While loop

Monday, 21 March 2022 8:52 AM

type: void

exp: (while exp exp) → cond, → body

(set! var exp) → var = exp

(begin es body) (let x rhs body)

(+)

set! x 1 → void

(let x 1 x)

(begin es body)

es → set of effects (void)

body → return (int | bool | void)

(begin (es) es body (body))

set! → vox exp-cont.

Type checker

(while exp exp) → void

exp → bool

exp → void

set! s 0 ;-

while (< i 10)

(+ i 1)

;set! i (+ i 1)

(+ (let x [x]) (<))

(while cnd body) → x

(x + 5)

set! x rhs

type x = type rhs and

(set! x 2) (type ≠ void)

(set! x 2) (type ≠ void)

(set! x #t) ✗

int x = 2;

(begin

es ⟶ void

vody) → (void)

type begin = type vody

(let ([sum 0])
  (let ([i 5])
    (begin
      (while (> i 0)
        (begin
          (set! sum (+ sum i))
          (set! i (- i 1)))
        sum))))

(let ([sum 0])

(let ([i 5])

(begin

(while (> i 0)

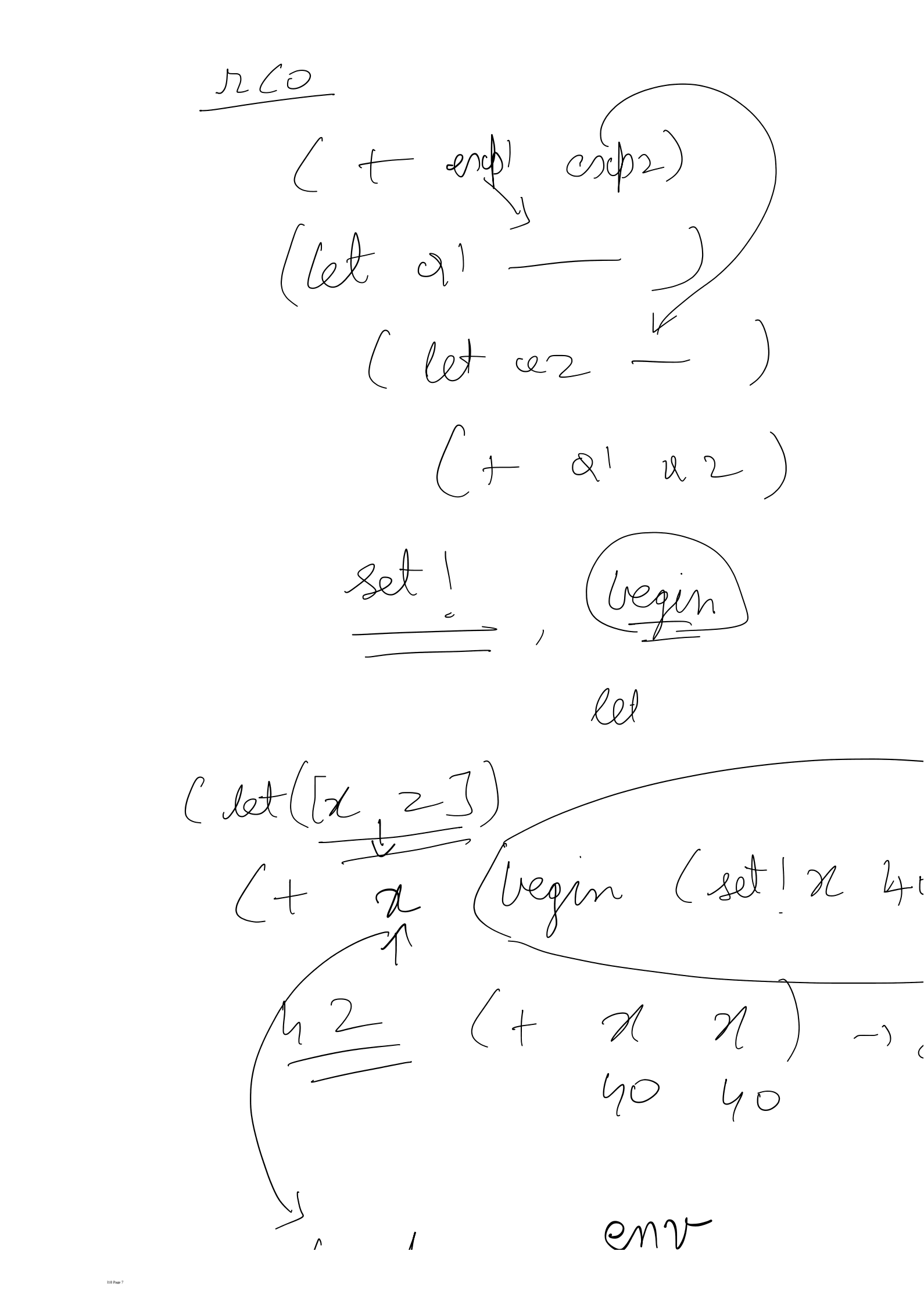
(begin

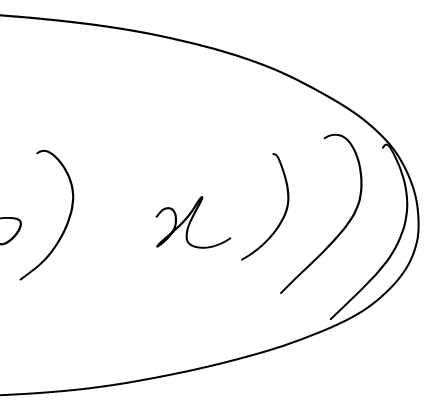
((set! sum (+ sum i))

(set! i (- i 1))))))

sum)))

RCO


RCO
(+ exp1 exp2)
(let a1 ——)
(let a2 —)
(+ a1 a2)
set!
=
begin
let
(let([x 2])
(+ x
begin (set! x 4
42
(+ x x)
40 40
env



bound. env

(get!)

(let [x 2]

(let [y 0]

(+ y (+ x (begin

(get! x))

(let t1 (get!

set! x

get!

(define collet-set! e

match

(set! x 40))))

x) t1)

rly

))

set!

(set! x rhs

(set-union (x

get!

(var x)

if (x ∈ set

(get! x

(var x)

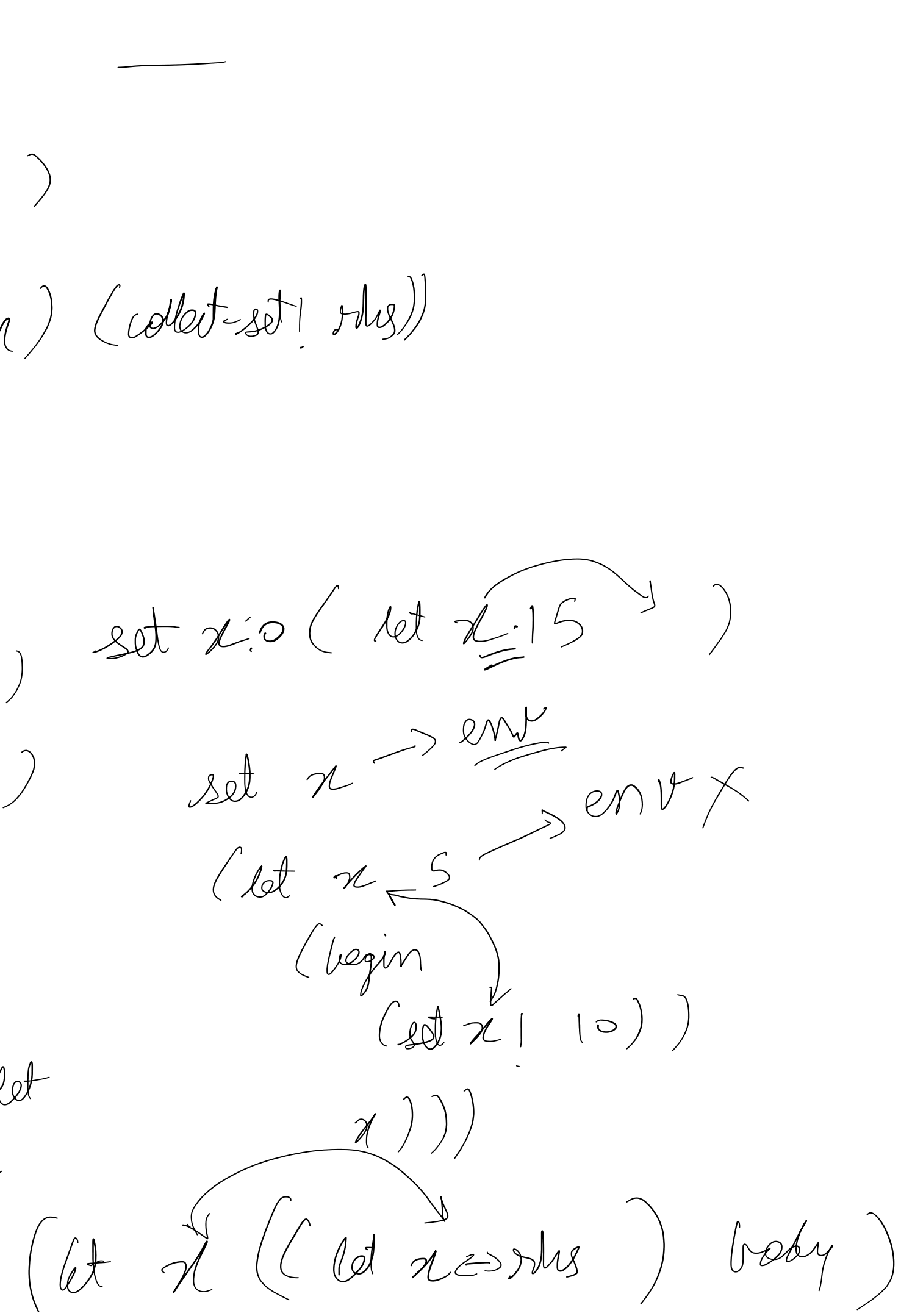
set! -> (set

(

get!

env

end


set x:0 (let x:15)
set x -> env
env x
(let x 5
(begin
(set x 10))
x)))
(let x ((let x=>rhs) body))

(let x ((e x2 rhs) ... body))

set!

(rco-atom (Begin es body)

-> es1 = (map rco-exp es)

body1 = (rco_exp body)

tmp <- (+ (begin set! x 0 y) -> (le

(begin set! y 0 x))

rco_exp (+ 0 0)

alues (tmp1) rco_atom (len)

tmp2 rco. t (——)

(+ tmp1 tmp2)

(tmp. (Begin es1 body1)

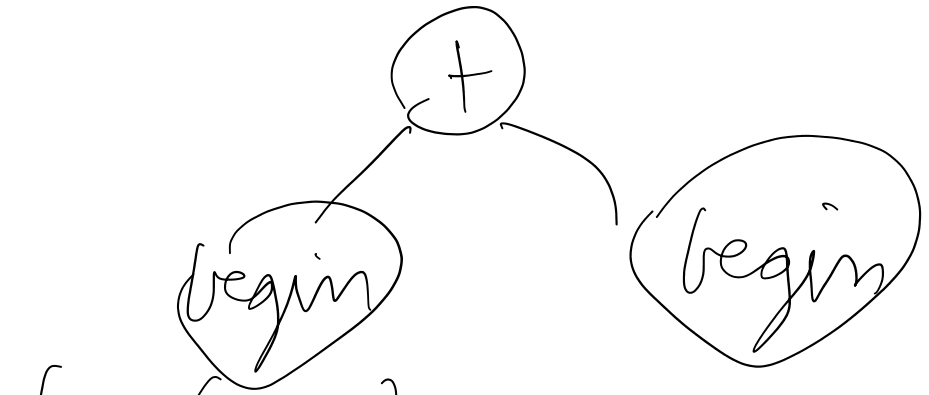
t - (_)

(let _ (_)

(+ x y))

(values (var tmp))

(tmp. (Begin es) body))



(let (x 2)

(+ (begin set! x 4e

))

- x) x)))