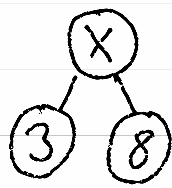
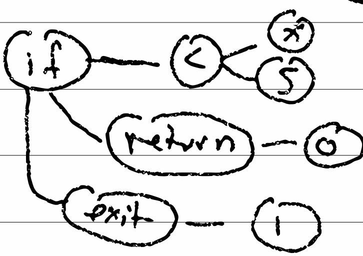
1-1 1 + 1 5 1+

1 x 3 " 3 x 8 "



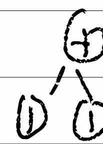
```
if (x < 5) {
  return 0; }
else {
  exit(1); }
```



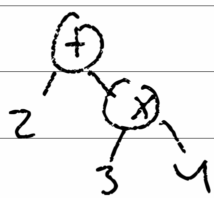
(+ 1 1)

top

children



(+ 2 (x 3 4))



1-2) $J_0 \Rightarrow$ e := v | (+ e e)

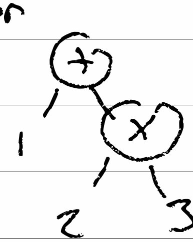
v := number | (* e e)

(+ 1 (+ 2 3)) $\in J_0$

```
interface Joe { }
class JNumber implements Joe {
  int n;   JNumber( int _n) { n=_n;}}
class JPlus imp Joe {
  Joe left, right;   JPlus( . . . ) }
class JMult imp Joe {
  Joe l, r;   JMult () ... {} }
```

$\to$ Sexpr

(+ 1 (* 2 3)) =



```
new JPlus (
  new JNum (1),
  new JMult(
    new JNum (2))
    new JNum (3)))
```

= JP(JN(1), JM (JN(2), JN(3)))

```
class JPlus :
  def _init (l, r) :
    this.l = l;
    this.r = r;
```

BST n := mt | (br num n n)

1-3/ ① pp : $J_0 \Rightarrow$ string

② pp n = itos (n)

③ pp (+ e_L e_R) = "(" ++ pp(e_L) ++ "+" ++ pp(e_R) ++ ")"

④ pp (x e_L e_R) = "(" ++ pp(e_L) ++ "*" ++ pp(e_R) ++ ")"

① interface Joe { public String pp(); }

② class JNum { ...

public String pp() {

return intToStr(n); } }

③ class JPlus {

public String pp() {

return this.left.pp() ~~++~~ "+" + this.right.pp() }

1-4) big-step interpreter

interp : e → v

interp n = n

interp (+ eL eR) = interp eL + interp eR

(1) interp (* eL eR) = interp eL * interp eR

→ class JMult {

public int interp () {

return this.left.interp() * this.right.interp(); }}

(+ 1 2 3) = (+ 1 (+ 2 3))

desugar →

se = empty | (cons se se) | string

pair

(a b c) = (pair "a" (pair "b" (pair "c" mt)))

(+ 1 2) = (p "+" (p "1" (p "2" mt)))

(+ 1 (+ 2 3)) = (p "+" (p "1"

(p (p "+" (p "2"

(p "3" mt)))

mt)))

1-5) desugar for J_0

```
("-" e)      => (* -1 (desugar e))
("-" e1 e2)  => (+ (d e1) (de ("-" e2)))
("+")        => 0
("+" e1 more ...) =>
   (+ (d e1) (d ("+" more ....)))
'*'                    = 1
"x"
(x                            "x"
```

$$se \xrightarrow[\text{desugar}]{} J_0 \xrightarrow[\text{interp}]{} V$$

$$\xrightarrow[\text{compile}]{} bc \xrightarrow[\text{vm}]{} V$$

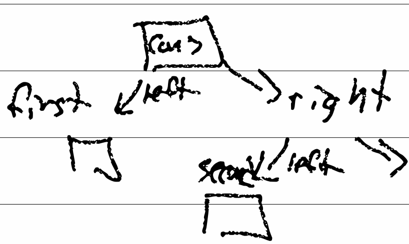
2-)) b

desugarer   (- e1) => (* -1 e1')
(- e1 e2) => (+ e1' (- e2))
(+) => 0
(+ e1 e2 ...) =>
(+ e1' (+ e2 ...))

def
desugar (se) :
if ~~isCons~~(se) && length (se) = 2 &&
first (se) == "-" then

> def length (se) :
> if is Null (se) : return 0
> else is Cons (se) : return 1 + length (right(se))
> else ~~false~~ false

new JMult (new JNum (-1), desugar(
Second (se)))



if isList(se) && len(se) = 3 && first (se)=="+"
then : return new JAdd(desugar(sec(se)),
desugar(new Cons("-", ~~...~~
newCons(third(se), null)

2-2] cons(a, cons(b, cone(c, null))))

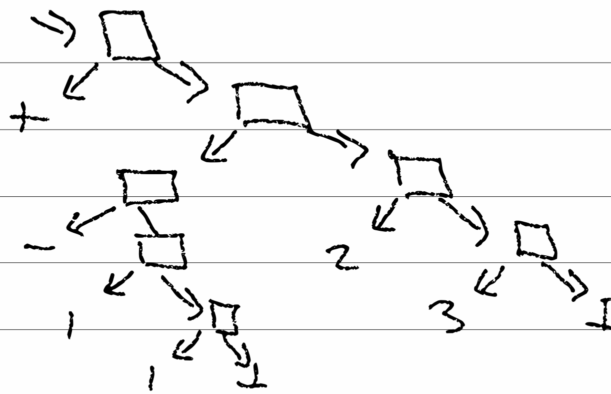
len = 3

(+ (- 1 1) len = 4

1 2 2 ← 3

3)

4



isList(se):

Null: True

Cons: isList(right(se))

ow: False

Len(se):

Null: 0

cons: 1 + len(right(se))

2-3 $J_0 \Rightarrow J_1$

$$\begin{aligned} e &::= v \mid (\overset{\text{fun}}{e}\ \overset{\text{args}}{e \ldots}) \\ &\mid (\text{if}\ \underset{c}{e}\ \underset{t}{e}\ \underset{f}{e}) \end{aligned}$$

$v ::= b$

$b ::=$ some set of constants

// in J_0, b = num | + | *

numbers | bools | prim

prim = $+, -, *, /, \leq, <, =, >, \geq, \ldots$

interp $v = v$

interp (if e_c e_t e_f) = interp e_k

where e_k = if interp e_c then e_t o.w. e_f

interp (e_f e_a ...) = $\delta(p, v_a \ldots)$

where p = interp e_f

$v_a \ldots$ = interp e_a ...

$\delta : b \ldots \to b$

$\delta(+, 1, 2) = 3$ $\delta(/, 1, 0) = \perp$

$\delta(<, 1, 3) = $ true

2/4 "small step interp"   "big step"

$e \Rightarrow e$ $e \Rightarrow v$

until its the same

interp   Interp

```
Interp e =
  let e' = interp(e)
  if e == e' then
     ret e
  o.w.
     Interp(e')
```

(+ (+ 1 1) 2) ⇐ (+ (+ 1 1) (+ 1 1))

⇒ (+ 2 2)

⇒ 4

↓

(+ 2 (+ 1 1))

```
int x = 1;
f( x--, x++)
```

(1, 0)

(2, 1)

2-5/ step : e → e

step (if true et ef) = et
step (if false et ef) = ef
step (p va ...) = δ(p, va ...)
step v = v
step (if e(≠v) et ef) =
(if (step e) et ef) or (if e (step et) ef)
step (vb .... e(≠v) ea ...) =
(vb ... (step e) ea ...)

A context
C ::= hole | if0 C e e
| if1 e C e
| if2 e e C
| (e... C e ...)
plug C e (C[e])
plug hole x = x
plug (if0 C e1 e2) x = if x e1 e2
plug (if1 e1 C e2) x = if e1 x e2
plug (e1 ... C e2 ..) x = (e1 ... x e2 ...)

2-6

step C [if true et ef] =
C [et]

step C [if false et ef] =
C [ef]

step C [p va ...] = C [δ(p, va ...)]

~~step~~ "parse" : e $\Rightarrow$ C $\times$ e

step $\Longrightarrow$ e

interp e = if e $\in$ V then e

C, e' = parse e e

e'' = step e'

plug C e''

redex

parse : e $\Rightarrow$ C $\times$ e

parse (if ec et ef) =

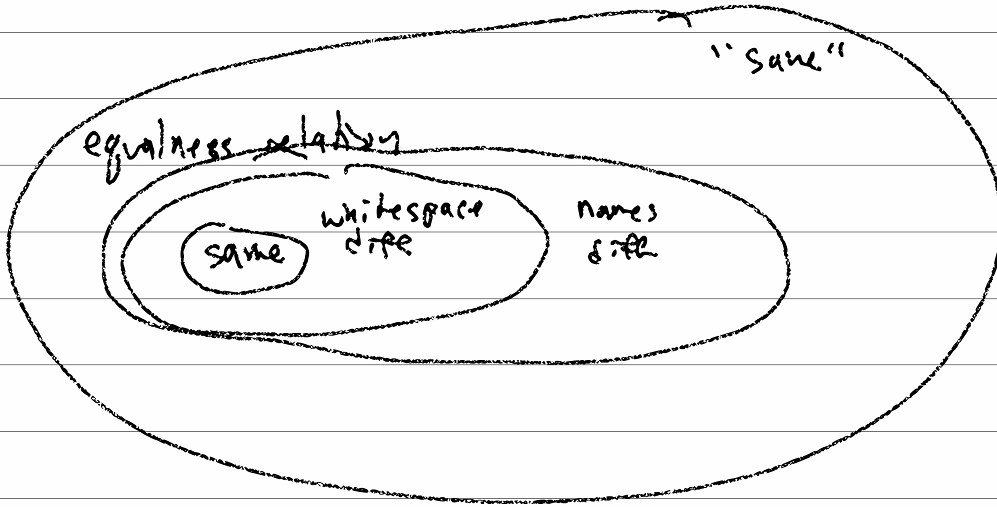
if ec $\in$ V then (hole , e)

o.w. let C', e' = parse ec

(if0 C' et ef , e')

2-7) Answer: Contexts

Question: How do I know when two programs do the same thing?



$x = y$

$\forall x.\ f\,x = g\,x$

$\forall C.\ C[x] = C[y]$

C=hole $\quad x = y$

C= (+ hole 2) $\quad x+2 = y+2$

C= (map hole (list 1 2))

....

Observational Equivalence

2-8)

$C ::= \text{hole} \mid \text{if } C\ e\ e$
$\mid \text{if } e\ C\ e$
$\mid \text{if } e\ e\ C$
$\mid (e \ldots\ C\ \ e \ldots)$

$E ::= \text{hole} \mid \text{if } E\ e\ e$
$\mid (v \ldots\ E\ \ e \ldots)$

"unique decomposition"

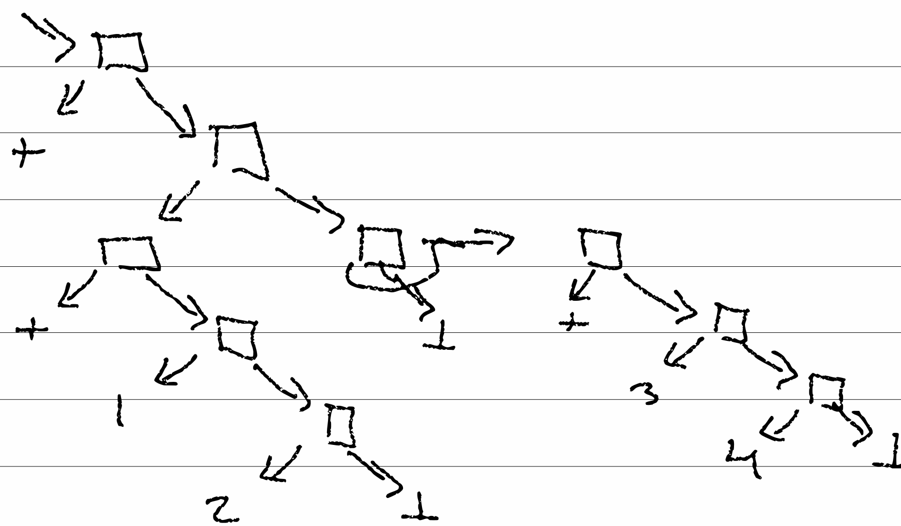
$\forall e.\ e \in v$ or $e = \underset{\text{unique } E}{E}[e']$ where $e' \notin v$

$\delta(1, 2) = \bot \qquad \delta(1, 1, 0) = \bot$

(+ (+ 1 2) (+ 3 4)) ← tree
← java tree

```
new JPlus (new JPlus (new JNum (1),
                      new JNum (2))
           new JPlus (new JNum (3),
                      new JNum (4)))
```

2-9)



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

```
3-1/  E = hole  | (if E e e)
       | (v ... E e ...)

step  E[if true et ef] → E[et]
step  E[if false et ef] → E[ef]
step  E[p va ...] → E[δ(p, va ...)]

interp e = case (parse e) of
             false → e
             (E, e) → e' = step e
               E[e']

gigantic program [(+ (+ 1 1)
                     (+ 2 3))]
```

3-2/ Jp "language"

CC_0 "machine" $e \to e$

$st \longmapsto st$

$$\begin{array}{ccc} \text{lang } e & \xrightarrow{\text{inject}} & \text{machine } st \\ \downarrow \text{step} & & \downarrow \text{step}^{(3)} \\ e' & \xleftarrow{\text{extract}} & st' \end{array}$$

done?
fv

$st = \langle e, E \rangle$ done? $\langle v, \text{hole} \rangle$

inject $e = \langle e, \text{hole} \rangle$

extract $\langle e, E \rangle = E[e]$

$\langle \text{if } e_c\ e_t\ e_f, E \rangle \longmapsto \langle e_c, E[\text{if hole } e_t\ e_f] \rangle$

$\langle \text{true}, E[\text{if hole } e_t\ e_f] \rangle \longmapsto \langle e_t, E \rangle.$

$\langle \text{false}, E[\text{if hole } e_t\ e_f] \rangle \longmapsto \langle e_f, E \rangle$

$\langle e_0\ e_1 \ldots, E \rangle \longmapsto \langle e_0, E[\text{hole } e_1 \ldots] \rangle$

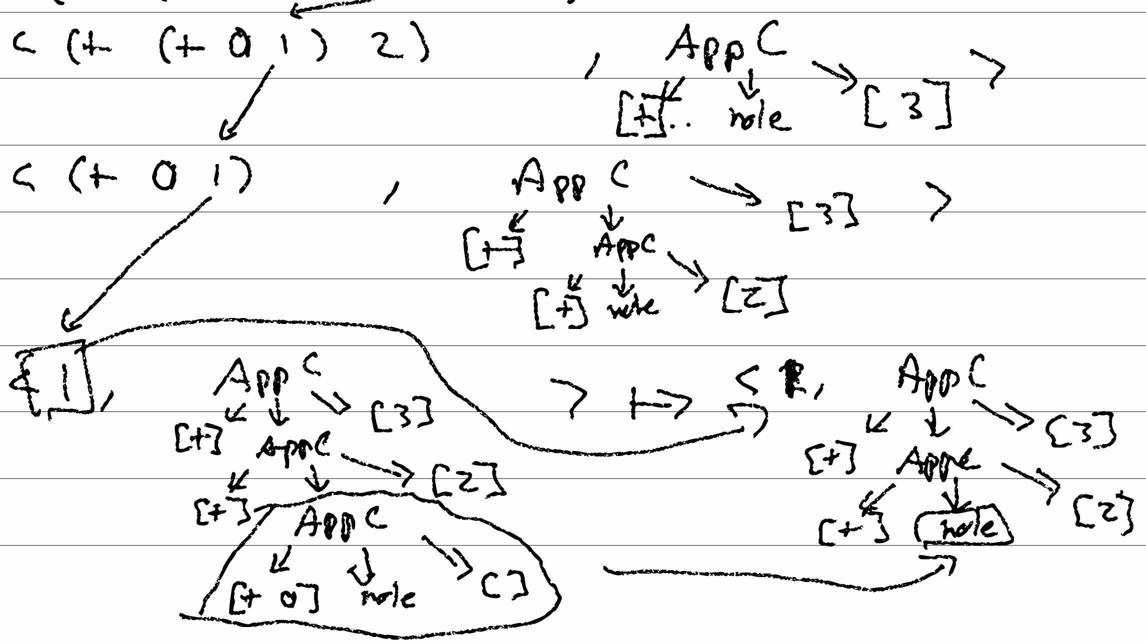
$\langle v_1, E[v_0 \ldots \text{hole } e_1\ e_2 \ldots] \rangle \longmapsto \langle e_1, E[v_0 \ldots v_1 \text{ hole } e_2 \ldots] \rangle$

$\langle v_n, E[v_0 \ldots \text{hole}] \rangle \longmapsto \langle \delta(v_0 \ldots v_n), E \rangle$

3) E = hole | if E e e | (v ... E e ...)

```
interface Context {
   Expr plug (Expr); }
Hole : Context {
   plug (e) = e; }
IfC : Context {
  Context c; Expr t, f;
   plug (e) = new If( c.plug(e), t, f); }
AppC : Context {
  List<V> vs; Context c; List<Expr> es;
   plug(e) = new App( vs ++ [c.plug(e)] ++ es);}
```

< (+ (+ (+ a 1) 2) 3) , hole >



3-4) $E = \text{hole} \mid \text{if } E\ e\ e \mid v \ldots E\ e \ldots$

$\quad = \text{top} \mid \text{if } e\ e\ \square \mid (v \ldots)(e \ldots)\ []$

$K = \text{kret} \mid \text{kif } e\ e\ k \mid \text{kapp } \vec{v}\ \vec{e}\ k$

CK$_0$ machine $\quad st = \langle e, k \rangle$

inject $e = \langle e, \text{kret} \rangle$

extract $\langle e, \text{kret} \rangle = e$

$\langle e, \text{kif } e_t\ e_f\ k \rangle = \text{extract } \langle \text{if } e\ e_t\ e_f, k \rangle$

$\langle e, \text{kapp } (v \ldots)(e_1 \ldots)\ k \rangle = \text{extract } \langle (v \ldots e\ e_1 \ldots), k \rangle$

done $\langle v, \text{kret} \rangle$

1 $\langle \text{if } e_c\ e_t\ e_f, k \rangle \mapsto \langle e_c, \text{kif } e_t\ e_f\ k \rangle$

2 $\langle \text{true}, \text{kif } e_t\ e_f\ k \rangle \mapsto \langle e_t, k \rangle$

3 $\langle \text{false}, \text{kif } e_t\ e_f\ k \rangle \mapsto \langle e_f, k \rangle$

4 $\langle e_0\ e_1 \ldots, k \rangle \mapsto \langle e_0, \text{kapp } ()\ (e_1 \ldots)\ k \rangle$

5 $\langle v_1, \text{kapp } (v_0 \ldots)\ (e_0\ e_1 \ldots)\ k \rangle$

$\quad \mapsto \langle e_0, \text{kapp } (v_0 \ldots v_1)\ (e_1 \ldots)\ k \rangle$

6 $\langle v_n, \text{kapp } (v_0 \ldots)\ ()\ k \rangle$

$\quad \mapsto \langle \delta(v_0 \ldots v_n), k \rangle$

3-5

while (1) {

- is e a value?
  - yes: is k a ret?
    - Y: return e
    - N: is k an if?
      - Y: is e true
        - rule 2
        - rule 3
      - N: is k_1 es empty
        - Y: rule 6
        - N: rule 5
  - no: is e an if
    - Y: rule 1
    - N: rule 4

rule 1:

    k = new kif (e.true, e.false, k)
    e = e.cond;
    ~~jump B.pc~~

k is a stack and the stack (of C)

= kontinuation

continuation

3-6/ struct if {
```
struct if {            struct expr {
  expr h;                enum tag; }
  expr *c, *t, *f; }

struct num {           enum tag {
  expr h;                IF, NUM, APP,
  int n; }               BOOL, PRIM,
                         KRET, KIF,
struct app {             KAPP, CONS, NIL};
  expr h;
  expr *f, *args; }

expr * make_if(expr *c, t, f){
  if * p = malloc(size ...))
  p -> h. tag = IF;
  p -> c = c;   ...
  return p; }
```

(+ (+ 1 1) 2)

```
make-add ( make_add (make-num(1),
                     make_num(1)),
           make_num(2) );
```

0 2 0 1 1 2 2 1 4 0 7

4-1 / J_0 $e = n \mid (+\ e\ e)$

$\mid (\cancel{*}\ e\ e)$

J_1 p = unary − (neg) , not (1)

\+ , * , between

$e = n \mid p \mid^{(2)} (\text{if}\ e\ e\ e)$

$\boxed{\mid (e\ \ldots)}$

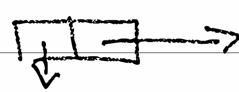
class JApp impl JExpr {
List < JExpres contents }

(+ 1 2) ⇒ App List (+ , 1 , 2)
↓ p ↓ n ↓ n

desugar

App ([Prim (PLUS), Num (1), Num (2)])

(cons "+" (cons "1" (cons "2" null)))



(+ 1 (+ 2 3))

App([Prim (PLUS), Num (1), Num (2)
App([Prim (PLUS), Num (2), Num (3)]])

(+ 1 3 (+ 2 3))

δ(+ 1 3 5) = ⊥

4-2)

```
Expr * Delta ( List <Expr *) args ) {
    if ( len (args ) == 3
        && args[0] == Prim ( PLUS)) {
        ret new Num ( (Num) arg[1]).n + ((Num) args[2]).n );}
}
```

(+) => 0

~~(+ n) => n~~

(+ n more ...) => (+ n de(+ more ...))

desugar ((cons "+" empty) = new Num (0);

```
delta (args)
  args[0] (args)
-->
  args[0].apply (args[1...])
```

J_0 or J_1 : prog = e

4-3 / J_2 :

e := v | (e e ...) | (if e e e)
  | x

v := number | bool | prim | f

X ∈ some set of variable names

f ∈ some set of function names

prog := d ... e

d := (define (f x ...) e)

(define (Double x) (+ x x))

— (Double (+ (Double 1) 3))

(define (Quad x) (Double (Double x)))

(Quad (+ 1 (Double 3)))

$f(x) = 1 + x$

$f(3)? = 1 + 3 = 4$

$f(x) = 1 - x$

$f(3+4) = 1 - (3+4)$

$\cancel{= -2 + 4 = 2}$

$f(7)$

$= 1 - 7 = -6$

Double (1 + 1) = Double (2)

(1+1) + (1+1) = 2 + 2

4-3

$E = \text{hole} \mid \text{if } E\ e\ e$
$\mid (v \ldots E\ e \ldots)$

~~$E[x] = \ldots$~~ $\quad \Sigma / E[\text{if } T\ e_t\ e_e] = E[e_t]$

$\Sigma / E[f\ v \ldots] = \ldots$

eval : $e \Rightarrow v \ldots$ smallstep $e \to e$
eval' : $p \Rightarrow v$
$\Sigma \times$ ... smallstep $\Sigma\ e \Rightarrow e$
?

$\Sigma : f \Rightarrow d$

eval' Σ ~~do~~ (define (~~f~~ x ...) e) : more
= eval' $\Sigma[f \mapsto d]$ more
eval' Σ e = do smallstep

$f(x) = 1 + x$
$f(4)$

$\Sigma / E[f\ v \ldots] = E[e[x \leftarrow v] \ldots]$
if $\Sigma(f) = $ (define (f x ...)
e)

4-4 / $e[x \leftarrow v]$ is pronounced
e where xs are replaced with v

subst ~~x~~ v ~~e~~ $\rightarrow$ e

subst x v v' = v'
subst x v x = v
subst x v x' = x'
subst x v (if e_c e_t e_f) =
(if $e_c[x \leftarrow v]$ $e_t[x \leftarrow v]$ $e_f[x \leftarrow v]$)
subst x v (e ...) =
(e[x←v] ...)

```
interface JExpr {
    JExpre subst (Variable x, JExpre v); }
class JVar {
    subst(x, v) { if (x == this.x)
                      return v
                  return this } }
class JIf {
    subst(x,v) {
    new JIf ( this.c.subst(x,v), this.t.subst(x,v),
              this.f.subst(x,v)); } }
```

4-5

CK

6: $\langle v_n, \text{kapp}((p\ v_0 \ldots), (), k) \rangle$

$\quad \mapsto \langle \delta(p\ v_0 \ldots v_n), k \rangle$

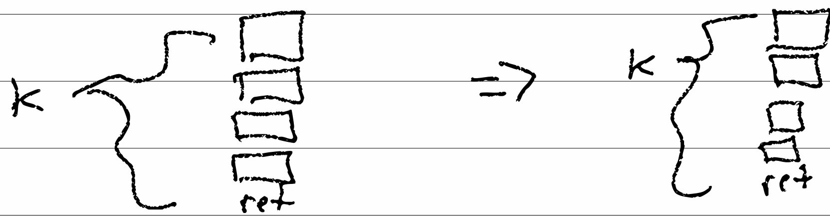
7: $\langle v_n, \text{kapp}((f\ v_0 \ldots), (), k) \rangle$

$\quad \mapsto \langle e[x_i \leftarrow v_i], k \rangle$

where $\Sigma(f) = \text{define } (f\ x_0 \ldots x_n)\ e)$

c = ... 7 ... 2 ...

c = 2   kapp (Expt 7)



$\langle (e_0\ e_1 \ldots), k \rangle$

$\quad \mapsto \langle e_0, \text{kapp}((), (e_1 \ldots), k) \rangle$

4-6 (define (F x) (F x))
(F 10)

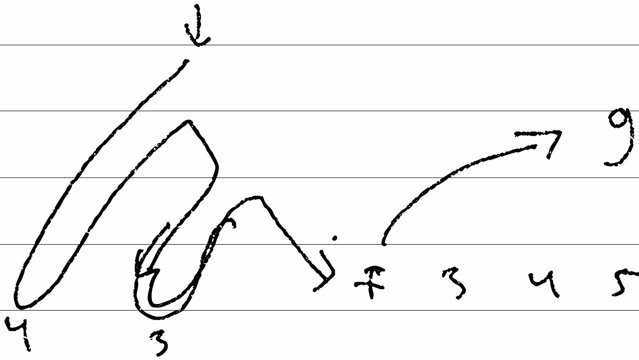
$\Rightarrow$ Σ = [F $\mapsto$ (define (F x) (F x))]
e = (F 10)
k = kret

< F 10 , kret> $\Leftarrow$
< F , kapp ((), (10), kret) >
< 10, kapp ((F), (), kret) >
& Σ(F) = (define (F x) (F x))
f= x... e=

< e [x ... $\leftarrow$ v...] , kret>
(F x) [x $\leftarrow$ 10]
< (F 10) , kret>

error $\Rightarrow$ Stack trace
$\Rightarrow$ fun "at fault" – F
fun that called ~~F~~ – G
"past" G – H

4-7)



g(. . —)

x = f 3 4 5

n(x)

6-1

$C = \text{hole} \mid \text{if } C\ e\ e$
$\mid \text{if } e\ C\ e$
$\mid \text{if } e\ e\ C$
$\mid e \ldots C\ e \ldots$

$E = \text{hole} \mid \text{if } E\ e\ e$
$\mid v \ldots E\ e \ldots$

if true (+ 1 2) 4

$E = \text{hole}$ $e = \uparrow$

$C = \text{hole}$ $e = \uparrow$

$C = \text{if true hole } 4$
$e = (+\ 1\ 2)$

(+ 1 2)

$E = \text{hole}$ $e = (+\ 1\ 2)$

find-redex $\underbrace{B = E[e]}$

step $e = e'$

6-2) fr (if e t f) =

if (value? e)

(hole , e)

o.w. (E , e) = fr e

(if(E t f , e)

fr (app es) =

for e in es

if (value? e)

(+ 1 2)

$\langle e_0 \; e \ldots , k \rangle$

$\mapsto \langle e_0 , \mathrm{kapp}((), (e \ldots), k) \rangle$

$\langle v_n , \mathrm{kapp}((v_0 \ldots), (\overset{e_1}{e_{n+1}} \ldots), k) \rangle$

$\mapsto \langle e_1 , \mathrm{kapp}((v_0 \ldots v_n), (e_{n+1} \ldots), k \rangle$

$\langle v_n , \mathrm{kapp}((p \; v_0 \ldots), (), k) \rangle$

$\mapsto \langle \delta(p, v_0 \ldots v_n), k \rangle$

6-3/ J_2 = PASCAL or C

top-level functions

CK = have the map $f \to d$

and we have subst

$\langle v_n, \text{kapp}((f\ v_0 \ldots), (), k)\rangle$

$\longmapsto \langle e[x_0 \leftarrow v_0] \ldots [x_n \leftarrow v_n], k\rangle$

where $\Sigma(f) =$ define $(f\ x_0 \ldots x_n)\ e$

$x[x \leftarrow v] = v$

$y[x \leftarrow v] = y$

$u[x \leftarrow v] = u$

$(\text{if } c\ t\ f)[x \leftarrow v] = (\text{if } c[x \leftarrow v]\ \ t[x \leftarrow v]\ \ f[x \leftarrow v])$

$(e \ldots)[x \leftarrow v] = (e[x \leftarrow v] \ldots)$

$\langle \text{if } c\ t\ f, k\rangle$

$\longmapsto \langle c, \text{kif}(t, f, k)\rangle$

OLD: NO rule for a variable in the c pos

$\langle x, k\rangle \longmapsto \ldots$

NEW:

$\langle x, k\rangle \longmapsto$ finally do the subst

(Σ)

G-Y) CEK $st = \langle e, env, k \rangle$

$env = \emptyset \mid env[x \leftarrow v]$

$k = kret \mid kif\ e\ e\ k$

$\mid kapp\ \vec{v}\ \vec{e}\ k$

$\langle x, env, k \rangle \mapsto \langle env(x), \emptyset, k \rangle$

$\langle if\ c\ t\ f, env, k \rangle$

$\mapsto \langle c, env, kif\ t\ f\ k \rangle$

$\langle true, env, kif\ t\ f\ k \rangle$

$\mapsto \langle t, env, k \rangle$

$\langle e_0\ e_1 \ldots, env, k \rangle$

$\mapsto \langle e_0, env, kapp\ ()\ (e_1 \ldots)\ k \rangle$

$\langle v_1, \cancel{env}, kapp\ (v_0 \ldots)\ (e_0\ e_1 \ldots)\ k \rangle$

$\mapsto \langle e_0, env, kapp\ (v_0 \ldots v_1)\ (e_1, \ldots)\ k \rangle$

$\langle v_n, env, kapp\ (v_0 \ldots)\ ()\ k \rangle$

$\mapsto \langle e, \emptyset[x_0 \leftarrow v_0] \ldots [x_n \leftarrow v_n], k \rangle$

where $\Sigma(f) = define\ (f\ x_0 \ldots x_n)\ e$

6-5/ define f x = x + z

define g z = f 4

g 2

define f x = 3

define g z = (f 1) + x

g 2

6-6

$CEK = \langle e, env, k \rangle$

$env = \emptyset \mid env[x \leftarrow v]$

$k = kret \mid kif\ env\ t\ f\ k$

$\quad \mid kapp\ \vec{v}\ env\ \vec{e}\ k$

$\langle x, env, k \rangle \mapsto \langle env(x), \emptyset, k \rangle$

$\langle if\ c\ t\ f, env, k \rangle \mapsto \langle c, env, kif\ env\ t\ f\ k \rangle$

$\langle true, \_, kif(env, t, f, k) \rangle \mapsto \langle t, env, k \rangle$

$\langle e_0\ e_1 \ldots, env, k \rangle$

$\quad \mapsto \langle e_0, env, kapp((), env, (e_1 \ldots), k) \rangle$

$\langle v_n, \_, kapp((v_0 \ldots), env, (e_0\ e_1 \ldots), k) \rangle$

$\quad \mapsto \langle e_0, env, kapp((v_0 \ldots v_n), env, (e_1 \ldots)\ k) \rangle$

$\langle v_n, \_, kapp((f\ v_0 \ldots), \ , (), k) \rangle$

$\quad \mapsto \langle e, \emptyset[x_0 \leftarrow v_0] \ldots [x_n \leftarrow v_n], k \rangle$

where $\Sigma(f) = define\ (f\ x_0 \ldots x_n)\ e$

"dynamic scope" — equal

- emacs lisp
- JS / Py / Ruby / perl / PHP / etc

specific vars are always dynamic

$new = \emptyset[A \leftarrow env(A)][B \leftarrow env(B)]$

"this"

2/5-7 PASCAL/C - all funs are top-level

$p = d \ldots \; e$

JS = (x) => 1 + x

Py = lambda: x : 1 + x

C++ = [](int x){return 1+x;}

J_3

$e = v \mid e \; e \ldots \mid \text{if } e \; e \; e \mid x$

$v = b \mid \boxed{(\lambda \; (x \ldots) \; e)}$ — new

$b = \text{num} \mid \text{bools} \mid \text{prim}$ // No f's

$E = \text{hole} \mid \text{if } E \; e \; e \mid v \ldots E \; e \ldots$

$$E[(\lambda \; (x_0 \ldots x_n) \; e) \; v_0 \ldots v_n] = E[e[x_0 \leftarrow v_0] \ldots [x_n \leftarrow v_n]]$$

$((\lambda x. ((\lambda y. x + y) \; 7)) \; 8) \Rightarrow 15$

let x = 8 in
let y = 7 in
x + y

let x = e_1 in e_2 =>

$(\lambda x. e_2) \; e_1$

let x = 8 in
let x = x + 1 in
x + x

6-8

$(\lambda\ (x_0 \ldots x_n)\ e)\ [y \leftarrow v]$

$= (\lambda\ (x_0 \ldots x_n)$

$e\ [y \leftarrow v])$

unless $y \in x_0 \ldots x_n$

$(\lambda x.\ (\lambda x.\ x+1))\ 7$

$(\lambda x.\ x+1)$

OLD

machine v = theory v

new

$(v ::= b \mid \star) \neq (v ::= b \mid \lambda(x\ldots)\ e)$

closure $(\lambda (x \ldots)\ e,\ env)$

$\langle \lambda(x\ldots).\ e,\ env,\ k \rangle \longmapsto \langle clo(\lambda(x\ldots) e,\ env),\ \emptyset,\ k \rangle$

$\langle v_n,\ -,\ kapp(\ (clo(\lambda (x_0 \ldots x_n).\ e,\ env)\ v_0 \ldots),$

$(),\ k) \rangle$

$\longmapsto \langle e,\ env[x_0 \leftarrow v_0] \ldots [x_n \leftarrow v_n],\ k \rangle$

7-1) J_3 : $v : \ldots \mid \lambda\,(x \ldots)\ e$

$\text{let } x = e_1 \text{ in } e_2$

$\Rightarrow (\lambda\,(x)\ e_2)\ e_1$

$\text{let } x = 1 \text{ in}$ $\quad (\lambda\,(x)$

$\text{let } y = 2 \text{ in}$ $\quad (\lambda\,(y)$

$x + y$ $\quad (+\ x\ y)\ 2)\ 1$

CEK_1 : $v = \ldots$ $\quad (\lambda\,(y)\ (+\ 1\ y))\ 2$

$\mid \cancel{\lambda\,(x)\ e}$

$\mid \text{clo}(\lambda\,(x \ldots)\ e,\ env)$

$\langle \lambda\,(x \ldots)\ e,\ env,\ k \rangle$

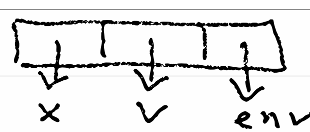
$\longmapsto \langle \text{clo}(\lambda\,(x \ldots)\ e,\ env),\ \emptyset,\ k \rangle$

$\langle v_n,\ \text{—},\ \text{kapp}((c\ v_0 \ldots),\ \text{—},\ (),\ k) \rangle$

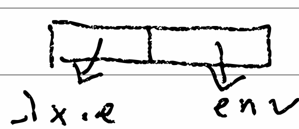
where $c = \text{clo}(\lambda\,(x, \ldots) e,\ env)$

$\longmapsto \langle e,\ env[x_0 \mapsto v_0] \ldots [x_n \mapsto v_n],\ k \rangle$

$env = \bot \mid$



$clo =$



let y = 3 in
let z = 8 in

7-2) λ (x) (+ x y)   clo   [8, 3, 19, 22, 36]

z → 8

y → 3

$\Downarrow$

$(\lambda. (+ \underbrace{\hat{0}}_{\text{static address}}\ \hat{1}), [\hat{1}])$

x → 19

a → 22

b → 36

...

SA = nat   env = vector v

FLAT - CLOSURES   [␣ , 3]

SA = (nat, nat)   env = ↓ , vector v

[8, 3, 19, 22, 36] env

$(\hat{0}, 0)$ $(\hat{1}, 1)$

^ , [␣ , ␣]   NESTED CLOSURES

7-3

$= v \mid (\text{if } e\ e\ e)$

$\quad x \mid (e\ e) \mid (p\ e\ e)$

$v = b \mid \lambda\ (x)\ e$

$b = \text{num} \mid \text{bools} \mid \text{prim}$

$\text{prim} = + \mid - \mid * \mid \div \mid <$

$(+\ 1\ 2)$ $\qquad \cancel{((\lambda\ (x\ y)\ (+\ x\ y))\ 1\ 2)}$

$(((\lambda x.\ \lambda y.\ (+\ x\ y))\ 1)\ 2)$

$= (+\ 1\ 2)$

$e = v \mid x \mid e\ e \qquad E = \text{hole}$

$v = \lambda x.\ e \qquad\qquad\quad \mid (E\ e)$

$\qquad\qquad\qquad\qquad\quad \mid (v\ E)$

$E[(\lambda x.e)\ v] \mapsto E[e[x \leftarrow v]]$

7-4/ What is a Bool really, man?

if True A B = A
if False A B = B

True = $\lambda x. \lambda y. x$
False = $\lambda x. \lambda y. y$
if = $\lambda c. \lambda x. \lambda y.\ c\ x\ y$ = $\lambda c. c$

if True A B = True A B = A

NOT T = F
NOT F = T
NOT = $\lambda b. \lambda x. \lambda y.\ b\ y\ x$

```
interface Bool { int choose (int, int); }
class True : Bool { True() int choose( x , y ) = x }
class False : Bool { False() int choose (x, y) = y }
class Not : Bool { Bool b. Not (Bool b) { this. b=b;}
                   int choose (x, y) {
                       return b. choose (y, x); } }
```

7-5/ What is a number?

zero := doesn't do something

one := does something once

two := does it twice

$$\text{add } \overset{n}{x} \ \overset{m}{y} = \cancel{xy} \underbrace{\qquad}_{}^{n+m}$$

$$\text{zero} := \lambda f. \lambda x. \quad x$$

$$\text{one} := \lambda f. \lambda x. \ f \ x$$

$$\text{two} := \lambda f. \lambda x. \ f \ (f \ x)$$

$$\text{add1} := \lambda n. \lambda f. \lambda x. \ f \ (n \ f \ x)$$

$$\text{add} := \lambda n. \lambda m. \lambda f. \lambda x. \quad n \ f \ (m \ f \ x)$$

$$\text{zero?} := \lambda n. \ n \ (\lambda x. \text{FALSE}) \ \text{TRUE}$$

$$\text{mult} := \lambda n. \lambda m. \lambda f. \lambda x. \ n \ (m \ f) \ x$$

two two

$$\text{two} \ (\text{two} \ f) \ x$$

$$(\lambda x. \ f \ f \ x)$$

$$(\lambda x. \ f \ f \ x)((\lambda x. \ f \ f \ x) \ x)$$

$$f \ f \ f \ f \ x$$

7-6 / Pair

$$\text{fst } (\text{pair } A \quad B) = A$$
$$\text{snd } (\text{pair } A \quad B) = B$$

$$\text{pair} = \lambda a. \lambda b. \lambda c. \text{ if } c \;\; a \;\; b$$
$$\text{fst} = \lambda p. \;\; p \text{ TRUE}$$
$$\text{snd} = \lambda p. \;\; p \text{ FALSE}$$

$$\text{sub1} := \lambda n. \text{ fst } (n \; (\lambda p. \text{ pair } (\text{snd } p) \; (\text{add1 } (\text{snd } p))) \; (\text{pair } z \;\; z))$$

$$\text{mkfac} := \lambda \text{fac}. \lambda n.$$
$$\text{if } (\text{zero? } n)$$
$$1 = \text{one}$$
$$(\text{mult } n \;\; (\text{fac } (\text{sub1 } n)))$$

$g(x) = x \cup \{a, b\}$ $\quad f(x) = 17 * x$

$$\underset{x}{\text{fac}} := \underset{F}{\text{mkfac}} \; \underset{x}{\text{fac}} \qquad x = F \;\; x$$

7-7) Fixed point of a lambda?

$$\text{FIX } F = x \qquad F\ x = x$$

$$F\ (\text{FIX } F) = \text{FIX } F$$

Z-combinator

$$\text{FIX} := \lambda F. ((\lambda x. F\ (\lambda v. x\ x\ v))\ (\lambda x. F\ (\lambda v. x\ x\ v)))$$

$$\text{FIX } F := ((\lambda x. F\ (\lambda v. x\ x\ v))\ (\lambda x. F\ (\lambda v. x\ x\ v)))$$

A

$$= F\ (\lambda v. A\ A\ v)$$

$$= F\ (A\ A)$$

$$= F\ ((\lambda x. F(\lambda v. x\ x\ v))\ (\lambda x. F(\lambda v. x\ x\ v)))$$

$\lambda x. \lambda y. x\ y$ — x

$$= F\ (\text{FIX } F)$$

Lambda-Calculus

n add1 0

$\lambda x. x+1$

Church Numeral / Church encoding

Talk about cc0

8-1 Lambda - Calculus

tiny : $e \to e$

tiny (if T et ec) = et
tiny (if F et ef) = ef
tiny (p $v_0 \ldots v_n$) = $\delta(p, v\ldots)$

small : $e \to e$

small e = let C, e' = find-redex e
let e'' = tiny e'
ret C[e'']

big : $e \to v$

big e =
  if $e \in v$: return e
  o.w. big (small e)

cc0 : $C \times e \to C \times e$

cc0 C e =
  $C' \times e'$ = move-context C e
  e'' = tiny e'
  ret (C', e'')

cc0* C e = if $e \in v$ and C = hole, ret C e
o.w. cc0* (cc0 C e)

big' : $e \to e$

extract (cc0* (inject e))

inject = (hole, e)   extract (hole, v) = v

8-2)

$\langle \text{if } e_c\ e_t\ e_f,\ E \rangle$

$\qquad \mapsto \langle e_c,\ E[\text{if hole } e_t\ e_f] \rangle$

A
(+ 1
B   C   D   F
(if (zero? 1) (+ 2 3) (+ 4 5)))

inject $A = \langle A, \text{hole} \rangle$

cc0 $\langle A, \text{hole} \rangle$

$\mapsto \langle +,\ \text{hole}[(\text{hole}\ 1\ B)] \rangle$

$= \langle +,\ (\text{hole}\ 1\ B) \rangle$

$\mapsto \langle 1,\ (+\ \text{hole}\ B) \rangle$

$\mapsto \langle B,\ (+\ 1\ \text{hole}) \rangle = E[(\text{if hole } e_t\ e_f)]$

$\mapsto \langle C,\ (+\ 1\ (\text{if hole } D\ F)) \rangle$

$\mapsto\mapsto \langle \text{False},\ = \rangle$

$\mapsto \langle \cancel{B},\ (+\ 1\ \text{hole}) \rangle$

$\mapsto\mapsto\mapsto \langle 9,\ - \rangle$

$\mapsto \langle 10,\ \text{hole} \rangle$

$\rightarrow$ extract $\rightarrow 10$

8-3/ Lambda-calculus

$e = x \mid e \; e \mid \lambda x . e$

J_3 doesnt recursion (except via Z)

$J_3 \Rightarrow J_4$

$v = \ldots \mid$ ~~$\lambda (x \ldots) \; e$~~

$\mid \lambda \; \cancel{x} \; (x \ldots) \; e$

a.map(lambda x: x+1)

lambda fib (n): ....)

rec args

unbound

| let fib = $\lambda n.$ .... (fib (sub1 n))

| let x = xe in be

:= $(\lambda x.$ be) xe

let fib = λ inner_fib : n. ....

inner_fib (sub1 n)

"(define (f x...) xe) ; b"

$\Rightarrow$ "let f = λ f (x...) . xe in b"

8-4

$$E[(\ell \; v_0 \; \cdots \; v_n)] = E[\,b[f \leftarrow \ell][x_0 \leftarrow v_0] \cdots [x_n \leftarrow v_n]\,]$$

where $\ell = (\lambda \; f \; (x_0 \; \cdots \; x_n) \; b)$

$$\langle \lambda f \; (x \ldots) \; b \;, \; env \;, \; k \rangle$$

$$\mapsto \langle \, c \,, \; \emptyset \,, \; k \rangle$$

where $c = clo(\lambda f \; (x \ldots) \; b, \; env')$

$env' = env[f \leftarrow c]$

```
switch ( tag ( c ) ) {
  case LAMBDA:
    envp = make_env(env, c->fun, NULL);
    c = make_clo( c , envp )
    envp->val = c;
    env = NULL;
    break;
```

8-5

```
while(1) {
  int x;; y;
  scanf ("%d", &x);
  scanf ("%d", &y);
  *x = *y; }
```

deref (malloc (4), 5) = $\perp$

Algebraic data types

dt ::=	0	—	___
|	1	—	void
|	dt + dt	—	interface/variants
|	dt × dt	—	pair

type	construct	destruct
1	void	—
0	—	—
dt × dt	pair	fst, snd
dt + dt	left ___, right ___	case/switch/if ___

case (left a) X Y => X a

case (right a) X Y => Y a

8-b)

$$\mathrm{Bool} = 1 + 1$$

$$\mathrm{Nat} = 1 + \mathrm{Nat}$$

$$\mathrm{Bin} = 1 + \mathrm{Bin} + \mathrm{Bin}$$

$$\mathrm{List}(A) = 1 + (A \times \mathrm{List}(A))$$

$$\mathrm{BinT}(A) = 1 + (A \times \mathrm{BinT}(A) \times \mathrm{BinT}(A))$$

$$\mathrm{BinT}'(A) = A + (\mathrm{BinT}'(A), \mathrm{BinT}'(A))$$

$$SE = 1 + \mathrm{Atom} + (SE, SE)$$

$$d_A\, 0 = 0$$

$$d_A\, 1 = 0$$

$$d_A\, A = 1$$

$$d_A\, B = 0$$

$$d_A\, X + Y = d_A X + d_A Y$$

$$d_A\, X * Y = d_A X * Y + X * d_A Y$$

$$d_A\, \mathrm{List}(A) = \mathrm{Zipper}(A)$$

9-1/ (+ 1 2)

JApp(+, 1, 2).asc()

prim v v

= "make_japp(make_jprim(PLUS),

...)"

write to file("x.c", o.asc())

$J_4 \to J_5$

e := x | v | (e e...) | (if e e e)

case e as (inl x) ⇒ e or (inr x) ⇒ e

v := num | bool | prim | λ x (x...) e

unit | pair v v | inl v | inr v

prim := .... | pair | inl | inr

| fst | snd

E[fst (pair v₁ v₂)] = E[v₁] E[snd (pair v₁ v₂)] = E[v₂]

E[case (inl v) as (inl x₁) ⇒ e₁ or (inr xr) ⇒ er]

⇒ E[e₁[x₁ ← v]]

E[(inr v)] ⇒ E[er[xr ← v]]

9-3/// List is either empty
or a cons with a thing
and another list

```
empty := inl unit
cons := λ (data rest). inr (pair data rest)

length := λ rec (l).
    case l of
        (inl _) => 0
        (inr p) => 1 + rec (snd p)

map := λ rec (f l).
    case l of
        inl _ => l
        inr p => cons (f (fst p))
                      (rec f (snd p))

reduce := λ rec (f ~~z~~ l).
    case l of inl _ => z
              inr p => rec f (f z (fst p))
                             (snd p)
```

9-3/ Reduce (+) 0 (cons 1 (cons 2
(cons 3 empty)))

= reduce (+) 1 (cons 2 (cons 3 mt))
= reduce (+) 3 (cons 3 mt)
= reduce (+) 6 mt
= 6

true := inl unit
false := inr unit
if e_c e_t e_f := case e_c of inl _ ⇒ et
inr _ ⇒ ef

inl _ inr inl _ inr inr _

pair → tuple fst/snd → π / .proj
fst = π_0
snd = π_1

case$^{(2)}$ → case (&) inl/inr → choice I

```
obj_t* delta_pair (obj_t* l, obj_t* r){
   ret make_pair (l, r); }
obj_t* delta_fst (obj_t* o) {
   ret ((pair_t*) o) -> fst; }
```

9-4/ $e ::= obj\ \{ x : e,$

$\dots \}$

$\mid\ e . x$

$v ::= obj\ \{ x : v\ \dots \}$

$E[\,obj\ \{ x_0 : v_0 \dots x_i : v_i \dots x_n : v_n \}$

$.\ x_i] \Rightarrow E[v_i]$

$\{\}$

empty % = empty

$\{o, x : e\}$

added

set o x e = (cons $\underbrace{\text{(pair "x" e)}}$ o)

e.x = lookup "x" e

lookup := λrec (field obj).

case obj of inl _ $\Rightarrow$ (rec field obj)

(fst

inr p $\Rightarrow$ if string=? field (fst p))

(fst

(snd p)

(rec field (snd p)