

H/

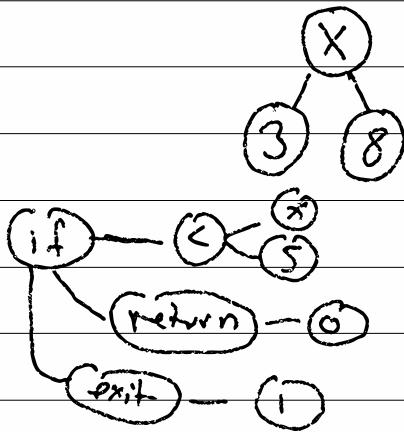
$1 + 1$

$5$

$1 +$

$1 \times 3$

" $3 \times 8$ "



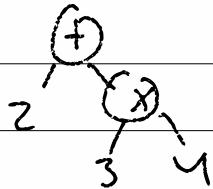
```

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

$(+ 1 1)$   
↳ ↳ ↳  
children



$(+ 2 (\times 3 4))$



$\vdash \text{J}_0 \Rightarrow e := v \quad | \quad (+\ e\ e)$   
 $v := \text{number} \quad | \quad (*\ e\ e)$

$(+ 1 (* 2 3)) \in \text{J}_0$

interface  $\text{Joe} \in \Sigma$

class  $\text{JNumber}$  implements  $\text{Joe} \in \Sigma$

int  $n$ ;  $\text{JNumber}(n)$  {  $n = -n$ ; }

class  $\text{JPlus}$  imp  $\text{Joe} \in \Sigma$

$\text{Joe} \text{ left, right; } \text{JPlus}(\dots) \in \Sigma$

class  $\text{JMlt}$  imp  $\text{Joe} \in \Sigma$

$\text{Joe} \text{ l, r; } \text{JMlt}(\dots) \in \Sigma$ ?

$(+ 1 (* 2 3)) \Rightarrow \text{Expr}$

$\text{new JPlus($

$\text{new JNum}(1),$

$\text{new JMlt(} \quad = \text{JP}(\text{JN}(1), \text{JM}(\text{JN}(2), \text{JN}(3)))$

$\text{new JNum}(2))$

$\text{new JNum}(3)))$

class  $\text{JPlus}:$

def \_\_init\_\_(l, r):

$\text{this.l} = l;$

$\text{this.r} = r;$

BST  $n = \text{mt} \quad | \quad (\text{br num}$

$n \ n)$

1-3/6 pp : J<sub>0</sub>  $\Rightarrow$  string

③ pp n = itos(n)

④ pp ( + e<sub>L</sub> e<sub>R</sub>) = "(#pp(e<sub>L</sub>) ++ "+" ++ pp(e<sub>R</sub>)  
++ ")"

⑤ pp ( \* e<sub>L</sub> e<sub>R</sub>) = "(" ++ pp(e<sub>L</sub>) ++ ">\*" ++  
pp(e<sub>R</sub>) ++ ")"

① interface J<sub>0</sub> { 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(); }

# I-9) big-step interpreter

interp : e → v

interp n = n

interp (t e<sub>L</sub> e<sub>R</sub>) = interp e<sub>L</sub> + interp e<sub>R</sub>

interp (\* e<sub>L</sub> e<sub>R</sub>) = interp e<sub>L</sub> \* interp e<sub>R</sub>

→ class JMult {

public <sup>int</sup> interp () {

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

$$(+ \ 1 \ 2 \ 3) = (+ \ 1 \ (+ \ 2 \ 3))$$

desugar →

se = empty | (cons <sup>min</sup> se se) | string

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

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

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

mt)))

I-5) desugar for  $\mathcal{J}_0$

$$(" - " \ e) \Rightarrow (* \ -1 \ (\text{desugar } e))$$

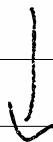
$$(" - " \ e_1 \ e_2) \Rightarrow (+ \ (\text{d } e_1) \ (\text{de } (" - " \ e_2)))$$

$$(" + ") \Rightarrow 0$$

$$(" + " \ e_1 \ \text{more} \dots) \Rightarrow$$

$$(+ \ (\text{d } e_1) \ (\text{dd } (" + " \ \text{more} \dots)))$$

"\*



= 1



"x"

"x"

$$\text{se} \Rightarrow \mathcal{J}_0 \Rightarrow v$$

desugar interp

compiie bc  $\xrightarrow{\text{vm}} v$

$\Sigma \cup \{ \}$  b  
 desugarer  $(- e_1) \Rightarrow (\widehat{*} -1 e'_1)$   
 $(- e_1 e_2) \Rightarrow (\widehat{*} e'_1 (-e_2))$   
 $(+) \Rightarrow 0$   
 $(+ e_1 e_2 \dots) \Rightarrow (\widehat{*} e'_1 (+ e_2 \dots))$   
 def  
 desugar(se) :  
 if isList(se) & length(se) = 2 &&  
     first(se) == "-" then  
 > def length(se) :  
 > if isNull(se) : return 0  
 > else if Cons(se) : return 1 + length(right)  
 > else false  
     new JMut( new JNum(-1), desugar(  
                 second(se)))

first  $\leftarrow$  left  $\rightarrow$  right

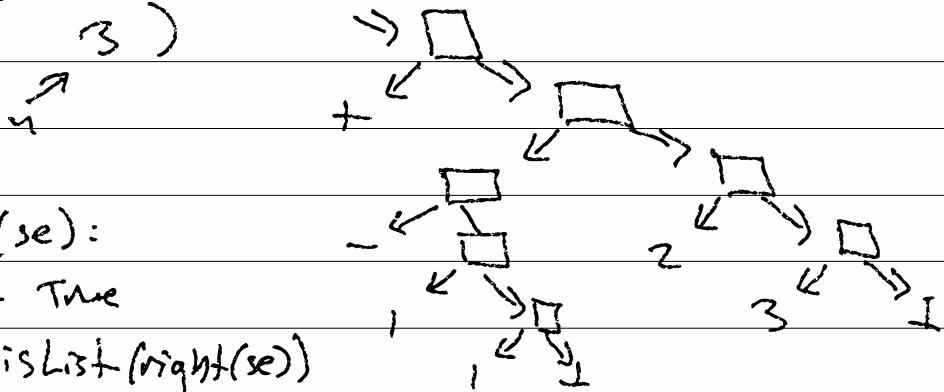
J second/left  $\rightarrow$  right

first

if isList(se) & len(se) = 3 && first(se) == "-"  
 then : return new JAdd( desugar(second(se)),  
                     desugar( new Cons("-", ~~new Cons(first(se), null)~~, null)))

22]  $\text{Cons}(a, \text{Cons}(b, \text{Cons}(c, \text{null})))$   
 $\text{len} = 3$

$(+ (- 1 1) \text{len} = 4$   
 $1 \xrightarrow{1} 2 \xrightarrow{2} 2 \leftarrow 3$   
 $3)$



$\text{isList}(\text{se})$ :

$\text{Null} = \text{True}$

$(\text{cons} : \text{isList}(\text{right}(\text{se})))$

$\text{ow} = \text{False}$

$\text{Len}(\text{se})$ :

$\text{Null}: 0$

$(\text{cons} : 1 + \text{len}(\text{right}(\text{se})))$

2-3 /  $J_0 \Rightarrow J_1$

$\downarrow$  fun     $\downarrow$  args

$$e := v \quad | \quad (e \ e \ \dots)$$
$$| \quad (\text{if } e \ e \ e)$$
$$\begin{matrix} \nearrow & \nearrow & \nearrow \\ c & f & f \end{matrix}$$

$v := b$

$b$  = some set of constants

/l in  $J_0$ ,  $b = \text{num} \mid + \mid *$   
numbers    |    bools    |    prim

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

interp  $v = v$

interp (if  $e_c$   $e_t$   $e_f$ ) = interp  $e_k$

where  $e_k = \text{if interp } e_c \text{ then}$   
 $e_t \text{ o.w. } e_f$

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

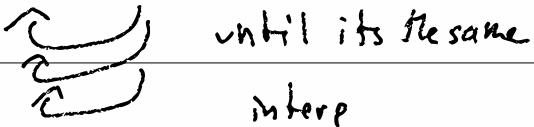
where  $p = \text{interp } e_f$

$v_a = \text{interp } e_a \dots$

$\delta : b \dots \rightarrow b$

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

$\delta(\leq, 1, 3) = \text{true}$

$\Sigma^4 /$  "small step interp"      "big step"  
 $e \rightarrow e$        $e \rightarrow v$   
  
 Interp      Interp

Interp  $e =$

let  $e' = \text{interp}(e)$

if  $e == e'$  then

ret  $e$

O.V.

Interp ( $e'$ )

$(+ (+ 1 1) z) \leftarrow (+ (+ 1 1))$   
 $\Rightarrow (+ z z) \leftarrow (+ 1 1))$   
 $\Rightarrow 4 \downarrow (+ z (+ 1 1))$

int  $x = 1;$

$f(x--, x++)$        $(1, 0)$   
 $(2, 1)$

(2-3) step :  $e \rightarrow e$

step (if true  $e_1$   $e_2$ ) =  $e_1$

step (if false  $e_1$   $e_2$ ) =  $e_2$

step ( $P \vee a \dots$ ) =  $\delta(P, va \dots)$

step  $v = v$

step (if  $e(\&v)$   $e_1 e_2$ ) =

(if (step  $e$ )  $e_1 e_2$ ) or (if  $e$  (step  $e$ )  $e_2$ )

step ( $v_b \dots e(\&v)$   $e_a \dots$ ) =

( $v_b \dots$  (step  $e$ )  $e_a \dots$ )

A context

$C := \text{hole} \quad | \quad \text{if0 } C \ e \ e$

$| \quad \text{if1 } e \ C \ e$

$| \quad \text{if2 } e \ e \ C$

$| \quad (e \dots C \ e \dots)$

plug  $C \ e \ (C[e])$

plug hole  $x = x$

plug (if0  $C \ e_1 \ e_2$ )  $x = \text{if } x \ e_1 \ e_2$

plug (if1  $e_1 \ C \ e_2$ )  $x = \text{if } e_1 \ x \ e_2$

plug ( $e_1 \dots C \ e_2 \dots$ )  $x = (e_1 \dots x \ e_2 \dots)$

2-6

$$\text{step } C[\text{if true } e_1 \text{ et } e_2] = \\ C[e_1]$$

$$\text{step } C[\text{if false } e_1 \text{ et } e_2] = \\ C[e_2]$$

$$\text{step } C[p \text{ va } \dots] = C[S(p, \text{va } \dots)]$$

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

step  $\xrightarrow{\quad\quad\quad} e$

$$\text{interp } e = \text{if } e \in V \text{ then } e$$

$$C, e' = \text{parse } e \quad e$$

$$e'' = \text{step } e'$$

$$\text{plug } C \ e''$$

$$\text{parse} : e \Rightarrow C \times e \quad \leftarrow \text{redex}$$

$$\text{parse "(if } e_1 \text{ et } e_2) =$$

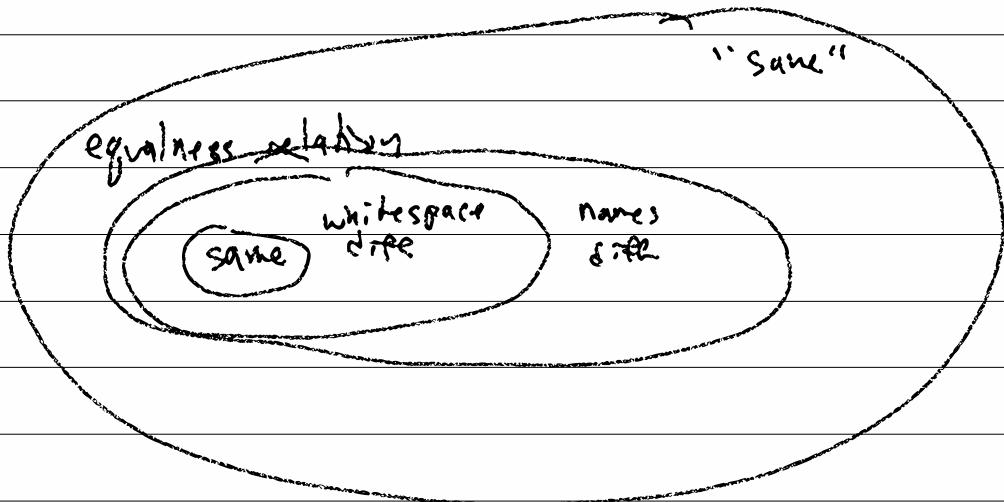
$$\text{if } e_1 \in V \text{ then } (\text{hole}, e)$$

$$\text{o.w. let } C', e' = \text{parse } e_1$$

$$(\text{ifO } C' \text{ et } e_2, e')$$

## 2-7) Answer: Contexts

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



$$x = y$$

$$\forall x, f_x = g_x$$

$$\forall c, C[x] = C[y]$$

$$C[\text{hole } x] = y$$

$$C[+ \text{hole } z] = x + z = y + z$$

$$C[\text{map hole } (l; r + z)]$$

....

Observational Equivalence

$\rightarrow C := \text{hole} \mid \text{if } C \text{ e e}$   
 $\quad \mid \text{if } e C e$   
 $\quad \mid \text{if } e e C$   
 $\mid (e \dots C \dots e \dots)$

$E := \text{hole} \mid \text{if } E \text{ e e}$   
 $\mid (\vee \dots E \dots e \dots)$

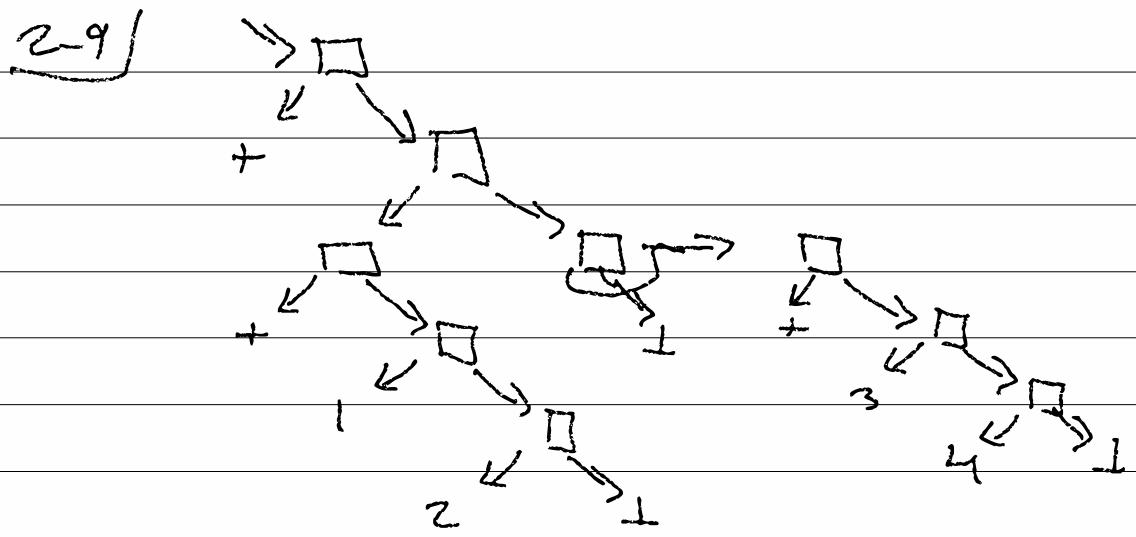
"mique decomp:  $\vdash_{\text{mique}}$ "       $\downarrow^{\text{mique } E}$   
 & e.  $e \in \vee$  or  $e = E[e']$  where  
 $e' \in \vee$

$$\delta(1, 2) = 1 \quad \delta(1, 1, 0) = 1$$

$(+ (+ 1 2) (+ 3 4))$        $\stackrel{\text{tree}}{\sim} \text{java tree}$

new JPlus (new JPlus (new JItem (1),  
 new JItem (2)))

new JPlus (new JItem (3),  
 new JItem (4)))



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

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

step  $E[\text{if true } e + ef] \rightarrow f[e]$

step  $E[\text{if false } e + ef] \rightarrow f[ef]$

step  $E[p \; va \dots] \rightarrow E[\delta(p, va \dots)]$

interp  $e = \text{case (parse } e) \text{ of}$   
false  $\rightarrow e$

$(E, e) \rightarrow e$ : step e

$E[e']$

gigantic program  $\left[ (+ \quad (+ 1 \quad 1) \atop (+ 2 \quad 3)) \right]$

3-2/  $\Sigma$  "language"

$(C_0$  "machine"

$e \rightarrow e$

$st \mapsto st$

lang e  $\xrightarrow{\text{inject}}$  machine st

$\downarrow$  step

$\downarrow$  step (3)

done?  
fv

$\leftarrow$  extract

$st'$

$st = \langle e, E \rangle$

done?  $\langle v, \text{hole} \rangle$

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

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

$\langle \text{if } ec \text{ et } ef, E \rangle \mapsto \langle ec, E[\text{if hole et } ef] \rangle$

$\langle \text{true}, E[\text{if hole et } ef] \rangle \mapsto \langle \text{et}, E \rangle.$

$\langle \text{false}, E[\text{if hole et } ef] \rangle \mapsto \langle ef, E \rangle$

$\langle e_0 e_1 \dots, E \rangle \mapsto \langle e_0, E[\text{hole } e_1 \dots] \rangle$

$\langle v, E[v_0 \dots \text{hole } e_1 e_2 \dots] \rangle \mapsto \langle e_1, E[v_0 \dots v_1 \text{hole } e_2 \dots] \rangle$

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

33)  $E = \text{hole} \mid \text{if } E \ e \ e \mid (\& \dots E \ e \dots)$

interface context  $\Sigma$

Expr plug (Expr); }

Hole : Context +  $\Sigma$

plug (e) = e; }

If C : Context +  $\Sigma$

Context c; Expr t, f;

plug (e) =  $\text{new If}(C \cdot \text{plug}(e), t, f);$  }

AppC : Context  $\Sigma$

List <V> vs; Context q; List <Expr> es;

plug (e) = new App( vs ++ [C · plug (e)] ++ es ); }

< (+ ( + ( + 0 1 ) 2 ) 3 ) , hole >

< (+ ( + 0 1 ) 2 ) , AppC  
[+] hole [3] >

< (+ 0 1 ) , AppC  
[+] hole [3] >

{ 1 }  
AppC  
[+] AppC  
[+] hole [3] >  
+ >  $\Rightarrow$  < R, AppC  
[+] AppC  
[+] hole [3] >  
+ >  $\Rightarrow$  < R, AppC  
[+] AppC  
[+] hole [3] >

$\underline{3-4}/$   $E = \text{hole} \quad | \text{if } E \in \{ v \dots E \dots \}$   
 $= \text{top} \quad | \text{if } ee \quad \square \quad | \quad (v \dots)(e \sim) \square$   
 $K = \text{kre} : \quad | \text{kif } ee \ K \quad | \ K_{\text{app}} \xrightarrow{\vec{v}} \vec{e} \vec{k}$

CK<sub>0</sub> machine       $st = \langle e, k \rangle$

inject  $e = \langle e, k_{\text{ret}} \rangle$

extract  $\langle e, k_{\text{ret}} \rangle = e$

$\langle e, \text{kif } ee \text{ et } ef \ K \rangle = \text{extract}$

$\langle \text{if } e \text{ et } ef, k \rangle$

$\langle e, k_{\text{app}} (v \dots)(e \dots) k \rangle =$

$\text{extract } \langle (v \dots e e_i \dots), k \rangle$

done  $\langle v, k_{\text{ret}} \rangle$

0  $\langle \text{if } ee \text{ et } ef, k \rangle \mapsto \langle ee, \text{kif } ee \text{ et } ef \ K \rangle$

1  $\langle \text{true}, \text{kif } ee \text{ et } ef \ K \rangle \mapsto \langle ee, k \rangle$

2  $\langle \text{false}, \text{kif } ee \text{ et } ef \ K \rangle \mapsto \langle ef, k \rangle$

3  $\langle e_0 e_1 \dots, k \rangle \mapsto \langle e_0, k_{\text{app}} () (e_1 \dots) k \rangle$

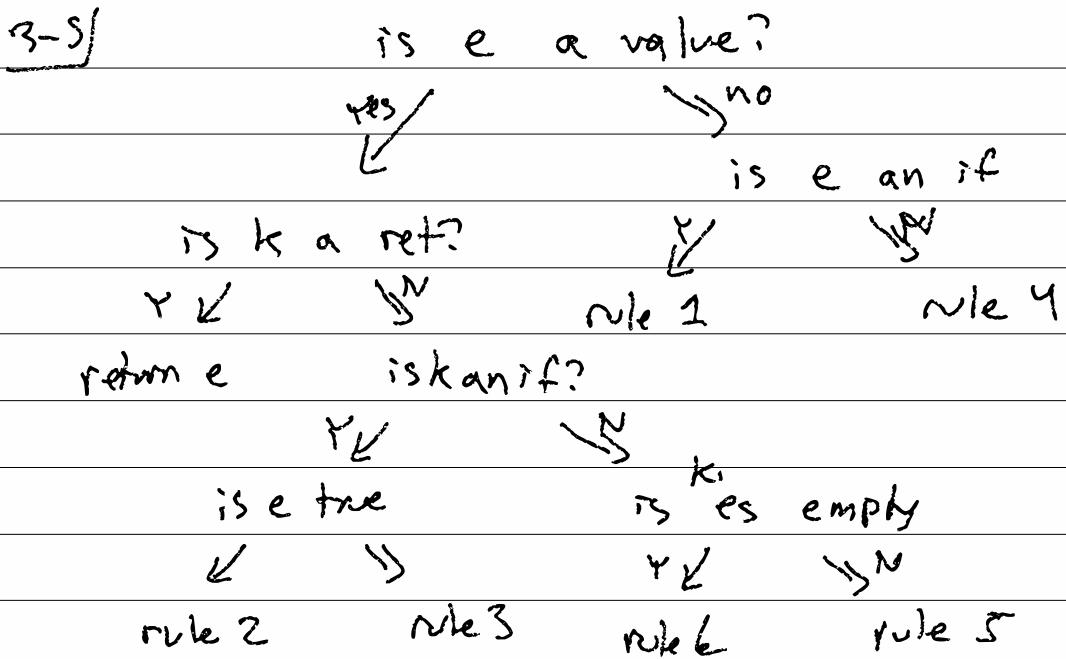
4  $\langle v_1, k_{\text{app}} (v_0 \dots) (e_0 e_1 \dots) k \rangle$

$\mapsto \langle e_0, k_{\text{app}} (v_0 \dots v_1) (e_1 \dots) k \rangle$

5  $\langle v_n, k_{\text{app}} (v_0 \dots) () k \rangle$

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

while(1) {



rule 1:

$K = \text{new\_if } (e, \text{true}, e, \text{false}, k)$

$e = e.\text{cond};$

~~jump  $R + PC$~~

$K$  is a stack and the stack (of c)

= Kontinuation

continuation

3-6/ struct if {  
 expr \* c, t, f; }  
 struct num {  
 int n; }  
 struct app {  
 expr \* f, \* args; }  
 expr \* make\_if(expr \* t, f) {  
 if \* p = malloc(sizeof ...))  
 p->h.tag = IF;  
 p->c = c; ...  
 return p; }

struct expr {  
 enum tag; }  
 enum tag {  
 IF, NUM, APP,  
 BOOL, PRIM,  
 RET, KIF,

KAPP, CONS, NIL};

(+ (+ 1 1 7 2)

make-add (make-add (make-num(1),  
 make-num(1)),  
 make-num(2)));

02011221\$07

4-1 /  $\Sigma_0$      $e = n \mid (+ \ e \ e)$   
                           |  $(\neq \ e \ \epsilon)$

$\Sigma_1$      $p = \text{unary} - (\text{neg})$ , not ()  
                   + , \* , / between

$e = n \mid p \mid^{(2)} (\text{if } e \in e)$   
 $\boxed{T(e \dots)}$

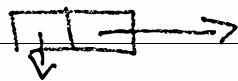
class TApp impl TExpr {  
     List<TExpr> contents }

App

$$(+ \ 1 \ 2) \Rightarrow \text{List}(+, 1, 2)$$

$\downarrow \quad \downarrow \quad \downarrow$   
     p      n      n

*desugar*  
 $\rightarrow \text{App}([\text{Prim(PLUS)}, \text{Num}(1), \text{Num}(2)])$   
 $(\text{cons} \ "+"\ (\text{cons} \ "1" (\text{cons} \ "2" \ \text{null})))$



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

$\text{App}([\text{Prim(PLUS)}, \text{Num}(1), \text{Num}(2)])$

$\text{App}([\text{Prim(PLUS)}, \text{Num}(2), \text{Num}(3)])$

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

$$\delta(+ \ 1 \ 3 \ 5) = 1$$

4-2) Expr \* Delta (List < Expr >) args) {  
 if (len(args) == 3  
 && args[0] == prim(PLUS)) {  
 ret new Num((Num(args[1]) + num(args[2]))  
 ); } };

$$(+ \rightarrow 0$$

$$\cancel{(+ \rightarrow A)} \rightarrow A$$

$$(+ n \text{ more } \dots) \Rightarrow (+ n (+ \text{ more } \dots))$$

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

delta (args)  
 args[0] (args)  
 $\rightarrow$   
 args[0].apply (args[1...])

$T_0$  or  $T_1$  :  $\text{prog} = e$

4-3 /   $T_2$  :

$e := v \mid (\overset{e}{\overset{e}{e}} \dots) \mid (\text{if } eee)$   
|  $x$

$v := \text{number} \mid \text{bool} \mid \text{prim} \mid f$

$X \in \text{some set of variable names}$

$f \in \text{some set of function names}$

$\text{prog} := d \dots e$

$d := (\text{define } (f \ x \dots) \ e)$

$(\text{define } (\text{Double } x) \ (+ \ x \ x))$

$\rightarrow (\text{Double } (+ \ (\text{Double } 1) \ 3))$

$(\text{define } (\text{Quad } x) \ (\text{Double } (\text{Double } x)))$

$(\text{Quad } (+ \ 1 \ (\text{Double } 3)))$

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

$$f(x) = 1 - x$$
$$f(3+4) = 1 - (3+4)$$
$$\begin{matrix} " \\ f(7) \end{matrix} = \cancel{3+4} = 2$$
$$= 1 - 7 = -6$$

$$\text{Double}(1+1) = \text{Double}(2)$$

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

4-3)

$$E = \text{hole} \quad | \quad \text{if } E \ e \ e \\ | \quad (\vee \dots E \ e \dots)$$

~~$E[x] = \dots$~~   $\Sigma / E[\text{if } f \ t \ e \ e] = E[e]$   
 $\Sigma / E[f \ \vee \ \dots] = \dots$

$\text{eval} : e \Rightarrow \Sigma \dots \text{smallstep } e \rightarrow e$   
 $\text{eval}' : p \Rightarrow \Sigma$   
 $\Sigma^x \vdash \dots \text{smallstep } \Sigma \underset{?}{\overset{!}{\text{c}}} e \Rightarrow e$

$\Sigma : f \rightarrow d$

$\text{eval}' \Sigma \text{ do}(\text{define } (f x \dots) e) : \text{more}$

$= \text{eval}' \Sigma [f \mapsto d] \text{ more}$

$\text{eval}' \Sigma e = \text{do smallstep}$

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

$\Sigma / E[f \ \vee \ \dots] = E[e[x \leftarrow v] \dots]$

$\text{if } \Sigma(f) = (\text{define } (f x \dots) e)$

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

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

subst  $x \ v \ v' = v'$

subst  $x \ v \ x = v$

subst  $x \ v \ x' = x'$

subst  $x \ v$  (if  $e_c \ e + e_c$ ) =

(if  $e_c [x \leftarrow v] \ e + [x \leftarrow v] \ e_f [x \leftarrow v]$ )

subst  $x \ v$  (e ...) =

( $e [x \leftarrow v]$  ...)

interface JExpr {

JExpr subst (Variable x, JExpr v); }

class JVar {

subst(x, v) { if ( $x == \text{this}, x$ )  
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)

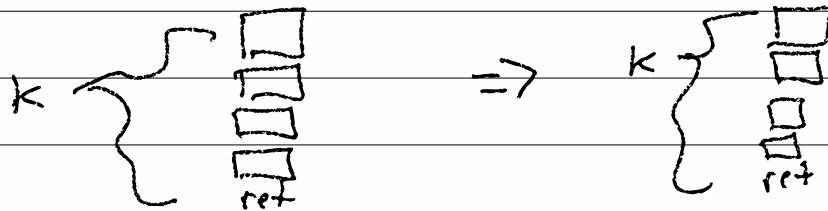
CK

$$6: \langle v_n, k_{app}((v_0 \dots), (), k) \rangle \\ \mapsto \langle \delta_{(v_0 \dots v_n)}, k \rangle$$

$$7: \langle v_n, k_{app}((f v_0 \dots), (), k) \rangle \\ \mapsto \langle e[x_i \leftarrow v_i], k \rangle$$

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

$$c = 2 \quad k_{app}(\text{Expt 7}) \quad c = \dots 7 \dots 2 \dots$$



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

$$\mapsto \langle e_0, k_{app}(( ), (e_1 \dots), k) \rangle$$

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

$\Rightarrow \Sigma = [F \mapsto (\text{define } (F x) (F x))]$

$e = (F 10)$

$k = kret$

$\langle F 10, kret \rangle$

$\langle F, kapp ((), 10), kret \rangle$

$\langle 10, kapp ((F), (), kret) \rangle$

$\Sigma(F) = (\text{define } (F x) (F x))$

$f \leftarrow x \dots \hat{e}$

$\langle e[x \dots \leftarrow v \dots], kret \rangle$

$(F x)[x \leftarrow 10]$

$\langle (F 10), kret \rangle$

error  $\rightarrow$  stack trace

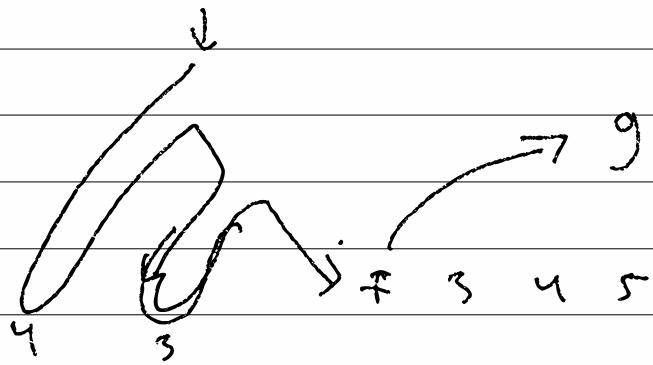
$\rightarrow$  fun "at fault"  $\sim F$

fun that called ~~F~~  $\sim f$

"past"

G - H

$y=f$ )



$g(\cdot \cdot \cdot)$

$x = f \quad 3 \ 4 \ 5$

$n(x)$

$$\begin{array}{l}
 \underline{6-1} \quad C = \text{hole} \quad | \quad \text{if } C \in e \\
 \qquad\qquad\qquad | \quad \text{if } e \in C \\
 \qquad\qquad\qquad | \quad \text{if } e \in C \\
 \qquad\qquad\qquad | \quad e \dots E e \dots \\
 E = \text{hole} \quad | \quad \text{if } E \in e \\
 \qquad\qquad\qquad | \quad v \dots E e \dots
 \end{array}$$

if true (+ 1 2) 4

$$\begin{array}{ll}
 E = \text{hole} & e = \uparrow \\
 C = \text{hole} & e = \uparrow \\
 & C = \text{if true hole 4} \\
 & e = (+ 1 2)
 \end{array}$$

(+ 1 2)

$E = \text{hole}$     $e = (+ 1 2)$

find-reduce  $\underbrace{B}_{, B} = E[e]$

step  $e = e'$

6-2]  $\text{fr} (\text{if } e + f) =$

$\text{if } (\text{value? } c)$

$(\text{hole}, e)$

O.W.  $(E, e) = \text{fr } c$

$(\text{if } (E + f), e)$

$\text{fr} (\text{app } es) =$

for  $e$  in  $es$

$\text{if } (\text{value? } e)$

---

$(+ \mid \times)$

$\langle e_0 \ e \dots , k \rangle$

$\mapsto \langle e_0, \text{kapp} ((), (e \dots), k) \rangle$

$\langle v_n, \text{kapp} ((), (v_0 \dots), (e_{n+1} \dots), k) \rangle$

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

$\langle v_n, \text{kapp} ((p \ v_0 \dots), (), k) \rangle$

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

6-3 /  $\mathcal{I}_2 = \text{PASCAL or C}$

top-level functions

$Ck = \text{have the map } f \rightarrow d$   
and we have subst

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

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

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

$$x[x \leftarrow v] = v$$

$$y[x \leftarrow v] = y$$

$$u[x \leftarrow v] = u$$

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

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

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

$\mapsto \langle c, k \text{ if } (+, f, k) \rangle$

OLD: no rule for a variable in the case  
 $\langle x, k \rangle \mapsto \dots$

NEW:

$\langle x, k \rangle \mapsto \text{finally do the subst}$

(Σ)

G-γ) C E K    st = < e, env, k >

env = Ø | env [x ← v]

k = k<sub>net</sub> | k<sub>if</sub> ∈ e k

| k<sub>app</sub> → e → k

< x, env, k > ↪ < env(x), Ø, k >

< if c t f, env, k >

↪ < c, env, k<sub>if</sub> t f k >

< true, env, k<sub>if</sub> t f k >

↪ < t, env, k >

< e<sub>0</sub> e<sub>1</sub> ... , env, k >

↪ < e<sub>0</sub>, env, k<sub>app</sub> () (e<sub>1</sub>...) k >

< v<sub>1</sub>, , ~~v<sub>0</sub>~~, k<sub>app</sub> (v<sub>0</sub>...) (e<sub>0</sub> e<sub>1</sub>...) k >

↪ < e<sub>0</sub>, env, k<sub>app</sub> (v<sub>0</sub>...v<sub>1</sub>) (e<sub>1</sub>...) k >

< v<sub>n</sub>, env, k<sub>app</sub> (~~v<sub>0</sub>...~~) () k >

↪ < e, ~~v<sub>0</sub>~~ [x<sub>0</sub> ← v<sub>0</sub>] ... [x<sub>n</sub> ← v<sub>n</sub>], k >

where Σ(f) = define (f x<sub>0</sub> ... x<sub>n</sub>) ∈

6-5/ define  $f(x) = x + z$   
define  $g(z) = f(y)$   
 $g(2)$

define  $f(x) = 3$   
define  $g(z) = (f(1)) + x$   
 $g(2)$

6-6]

$\text{CEK} = \langle e, \text{env}, k \rangle$

$\text{env} = \emptyset \mid \text{env}[x \leftarrow v]$

$k = \text{tret} \mid \text{kif env} + f \ k$

$| \ kapp \xrightarrow{\vec{v}} \text{env} \xrightarrow{\vec{e}} k$

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

$\langle \text{if } c + f, \text{env}, k \rangle \mapsto \langle c, \text{env}, \text{kif env} + f k \rangle$

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

$\langle e_0 e_1 \dots, \text{env}, k \rangle$

$\mapsto \langle e_0, \text{env}, \text{kapp}((\_), \text{env}, (e_1 \dots), k) \rangle$

$\langle v_n, \_, \text{kapp}((v_0 \dots), \text{env}, (e_0 e_1 \dots), k) \rangle$

$\mapsto \langle e_0, \text{env}, \text{kapp}((v_0 \dots v_n), \text{env}, (e_1 \dots) k) \rangle$

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

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

where  $\Sigma(f)$  define  $(f/x_0 \dots x_n) e$

"dynamic scope" ~ equal

- emacs lisp

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

specific vars are always dynamic

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

"this"

Q7) PASCAL/C - all funcs are top-level  
 $p = \lambda \dots e$

$\text{JS} = (\lambda) \Rightarrow 1 + x$

$\text{Py} = \text{lambda: } x : 1 + x$

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

$\text{J}_3$

$e = v \mid e \ e \dots \mid \text{if } e \ e \ e \mid x$

$v = b \mid \boxed{(\lambda (x \dots) \ e)} \text{--- new}$

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

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

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

$((\lambda x. \ ((\lambda y. \ (x + y) \ 7)) \ 8) \Rightarrow 15$   
let  $x = 8$  in  
let  $y = 7$  in  
 $x + y$

let  $x = e_1$  in  $e_2 \Rightarrow$   
 $(\lambda x. \ e_2) \ e_1$

let  $\overbrace{x}^{\leftarrow 8} = 8$  in  
let  $\overbrace{x}^{\leftarrow x + 1} = x + 1$  in  
 $x + x$

6-8/

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

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

$$e[y \leftarrow v])$$

unless  $y \notin x_0 \dots x_n$

$$(\lambda x_1 (\lambda x_1 x+1)) z$$

$$(\lambda x_1 x+1)$$

old  
machine  $v = \text{theory } v$

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

closure  $(\lambda(x \dots) e, \text{env})$

$$< \lambda(x \dots) e, \text{env}, k > \mapsto < \text{clo}(\lambda(x \dots) e, \text{env}), \emptyset, k >$$

$$< v_0, \dots, k \text{app}((\text{clo}(\lambda(x_0 \dots x_n) e, \text{env}) v_0 \dots), \dots, (), k) >$$

$$\mapsto < e, \text{env}[x_0 \leftarrow v_0] \dots [x_n \leftarrow v_n], k >$$

7-1  $\vdash J_3 : v = \dots \mid \lambda(x\dots) e$

let  $x = e_1$  in  $e_2$

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

let  $x = 1$  in

$\lambda(x)$

let  $y = 2$  in

$\lambda(y)$

$x+y$

$(+ x y) 2 \mid 1$

$\text{CEK}_1 : v = \dots \quad (\lambda(y) (+ 1 y)) 2$

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

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

$\langle \lambda(x\dots) e, \text{env}, k \rangle$

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

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

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

$\mapsto \langle e, \text{env}[x_0 \mapsto v_0] \dots [x_n \mapsto v_n], k \rangle$

$\text{env} = \perp \quad | \quad \boxed{\quad \quad \quad \quad \quad} \quad | \quad \begin{array}{ccccccc} \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ x & v & & & \text{env} \end{array}$

$\text{clo} = \boxed{\quad \quad \quad} \quad | \quad \lambda x.e \quad \text{env}$

let  $y = 3$  in  
let  $z = 8$  in [8, 3, 19, 22, 36]

7-2)  $\lambda(x)(+x+y) \xleftarrow{\text{clo}} z \rightarrow 8$

$\downarrow$   $y \rightarrow 3$   
 $(\lambda. (+ \underset{\text{static}}{\hat{0}} \underset{\text{address}}{\hat{1}}), [\hat{1}]) \quad x \rightarrow 19$   
 $a \rightarrow 22$   
 $b \rightarrow 36$

...

SA = nat env = vector v

FLAT-CLOSURES  $[\hookrightarrow, 3]$

SA = (nat, nat) env =  $\downarrow$ , vector v

$(\overset{\wedge}{0}, 0) (\overset{\wedge}{1}, 1) \quad [8, 3, 19, 22, 36] \text{ env}$   
 $\cap, [\hookrightarrow, \hookrightarrow]$  NESTED CLOSURES

$$\begin{aligned}
 7-3) &= v \mid (\text{if } e \leq e) \\
 &\quad x \mid (e \leftarrow e) \mid (p \leftarrow e) \\
 v &= b \mid \lambda(x)e \\
 b &= \text{num} \mid \text{bools} \mid \text{prim} \\
 \text{prim} &= + \mid - \mid * \mid \div \mid \lt
 \end{aligned}$$

7-y/ what is a Bool really, man?

$$\text{if True } A \ B = A$$

$$\text{if False } A \ B = B$$

$$\text{True} = \lambda x. \lambda y. x$$

$$\text{False} = \lambda x. \lambda y. y$$

$$\text{if} = \lambda c. \lambda x. \lambda y. c \times y = \lambda c. c$$

$$\underbrace{\text{if}}_{\text{True}} \ A \ B = \text{True} \ A \ B = A$$

$$\text{NOT T} = F$$

$$\text{NOT F} = T$$

$$\text{NOT} = \lambda b. \lambda x. \lambda y. b \ y \ x$$

interface Bool { int choose (int, int); }

class True : Bool { <sup>True</sup> int choose (x, y) = x }

class False : Bool { <sup>False</sup> int choose (x, y) = y }

class Not : Bool { 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

$$\text{add } \lambda^n x. \lambda^m y. \underline{\text{_____}}^{n+m} = \cancel{\lambda f. \lambda x. f x}$$

zero :=  $\lambda f. \lambda x. x$

one :=  $\lambda f. \lambda x. f x$

two :=  $\lambda f. \lambda x. f(f x)$

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

add :=  $\lambda n. \lambda m. \lambda f. \lambda x. n f(m f x)$

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

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

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

$f f f f x$

## 7-6/ Pair

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

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

$$\text{pair} = \lambda a. \lambda b. \lambda c. \text{if } c \ a \ b$$

$$\text{fst} = \lambda p. \ p \ \text{TRUE}$$

$$\text{snd} = \lambda p. \ p \ \text{FALSE}$$

$$\text{subl} := \lambda n. \ \text{fst} (n (\lambda p. \text{pair} (\text{snd} p) (\text{pair} z \ z))) \\ (\text{addl} (\text{snd} p)))$$

$\lambda$  fac.

$$\text{mkfac} := \lambda n.$$

$$\text{if } (\text{zero? } n)$$

1 = one

$$(\text{addlt } n (\text{fac} (\text{subl } n)))$$

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

$$\text{fac} := \text{mkfac fac} \quad x = F \ x$$

$$\begin{matrix} \uparrow & \uparrow & \uparrow \\ x & F & x \end{matrix}$$

7-7) Fixed point of a lambda?

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

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

$\in \mathbb{Z}$ -combinator

$$\begin{aligned} \text{FIX} := & \lambda F. ((\lambda x. F (\lambda v. x x v)) \\ & (\lambda x. F (\lambda v. x x v))) \end{aligned}$$

$$\begin{aligned} \text{FIX } F &:= ((\lambda x. F (\lambda v. x x v)) \\ & (\lambda x. F (\lambda v. x x v))) \end{aligned} \quad A$$

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

$$= F(A A)$$

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

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

Lambda-Calculus

$$\begin{array}{c} n, \text{add} \\ \downarrow \\ \lambda x. x + 1 \end{array} \quad 0$$

Church Numeral / Church encoding

## 8-1/ Lambda - Calculus

---

tiny (if  $T \in PC$ ) =  $e$

tiny :  $e \rightarrow e$  tiny (if  $F \in PF$ ) =  $eF$

tiny ( $p, v_0 \dots v_n$ ) =  $\delta(p, v_n)$

small :  $e \rightarrow e$

small  $e = \text{let } C, e' = \text{find-addr } e$

big :  $e \rightarrow v$  let  $e'' = \text{tiny } e'$   
let  $C[e'']$

big  $e =$

if  $e \in V$  : return  $e$

o.w. big (small  $e$ )

$CC()$  :  $C \times e \rightarrow C \times e$   
 $CC(C, e) = C' \times e' = \text{move-context } C \ e$   
 $e'' \rightarrow \text{tiny } e'$

ret  $(C', e'')$

$CC^* C \ e = \text{if } e \in V \text{ and } (= \text{hole}, \text{ret } C \ e)$

big! :  $e \rightarrow e$  o.w.  $CC^*(CC^* C \ e)$

extract ( $CC^* (\text{inject } e)$ )

inject =  $(\text{hole}, e)$  extract(hole,  $v$ ) =  $v$

B-3

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

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

A

$(\frac{+}{8} \ 1 \ c \ 0 \ F)$   
 $(\text{if zero? } 1) \quad (+ 2 \ 3) \quad (+ 4 \ 5))$

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

$\ll 0 \ \langle A, \text{hole} \rangle$

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

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

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

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

$\mapsto \langle C, (+ 1 \ (\text{if hole } 0 \ F)) \rangle$

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

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

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

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

$\mapsto \text{extract} \rightarrow 10$

## 8-3) Lambda-calculus

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

$\mathcal{T}_3$  doesn't recursion (except via  $\Xi$ )

$$\mathcal{T}_3 \Rightarrow \mathcal{T}_4$$

$$v = \dots \mid \cancel{\lambda(x\dots)e}$$

$$\mid \lambda * (x\dots)e$$

a. map (lambda  $x$ :  $x+1$ )

lambda fib (n): ...  
 $\xrightarrow{\text{rec}}$   $\xrightarrow{\text{args}}$

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

| let  $x = xe$  in be  
 $\quad := (\lambda x. be) xe$

let fib =  $\lambda$  inner-fib : n . ...

inner-fib (sub1 n)

"(define (f xe) ; b")  
 $\xrightarrow{x\dots}$

$\Leftrightarrow$  "let f =  $\lambda x. f(x\dots) \cdot xe$  in b"

8-4

$$E[(\lambda v_0 \dots v_n)] = E[b[f \leftarrow \ell][x_0 \leftarrow v_0] \dots]$$

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

$\langle \lambda f (x \dots) b, \text{env}, k \rangle$

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

where  $c = \text{clo}(\lambda f (x \dots) b, \text{env}')$

$\text{env}' = \text{env}[f \leftarrow c]$

switch (tag(c)) {

case LAMBDA:

$\text{envp} = \text{make\_env}(\text{env}, c \rightarrow \text{fun}, \text{NULL});$

$c = \text{make\_clo}(c, \text{envp})$

$\text{envp} \rightarrow \text{val} = c;$

$\text{env} = \text{NULL};$

break;

while(1) {

8-5/    vint x, y;

scanf ("%d", &x);  
scanf ("%d", &y);  
 $x = \{y\}$



deref (malloc (4), 5) = ⊥

## Algebraic data types

dt == O — —

| 1 — void

| dt + dt — interface variants

| dt × dt — pair

type	constraint	destruct
1	void	—
O	—	—
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

$$\begin{aligned}
 8-6) \quad \text{Bool} &= 1 + 1 \\
 \text{Nat} &= 1 + \text{Nat} \\
 \text{Bin} &= 1 + \text{Bin} + \text{Bin} \\
 \text{List}(A) &= 1 + (A \times \text{List}(A)) \\
 \text{BMT}(A) &= 1 + (A \times \text{BMT}(A) \times \text{BMT}(A)) \\
 \text{BMT}'(A) &= A + (\text{BMT}'(A), \text{BMT}'(A)) \\
 \text{SE} &= 1 + \text{Atom} + (\text{SE}, \text{SE})
 \end{aligned}$$

$$\begin{aligned}
 d_A 0 &= 1 \\
 d_A 1 &= 0 \\
 d_A A &= 1 \\
 d_A B &= 0 \\
 d_A X + Y &= d_A X + d_A Y \\
 d_A X \times Y &= d_A X \times Y + X * d_A Y
 \end{aligned}$$

$$d_A \text{List}(A) = \text{Zipper}(A)$$

q-1/ (+ 1 2)

$\text{TAAPP} (+, 1, 2) . \text{asc}()$

prim  $\downarrow \downarrow$

= "make\_japp(make\_jprim(PLS),  
... ) "

writeToFile("x.c", 0, asc())

$J_4 \rightarrow J_5$

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

case  $e$  as  $(\text{inl } x) \rightarrow e$  or  $(\text{inr } x) \rightarrow e$

$v := \text{num} \mid \text{bool} \mid \text{prim} \mid \lambda x (x \dots) \ e$   
 $\text{unit} \mid \text{pair } v \ v \mid \text{inl } v \mid \text{inr } v$

$\text{prim} := \dots \mid \text{pair} \mid \text{inl} \mid \text{inr}$   
 $\mid \text{fst} \mid \text{snd}$

$E[\text{fst } (\text{pair } v, v)] = E[v] \quad E[\text{snd } (\text{pair } v, v)] = E[v]$

$E[\text{case } (\text{inl } v) \text{ as } (\text{inl } x_i) \rightarrow e_i \text{ or } (\text{inr } x_r) \rightarrow e_r]$   
 $\Rightarrow E[e_i[x_i \leftarrow v]]$

$E[(\text{inr } v)] \Rightarrow E[e_r[x_r \leftarrow v]]$

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

empty :=  $\text{inl } \text{unit}$

cons :=  $\lambda (\text{data rest}) . \text{inn} (\text{pair data rest})$

length :=  $\lambda \text{rec } (1) .$

case 1 of

$\text{inl } \_ \rightarrow 0$

$\text{inn } p \rightarrow 1 + \text{rec } (\text{snd } p)$

map :=  $\lambda \text{rec } (f 1) .$

case 1 of

$\text{inl } \_ \rightarrow 1$

$\text{inn } p \rightarrow \text{cons } (f (\text{fst } p))$

$(\text{rec } f (\text{snd } p))$

reduce :=  $\lambda \text{rec } (f \ \underline{\underline{z}} \ 1) .$

case 1 of  $\text{inl } \_ \rightarrow \underline{\underline{z}}$

$\text{inn } p \rightarrow \text{rec } f (f \ \underline{\underline{z}} (\text{fst } p))$   
 $(\text{snd } p)$

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

= reduce (+) 1 (cons 2 (cons 3 m+))

= reduce (+) 3 (cons 3 m+)

= reduce (+) 6 m+

= 6

true := int unit

false := int unit

if ec et ef == case ec of int → et  
int → ef

int -

int int -

int int -

pair → tuple

fst/snd → π/.proj

$\text{fst} = \text{π}_0$   
 $\text{snd} = \text{π}_1$

case<sup>(?)</sup> → case (a)

int/int → choice I

---

obj-+\* delta-pair (obj-+\* 1, obj-+\* r) {

ret make-pair (1, r); }

obj-+\* delta-fst (obj-+\* o) {

ret (posi-+\* o) → fst; }

q-y /  $e := \text{obj} ; \{ x : e, \dots \}$

$| e \cdot x$   
 $v := \text{obj} ; \{ x = v \dots \}$

$E[\text{obj} ; \{ x_0 : v_0 \dots x_i : v_i \dots x_n : v_n \}]$   
 $\cdot x_i] \Rightarrow E[v_i]$

$\{ \dots \}$

$\{ \xrightarrow{\text{empty}} \text{empty} \} = \text{empty}$

$\xrightarrow{\text{Set } o \times e = (\text{cons} (\text{pair } "x" e) o)}$

$\emptyset, x = \text{lookup } "x" e$

$\text{lookup} := \lambda \text{rec} (\text{field obj}).$

$\text{case obj of int} \rightarrow (\text{rec field obj})$

$\text{inr } p \rightarrow \text{if string=? field } (\text{fst } p)$

$(\text{snd } p)$

$(\text{rec field } (\text{snd } p))$

---

$(\text{Ek } K = \text{Kcase } (x_1, e_1, x_r, er, env, tc) \text{ Krase}(C, tc))$

$\langle \text{case } e_1 \text{ of int } x_1 \rightarrow e_1 \text{ or inr } x_r \rightarrow er, env, tc \rangle$

$\mapsto \langle e_1, env, \text{Kcase}(x_1, e_1, x_r, er, env, tc) \rangle$

$\langle \text{inr } v, \dots, \text{Kcase}(x_1, e_1, x_r, er, env, tc) \rangle$

$\mapsto \langle e_1, env[x \leftarrow v], tc \rangle$

## 9-5 / Mutation

$$A \rightarrow B \supseteq \left\{ \begin{array}{l} (a_1, b) \\ \dots \\ (a_n, b) \end{array} \right\}$$

JS

```
const x = f(3);
```

```
console.log(x); "42"
```

```
const y = f(3); y=x
```

```
console.log(y); "??"
```

```
function f(x){  
    return x + 39; }
```

let c = 0; const f = (x)  $\rightarrow$  x + 39 + c++;

if p f()

...

if q c()

...

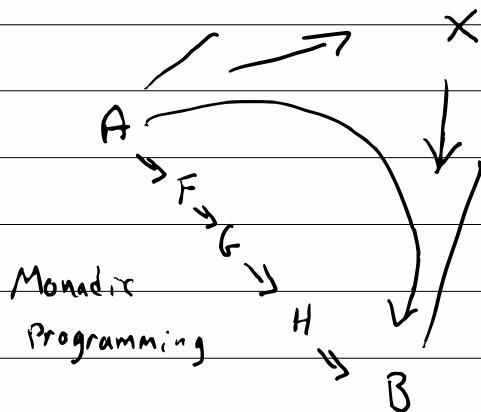
f(3)

Math

$$x = f(3)$$

$$y = f(3)$$

$x = y ?$

$$f(3) = f(3)$$


12-1

Goal: add mutation

$$\begin{array}{ll}
 e ::= \dots \mid \text{box } e & E = \dots \mid \text{box } E \\
 \mid \text{unbox } e & \mid \text{unbox } E \\
 \mid \text{set-box! } e \leftarrow e & \mid \text{set-box } E \leftarrow e \\
 & \mid \text{set-box } v E
 \end{array}$$

struct box { void \*p; }

box (int x) { ip = malloc (int)

box b = { p = ip }

\*ip = x;

ret b; }

let b = box b in

(set-box! b 8; ) + (unbox b)

b  $\rightarrow \sigma_0$

(set-box! (box b) 8; ) + (unbox (box b))

unbox (box b) )

$\phi_{\text{fun}}$   $(\text{set-box! } \sigma_0 8; ) + (\text{unbox } \sigma_0)$

$\Rightarrow \phi[\sigma_0 \mapsto 8] (\text{unbox } \sigma_0 + \text{ub } \sigma_0) \Rightarrow \sigma[\sigma_0 \mapsto 8] / (8 + 8) = 16$

12-2 / small step :  $e \rightarrow e'$   
 $\Sigma \times e \rightarrow \Sigma \times e'$   
 $(M, S, \rho) \rightarrow (M', S', \rho')$

$\Sigma = \text{store } (\text{memory / heap})$   
 $\text{ptrs} \Rightarrow \text{vals}$   
 $\sigma \rightarrow v$

$\Sigma / E[\text{if } T \text{ et } e]$   
 inject  $e = \emptyset / e \rightarrow \Sigma / E[e]$   
 extract  $\Sigma / v = v$   
 $v := \dots | \sigma$

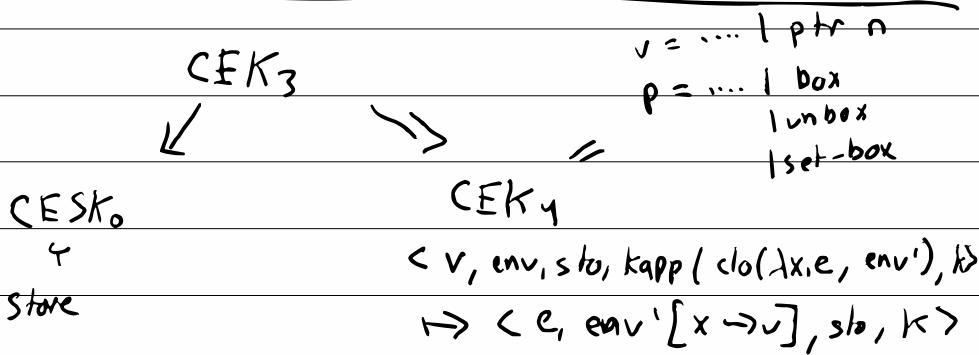
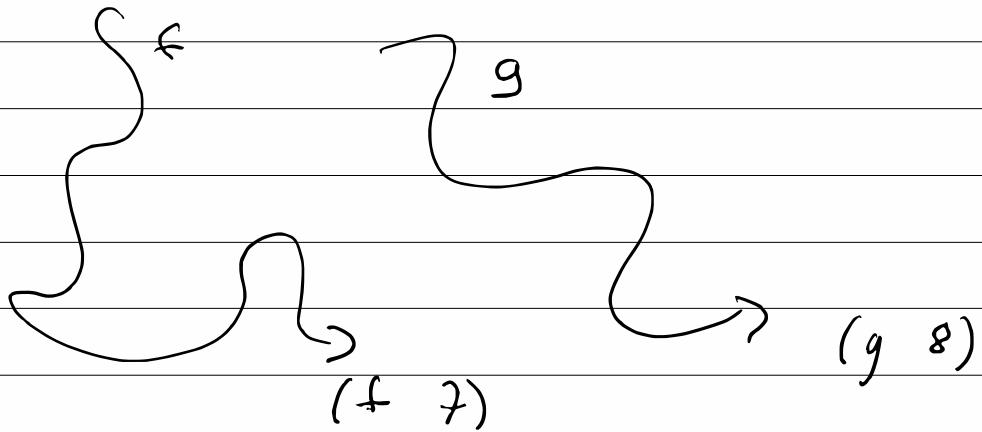
$\Sigma / E[\text{box } v] \rightarrow \Sigma[\sigma \mapsto v] / E[\sigma]$   
 where  $\sigma$  d.n.o..  $\Sigma$  ( $\sigma = \text{malloc}$ )

$\Sigma / E[\text{unbox } \sigma] \rightarrow \Sigma / E[\Sigma(\sigma)]$   
 $\Sigma / E[\text{set-box! } \sigma \ v] \rightarrow \Sigma[\sigma \mapsto v] / E[v]$   
 $E[\text{unit}]$   
 $\text{void}$

12-3 / let  $b = \text{box } 0$  in

let  $f = \lambda x, \text{set-box! } b (x+1);$   
 $x = 2$  in

let  $g = \lambda y, y + \text{unbox } b$  in



$\langle \text{if } e_c \text{ et } cf, \text{ env}, \text{ sto}, \text{k} \rangle$

$\mapsto \langle pc, \text{env}, \text{sto}, \text{kif}(\text{env}, \text{et}, \text{ef}, \text{k}) \rangle$

$\langle v, \text{env}, \text{sto}, \text{kbox}(\text{box } e, \text{env}, \text{sto}, \text{k}) \rangle$

$\langle \text{box } e, \text{env}, \text{sto}, \text{k} \rangle$

$\mapsto \langle \sigma, \text{env}, \text{sto}[\sigma \mapsto v], \text{k} \rangle$

$\mapsto \langle e, \text{env}, \text{sto}, \text{k} \rangle$

where  $\sigma = \text{malloc}(\text{sto})$

$\langle \text{box } (k) \rangle$

(2-γ)  $\Phi^r = \dots$  | pair  $\checkmark$   $\checkmark$   
| ~~max~~ σ

Question : Should we add

`set-fst` : pair A B × A → I

Set-Snd : Pair A B × B → ()

$\text{mpair } a \ b = \text{pair } (\text{box } a) \ (\text{box } b)$

`mpair-set-fst p a' = setbox (fst p) a'`

`mpair-fsL p = unbox (fst p)`

let  $f$   $x_i =$   
let  $x - \text{box } x_i, \rightarrow$   
set!  $x S;$

Let  $v = (\log 7)$  in

set! v 8

v t v

$e := \dots$  | set!  $x$   $e$

6

12-5 / Always store vars as pointers

ULD:

$$\Sigma / E[(\lambda x. e) \ v] \rightarrow \Sigma / E[e[x \leftarrow v]]$$

NEW:

$$\Sigma / E[(\lambda x. e) \ v] \rightarrow \Sigma[\sigma \mapsto v] / \\ \text{where } \sigma = \text{mallc}(\Sigma) \quad E[e[x \leftarrow \text{unbox } \sigma]]$$

$$\Sigma / E[\text{self! } (\text{unbox } \sigma) \ v] \rightarrow \\ \Sigma[\sigma \mapsto v] / E[v]$$

desugar  $(e_1; e_2) =$

let  $\text{marnihcarinacgnymng}$  =  $e_1$  in

$e_2$

$$= (\lambda \_ . e_2) \ e_1$$

## 12-6/ Fancy desugar

desugar  $(\lambda x. \dots)$

$x$

set!  $x \leftarrow$

$\vdots$   
)

$= (\lambda x. \dots \text{let } x = \text{box } x_i \text{ in} \dots (\text{unbox } x))$

set-box!  $x \leftarrow$

desugar  $M (\lambda x. e)$

if modified  $x \in M$  then

$\lambda x. \text{let } x = \text{box } x_i \text{ in desugar } (\overbrace{M \cup \{x\}}^M) e$

o.w.

$\lambda x. \text{desugar } M e$

desugar  $M x =$

if  $x \in M$  then  $\text{unbox } x$

o.w.  $x$

12-7

$\text{eval } e$   
=  $\text{eval} \left( \text{let } \overbrace{\dots}^{\text{stdlib}} \text{ in } e \right)$

→ prints 11

→ compile

→ nys

desarrollar (map) =  $\lambda f. \dots \dots \dots$

$\underline{15-1}$        $1 / 0$        $\delta \xrightarrow{\text{partial}}$   
 $(S \quad 1)$        $\delta$  is undefined  
 set-box!       $\exists \quad 2$       function app needs a box

$$x \rightarrow y \rightarrow z \rightarrow (\cancel{*} \div 1 \ 0) \rightarrow$$

$$\text{eval}(p) = v \text{ iff } p \xrightarrow{\text{partial}} v$$

$v = \dots | \text{bad bad bad}$   
 $E[(v_0 \ v \dots)] \rightarrow \text{bad bad bad}$   
 where  $v_0 \in p, \in \lambda \dots$

$$v = \dots | \text{err}, [ \dots | \text{err}, \dots ] \text{ or } k$$

$$E[(p \ v \dots)] \rightarrow \text{err}, \exists$$

$$\delta(p, \vec{v}) = \perp \qquad E[a] \rightarrow E[b]$$

$$e = \dots | \text{abort } e \qquad E' = E$$

$$E[\text{abort } e] \rightarrow e$$

$$\langle \text{abort } e, \text{env}, k \rangle \mapsto \langle e, \text{env}, \text{kret} \rangle$$

J<sub>1</sub>:

15-2)  $e = \dots | \text{throw } e$   
 $| \text{try } e \text{ with catch } e$

$\text{try } (+ 1 \ (\text{throw } 2))$   $\Rightarrow |$   
with catch  $(\lambda x. (- x 1))$

$E = \dots | \text{try } \overset{e}{\cancel{\text{try}}} \text{ with catch } E$   
 $| \text{try } E \text{ with catch } v$

$E [\text{try } v \text{ with catch } u] \rightarrow E[v]$

$E [\text{try } L[\text{throw } v] \text{ with catch } u]$   
 $\rightarrow E[u v]$

$L = E - (\text{try } E \text{ with catch } v)$

$\text{try } (+ 1$   
 $(\text{try } (+ 2 \ (\text{throw } 3)))$   $\Rightarrow 7$   
with catch  $(\lambda(v) (* v 2)))$   
with catch  $(\lambda(x) (* x 3))$

15-3 /  $K = \dots \mid \text{preTry } K \ e \ \text{env} \ k$   
 $\mid \cancel{\text{try }} K \ v \ k$

$\langle \text{try } e_b \text{ with catch } e_k, \text{ env}, k \rangle$

$\mapsto \langle e_h, \text{ env}, \text{ preTry } K \ e_b \ \text{env} \ k \rangle$

$\langle v_h, \_, \text{ preTry } K \ e_b \ \text{env} \ k \rangle$

$\mapsto \langle e_b, \text{ env}, \text{ Try } K \ v_h \ k \rangle$

$\langle \text{vars}, \_, \text{ Try } K \ v_h \ k \rangle$

$\mapsto \langle \text{vars}, \_, k \rangle$

$\langle \text{throw } e, \text{ env}, \text{ Try } K \ v_h \ k \rangle$

$\mapsto \langle v_h \ e, \text{ env}, k \rangle$

$\mapsto \langle e, \text{ env}, \text{kapp } (v_h) \ _- \ () \ k \rangle$

$\langle \text{throw } e, \text{ env}, \text{kapp } (v...) \ \text{env}' (e...) k \rangle$

$\mapsto \langle \text{throw } e, \text{ env}, k \rangle$

$\text{preTry } K \ e \ \text{env}' \ k$

$\langle \text{throw } e, \text{ env}, \text{kret} \rangle \mapsto \langle e, \text{ env}, \text{kret} \rangle$

15-y ~~(f a b c)~~ →

(let ([av a])

(if (function? av)

(if (= (function-arity av) 2)  
<sup>arity</sup>  
(av b c))

[throw "wrong num args"] )

(throw "not fun"))

(+ 2  
    <sup>(throw o)</sup>)

try ...  
    <sup>L</sup> [ (throw e) ]

with catch

(λ (x tryagain)

(tryagain 8)

E [ try L [ throw e ] with catch u ]

→ E [ u e  
      (λ (x) try L [ x ] with catch u) ]

## 15-5/ First-class continuations

$e = \dots | \text{callcc } e$

$E = \dots | \text{callcc } E$

$$E[\text{callcc } v] \rightarrow E[v (\lambda(x) \text{ abort } E[x])]$$

$\text{cek } v = \dots | \text{Kont } K$

$K = \dots | k\text{callcc } K$

$\langle v, -, k\text{callcc } K \rangle$

$\mapsto \langle v^{(\text{kont})}, -, K \rangle$

$\langle \text{callcc } e, \text{env}, K \rangle \mapsto \langle e, \text{env}, k\text{callcc } K \rangle$

$\langle \text{if } c + f, \text{env}, K \rangle \mapsto \langle c, \text{env}, k\text{if env} + f K \rangle$

$\langle v, -, K\text{app } (\text{kont } K) - () - \rangle$

$\mapsto \langle v, -, K \rangle$

15-6 / ~~Ex~~

$f = (\lambda (x) \quad \text{int } f(\text{int } x)\}$   
(callee  $\lambda$  (return))  
(if ( $x == 0$ )  
    (return 2);  
    printf("0x%lx",  
          return 2/x);  
    ~~return~~  
(print x)  
(/ 2 x)))

$(+ 1 (f 7))$   
 $(+ 1 (f 0))$   
return =  $\lambda x. \text{abort } (+ 1 x)$

$(\lambda (x \dots) b) \Rightarrow (\lambda (x \dots) \quad \text{callee } (\text{if } (x == 0) \quad \text{return } 2; \quad \text{printf } ("0x%lx", \quad \text{return } 2/x); \quad \text{print } x) \quad b))$

15-7 /

(define last-handler  
(box (λ (x) (abort x))))

(define throw  
(λ (v) ((unbox last-handler) v)))

(esugar (try e<sub>1</sub> with catch e<sub>2</sub>)  
= (try-catch\* (λ () e<sub>1</sub>) e<sub>2</sub>)

try-catch\* := (λ (body new-handler)  
(let ([old-handler (unbox last-handler)])  
(catch (λ (here)  
(set-box! last-handler (λ (x) (set! lh oh)  
(here (nh x))))

~~(let ([ans (body)])  
(set-box! lh oh)  
ans))))))~~