Logic for Linguists:

Lecture 7

Gregory Wilsenach

University of Cambridge

27th November 2019

Review

Last week we discussed:

- the relationship between the grammars of interest in formal linguistics and various algorithmic models;
- the most general algorithmic model, the Turing machine; and
- the limitations of this model.

In today's lecture we will discuss a different model of computation, the lambda calculus. However, we will motivate our interest from another angle...

First-Order Semantics

Example:

"There exists a student learning logic."

There are a number of approaches for assigning to this sentence a formal semantics. We have so far met the first-order approach. We might assign it the formula:

$$\exists x(\text{student}(x) \wedge \text{learning-logic}(x))$$

Problems

This approach is unsatisfying for so many reasons.

It seems ad-hoc, how does one construct the FO-sentence from the English sentence? We would like some natural compositional approach to semantics.

In natural language we can perform high-order reasoning, i.e. we can reason about reasoning about reasoning, FO cannot define these higher-order functions.

FO seems too simple in other respects. We can represent:

- verbs, common nouns, and adjectives → predicates;
- proper nouns → constants; and
- variables → pronouns.

What about: Prepositions, verb phrases, adjective phrases, adverbs, etc.

There are other problems, e.g. with conjunctions.

Lambda Calculus to the Rescue!

The lambda calculus gives us a neat compositional approach to semantics, one that allows us to naturally encode higher-order functions and deal with some of the additional complexities of language.

The lambda calculus was developed by Alonzo Church in the 1930's as part of his work on the theory of algorithms.

We will spend the rest of this lecture taking about the lambda calculus...

Functions

The lambda calculus is a language for defining functions abstractly and applying (or composing) functions.

Let's first talk about functions. We are used to functions such as:

$$f(x) = x^2 + x + 1 \text{ or } g(y) = y + 1$$

We can apply a function to an input and we can compose functions together.

We apply $f(x)$ to 2 by replacing each x appearing in the definition of $f(x)$ with the number 2 and then applying the definitions of the functions $+$ and $\times$.

We have something like:

$$f(2) = 2^2 + 2 + 1 = 4 + 2 + 1 = 6 + 1 = 7$$

Composition

We can also apply one function to the output of another function (assuming matching types). For example:

$$g(f(2)) = g(f(2)) = f(2) + 1 = 7 + 1 = 8$$

We could also do this abstractly! We could define:

$$g(f(x)) = (x^2 + x + 1) + 1 = x^2 + x + 2$$

This is called function composition. Notice that function composition is just function evaluation, except we are evaluating one function abstractly on another.

We can use function composition to define new functions from old ones! We can think of composition as an operation that takes in two functions and gives us back a single function.

Composition (2)

We should notice something else. Consider the function:

$$f(x) = x^2 + x + 1$$

It can be defined by composing two functions add and multiply such that

$$f(x) = \text{add}(\text{add}(\text{multiply}(x, x), x), 1)$$

The thought behind the lambda calculus is: What can we build up from basic syntax and function composition?

The answer, it seems, is essentially everything.

Informal Lambda Calculus

At it's most basic level the lambda calculus provides a language for describing functions, where instead of writing

$$f(x) = x^2 + x + 1$$

we write

$$\lambda x.x^2 + x + 1.$$

The lambda calculus also allows us to apply a function to a value (or another function) such that

$$(\lambda x.x^2 + x + 1)(2) \to 2^2 + 2 + 1 \to 4 + 2 + 1 \to 7.$$

These sort of repeated applications and simplifications is how the lambda calculus computes!

λ-Terms

We start with a sequence of variables $x, y, z, \ldots$.

The λ-terms are built up as follows:

- all variables are λ-terms
- λ-abstraction: if x is a variable and M a λ-term then $\lambda x.M$ is a λ-term
- application: if M and N are λ-terms then (MN) is a λ-term.

Examples and Notation

Examples:

$$x \qquad \lambda x.x \qquad \lambda x.y \qquad (\lambda y.(\lambda x.(xy))z)$$

Notational Conventions:

- We usually write $\lambda xy.M$ rather than $\lambda x.(\lambda y.M)$ and
- often omit brackets when we can to simplify things and write $(\lambda x.x)z$ rather than $((\lambda x.x)z)$ and xy rather than (xy).

α-Reductions

This is an essentially simple concept. We say M is α-equivalent to N if we can derive N from M by renaming the bound variables.

We write $M =_\alpha N$ to denote that M and N are α-equivalent.

Example 1:

$$\lambda x.xy =_\alpha \lambda z.zy$$

Example 2:

$$\lambda x.(\lambda xz.x)z =_\alpha \lambda y.(\lambda xz.x)z$$

β-Reductions

The λ-term $\lambda x.F$ is intended to encode a function of x.

This function is applied to a value by taking the description of the function (F) and replacing each free occurrence of x with the value for which we want to compute the function.

A β-reduction is the formalisation of this process.

The idea is that a term of the form $(\lambda x.F)z$ β-reduces to the term corresponding to F with every free occurrence of x replaced by z.

Example 1:

$$(\lambda x.x^2 + 1)5 \rightarrow_\beta 5^2 + 1 \rightarrow_\beta 25 + 1 \rightarrow_\beta 26$$

Example 2:

$$(\lambda x.x \text{ runs})(\text{John}) \rightarrow_\beta \text{John runs}$$

β-Conversion

The opposite of a β-reduction is a β-expansion. This is just the inverse process.

A β-expansion of xz is $(\lambda y.yz)x$. We could have chosen any variable instead of y, but any two β-expansions of this form will be α-equivalent.

We say that two λ-terms M and N are β-equivalent (and write $M =_\beta N$) if we can get from M to N via a series of β-reductions, β-expansions, and α-equivalences.

This is the notion of computation! For example:

$$(\lambda x.x^2 + 1)5 =_\beta 26$$

Examples/Exercises

β-reduce the following:

- $(\lambda x.x)z$
- $(\lambda x.x)(\lambda y.y)$
- $(\lambda x.y)(\lambda y.y)$
- $(\lambda xy.yy)zw$

What About These Numbers?

Wait a second...I keep on giving examples in terms of numbers and addition, but when I formally defined λ-terms we only allowed variables, abstraction, and application.

Can we formalise these numbers in our system?

We can! I'll sketch the idea very briefly.

Numbers

Let's denote the encoding of a number $n \in \mathbb{N}$ by $\underline{n}$. We define this encoding as follows:

$$\begin{aligned}
\underline{1} &:= \lambda f x.x \\
\underline{2} &:= \lambda f x.f x \\
\underline{3} &:= \lambda f x.f(f x) \\
&\vdots \\
\underline{n} &:= \lambda f x.\underbrace{f(\ldots(f}_{n} x)\ldots)
\end{aligned}$$

It is essentially only for mathematical objects that we have such neat encodings in the lambda calculus. The natural language cases we will discuss in a moment are unfortunately too complex to yield so completely. We include the numerical encodings above as an example of what is possible in principal.

Example and Exercise

Let

$$P := \lambda x_1 x_2.\lambda f x.x_1 f(x_2 f x)$$

Exercise: Show that $P\underline{mn} =_\beta \underline{m+n}$.

In other words, we define numbers and addition from the ground up with just syntax and function composition! We can also define multiplication this way, and so we can define our earlier λ-term

$$\lambda x.(x^2 + x + 1)$$

completely without reference to any special functions $+$ or $\times$ or any number 1.

Turing Computation

I should pause here for a quick interlude.

It is possible to show that the lambda calculus can compute exactly what a Turing machine can compute.

I won't go into any detail about what exactly that means, but see here for details
https://www.cl.cam.ac.uk/teaching/1718/CompTheory/lecture-10.pdf

I hope I've convinced you that we can formalise a lot in lambda calculus using pure syntax and function composition (although formalising things outside of mathematics can be tricky)

Let's go back to being informal and let's just trust that we can build up any operation we might choose using pure syntax and function composition.

Back to Linguistics...almost

Notice that we can partially apply a function. For example

$$(\lambda xy.xy)z \rightarrow_{\beta} \lambda y.zy$$

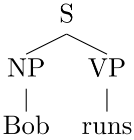
This is very useful for linguistics as we shall see in a moment...

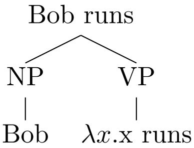
The idea is as follows:

- start with a syntax tree,
- each verb is denoted by a λ-term denoting a function where the arity of the function (the number of variables bound by the λ operator) is equal to the valence of the verb, and
- each noun is denoted by some fixed λ-term.

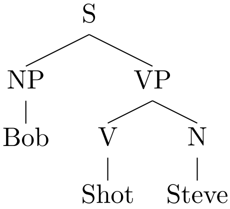
We then label the leaves of the syntax tree and β-reduce upwards in order to label the other nodes in the tree.

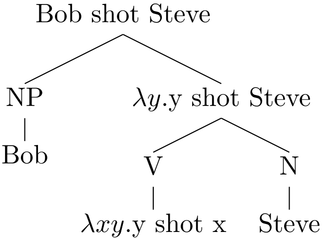
Example (1)





Example (2)


S
NP
VP
Bob
V
N
Shot
Steve


Bob shot Steve
NP
λy.y shot Steve
Bob
V
N
λxy.y shot x
Steve

Exercises

Write out the syntax trees for the following sentences and put them in lambda notation:

- "I teach logic to linguistics students."
- "Steve reads the Lord of the Rings to bill."
- "Her mother sits on the chair."

There are readings at the end of this presentation if you need more detail.

Summary

In this lecture we

- motivated the need for the lambda calculus;
- gave an informal introduction to the lambda calculus;
- introduced the lambda calculus formally;
- discussed how the lambda calculus arises within the theory of algorithms; and
- discussed how it can be used to give semantics to natural language sentences.

Reading

Here are some online course notes on semantics and the lambda calculus

- http://people.umass.edu/partee/MGU_2005/MGU052.pdf

Here is a cute introduction to the λ-calculus in linguistics entirely in pictures. It's really worth checking out!

- https://imgur.com/a/XBHub

For a detailed introduction to the λ-calculus see lectures 9, 10, 11, and 12 from

- https://www.cl.cam.ac.uk/teaching/1718/CompTheory/materials.html

Here is another (slightly mathematical) introduction to the subject

- http://www.inf.fu-berlin.de/lehre/WS03/alpi/lambda.pdf