As you can see, A represents addition, but it takes one number at a time. In particular, when you feed A with one natural number, you will get another function. For example, $A(3)$ and $A(5)$ will be both functions, but importantly, $A(3) \neq A(5)$, because $A(3)$ is the function that takes a natural number and adds 3 to it, while $A(5)$ is the function that takes a natural number and adds 5 to it. In the λ-notation, we can write them as:

(8) a. $A(3) = \lambda m \in \mathbb{N}.\ m + 3$
b. $A(5) = \lambda m \in \mathbb{N}.\ m + 5$

Lambda-notation allows you to represent A itself as well. It will look like (9). We will use '[' and ']' to indicate where each function starts and ends.

(9) $A+ = [\lambda n \in \mathbb{N}.\ [\lambda m \in \mathbb{N}.\ n + m]]$

This representation of A looks as if A has two λ's, but that's not the right way of understanding it. Recall that in lambda-notation, each function has three parts (apart from λ). The first two parts are collapsed into '$\lambda n \in \mathbb{N}$' here, which says that it takes as input any member n of the set $\mathbb{N}$, which is to say that it takes any natural number n. Then the output description specifies what function it turns, namely $[\lambda m \in \mathbb{N}.\ n + m]$. So, if $n = 3$, it returns $[\lambda m \in \mathbb{N}.\ 3 + m]$, if $n = 5$, it returns $[\lambda m \in \mathbb{N}.\ 5 + m]$, and so on. These resulting functions in turn have three parts. They take any member m of $\mathbb{N}$, i.e. any natural number, and returns $3 + m$, $5 + m$, etc.

In Lecture 4, we will analyse the denotations of transitive verbs as functions that map entities to functions, which in turn map entities to truth-values. For example, the denotation of *love* with respect to an arbitrary model M (whose domain is D) can be written as (10).

(10) $[\![\texttt{love}]\!]^M = [\lambda x \in D.\ [\lambda y \in D.\ 1 \text{ iff } y \text{ loves } x \text{ in } M]]$

This will be explained in Lecture 4.

3 λ-conversion

When a function applies to an appropriate argument, you can simplify the representation. This process is often called 'λ-conversion' (or β-reduction). Here is an example of $[\![\texttt{smokes}]\!]^M$ applied to an entity in the model, say Alice.

$$\begin{aligned}[\![\texttt{smokes}]\!]^M(\text{Alice}) &= [\lambda x \in D.\ 1 \text{ iff } x \text{ smokes in } M](\text{Alice}) \\ &= 1 \text{ iff Alice smokes in } M\end{aligned}$$

Recall that we are following the standard convention that the function comes to the left of the argument and the argument is indicated by '(' and ')'. Generally, for any function f and for any argument a for it, we write $f(a)$ for the result of applying f to a. So things like '(John)$[\lambda x \in D.\ 1 \text{ iff } x \text{ smokes in } M]$' and '$(\lambda x \in D.\ 1 \text{ iff } x \text{ smokes in } M)$[John]' don't make sense. Such expressions are not part of our metalanguage, and should never be used.

Here are some more examples of λ-conversion.

(11) $[\lambda x \in \mathbb{N}.\ x + 5](12) = 5 + 12 = 17$

(12) $[\lambda x \in \mathbb{N}.\ x + 5]([\lambda y \in \mathbb{N}.\ y^2](3)) = [\lambda x \in \mathbb{N}.\ x + 5](3^2) = [\lambda x \in \mathbb{N}.\ x + 5](9) = 9 + 5 = 14$

Recall that a function can return a function. Consider the following function S. This function performs subtraction.

(13) $S = [\lambda n \in \mathbb{N}.\ [\lambda m \in \mathbb{N}.\ n - m]]$

When one argument, say 3, is given, it returns the following function.

$$S(3) = [\lambda n \in \mathbb{N}.\ [\lambda m \in \mathbb{N}.\ n - m]](3) = [\lambda m \in \mathbb{N}.\ 3 - m]$$

And here is a particularly important convention. When two arguments follow a function, the function is applied to the left one (= the one closer to the function) first.

$$S(3)(2) = [\lambda n \in \mathbb{N}.\ [\lambda m \in \mathbb{N}.\ n - m]](3)(2) = [\lambda m \in \mathbb{N}.\ 3 - m](2) = 3 - 2 = 1$$

Notice that $S(2)(3) = -1$, so $S(3)(2) \neq S(2)(3)$, meaning that it matters (in this case) which argument comes first.

Also remember that square brackets are used to delimit functions. Notice the following inequality:

$$[\lambda n \in \mathbb{N}.\ [\lambda m \in \mathbb{N}.\ n - m]](3)(2) \neq [\lambda n \in \mathbb{N}.\ [\lambda m \in \mathbb{N}.\ n - m](3)](2)$$

The left-hand side is 1, as calculated above. The right-hand side is -1, because:

$$[\lambda n \in \mathbb{N}.\ [\lambda m \in \mathbb{N}.\ n - m](3)](2) = [\lambda n \in \mathbb{N}.\ n - 3](2) = 2 - 3 = -1$$

Here, 3 is the argument of the inner function, and 2 is the argument of the outer function that contains the inner function. Here we applied the inner function to 3 first, but generally, the order of application does not change the final output (a property called the Church-Rosser property).

$$[\lambda n \in \mathbb{N}.\ [\lambda m \in \mathbb{N}.\ n - m](3)](2) = [\lambda m \in \mathbb{N}.\ 2 - m](3) = 2 - 3 = -1$$