Lambda Calculus: Formal Definitions - Worksheet

Carlo Allietti
`spam@iscienceworld.com`

Amateur losing his mind on — August 23, 2022

What is lambda calculus?

According to Wikipedia:

> **Lambda calculus** is a formal system in mathematical logic for expressing computation based on function abstraction and application using variable binding and substitution.

In other words, it is a way to compute using only anonymous pure functions.

> **Info: Pure functions** are functions that only returns stuff based on it's arguments, like the mathematical function $f(x) = x + 1$, it has no side effects it just returns $x + 1$ given x.
> **Anonymous functions** are functions that are not defined with a specific **identifer** (name). These functions are often passed as one of the arguments to another function.

Expressions in lambda calculus are made from variables $(x, y, z, etc.)$; symbols $(\lambda, .)$; parentheses "()". Lambda calculus involves **lambda terms** (valid expression in lambda calculus) that we reduce/compute on.

1 Lambda terms

Lambda terms are built from the following 3 rules.

1.1 Variables

If x is a variable, then x is a lambda term.

Variables are characters/strings that represent a mathematical/logical value.

> **Info:** Variables can be free or bounded. A free variable is one that can be any value and is not restricted by being in a parameter of a lambda term. A bound variable is one that has been used in an abstraction as a parameter.
> Distiguishing between free and bound variables is crucial because name collision and other problems may arise that change the meaning of a lambda term. For our purposes we wont worry too much about these details.

1.2 Abstractions

If x is a variable and M is a lambda term, then $(\lambda x.M)$ is a lambda term.

Abstractions are definitions of anonymous pure functions that take in some arguments (such as x) and return an expression (such as M). As said before, the variable x in $(\lambda x.M)$ becomes bounded as a parameter.

1.3 Applications

If N and M are lambda terms, then (MN) is a lambda term.

Applications are when a function/abstraction M is applied to an argument N. In other words, it represents the calling of the function M using an input N, to output $M(N)$.

Info: If this is a little unclear, let's make a comparison to C-like syntax:
You can think of $(\lambda x.M)$ as: `l(x){return M;}`
You can think of (MN) as: `M(N);`

2 Notation

Here are a few common notational conventions in lambda calculus:

- Outermost parentheses are redundant: $MN \equiv (MN)$
- Application is left associative: $MNP \equiv ((MN)P)$
- Abstraction is right associative: $\lambda x.\lambda y.M \equiv (\lambda x.(\lambda y.M))$
- The body of an abstraction extends as far right as possible: $\lambda x.MN \equiv (\lambda x.(MN))$

3 Reduction

The reduction of lambda terms is how we compute in lambda calculus, these are the kinds of reduction:

3.1 α-conversion

α-conversion is the process of renaming the bound variables in a lambda term. Two terms are α-equivalent if they only differ by the names of their bound variables. For example:

$$\lambda x.Mx \rightarrow_\alpha \lambda y.My \tag{1}$$

α-conversion is used to avoid name collisions. You can think of this example as:
`l(x){return M(x);} /*equals*/ l(y){return M(y);}`

3.2 β-reduction

β-reduction is the application of an abstraction to its arguments. This is done by substituting the bound variables in the body of the abstraction with the arguments passed to it. In other words, the calling of the function with its inputs. For example (:= represents assignment/substitution):

$$(\lambda x.M)y \rightarrow_\beta M[x := y] \tag{2}$$

So $(\lambda x.M)y$ and $M[x := y]$ are β-equivalent. You can think of this as:
`l(x){return M;} l(y); /*equals*/ x = y; M;`
For example, lets reduce the following expression:

$$\begin{aligned} (\lambda x.\lambda y.yx)b(\lambda x.x) &\rightarrow_\beta \\ (\lambda y.yb)\lambda x.x &\rightarrow_\beta \\ (\lambda x.x)b &\rightarrow_\beta b \end{aligned} \tag{3}$$

Since multiple abstractions act like a multi-parameter function, you can think of $\lambda x.\lambda y.xy$ as: `l(x,y){x(y);}`

Question 1 (Exercise for the reader.)

Try to reduce the following lambda expression until its *done*.

$$((\lambda x.\lambda y.y)m((\lambda x.\lambda y.\lambda z.yzx)ndo))e \tag{4}$$