λ - CALCULUS

Mariam Ahmed

THE UNIVERSAL MODEL OF COMPUTATION BASED ON FUNCTIONS

Background

λ-Calculus is an incredibly powerful mathematical model of computation; a formal system of logic capable of reducing computation down to just function definition and application. Developed by Alonzo Church in the 1930s, λ-Calculus is often described as the 'basis of almost all functional programming', playing a huge role in computer science as we know it today.

Church Encoding

Identity and Successor functions can be used to encode Church Numerals, as below.

$I = \lambda x.x$ $\qquad$ $S = \lambda n.\lambda f.\lambda x.f\,(n\,f\,x)$

0 = λs.λz.z 1 = λs.λz.s z 2 = λs.λz.s (s z)	3 = λs.λz.s (s (s z)) 4 = λs.λz.s (s (s (s z) 5 = λs.λz.s (s (s (s (s z))

E.g. β-Reduction: Successor of 0

S0 = S(λf. λx. x)
= λn. λy. λx. y(nyx)(λf. λx. x)
= λy. λx. y((λf. λx. x)yx)
= λy. λx. y((λx. x)x)
= λy. λx. y(x) ← α-equivalent to **1**

Boolean Algebra

Mimicking the logic of conditional statements, we define the following Church Booleans.

true = λa.λb.a ← Chooses 1st argument
false = λa.λb.b ← Chooses 2nd argument

Logic Gates

not = λp.**p** false true
and = λp.λq.**p** q p
or = λp.λq.**p** p q
xor = λp.λq. **p** (not q) q

Result:
if **p** is true
If **p** is false

Additional Useful Functions

- **double** = λn.λf.λx.n(λx.f(f x)) x
- **add** = λm.λn.λf.λx.m f (n f x)
- **multiply** = λm. λn. λf. λx.m (n f) x
- **iszero** = λn.n (λy.false) true

Finding predecessors & performing subtraction:

- **pred** = λn.λf.λx.n (λg.(λh.h (g f))) (λu.x) (λu.u)
- **minus** = λa. λb.b pred a

Predicates

Minus, the function above, returns no –ve result. Hence, less than or equal, greater than or equal, and equal can be defined as follows.

- leq = λm. λn.(iszero (minus m n))
- eq = λm. λn.(and (leq m n) (leq n m))
- geq = λm. λn.(or (eq m n) (not(leq m n)))

The Y-Combinator

Recursion requires a method of infinitely looping until a base case is reached. Haskell Curry discovered the 'Y-combinator', which uses self-application to encode recursive functions in λ-Calculus.

$$\boxed{Y = \lambda f.(\lambda x.f\,(x\,x))\;(\lambda x.f\,(x\,x))}$$

E.g. Recursion in Euclid's Algorithm

Recall that Euclid's algorithm calculates the greatest common divisor of 2 inputs recursively.

$$\gcd(a,b) = \begin{cases} a & a = b \\ \gcd(a-b,b) & a > b \\ \gcd(a,b-a) & b > a \end{cases}$$

Encoding this in λ-Calculus, we use many previously defined functions and predicates. This includes the Y-Combinator to allow for recursion.

gcd = y (λf.λa.λb.eq a b (a) (geq a b (f (minus a b)(b)) (f (a)(minus b a))))

Conclusion & Turing Completeness

λ-Calculus is an abstract mathematical theory, simply and powerfully capturing a functional notion of computation. Alternatively, Turing Machines, created by Church's student Alan Turing, capture a state-based model of computation. Fascinatingly, these systems are equivalent, as outlined in the Church-Turing hypothesis. λ-Calculus is a Turing Complete system and can be used to emulate a Turing Machine.

References: Lambda Calculus – Graham Hutton on Computerphile
Modern Computation, A Unified Approach – S.S. Chandra