Lambda Calculus

Tutorial

Dr S. S. Chandra

The Lambda Calculus is an encoding scheme for computation based on just a single concept: the pure function. In much the same way that Turing 'built' an abstract encoding machine that represented the simplest computer, we will follow in the footsteps of Alonzo Church, who constructed an abstract encoding scheme that can represent computation itself, without any assumption of a physical machine.

In this tutorial, you will practice simplifying λ expressions and understanding how the encoding process is used to create the machinery of λ-Calculus.

Problem 1

Use β-reduction on the following λ expressions into β-normal form:

a) $(\lambda x.\lambda y.x + y)(3,5)$ 3+5=8
b) $(\lambda x.\lambda y.yxy)ab$
c) $(\lambda x.xx)(\lambda y.y)$
d) $(\lambda x.(\lambda y.xy))y$
e) $(\lambda x.(\lambda y.x))y$

b) (λx.λy.yxy)ab
(λy.yay)b
bab

c) (λx.xx)(λy.y)
(λy.y) (λy.y)
(λy.y)

d) (λx.(λy.xy))y
(λx.(λb.xb))y
λb.yb

e) (λx.(λy.x))y
(λx.(λy.x))b
λy.b

e.g. f(x)=3

Problem 2

Explain Church encoding and how Church numerals are derived. Derive the Church numerals up to 4 and describe how they can be used with the successor combinator. What is the consequence of developing these concepts within Lambda Calculus?

S = Ln.Ly.Lx.y(nyx)
N1 = Lf.Lx.f(x)
Sn1 = Ln.Ly.Lx.y(nyx)(Lf.Lx.f(x))
Ly.Lx.y((Lf.Lx.f(x))yx)
Ly.Lx.y((Lx.y(x))x)
Ly.Lx.y(y(x)) = Lf.Lx.f(f(x)) = N2
Lf.Lx.f(f(f(x))) = N3

Problem 3

Describe how Boolean logic can be defined in Lambda Calculus and therefore the three main operations of AND, OR and NOT can be encoded into it making sure to explain all the relevant combinators, their arguments and function bodies.

```
T := Lx.Ly.x == f(x,y)=x
-- T choose the first argument
F := Lx.Ly.y == f(x,y)=y
-- F choose the second argument

NOT := Lp.pFT
NOT T == (Lp.pFT)T = T(FT) = F
NOT F == (Lp.pFT)F = F(FT) = T

AND := Lp.Lq.pqF
AND T T = (Lp.Lq.pqF)TT = T(TF) = T
AND F T = (Lp.Lq.pqF)FT = F(TF) = F

OR := Lp.Lq.pTq
OR T F = (Lp.Lq.pTq)TF = (Lq.TTq)F = TTF = T
```

Problem 4

Define and discuss the significance of the Composition Combinator in Lambda Calculus. Describe how this combinator contributes to the Turing completeness of Lambda Calculus.

composition combinator: Lg.Lf.Lx.g(fx)
It allows us to formally encode the idea of composition. So you have two functions, g and f. you want to be able to apply them to each other in a structured way. So by feeding g and f into this combinator.
= Lx.g(fx)
Obviously composition is pretty essential for any kind of model of computation, it's because you need to be able to define things in terms of building blocks of simpler functions. And that's what composition does.

Problem 5

In a paragraph and using a few equations, describe how one can compute the factorial function using recursion in Lambda Calculus. Make sure to mention all the relevant combinators and theorems required to achieve your result.

```
Y := Lt.(Lx.t(xx))(Lx.t(xx))
Yt = (Lx.t(xx))(Lx.t(xx))
   = t((Lx.t(xx)(Lx.t(xx))
   = tYt
   = ttYt

FAC = Y(Lf.Ln.Z(n)(1)MUL(n)(f(PRED(n))))
PRED - predecessor, the factorial function applied to the predecessor
Z(n) - the predicate is zero
if n == 0:
        return 1
else:
        n*fac(n-1)   <-- in programming language

Z := Ln.Lf.Lx.n(Lx.F)T
MULT := (Ln.Lk.Lf.n(kf))
PRED := (Ln.Lf.Lx.n(Lg.Lh.h(gh))(Lu.x)(Lu.u))  <-- not necessary to remember
```