2-1

$$(+\ (+\ \overbrace{(+\ 1\ 2)}^{3}\ \text{read})$$
$$\quad (+\ \text{read}\ \underbrace{(+\ 7\ 3)}_{10}))$$

(+ 10 read)

e'_l = (+ 3 read)

e'_r = (+ 10 read)

return (+ 13 (+ read read))

$R_0 \rightarrow R_1$

R_1 = e ::= .... | var | let $\underset{x}{\text{var}}$:= $\underset{xe}{e}$ in $\underset{be}{e}$

R_0 interp : e → num

R_1 interp : $\underbrace{\text{env}}_{(\text{var} \rightarrow \text{num})}$ × e → num

interp env (Neg e) = -1 * (interp env e)

interp env (Var x) = env x

interp env (Let x xe be) = interp env' be

with env' = env [x → interp env xe]

interp env (Add l r) = (interp env l) + (interp env r)

(Add (Let x (Num 1) (Num 2)) (Var x))

R_0 randp : num → e

R_1 randp : set(var) × num → e

randp vs 0 = choice 0 → random number
1 → read

randp vs (1+n) = 2 → looking at var in vs

choices ...

→ let x' := (randp vs n) in (randp vs' n)

x' = random string vs' = vs ∪ {x'}

2-2

```
R0.opt : e → e
R1.opt : env × e → e
        (var→e)

opt env (Var x) = env x
opt env (Let x xe be) :=
  xe' = opt env xe
  if simple? xe' then
        opt (env[x→xe']) be
     else
        let x := xe' in (opt env[x→x] be)
```

```
let x := 7 in
  (+ ~~x~~7 ~~x~~7)
  => (+ 7 7)
  => 14

let x := read in
  (+ x x)

let x := (+ 7 read) in
```

```
simple? : e → bool
s? (Num _) = true
s? (Var _) = true
s? _ = false
```

compile : R_1 → X_0

step 3 : X_0 → X_0
(may vars) (won't be vars)

```
X0 : p   := program info [label → block] ...
     blk := block info instr ...
     instr := addq arg, arg  | subq arg, arg  | movq arg, arg
              retq           | negq arg       | callq label
              jmp label      | pushq arg      | popq arg
     arg  := number ($n)     | reg (%rn)      | mem %rn(offset)
             var (x)
                     rn := rsp | rbp | rax | rbx | rcx | rdx | rsi | rdi
                           r8 → r15
```

```
xi : Xp → num     xi (program _ l→blk) := xbi ms0         _main
  ms  := (rn → num) × (addr num → num) × (var → num)  × (l→blk)
  ms0 = (rn = 0) × (addr = 0) × (var = 0)  × l→blk
xbi : ms × label := xsi ms (l→blk label), instrs
xsi : ms × List(instr) := if~~empty? case~~ ~~xs ms (rest is)~~    where ms' = xii ms
                     ~~[illegible]~~ xii ms (first is) (rest is)
```

-3

```
xii ms (addq src, dst) k = xsi ms' k
  ms' = ms [dst -> ms(src) + ms(dst)]
movq   ms' = ms [dst -> ms(src)]
pushq src   ms' = ms [ %rsp(0) -> ms (src)
                       %rsp -> ms (%rsp) - 8 ]
popq dst    ms' = ms [ dst -> ms (%rsp(0))
                       %rsp -> ms (%rsp) + 8 ]
jmp label    := xbi ms label
retq         := escape ms (%rax)
callq _read-int := ms [%rax -> ask the user or debug]
```

R1	X0
tree-shaped & recursive (+ e e)	linear and structured (addq arg arg)
infinite variable	fixed registers
variables are scoped	variables/registers are global

```
C0  p = (program info [label -> tail] ...)
    tail = (return arg) | (seq stmt tail)
    stmt = set! var expr
    expr = arg | (read) | (- arg) | (+ arg arg)
    arg = number | var

interp.Cp = interp.Ct (empty env) (list main)
interp.Ct env (ret arg) = interp.Ca env arg
              (seq s t) = interp.Ct env'
                          env' = interp.Cs env s
interp.Cs env (set! x e) = env [x -> (interp.Ce env e)]
interp.Ce env (Arg a) = interp.Ca env a
interp.Ca env n = n   (read) = ask
          v = env v   (neg a) = -1 * Ca env a
                      (add l r) = (Ca env l) + (Ca env r)
```

2-4

```
r : R1          interp  =  ans        test (r)
r' = opt(r)  ⟶
r''          ⟶
r'''         ⟶
c : C0       ↗
c'           ↗
c''          ↗
             ↗
X : X0       ↗
x'           ↗
x''          ↗
```

uniquify : $R_1 \to R_1$

job: remove scopedness from vars

```
(+ (let x=7 in x)                 (+ (let A=7 in A)
   (let x=8 in          →            (let B=8 in
     let x = 1+x in                    let C = 1+B in
        x+x ) )                           C+C ) )
```

```
uniquify : (var → var) × e
            old    new

uni σ (Var x) = (σ x)
uni σ (Let x xe be) =
    Let x' (uni σ xe) (uni σ' be)
  where σ' = σ[x → x']
        x' = something new
```