3-1

Diffs from R —(eon)→ C → $\hat{X}$ → $\dot{X}$ → X

	rco	C	$\hat{X}$	$\dot{X}$		
	tree-like recursive	linear structured	linear structured			
	infinite vars	infinite vars	~~Part~~ infinite ~~regs~~		Initial	Init
uniquify →	static scope	global	global			
	sensible	sensible	~~Insane~~ sensible		sensible	insane

resolve-complex (rco) remove recursive structure

rco : R → Rco

rco := p = (program info e)

e = arg | (let x = c in e)

c = read | (- arg) | (+ arg arg)

arg = number | var

in: (+ (+ 2 3) (let x = read in (+ x x)))

rco: let y = (+ 2 3) in
let x' = read in
let z = (+ x' x') in
let x = (+ y z) in
x

(let x = 8 in (+ x x)) → let z = (+ 8 8) in z

rco : e × (x → e) → (list (x * e)) × e (arg)

rco (program i e) = (program i (let x = ex in ...
([x ex] ...), e' = rco e ∅ e'))

rco σ (num n) = nt, (num n)

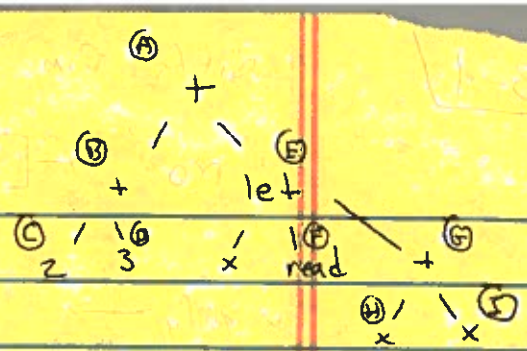
rco σ (var x) = nt, ~~σ(x)~~ σ(x)

rco σ (+ e1 er) = ~~nv1 ++ nvr~~ nv1, e1' = rco σ e1
~~(+ e1' er')~~ nvr, er' = rco σ er
((nv1 ++ nvr ++ [x, (+ e1' er')]), x)

rco σ (- ea) = ((nva ++ [x, (- ea')]), x)
(nva, ea') = rco σ ea

rco σ (read) = (([x, read]), x)

3-2



rco σ (let x = e_x in e_b) =

(nv_x, e'_x) = rco σ e_x (x)

(nv_b, e'_b) = rco σ' e_b

σ' = σ [x → e'_x]

$(nv_x$ ++ nv_b , $e'_b)$

rco ∅ A = $[rco_0, (+\ 2\ 3)]\ [rco_1, read]\ [rco_2, (+ rco_1\ rco_1)]\ [rco_3, (+\ rco_0\ rco_2)]$, rco_3

rco ∅ B = $[rco_0, (+\ 2\ 3)]$, rco_0

rco ∅ C = mt , 2

rco ∅ D = mt , 3

rco ∅ E = $[rco_1, read]\ [rco_2, (+ rco_1\ rco_1)]$, rco_2

rco ∅ F = $[rco_1, read]$, rco_1

rco $[x \to rco_1]$ G = $[rco_2, (+ rco_1\ rco_1)]$, rco_2

rco " H = mt , rco_1

rco " I = mt , rco_1

let rco_0 = (+ 2 3) in
let rco_1 = read in
let rco_2 = (+ rco_1 rco_1) in
let rco_3 = (+ rco_0 rco_2) in
rco_3

---

explicate-control (econ) : R_{rco} → C

econ (program r i e) = (seq program$_c$ i [BODY → t])

t = econe e

econe arg = return (econa arg)

econe (let x = e_x in e_b) = seq (set! x (econc e_x)) (econe e_b)

econe : e_{rco} → t

econa : arg_{rco} → arg_c

econc : complex rco → expr c

R C

econc (read) = (read)

econc (- a) = (- (econa a))

econc (+ l r) = (+ (econa l) (econa r))

R C

econa (num n) = (num n)

(var x) = (var x)

uncover : C → C vars →

before info = ∅

after: [{ rco_1, rco_2, rco_3, rco_0 }]

info' = info [vars → (x, ...)]

where t = seq (set! x ?) ... return ?

uncover (program info [BODY → t])

(program info' [BODY → t])