

H/

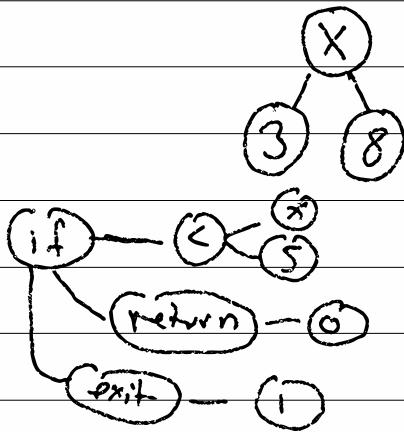
$1 + 1$

5

$1 +$

1×3

" 3×8 "



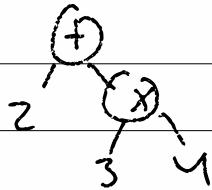
if ($x < 5$) {
 return 0; }
else {

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

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

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

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

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

$\text{Joe} \mid , \wedge; \text{JMlt}(\dots) \in \Sigma$?

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

$\text{new JPlus}($

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

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

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

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

class $\text{JPlus}:$

def __init__(l, r):

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

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

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

$n \quad n$

1-3/6 pp : J₀ \Rightarrow string

③ pp n = itos(n)

④ pp (+ e_L e_R) = "(#pp(e_L) ++ "+" ++ pp(e_R)
++ ")"

⑤ pp (* e_L e_R) = "(" ++ pp(e_L) ++ ">*" ++
pp(e_R) ++ ")"

① interface J₀ { 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_L e_R) = interp e_L + interp e_R

interp (* e_L e_R) = interp e_L * interp e_R

→ class JMult {

public ^{int} interp () {

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

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

desugar →

se = empty | (cons ^{min} 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

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

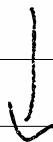
$$(" - " \ e_1 \ e_2) \Rightarrow (+ \ (\text{d } e_1) \ (\text{de } (" - " \ e_2)))$$

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

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

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

"*



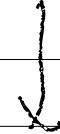
= 1



"x"

"x"

(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

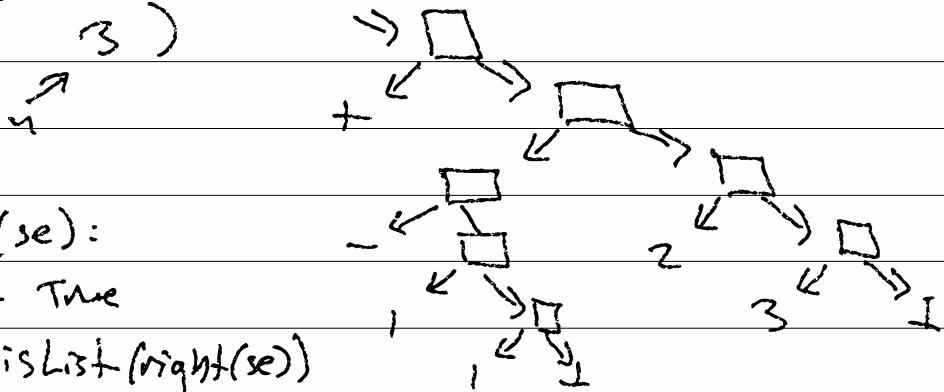
second \leftarrow left \rightarrow right

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

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



isList(se) :

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

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

$\text{or} : \text{False}$

$\text{Len}(se)$:

$\text{Null} : 0$

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

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

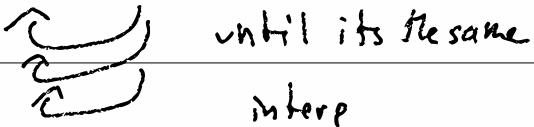
where $p = \text{interp } e_f$

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

$\delta : b \dots \rightarrow b$

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

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

$\Sigma^4 /$ "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')

$(+ (+ 1 1) z) \leftarrow (+ (+ 1 1))$
 $\Rightarrow (+ z z) \leftarrow (+ 1 1))$
 $\Rightarrow \frac{4}{(+ z (+ 1 1))}$

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

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

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

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

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

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

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

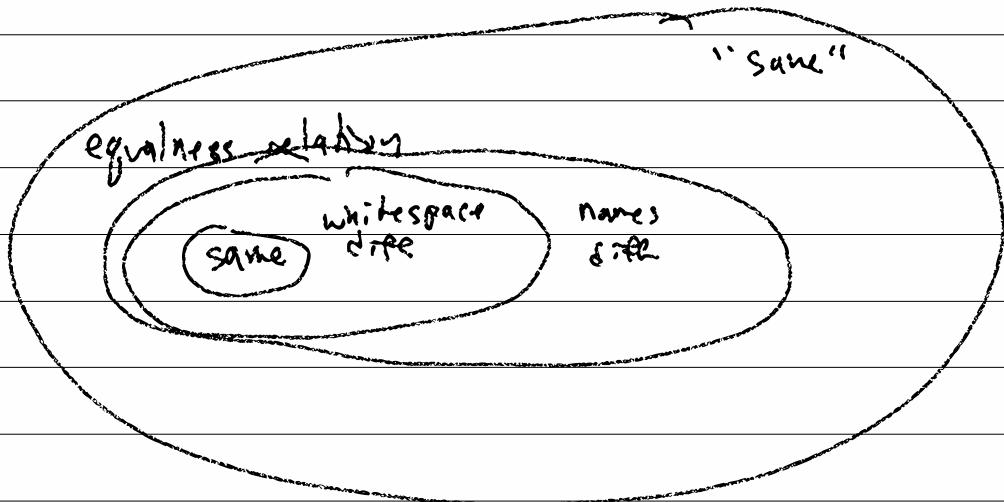
$$\text{if } e_1 \in V \text{ then } (\text{hole}, e)$$

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

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

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

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

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

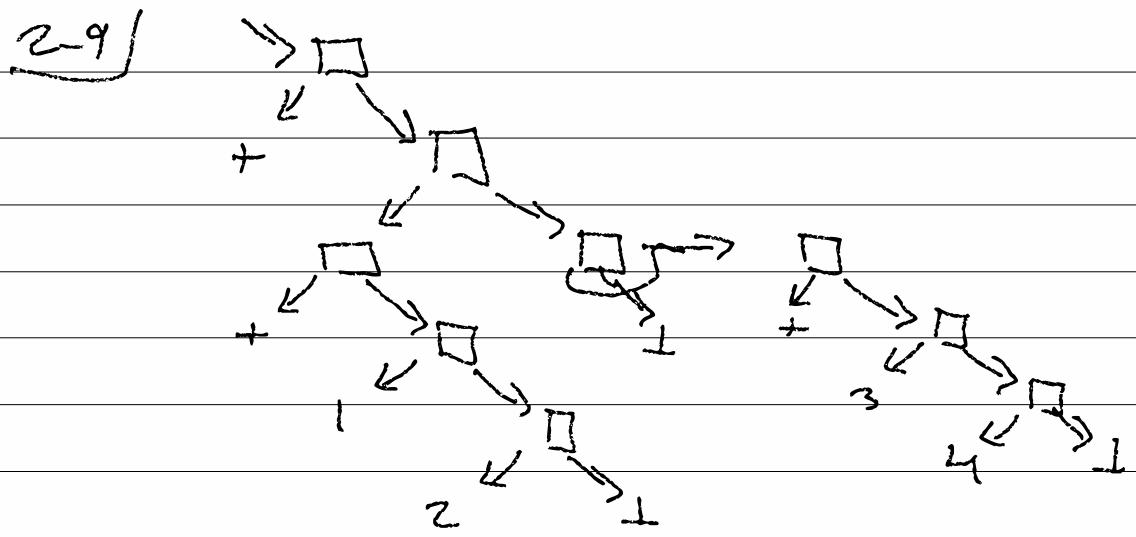
"mique decomp: \vdash_{mique} " $\downarrow^{\text{mique } E}$
 & e. $e \in \vee$ or $e = E[e']$ where
 $e' \in \vee$

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

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

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/ Σ "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 Σ

Expr plug (Expr); }

Hole : Context + Σ

plug (e) = e; }

If C : Context + Σ

Context c; Expr t, f;

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

AppC : Context Σ

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] > \mapsto < , AppC
[+] AppC
[+] hole [3] >

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

CK₀ 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 $\langle e_0 e_1 \dots, k \rangle \mapsto \langle e_0, k_{\text{app}} () (e_1 \dots) k \rangle$

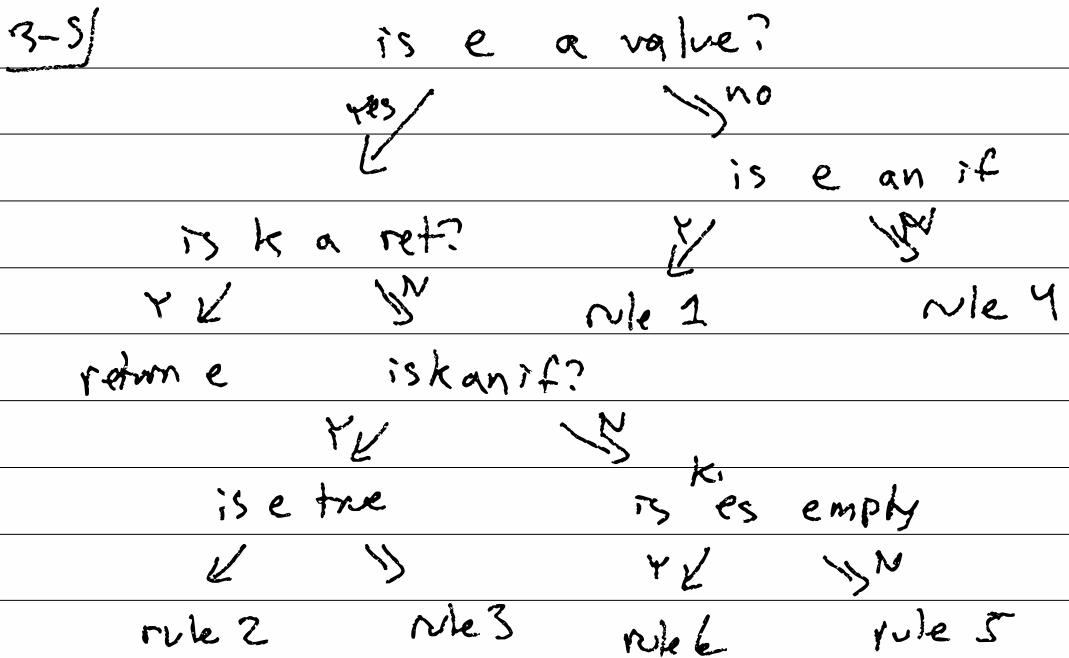
4 $\langle v_1, k_{\text{app}} (v_0 \dots) (e_0 e_1 \dots) k \rangle$

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

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 / Σ_0 $e = n \mid (+ \ e \ e)$
 | $(\neq \ e \ \epsilon)$

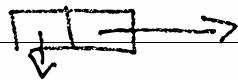
Σ_1 $p = \text{unary} - (\text{neg})$, not (1)

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

class TApp impl TExpr {
 List<TExpr> contents }

App
 $(+ \ 1 \ 2) \Rightarrow \text{List} (+, 1, 2)$
 $\quad \quad \quad \downarrow \quad \downarrow \quad \downarrow$
 $\quad \quad \quad p \quad n \quad n$

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



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

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

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

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

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

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)

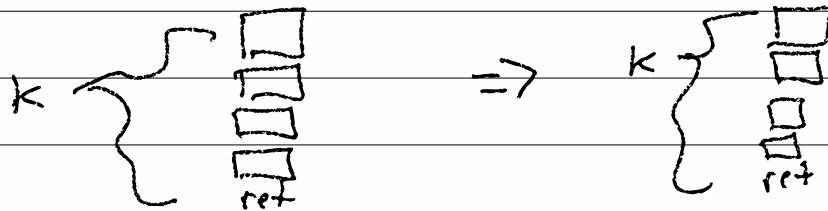
CK

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

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

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

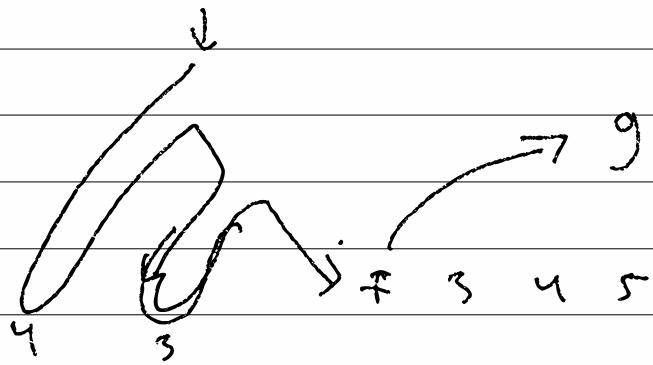
\rightarrow fun "at fault" $\sim F$

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

"past"

G - H

$y=f$)



$g(\cdot \cdot \cdot)$

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

$n(x)$