F10 The λ-calculus

Course in Semantics · Ling 531 / 731

McKenzie · University of Kansas

1 Introducing

Before we move on, we will introduce a final piece of the notation puzzle. The lambda (or λ-)calculus is a method of formalizing functions that makes it easier to see how they combine. It was developed in the 1930's by Alonzo Church, a mathematician and pioneer in computer science.

1.1 Rewriting functions

The λ-calculus rewrites functions as a variable and a binder.

(1) $f(x) = x^2$ becomes $\lambda x.x^2$
assuming $f = \lambda x.x^2$

(2) $f(3) = 3^2 = 9$ becomes $[\lambda x.x^2](3) = 3^2 = 9$

How does it work for us?

(3) a. ⟦*smokes*⟧ = f : D → {1,0}
For all x, f(x) = 1 if and only if **x smokes**

b. ⟦*smokes*⟧ = λx ∈ D. smokes(x)

1.2 Breaking it down

Every λ-expression has three basic parts.

λx	∈ D	. smokes(x)
↑	↑	↑
argument variable	domain condition	value condition

READ: lambda x in D, smokes (of) x

- Any domain can be used in the domain condition
- Any variable can be used
- You can switch out variables, so long as you don't change what binds what

(4) λx ∈ D. smokes(x) = λy ∈ D. smokes(y)

1.3 Abbreviation

- We will write the value condition in function(argument) style.[1]
- The VC is an abbreviation of: 1 if x smokes and 0 otherwise
- If the function in the VC has more than one word, put it in brackets.

(5) ⟦*has cats*⟧ = λx $\in$ D. [has cats](x)

2 Functional Application with Lambda Calculus

One of the basic operations of the λ-calculus is β-reduction, which involves applying an argument to a function and reducing the λ-expression. β-reduction corresponds nicely to the compositional rule of Functional Application.

Recall the following equivalence:

(6) a. ⟦*smokes*⟧ = f : D $\rightarrow$ {1,0}
For all x, f(x) = 1 if and only if **x smokes**

b. ⟦*smokes*⟧ = λx $\in$ D. smokes(x)

That means, in the semantic composition, we can switch these out seamlessly.

(7) *Alissa smokes*

a. ⟦*Alissa*⟧ = Alissa

b. ⟦*smokes*⟧ = f : D $\rightarrow$ { 1, 0 } = λx $\in$ D. smokes(x)
for all x $\in$ D,
f(x) = 1 iff x smokes

(8)
S
NP VP
N° V°
Alissa *smokes*

$\downarrow$

FA : ⟦*smokes*⟧(⟦*Alissa*⟧)
NN : ⟦*Alissa*⟧ **NN :** ⟦*smokes*⟧
TN : ⟦*Alissa*⟧ **TN :** ⟦*smokes*⟧

$\longrightarrow$

FA : [λx $\in$ D. smokes(x)](Alissa)
NN : Alissa **NN :** λx $\in$ D. smokes(x)
TN : Alissa **TN :** λx $\in$ D. smokes(x)

[1] In the literature, you will see a number of ways of writing the value condition. Some authors put the functions in SMALL CAPS: $\lambda x \in D$. SMOKES(y) Others will write it out as "x smokes": λx $\in$ D. x smokes. Others will place the value condition in brackets: $\lambda x \in D[smokes(x)]$. Some will combine different ways. I don't know of anyone, though, who abbreviates it as "λx $\in$ D. 1 if x smokes, 0 otherwise".

What do we do with $[\lambda x \in D.\ \text{smokes}(x)](\text{Alissa})$?

1. Start with the [function](argument) notation. Use brackets if you need to mark off the λ-expression.

   (1) $[\lambda x \in D.\ \text{smokes}(x)](\text{Alissa})$

2. Find the argument variable at the left edge of the expression.

   (2) $[\lambda x \in D.\ \text{smokes}(x)](\text{Alissa})$

3. Find every instance of the same variable in the value condition.

   (3) $[\lambda x \in D.\ \text{smokes}(x)](\text{Alissa})$

4. Replace every instance of the variable in the value condition with the argument.

   (4) $[\lambda x \in D.\ \text{smokes}(\text{Alissa})](\text{Alissa})$

5. Remove the argument from the right edge of the expression.

   (5) $[\lambda x \in D.\ \text{smokes}(\text{Alissa})]\cancel{(\text{Alissa})} \rightarrow [\lambda x \in D.\ \text{smokes}(\text{Alissa})]$

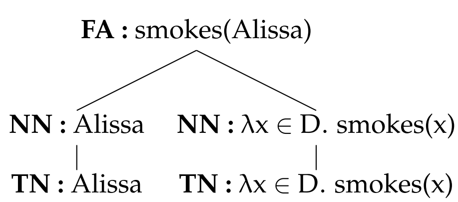
6. Remove the argument variable (and its domain condition)

   (6) $[\cancel{\lambda x \in D}.\ \text{smokes}(\text{Alissa})] \rightarrow [\text{smokes}(\text{Alissa})]$

7. You don't need the brackets anymore.

   (7) smokes(Alissa)

Now we can re-write the tree:



Recall that the 'smokes(x)' in $\lambda x \in D.\ \text{smokes}(x)$ is short for "1 if x smokes and 0 if x doesn't." Likewise, our expression, smokes(Alissa), is short for "1 if Alissa smokes and 0 if Alissa doesn't."

3 Bad lambdas

Watch out for these two common pitfalls.

1. Vacuous binding. Make sure that every argument variable is represented in the value condition. Otherwise, FA will have no effect. In the tree below, λx is not binding anything (hence 'vacuous' [ˈvæk.ju.ɪs]). So when we plug Bill into it, nothing happens; Bill just disappears.

(8) smokes(Alissa)

    λx ∈ D. smokes(Alissa)   Bill

If you unpack smokes(Alissa) you'll see why: λx ∈ D. smokes(Alissa) = f : D → {1,0}
For all x, f(x) = 1 if and only if **Alissa smokes**

2. Unbound variable. Make sure that every variable in the value condition is bound by a single argument variable binder. A free variable in a λ-structure is uninterpretable, except as a name. That is, it isn't really a variable and we can't change it via functional application.

In the tree below, the variable y is not bound by anything, so we cannot get rid of it. If we don't know what y is, we can't formulate truth-conditions, so the result would be nonsense.

(9) Bill sees y

    λx ∈ D. x sees y   Bill