3-1)

R_0 $\quad$ $interp_r : R_0 \to ans$

$\quad$ $opt : R_0 \to R_0$

X_0 $\quad$ $interp_x : X_0 \to ans$

R_1 $\quad$ ..

..

compile : $R_1 \to X_0$

R_1: tree-shaped & recursive, expr-oriented structure

(+ 1 (+ 2 (- (+ 3 4))))

infinite variable (let x := 2 in ....)

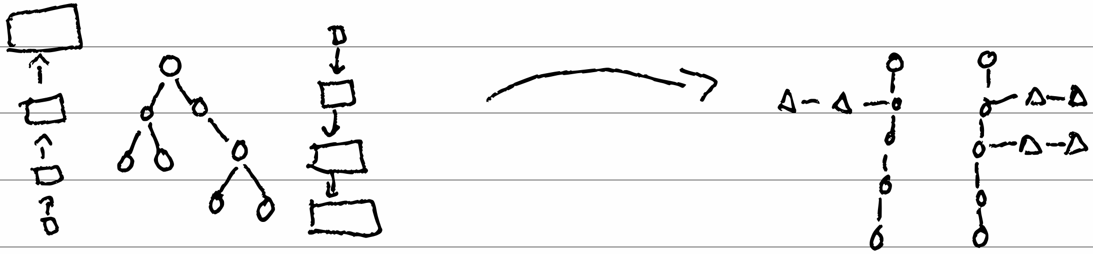
variables are scoped (+ (let x ...) (let x ...))

3-2)

X_0 : linear, heavily structured

fixed #/set of registers

registers/mem are global)



locally:

top-down

fixed-point/cyclic

bottom-up

3-3/ compilation w/ intermediate languages

R_1 opt

$$\begin{tikzcd}
R_1 \arrow[r, "\text{uniquify}"] \arrow[d, bend left, "\text{compile}"] & R_1 \arrow[r, "\text{flatten}"'] & C_0 \arrow[d, bend left, "\text{select-instr}"] \\
X_0 & X_0 \arrow[l, "\text{patch}"'] & X_0 \arrow[l, "\text{assign-homes}"']
\end{tikzcd}$$

- remove scope

removing nesting
rco
remove-complex

locally replace
w/ instructions

remove infinite variable

fix X86
weird

3-4/

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


Cip : p → ans            cip (progrm _ l→tm) =
                              cit ∅ (l→tm ("main"))

cit env (return arg) = cia env arg
        (seq stmt t) = cit (cis env stmt) t
```

3-51

```
cis env (set! v e)  = env[v ↦ cie env e]
cie env a           = cia env a
        (read)      = read from the user
        (Neg a)     = -1 * cia env a
        (Add l r)   = cia env l + cia env r
cia env (Num n)     = n
        (Var v)     = env[v]
```

3-b)

uniquify : $R_1 \to R_1$

job : remove scopedness from R

```
(+ (let x := 7 in x)                 (+ (let v0 = 7 in v0)
   (let x = 8 in           ↷            (let v1 = 8 in
   (let x = 1 + x in                     (let v2 = 1 + v1 in
      (+ x x))))                            (+ v2 v2))))
```

uni : $(var \Rightarrow \overset{\sigma}{var)} \Rightarrow e \Rightarrow e$

uni σ (Var v) = Var (σ v)

uni σ (Let x xe be) = Let x' (uni σ xe) (uni σ' be)

where x' = is the next variable $\sigma' = \sigma[x \mapsto x']$

3-7

```
uni σ   (Add l r) = Add (uni σ l) (uni σ r)
        (Neg e)   = Neg (uni σ e)
        (Num n)   = Num n

for i = 0 to 1024
    p = randp 6
    check (interp p)   (interp (uni p))
    check (in~~terp~~ p)   (interp (opt p))
check-all-eq  p , (uni p) , (opt (uni p))
```

3-8|

(assume that all vars are unique)

rco : $R_1 \Rightarrow R_1^*$ flatten = econ ∘ rco

```
p = (program info e)
e = arg | (let x = c in e)
c = (read) | (- arg) | (+ arg arg)
arg = num | var
```

```
in :  (+ (+ 2 3) (let x = read in (+ x x)))
out : (let v0 = (+ 2 3) in
       (let x = (read) in
       (let v1 = (+ x x) in
       (let v2 = (+ v0 v1) in v2))))
```

$\underline{3\text{-}9}$ $rco : (\text{~~xvar~~} \mapsto \underset{\text{any}}{e}) \times \underset{\text{any}}{e} \to (\text{list } (\text{xvar} \times \underset{c}{c})) \times \underset{\text{any}}{e}$

$rco\ (\text{program} ; e) = \text{let } x_0 = c_0 \text{ in } \text{~~let~~} \ldots \text{ let } x_n = c_n \text{ in}$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{ans}$

$[x_0, c_0] \ldots [x_n, c_n], \text{ans}) = rco\ \emptyset\ e$

$rco\ \sigma\ (\text{Num } n) = (\emptyset, (\text{Num } n))$

$(\text{Var } x) = (\emptyset, \sigma(x))$

$(\text{Add } e_L\ e_R) = ((nv_L +\!\!+\ nv_R +\!\!+ [(x, \text{Add } a_L\ a_R)],$
$x)$

where $(nv_L, a_L) = rco\ \sigma\ e_L$

$(nv_R, a_R) = rco\ \sigma\ e_R$

x is a fresh variable

$rco\ \sigma\ (\text{Neg } e_L) = (nv_L +\!\!+ [(x, (\text{Neg } a_L))], x)$

where $(nv_L, a_L) = rco\ \sigma\ e_L$

$rco\ \sigma\ (\text{Read}) = ([(x, (\text{Read}))], x)$

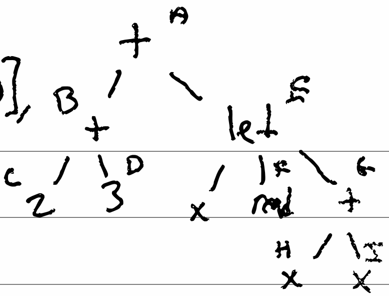
$rco\ \sigma\ (\text{Let } x\ xe\ be) = (nv_x +\!\!+\ nv_b, a_b)$

where $(nv_x, a_x) = rco\ \sigma\ xe$

$(nv_b, a_b) = rco\ \sigma'\ be$

$\sigma' = \sigma[x \mapsto a_x]$

3-10)



rco $\emptyset$ $A = [(v_0, (+2\ 3)), v_1, v_2, (v_3, (+ v_0\ v_2))],$

v_2

rco $\emptyset$ $B = [(v_0, (+ 2\ 3))], v_0$

rco $\emptyset$ $C = [\,], 2$

rco $\emptyset$ $D = [\,], 3$

rco $\emptyset$ $E = [(v_1, (\text{read})), (v_2, (+ v_1\ v_1))], v_2$

rco $\emptyset$ $F = [(v_1, (\text{read}))], v_1$

rco $[x \mapsto v_1]$ $G = [(v_2, (+\ v_1\ v_1))], v_2$

rco " $H = [\,], v_1$

rco " $I = [\,], v_1$

let $v_0 = (+2\ 3)$
let $v_1 = (\text{read})$
let $v_2 = (+ v_1\ v_1)$
let $v_3 = (+ v_0\ v_2)$
v_3

3-11/ → explicate-control

econ : R_1 (rco-style) → C_0

econ (Program : e) = (Program , [main ⇒ t])
where t = econ_e e

econ_e arg = (return arg)

econ_e (let x = xe in be) = (seq (set! x xe)
(econ_e be))