1 Functions with multiple arguments

So far we've been using intransitive verbs as functions. Functional application is very simple with these. Some verbs have two arguments, though: Transitives.

(1) *Becky chased Tom.*

If ⟦ *chased* ⟧ is a function, its meaning depends on two arguments. How do we formalize that?

Functions can have as many arguments as one likes. A **one-place** function has 1 argument. A **two-place** function has two, and so on.

With a two-place function, the two arguments need to be ordered, since the propositions x chased y and y chased x have distinct truth-conditions.

(2) ⟦ *chased* ⟧ = f : D × D → {1, 0}
for all ⟨x, y⟩ ∈ D × D,
f(⟨x, y⟩) = 1 if and only if **y chased x**

We can turn this into a λ-expression.

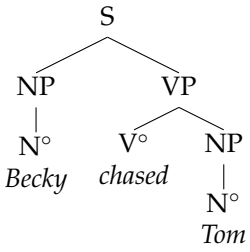
(3) ⟦ *chased* ⟧ = λ⟨x, y⟩ ∈ D × D. chased(⟨x, y⟩)

Plugging in the pair ⟨Becky, Tom, ,⟩ we get a truth-value:

(4) ⟦ *Becky chased Tom* ⟧ = [λ⟨x, y⟩ ∈ D × D. chased(⟨x, y⟩)](⟨Becky, Tom⟩) =
[~~λ⟨x, y⟩ ∈ D × D.~~ chased(⟨Becky, Tom⟩)] =
chased(⟨Becky, Tom⟩)

However, we run into a problem when we try to build the semantics off the syntax. For the subject and object of a verb do not merge simultaneously. The object merges first.

(5)



How can we get our two-place function to work with the syntax? With a process that's usually called **currying**.[1]

Basically, currying involves taking a two-place function, and breaking it into a sequence of one-place functions.

$$f(\langle x, y\rangle) \mapsto [f(x)](y)$$

[1]Named after mathematician Haskell Curry. It is also called **Schönfinkelization**, after Moses Schönfinkel, who developed it prior to Curry. Curry took it to the next level and it caught on then. But as it turns out, the first person recorded to do it was actually . . . Frege.

In the λ-calculus:

$$\lambda\langle x, y\rangle.\ x + y \mapsto \lambda x.[\lambda y.\ x + y]$$

Once curried, the function takes one argument, then gives you a new function, which takes the second argument. This is exactly what we see in a transitive verb.

(6) $[\![\ \textit{chased}\ ]\!]$ = f : D → { g | g : D → { 1, 0 } }
for all x ∈ D, f(x) = g: D → { 1, 0 }
for all y ∈ D, g(y) = 1 iff y chased x

READ: The denotation of *chased* is the function f such that f maps from D to the set of functions from D to {1, 0}, such that for all x in D, f of x equals the function g, such that g maps from D to the set containing 1 and 0, such that for all y in D, g(y) equals 1 if and only if y chased x.

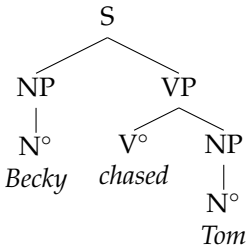
Now, that's quite a mouthful. In λ-notation it's easier:

(7) $[\![\ \textit{chased}\ ]\!]$ = λx ∈ D. λy ∈ D. chased(x)(y)
READ: *lambda x in D, lambda y in D, chased x y*

2 Composition

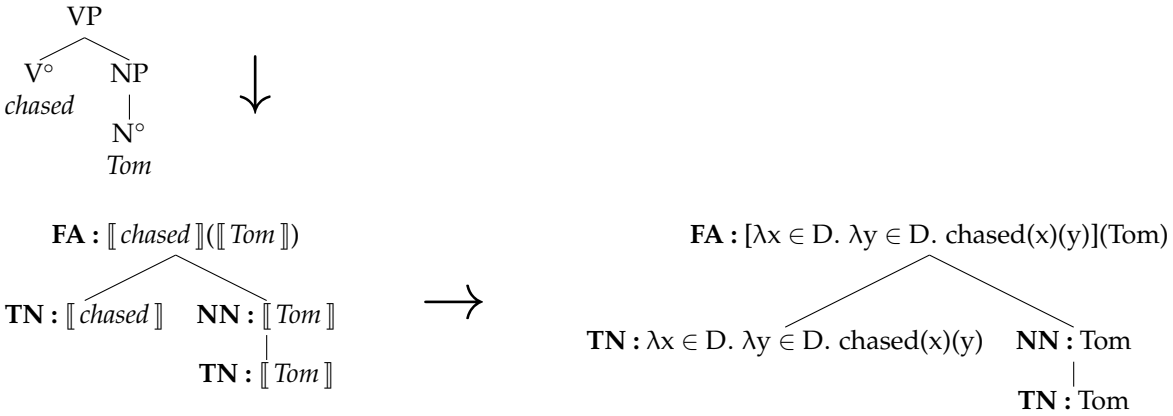
So, we have our sentence and its LF

(8)



We have our denotation of the verb, inserted via TN.

(9)



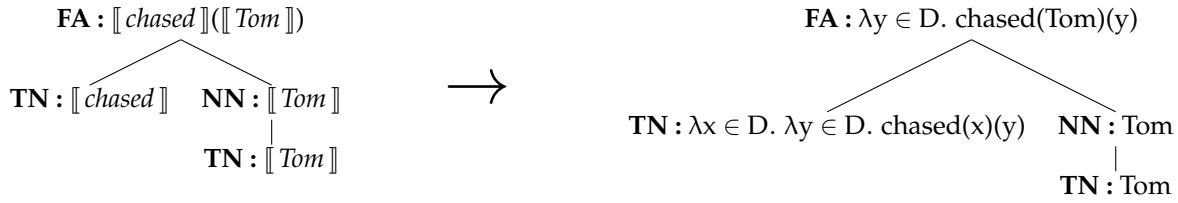
From here, the process is the same as with intransitives; we just do it twice.

1. Find the **leftmost** λ-operator and variable argument. Replace every instance of the variable in the value condition with the argument. Remove the argument, then remove the leftmost operator and variable argument.

   1. [λx ∈ D. λy ∈ D. chased(x)(y)](Tom)
   2. [λx ∈ D. λy ∈ D. chased(Tom)(y)](Tom)
   3. [λx ∈ D. λy ∈ D. chased(Tom)(y)]
   4. λy ∈ D. chased(Tom)(y)

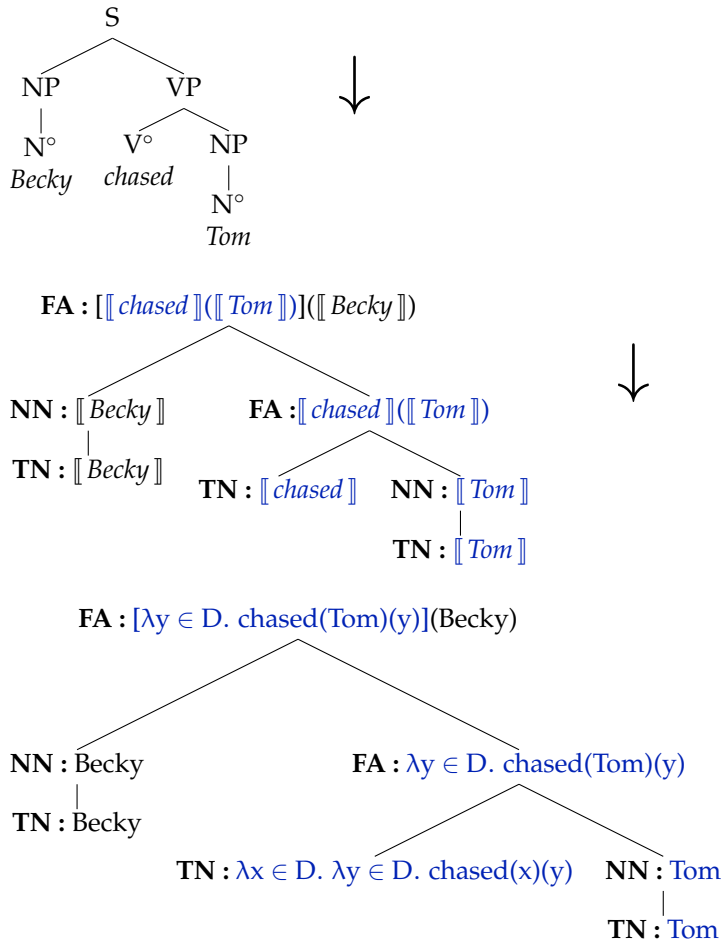
Notice how we're left with a function. The input to that will be the subject.

(10)



2. Repeat the process as usual. Note: The part we built above is highlighted in blue in the semantic trees. This part will not change.

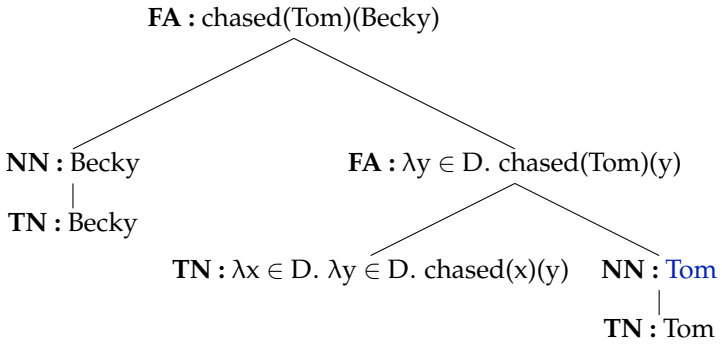
(11)



1. [λy ∈ D. chased(Tom)(y)](Becky)
2. [λy ∈ D. chased(Tom)(Becky)](Becky)
3. [λy ∈ D. chased(Tom)(Becky)]
4. chased(Tom)(Becky)

What we end up with, chased(Tom)(Becky), is an abbreviation for "1 if Becky chased Tom; 0 otherwise"

So, the whole semantic tree, if you were looking at it without the intermediate steps, would look like this:



Notice how the object is closer to the function than the subject. This is a result of the syntax. The first argument is 'closer' to the function, but it's the syntax that makes the first semantic argument happen to be the object. Notice that we wrote our result as chased(Tom)(Becky). A stricter version would be [chased(Tom)](Becky)…showing that chased(Tom) is a unit.