# NOTES ON LAMBDA CALCULUS

VAIBHAV KARVE

These notes were last updated September 17, 2018. They are notes taken from my reading of *Haskell Programming from First Principles* by *Chris Allen, Julie Moronuki.* I plan on expanding these notes further by reading the following at some unspecified time in the future:

- A tutorial introduction to the Lambda Calculus by *Raúl Rojas.*
- An algorithm for optimal lambda calculus reduction by *John Lamping.*
- Introduction to Lambda Calculus by *Henk Barendregt* and *Erik Barendsen.*
- Proofs and Types by *Jean-Yves Girard*, *Paul Taylor* and *Yves Lafont.*

## Contents



## 1. Basics and definitions

(1) Lambda calculus has been called the *smallest universal programming language of the world.* It consists of a single transformation rule (variable substitution) and a single function definition scheme.

(2) Lambda calculus is universal in that any computable function can be expressed and evaluated using this formalism. It is equivalent to Turing machines.

(3) Lambda calculus has three basic components or *lambda terms* – expressions, variables and abstractions.

(4) *Expressions* are variable names, abstractions, or combinations of other expression. *Variables* have no meaning or value, they are only names for potential inputs to functions. An *abstraction* is a function – it is a lambda term that has a head (a lambda) and a body and is applied to an argument. An *argument* is an input value.

(5) Expressions can be defined recursively as —

$$
\begin{aligned}
< \text{expression} > &:= < \text{name} > \mid < \text{function} > \mid < \text{application} > \\
< \text{function} > &:= \ \lambda < \text{name} > . < \text{expression} > \\
< \text{application} > &:= < \text{expression} >< \text{expression} >
\end{aligned}
$$

(6) Abstractions have two parts – a *head* and a *body*. The head of the function is a $\lambda$ followed by a variable name. The body of the function is another expression. For example: $\lambda x..x^2$
Lambda abstractions are anonymous functions.

(7) The variable named in the head is the *parameter* and *binds* all instances of that same variable in the body of the function. The dot (.) separates the parameters of the lambda from the function body.

## 2. Equivalences and reductions

(1) *Alpha equivalence* states that $\lambda x..x$ is the same as $\lambda y..y$, that is, the variables $x$ and $y$ are not semantically meaningful except in their role in their single expressions.

(2) *Beta reduction:* when applying a function to an argument, substitute the input expression for all instances of bound variables within the body of the abstraction.

$$(\lambda x.xx)3 = xx[x := 3] = 3\ 3$$

Hence, Beta reduction is the process of applying a lambda term to an argument, replacing the bound variables with the value of the argument, and eliminating the head.

$$\begin{aligned}(\lambda x.x)\lambda y.y &= x[x := (\lambda y.y)]\\ &= \lambda y.y\end{aligned}$$

(3) Another notation for beta reduction:

$$(\lambda x.x)y = [y/x]x = y$$

(4) Application in lambda calculus is left-associative.

$$\begin{aligned}(\lambda x.x)(\lambda y.y)z &= ((\lambda x.x)(\lambda y.y))z && \text{left-associativity}\\ &= (x[x := \lambda y.y])z && \text{beta reduction step 1}\\ &= (\lambda y.y)z && \text{beta reduction step 2}\\ &= y[y := z] && \text{beta reduction step 1}\\ &= z && \text{beta reduction step 2}\end{aligned}$$

(5) Variables in the body that are not bound by the head are called *free variables.* For example, $y$ is a free variable in the expression $\lambda x.xy$

$$(\lambda x.xy)z \ = \ xy[x := z] \ = \ zy$$

(6) Formally a variable $<$ name $>$ is free in an expression if one of the following three cases hold:
- $<$ name $>$ is free in $<$ name $>$
- $<$ name $>$ is free in $\lambda <$ name$_1$ $>$ . $<$ exp $>$, such that $<$ name $>\neq<$ name$_1$ $>$ and $<$ name $>$ is free in $<$ exp $>$.
- $<$ name $>$ is free in $E_1E_2$ if $<$ name $>$ is free in $E_1$ or it is free in $E_2$.

(7) Similarly, a variable $<$ name $>$ is bound if one of two cases hold:
- $<$ name $>$ is bound in $\lambda <$ name$_1$ $>$ . $<$ exp $>$, such that $<$ name $>=<$ name$_1$ $>$ or $<$ name $>$ is bound in $<$ exp $>$.
- 
- $<$ name $>$ is bound in $E_1E_2$ if $<$ name $>$ is bound in $E_1$ or if it is bound in $E_2$.

(8) The same identifier can occur free and bound in the same expression. For example, $y$ is both free and bound in the expression $(\lambda x.xy)(\lambda y.y)$.

(9) The alpha equivalence does not apply to free variables.

(10) *Currying:* named after Haskell Curry is the shorthand notation of the type $\lambda xy..xy$ for multiple lambda functions $\lambda x.(\lambda y.xy)$.

$$\begin{aligned} \lambda xy.xy\ 1\ 2 &= \lambda x.(\lambda y.xy)\ 1\ 2 \\ &= (\lambda y.xy)[x := 1]\ 2 \\ &= (\lambda y.1y)\ 2 \\ &= (1y)\ [y := 2] \\ &= 1\ 2 \end{aligned}$$

or by using currying we perform the same calculation in fewer steps,

$$\begin{aligned} \lambda xy.xy\ 1\ 2 &= (\lambda y.xy)[x := 1]\ 2 \\ &= (\lambda y.1y)2 \\ &= (1y)[y := 2] \\ &= 1\ 2 \end{aligned}$$

(11) A lambda term is in *beta normal form* when one cannot beta reduce (apply lambdas to arguments) its expressions any further. This corresponds to a fully evaluated function or fully executed program. The identity function $\lambda x.x$ is in normal form.

(12) A *combinator* is a lambda term with no free variables. Combinators serve only to combine the arguments that are given. The following are combinators: $\lambda x.x$, $\lambda xy.x$, $\lambda xyz.xz(yz)$ and the following are not: $\lambda y.x$, $\lambda x.xz$. The point of combinators is that they can only combine the arguments they are given, without injecting any new values or random data.

(13) A lambda term whose beta reduction never terminates is said to *diverge*. The lambda term *omega* defined as $(\lambda x.xx)(\lambda x.xx)$ diverges because

$$(\lambda x.xx)(\lambda x.xx) = (\lambda x.xx)(\lambda y.yy) = xx[x := \lambda y.yy] = (\lambda y.yy)(\lambda y.yy).$$

## 3. Examples

$$(\lambda xy.xy)(\lambda z.a)\ 1$$
$$= (\lambda y.(\lambda z.a)y)1$$
$$= (\lambda z.a)1$$
$$= a$$

$$(\lambda xyz.xz(yz))(\lambda x.z)(\lambda x.a)$$
$$= (\lambda xyb.xb(yb))(\lambda c.z)(\lambda d.a)$$
$$= (\lambda yb.(\lambda c.z)b(yb))(\lambda d.a)$$
$$= \lambda b.(\lambda c.z)b((\lambda d.a)b)$$
$$= \lambda b.z((\lambda d.a)b)$$
$$= \lambda b.za$$

$$(\lambda y.y)(\lambda x.xx)(\lambda z.zq)$$
$$= (\lambda x.xx)(\lambda z.zq)$$
$$= (\lambda z.zq)(\lambda z.zq)$$
$$= (\lambda z.zq)(\lambda x.xq)$$
$$= (\lambda x.xq)q$$
$$= qq$$

$$(\lambda a.aa)(\lambda b.ba)c$$
$$= (\lambda d.dd)(\lambda b.ba)c$$
$$= (\lambda b.ba)(\lambda b.ba)c$$
$$= (\lambda b.ba)(\lambda d.da)c$$
$$= ((\lambda d.da)a)c$$
$$= aac$$

$$(\lambda xyz.xz(yz))(\lambda mn.m)(\lambda p.p)$$
$$= (\lambda yz.(\lambda mn.m)z(yz))(\lambda p.p)$$
$$= \lambda z.(\lambda mn.m)z((\lambda p.p)z)$$
$$= \lambda z.(\lambda n.z)((\lambda p.p)z)$$
$$= \lambda z.z$$

$$(\lambda xy.xxy)(\lambda x.xy)(\lambda x.xz)$$
$$= (\lambda xy.xxy)(\lambda a.ay)(\lambda b.bz)$$
$$= (\lambda y.(\lambda a.ay)(\lambda c.cy)y)(\lambda b.bz)$$
$$= (\lambda a.a(\lambda b.bz))(\lambda c.c(\lambda b.bz))(\lambda b.bz)$$
$$= (\lambda a.a(\lambda b.bz))(\lambda c.c(\lambda d.dz))(\lambda e.ez)$$
$$= ((\lambda c.c(\lambda d.dz))(\lambda b.bz))(\lambda e.ez)$$
$$= ((\lambda b.bz)(\lambda d.dz))(\lambda e.ez)$$
$$= ((\lambda d.dz)z)(\lambda e.ez)$$
$$= (zz)(\lambda e.ez)$$
$$= yy(\lambda b.bz)$$

$$(\lambda x.\lambda y.xyy)(\lambda a.a)b$$
$$= (\lambda y.(\lambda a.a)yy)b$$
$$= (\lambda a.a)bb$$
$$= bb$$

$$(\lambda abc.cba)zz(\lambda wv.w)$$
$$= (\lambda bc.cbz)z(\lambda wv.w)$$
$$= (\lambda c.czz)(\lambda wv.w)$$
$$= (\lambda wv.w)zz$$
$$= (\lambda v.z)z$$
$$= z$$