Lambda Calculus

1 Introduction

Let us describe imagine a function

$$f : \mathbb{Z} \to \mathbb{Z} \tag{1}$$
$$f(x) = 3x \tag{2}$$

In various programming languages we may write this as

```
def triple(x):
    return 3*x

triple::Integer->Integer
triple x = 3*x

int triple(int x):
    return 3*x
```

In general a function take one argument and returns one body. We write a function in lambda calculus as $\lambda[argument].[body]$; f will be written in the lambda calculus as $f = \lambda x.(3x)$. To apply this function we say that $f5 = (\lambda x.(3 * x))5 = (3 * 5) = 15$.

In lambda calculus, we don't have an arithmatic as axioms; the only thing we have is λ, a symbol that allows us to construct functions. But what do these functions take as arguments and return? Other functions, of course. Why would we do this? You will see.

2 Lambda terms

The syntax of lambda calculus is particularly simple. We define a lambda term as:

1. a variable, x, is a lambda term
2. if t is a lambda term, and x is a variable, then $\lambda x.t$ is a lambda term (called a lambda abstraction)
3. if t and s are lambda terms, then ts is a lambda term (called an application)

A lambda abstraction $\lambda x.t$ represents an anonymous function that takes a single input, evalutes t with x bound to the input, and then returns t.

An application ts can be thought of as calling t with s as the input.

As a final point, the "=" sign is not part of lambda calculus but we use it to say that two lambda terms are equivalent.

3 Simple examples

The Identity function, $Ix = x$, is defined as $I = \lambda x.x$

The Constant function, C_r that always returns r, is defined as $C_r = \lambda x.r$

What does the function $T = \lambda a.\lambda b.a$ do?

[Currying]

4 Church numerals

Functions in LC take other functions as arguments and return other functions. This by itself is not very fun, so we must begin embedding data into functions. Let's start by embedding numbers.

The higher-order function that represents natural number n is a function that maps any function f to its n-fold composition f^n. Let us derive it.

$$nf = f^n \tag{3}$$
$$nfx = f^n x \tag{4}$$
$$n = \lambda fx.f^n x \tag{5}$$

so

$$0 = \lambda fx.x \tag{6}$$
$$1 = \lambda fx.fx \tag{7}$$
$$2 = \lambda fx.f(fx) \tag{8}$$
$$3 = \lambda fx.f(f(fx)) \tag{9}$$
$$\tag{10}$$

or $nfx = f^n x$.

let us derive the S or successor function. We observe that $Sn = \lambda fx.f^{(n+1)}x = \lambda fx.f(f^n x) = \lambda fx.f(nfx)$; doing an abstraction $S = \lambda nfx.f(nfx)$.

the + or addition function is $+ = \lambda mnfx.mf(nfx)$

the $\times$ or multiplication function is $\times = \lambda mnfx.m(nf)x$

5 Boolean

$$TRUE = \lambda ab.a \tag{11}$$
$$FALSE = \lambda ab.b \tag{12}$$

$$NOT = \lambda pab.pba \tag{13}$$
$$AND = \lambda pq.pqFALSE \tag{14}$$
$$OR = \lambda pq.pTRUEq \tag{15}$$