classmate

Date 14.10.2019

Page 1

RECURSION

1. AST annotation (circularity between closures & environment)
2. interpreter (extended-rec-env)
3. Y combinator (λ-calculus)

(define (G f)
  (λ (n)
    (if (= n 0)
        1
        (* n (f (- n 1))))...

(n!) if (n=0) 1 else n * (n-1)!

((G add1) 5)
= 25

((G (λ (x) x)) 5)
= 20

((G !) 5)
= 5! = 120

((G !) n) = ? (!n)

G ! = !

! is a fixed point of G

we want for Y that takes an arbitrary G and return its fixed point. Y is called the fixed-point combinator (or the Y combinator)

! : ℕ → ℕ

G : (ℕ → ℕ) → (ℕ → ℕ)

Y : ((ℕ → ℕ) → (ℕ → ℕ)) → (ℕ → ℕ)

```
(define (Y g)
  (let ((s (lambda (x) (g (x x)))))
  (s s)))
```

```
((Y G) 2) ⇒ ((G (s s)) 2) ⇒ (* 2 ((s s) 1))
          ⇒ (* 2 ((G (s s)) 1))
          ⇒ (* 2 (* 1 ((s s) 0)))
          ⇒ (* 2 (* 1 ((G (s s)) 0)))
          ⇒ (* 2 (* 1 1))
          ⇒ 2
```

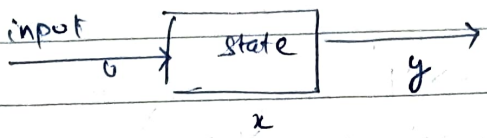
```
(define Ya
  (lambda (g)
    (let ((s (lambda (x)
               (lambda (n)
                 ((g (x x)) n)))))
      (s s))))

(define ! (Ya G))
```

ya : g

s = x → n →

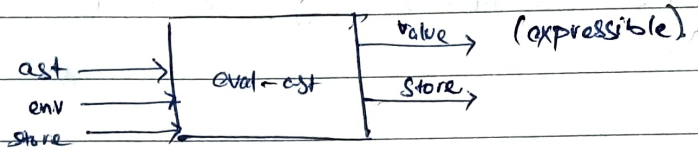
g (x (x)) (n)

s (s)



$x' = f(x, u)$

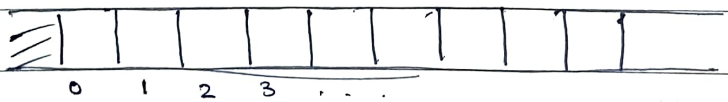
$y = h(x)$



eval-ast : [ast? env? store?] → [val? store?]

imperative programming.

modeling store.



STORE : $\mathbb{N}$ (Loc) → storable values?

ENV : $\mathbb{N}$ → Denotable Values.

C : $\mathbb{N}$ → memory address } ENV.

x = x + 1

mapping remains same

Store gets modified.

pointers : storable = location.

① g: f
f() × ✓

g(g) ×

g: f
<something>
f()

← g(g)

infinite loop. ×

② ~~g: f~~
f(f)

g: f
<something>
f(f)

← g(g)

infinite loop ✓

③ g: f n
if n==0 → 1
else → n * f(f n-1)

← g(g 5) 120.

goto 5!

③ g: f n
if n==0 → 1
else → n * f(f n-1)

← λ: n
g(g n)

get !

④ g : f g : f

λ : n

if n == 0 → 1

else → n * f(f)(n-1)

← g(g)

⑤ g : f

λ : n

if n == 0 → 1

else → n * f(n-1)

← g(g(g)) ✗

③ (g : f

λ : n

if n == 0 → 1

else → n * f(n-1)

y : y :

S = λ(x) : g(x(x))

(S S)

λ(S) :

(S(S)) (λ(x) g(x

inf (inf (x(x) : g(x(x)))

ggg : f
g (f (f))

inf (ggg)

inf : f
f (f)

g : f
x : n
if n == 0 → 1
else → n * f(n-1)

ggg : f
g (f (f))

inf : f x
x ()
f (f)

inf : f x
f (f x ())

← x
inf (inf x)

inf (ggg)

ya : g
s = x → n →
g (x (x)) (n)
s (s)

ya : g
s (s)
x → n →

ya: g
s =

g: (f) → g: f
(n) →

g: (f) → (n) →

g: f → n →
if n==0 → 1
else → n * f(n-1)

~~ya~~ s: x → n →
g(x(x)) (n)
← s(s)
⇓
s: x → n →
g(x) (n)
s(s) ✗

g: f
λ: n
if n==0 → 1
else → n * f(n-1)

s: x
λ: n
g (x) (n)
s(s)

s: x
g(x)
s(s)

inf: x
x(x)
inf(inf).

s: x
g(x(x))
s(s)

s: x
x: n
g(x)(n)
s(s)

s: x
x: n
g(x(x))(n)
s(s)

d: f
x: n
[illegible](n)

s: x
g(d(x))
s(s)

d: f
x: n
f(n)

s: x
g(d(x(n)))
s(s)

```
int fact(int n)
{
    if(n==0 || n==1)
    return 1
    else
    return n * fact(n-1)
}
```