λ-Calculus-Church Numerals

Professor: Suman Saha

Church Encoding

- We can write any natural number n using $\mathrm{add1}$ and 0 in a functional programming

- $(\mathrm{add1}\ (\mathrm{add}\ 0))$

2

- $(\mathrm{add1}\ (\mathrm{add1}\ (\mathrm{add1}\ 0)))$

3

Church Encoding

- We can write any natural number n using add1 and 0 in a functional programming

- (add1 (add 0))

2

- (lambda (f) (f (f (f 0))))

3

Church Encoding

- We can write any natural number n using add1 and 0 in a functional programming

- (add1 (add 0))

2

- (lambda (f) (lambda (x) (f (f (f x)))))

3

Church Encoding

- Can write any natural number n as:

- $1 + ... + 0 \;= \text{n times}$

- $0 = 0$
- $1 = 1 + 0$
- $2 = 1 + 1 + 0$
- $3 = 1 + 1 + 1 +0$

Church Encoding

- Represent the number n as a function that accepts another function g and returns a function that performs g n times

- $1 + \ldots. + 0 \ = n \text{ times}$

- $0 = (\lambda(f)\ (\lambda(x)\ x))$
- $1 = (\lambda(f)\ (\lambda(x)\ (f\ x)))$
- $2 = (\lambda(f)\ (\lambda(x)\ (f\ (f\ x))))$
- $3 = (\lambda(f)\ (\lambda(x)\ (f\ (f\ (f\ x)))))$

Church Encoding

- When we use this encoding, any two expressions that are alpha-equivalent to n is n

- (((λ(y) (y y)) (λ (x)x))
  (λ (z) (λ (x) (z (z x)))))

Church Encoding

- Given a number n. Its normal-form (when it is fully-reduced) must be something like
- n = (λ (f) (f (f … (f x) …)))
- How can you generate n + 1?

Church Encoding

- Given a number n. Its normal-form (when it is fully-reduced) must be something like
- $n = (\lambda\ (f)\ (f\ (f\ \ldots\ (f\ x)\ \ldots)))$
- How can you generate n + 1?
- $n + 1 = (\lambda\ (f)\ (f\ (f\ (f\ \ldots\ (f\ x)\ \ldots))))$

Church Encoding: SUCC

- Now, how could I wrote a function, succ, which computes n + 1 using only the lambda calculus?

- (λ (n)

  (λ (f) (λ (x) (f ((n f) x)))))

Church Encoding: SUCC

- (define succ

  (lambda (n) (lambda (f) (lambda (x) (f ((n f) x))))))
- ;; (succ 1) should equal 2

- ((λ (n)

  (λ (f) (λ (x) (f ((n f) x)))))

  (λ (f) (λ (x) (f x))))

Church Encoding: PLUS

- Now how do you do addition? We need two arguments using currying.

- plus = (lambda (n) (lambda (k) ..))
- one = (lambda (f) (lambda (x) (f x)))

- We can call this like: ((plus one) one) ;; compute 2

Church Encoding: PLUS

- ((n f) x) ;; applies f to x n times
- ((k f) x) ;; applies f to x k times

- plus = (λ (n) (λ (k) (λ (f) (λ (x) ((k f) ((n f) x))))))

Church Encoding: Try at Home

- (plus 0 1);; (λ (f) (λ (x) (f x)))
- (plus 1 1);; (λ (f) (λ (x) (f (f x))))
- (plus 2 0);; (λ (f) (λ (x) (f (f x))))

Church Encoding: MULT

- ((n f) x) ;; applies f to x n times
- ((k f) x) ;; applies f to x k times

- mult = (λ (n) (λ (k) (λ (f) (λ (x)(((n k) f) x)))))

Church Encoding: Try at Home

- (mult 1 1);; (λ (f) (λ (x) (f x)))
- (mult 2 1);; (λ (f) (λ (x) (f (f x))))
- (mult 2 0);; (λ (f) (λ (x) x))