NOTES ON LAMBDA CALCULUS

VAIBHAV KARVE

These notes were last updated September 11, 2018. They are notes taken from my reading of *Haskell Programming from First Principles* by *Chris Allen, Julie Moronuki.*

1. Basics and definitions

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

(2) *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.

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

Lambda abstractions are anonymous functions.

(4) 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.

$$\lambda x.x^2\ 3 = 3^2 = 9$$

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) Application in lambda calculus is left-associative.

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

(4) 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 $\lambda x.xy$

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

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

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

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

3. Worked-out examples

REDO THIS

$$\begin{aligned}(\lambda xy.xxy)(\lambda x.xy)(\lambda x.xz) &= (\lambda xy.xxy)(\lambda a.ay)(\lambda b.bz)\\ &= (xxy)[x := \lambda a.ay][y := \lambda b.bz]\\ &= (\lambda a.ay)(\lambda a.ay)(\lambda b.bz)\\ &= (\lambda a.ay)(\lambda c.cy)(\lambda b.bz)\\ &= (ay)[a := \lambda c.cy](\lambda b.bz)\\ &= ((\lambda c.cy)y)(\lambda b.bz)\\ &= (cy)[c := y](\lambda b.bz)\\ &= yy(\lambda b.bz)\end{aligned}$$

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

$$
\begin{aligned}
(\lambda xyz.xz(yz))(\lambda mn.m)(\lambda p.p) &= (\lambda yz.xz(yz)[x := \lambda mn.m])(\lambda p.p) \\
&= (\lambda yz.(\lambda mn.m)z(yz))(\lambda p.p) \\
&= (\lambda z.(\lambda mn.m)z(yz))[y := \lambda p.p] \\
&= \lambda z.(\lambda mn.m)z((\lambda p.p)z) \\
&= \lambda z.(\lambda n.m[m := z])((\lambda p.p)z) \\
&= \lambda z.(\lambda n.z)((\lambda p.p)z) \\
&= \lambda z.z[n := (\lambda p.p)z] \\
&= \lambda z.z
\end{aligned}
$$