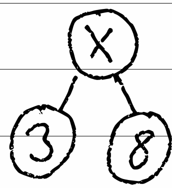
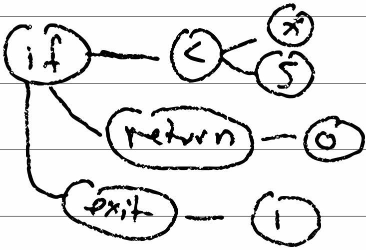
1-1   1 + 1   5   1+

1 × 3   " 3 × 8 "



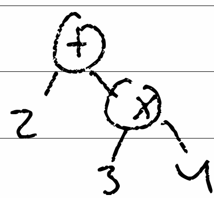
```
if (x < 5) {
    return 0; }
else {
    exit(1); }
```



(+ 1 1)

top   children

(+ 2 (x 3 4))



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

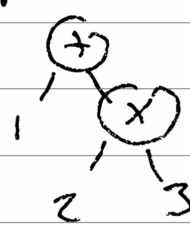
v := number | (* e e)

(+ 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() ... {} }
```

→ 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~~ pair se se) | string

(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))

("-" e_1 e_2) => (+ (d e_1) (de ("-" e_2)))

("+") => 0

("+" e_1 more ...) =>

(+ (d e_1) (d ("+" mor ....)))

'*'

"x"

(x

= 1

"x"

$$se \xRightarrow{\text{desugar}} J_0 \xRightarrow{\text{interp}} V$$

$$\xRightarrow{\text{compile}} bc \xRightarrow{\text{vm}} V$$

2-)) b

desugarer $(- \ e_1) \Rightarrow (* \ -1 \ e_1')$

$(- \ e_1 \ e_2) \Rightarrow (+ \ e_1' \ (- e_2))$

$(+) \Rightarrow 0$

$(+ \ e_1 \ e_2 \ldots) \Rightarrow$

$(+ \ e_1' \ (+ \ e_2 \ldots))$

def

desugar (se) :

if is~~Cons~~(se) && length (se) = 2 &&

first (se) == "-" then

> def length (se) :

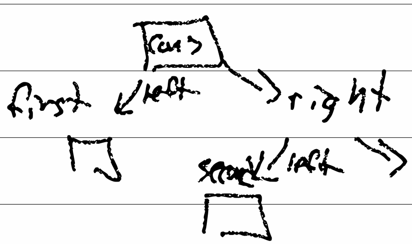
> if is Null (se) : return 0

> else is Cons (se) : return (+ length (right(se)))

> else ~~false~~ false

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

second (se)))



if isList(se) && len(se) = 3 && ~~fisf~~ first (se)=="-"

then : return new JAdd(desugar(sec(se)),

desugar(new Cons("-", ~~[illegible]~~

new Cons(third (se), Null)

2-2] Cons(a, Cons(b, Cone(c, null))))

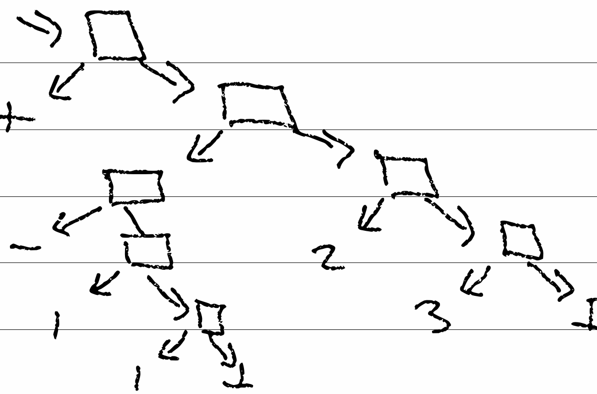
len = 3

(+ (- 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/ $J_0 \Rightarrow J_1$

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

$v ::= b$

$b ::=$ some set of constants

// in J_0, $b = \text{num} \mid + \mid *$

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 $e_a \ldots$

$\delta : b \ldots \to b$

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

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

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 e_1 e_2) x = if x e_1 e_2
plug (if1 e_1 C e_2) x = if e_1 x e_2
plug (e_1 ... C e_2 ..) x = (e_1 ... x e_2 ...)

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 ⇒ C × e

step ⟶ e

interp e = if e ∈ v then e

C, e' = parse e e

e'' = step e'

plug C e''

redex

parse : e ⇒ C × e

parse (if ec et ef) =

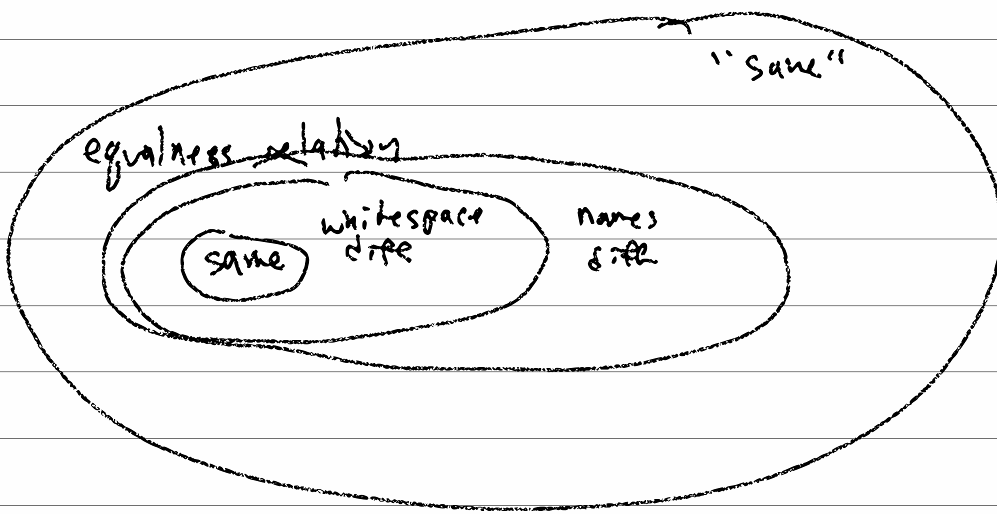
if ec ∈ v then (hole , e)

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 $x = y$

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

C = (map hole (list 1 2))

....

Observational Equivalence

2-8) $C ::=$ hole | if C e e
| if e C e
| if e e C
| $(e \ldots C \; e \ldots)$

$E ::=$ hole | if E e e
| $(v \ldots E \; e \ldots)$

"unique decomposition"

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

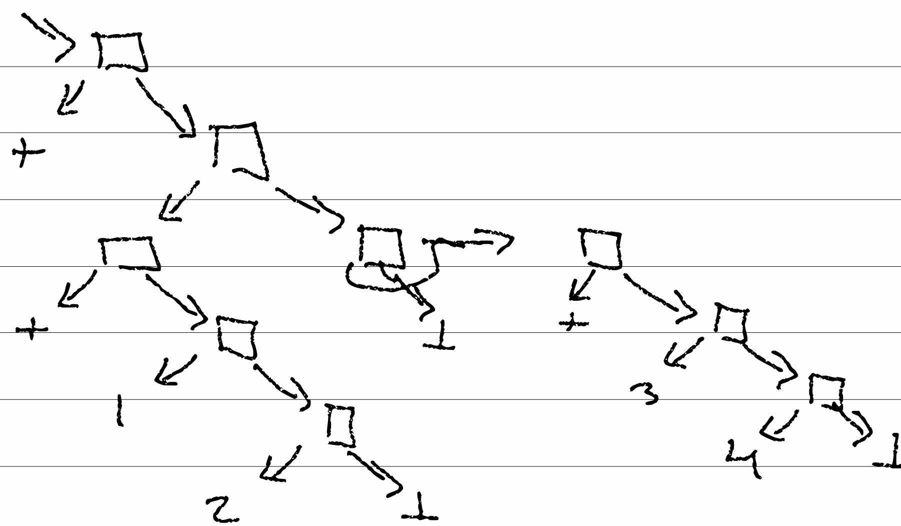
$\delta(1, 2) = \bot$ $\quad\quad$ $\delta(1, 1, 0) = \bot$

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

step $E[\text{if true } e_t\ e_f] \to E[e_t]$
step $E[\text{if false } e_t\ e_f] \to E[e_f]$
step $E[p\ \ v_a \ldots] \to E[\delta(p, v_a \ldots)]$

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"

$$e \to e$$
$$st \mapsto st$$

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

done?
fv

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

inject $e = \langle e, hole \rangle$

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

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

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

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

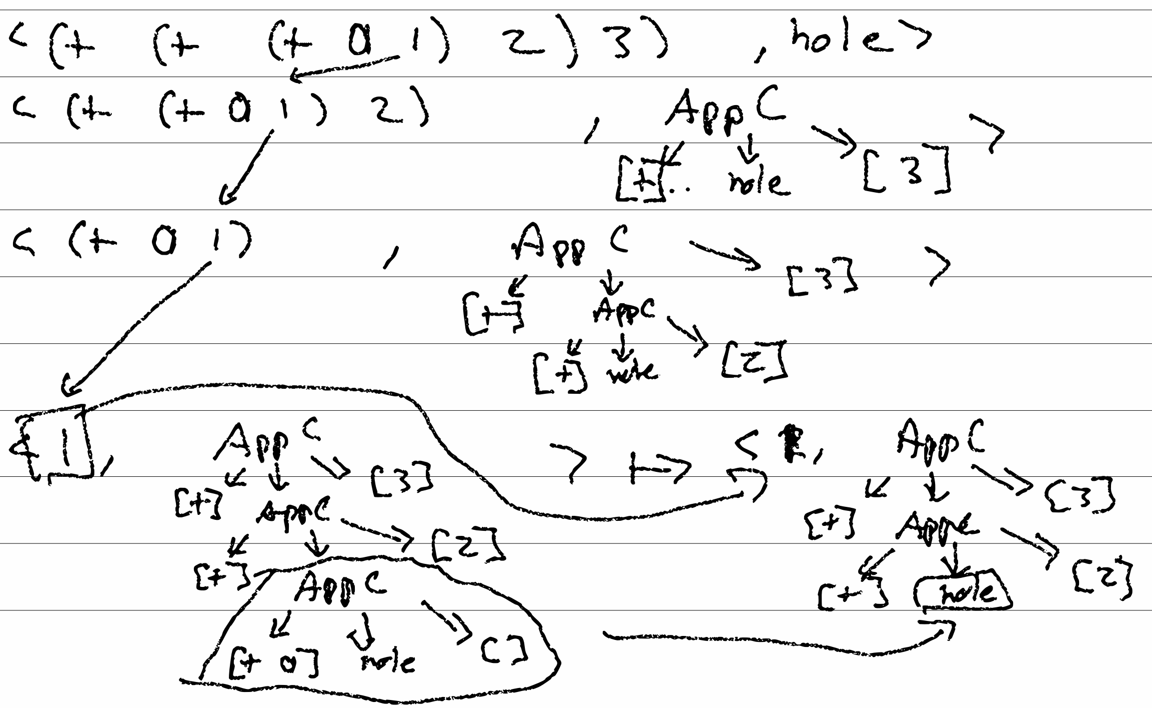
$\langle e_0\ e_1 \ldots, E \rangle \mapsto \langle e_0, E[\text{hole } e_1 \ldots] \rangle$

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

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

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);}
```



3-4

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

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

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

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

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

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

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

while(1) {

3-5

- 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.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 {
  expr* c, *t, *f; }
        h;
struct num {
  int n; }
      h;
struct app {
  expr *f , *args ; }
          h;

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

```
struct expr {
  enum tag; }

enum tag {
  IF, NUM, APP,
  BOOL, PRIM,
  KRET, KIF,
  KAPP, CONS, NIL};
```

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

0 2 0 1 1 2 2 1 $ 0 7

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

$\qquad\qquad\quad \mid (*\ e\ e)$

J_1 $\quad p = $ unary− (neg) , not (1)

$\qquad\qquad$ + , * , between

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

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

```
class JApp impl JExpr {
  List<JExpr> contents }
```

App

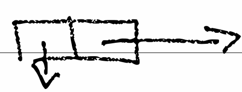
(+ 1 2) ⇒ List(+ , 1 , 2)

$\qquad\qquad\qquad\qquad\qquad\qquad$ ↓ p $\quad$ ↓ n $\quad$ ↓ 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)

$\qquad$ 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

~~(+ A) => A~~

(+ 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 | ($\overset{e}{e}$ ...) | (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)))

$f(x) = 1 + x$

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

$f(x) = 1 - x$

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

$\cancel{= -3 + 4 = 2}$

"

$f(7)$

$= 1 - 7 = -6$

Double (1+1) = Double (2)

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

4-3

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

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

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

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

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

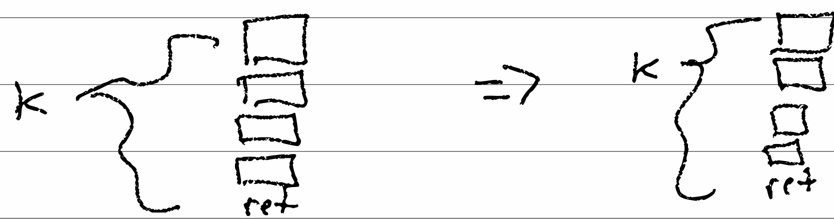
7: $\langle v_n, \text{kapp}((f\ v_0 \ldots), (), k) \rangle$

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

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

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

c = 2 kapp (Expt 7)



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

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

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

(F 10)

$\Rightarrow$ Σ = [F $\mapsto$ (define (F x) (F x))]

e = (F 10)

k = kret

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

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

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

& Σ(F) = (define (F x) (F x))

f ⇐ x... = e ⇐

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

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

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

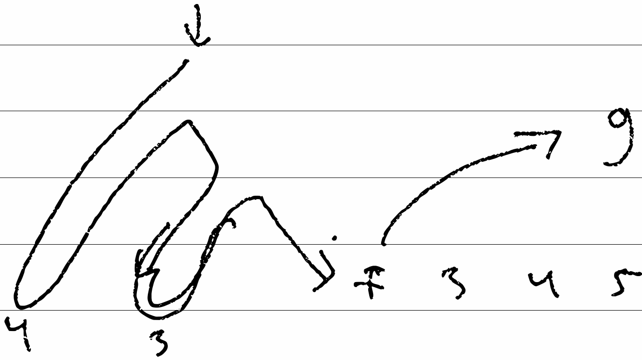
error $\Rightarrow$ stack trace

$\Rightarrow$ fun "at fault" — F

fun that called ~~F~~ — G

"past" G — H

4-7)



g (. . ——)

x = f 3 4 5

h (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 = hole e = ↑

C = hole e = ↑

C = if true hole 4
e = (+ 1 2)

(+ 1 2)
E = 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 c
(if(E t f , e)

fr (app es) =
for e in es
if (value? e)

(+ 1 2)

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

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

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

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

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

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

6-3/ J_2 = PASCAL or C

top-level functions

CK = have the map $f \Rightarrow d$

and we have subst

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

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

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

$x[x \leftarrow v] = v$

$y[x \leftarrow v] = y$

$u[x \leftarrow v] = u$

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

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

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

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

OLD: No rule for a variable in the c pos

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

NEW:

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

(Σ)

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

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

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

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

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

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

$\quad \longmapsto \langle c, env, kif\ t\ f\ k \rangle$

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

$\quad \longmapsto \langle t, env, k \rangle$

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

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

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

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

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

$\quad \longmapsto \langle e, \cancel{env}\,\emptyset\,[x_0 \leftarrow v_0] \ldots [x_n \leftarrow v_n], k \rangle$

$\quad$ 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

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

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

$$k = kret \mid kif\ env\ t\ f\ k$$
$$\mid kapp\ \vec{v}\ env\ \vec{e}\ k$$

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

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

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

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

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

$$\langle v_n, \_, kapp((f\ v_0 \ldots), \quad, (), k) \rangle$$
$$\mapsto \langle e, \emptyset[x_0 \leftarrow v_0] \ldots [x_n \leftarrow v_n], k \rangle$$

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

"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 ... e

JS = (x) => 1 + x

Py = lambda: x : 1 + x

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

J3

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

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

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

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

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

$E[e\ [x_0 \leftarrow v_0] \ldots [x_n \leftarrow v_n]]$

$((\lambda x.$
$((\lambda y.$
$x + y)\ 7))\ 8)$ => 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
$$(v ::= b \mid \star) \neq (v ::= b \mid \lambda(x\ldots)\ e)$$
$$\text{closure}(\lambda(x\ldots)\ e,\ env)$$

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

$$\langle v_n,\ -,\ kapp(\ (clo(\lambda(x_0 \ldots x_n).e,\ env)\ v_0 \ldots),$$
$$(),\ k) \rangle$$
$$\longmapsto \langle e,\ env[x_0 \leftarrow v_0] \ldots [x_n \leftarrow v_n],\ k \rangle$$

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

let $x = e_1$ in e_2

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

let $x = 1$ in

let $y = 2$ in

$x + y$

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

$(\lambda\,(y)\ (+\ 1\ y))\ 2$

CEK_1 : $v =$ ....

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

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

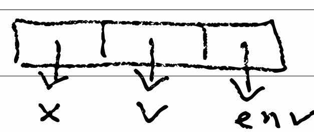
$\langle \lambda\,(x \ldots)\ e,\ env,\ k \rangle$

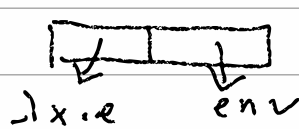
$\longmapsto \langle clo(\lambda\,(x \ldots)\ e,\ env),\ \emptyset,\ k \rangle$

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

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

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

$env = \perp \mid$ 

$clo =$ 

let y = 3 in
let z = 8 in

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

z → 8

y → 3

x → 19

a → 22

b → 36

...

$$\Downarrow$$

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

SA = nat   env = vector v

FLAT-CLOSURES [_ , 3]

SA = (nat, nat)   env = ↓ , vector v

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

$\widehat{(0,0)}$ $\widehat{(1,1)}$ ^, [_ , _]   NESTED CLOSURES

7-3/

$$= v \mid (\text{if } e\ e\ e)$$
$$x \mid (e\ e) \mid (p\ e\ e)$$
$$v = b \mid \lambda\,(x)\ e$$
$$b = \text{num} \mid \text{bools} \mid \text{prim}$$
$$\text{prim} = + \mid - \mid * \mid \div \mid <$$

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

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

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

$$e = v \mid x \mid e\ e \qquad E = \text{hole}$$
$$v = \lambda x.\ e \qquad \mid (E\ e)$$
$$\mid (v\ E)$$

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

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

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

$\text{True} = \lambda x.\, \lambda y.\, x$
$\text{False} = \lambda x.\, \lambda y.\, y$
$\text{if} = \lambda c.\, \lambda x.\, \lambda y.\, c\ x\ y = \lambda c.\, c$

if True A B = True A B = A

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

```
interface Bool { int choose (int, int); }
                 True()
class True : Bool { int choose( x , y ) = x }
                  False()
class False : Bool { int choose (x, y) = y }
                  Bool b.
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

add $\overset{n}{x}$ $\overset{m}{y}$ = ~~[illegible]~~ $\underbrace{\qquad}_{}^{n+m}$

$zero := \lambda f. \lambda x. \quad x$
$one := \lambda f. \lambda x. \; f \; x$
$two := \lambda f. \lambda x. \; f \; (f \; x)$

$add1 := \lambda n. \lambda f. \lambda x. \; f \; (n \; f \; x)$
$add := \lambda n. \lambda m. \lambda f. \lambda x. \; n \; f \; (m \; f \; x)$
$zero? := \lambda n. \; n \; (\lambda x. FALSE) \; TRUE$
$mult := \lambda n. \lambda m. \lambda f. \lambda x. \; n \; (m \; f) \; x$

two two

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

7-6/ Pair

fst (pair A B) = A
snd (pair A B) = B

pair = λa. λb. λc. if c a b
fst = λp. p TRUE
snd = λp. p FALSE

sub1 := λn. fst (n (λp. pair (snd p) (add1 (snd p))) (pair z z))

mkfac := λfac. λn.
if (zero? n)
1 = one
(mult n (fac (sub1 n)))

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

fac := mkfac fac $x = F\ x$

$\uparrow$ x $\uparrow$ F $\uparrow$ x

7-7) Fixed point of a lambda?

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

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

Z-combinator

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

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

A

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

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

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

$\lambda x. \overbrace{\lambda y. x\ y}^{x}$

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

Lambda-Calculus

n add1 0

$\lambda x. x+1$

Church Numeral / Church encoding

Talk about
cc0

8-1 Lambda - Calculus

tiny : e → e

tiny (if T et ec) = et
tiny (if F et ef) = ef
tiny (p v_0 … vn) = δ(p, v…)

small : e → e

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

big : e → v

big e =
  if e ∈ v : return e
  o.w. big (small e)

cc0 : C × e → C × e
cc0 C e =
  C' × e' = move-context C e
  e'' = tiny e'
  ret (C', e'')

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

big' : e → e
extract (cc0* (inject e))
inject = (hole, e)   extract (hole, v) = v

8-2)

$\langle$ if ec et ef , E $\rangle$

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

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

inject A = $\langle$ A, hole $\rangle$

cc0 $\langle$ A , hole $\rangle$

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

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

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

$\mapsto$ $\langle$ B , (+ 1 hole) $\rangle$ = E[(if hole et ef)]

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

$\mapsto\mapsto$ $\langle$ False , = $\rangle$

$\mapsto$ $\langle$ F , (+ 1 hole) $\rangle$

$\mapsto\mapsto\mapsto$ $\langle$ 9 , = $\rangle$

$\mapsto$ $\langle$ 10 , 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)

$J_3 \Rightarrow J_4$

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

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

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 = λf (x...) . xe in b"

8-4

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

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

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

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

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

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

```
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), ...)"

writetofile("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

$E[\text{fst } (\text{pair } v_1\ v_2)] = E[v_1]$ $E[\text{snd } (\text{pair } v_1\ v_2)] = E[v_2]$

$E[\text{case } (\text{inl } v) \text{ as } (\text{inl } x_l) \Rightarrow e_l \text{ or } (\text{inr } x_r) \Rightarrow e_r]$

$\Rightarrow E[e_l[x_l \leftarrow v]]$

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

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

reduce := λrec (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

$\text{case}^{(2)}$ → case (n)   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/ $e ::= obj\ \{ x : e, \ldots \}$

$\mid\ e . x$

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

$E[\, obj\ \{ x_0 : v_0 \ \ldots \ x_i : v_i \ \ldots\ x_n : v_n \} . x_i] \Rightarrow E[\, v_i]$

$\{\}$

empty % = empty

$\{o, x : e\}$

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

e.x = lookup "x" e

lookup := λrec (field obj) .

case obj of inl _ → (rec field obj)

inr p → if string=? field (fst (fst p))

(snd (fst p))

(rec field (snd p))

CEK K = Kcase (x₁, e₁, xr, er, env, k) Kcase($\vec{c}$, k)

⟨case e_s of inl $x_l \rightarrow e_l$ or inr $x_r \rightarrow e_r$, env, k⟩

↦ ⟨e_s, env, kcase (x_l, e_l, x_r, e_r, env, k)⟩

⟨inl v, _, Kcase (x_l, e_l, x_r, e_r, env, k)⟩

↦ ⟨e_l, env[x ← v], k⟩

9-5/ Mutation

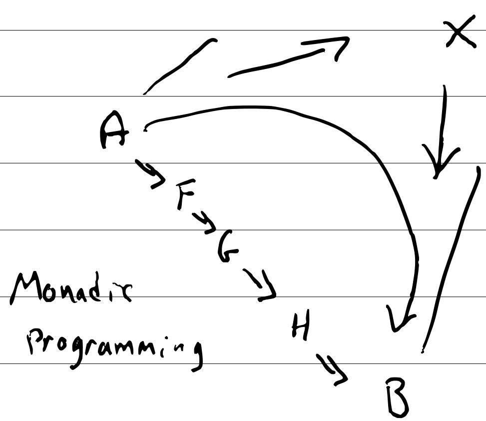
$A \to B = \{(a, b), \dots, (a, b)\}$

JS	Math
`const x = f(3);` ($g(f)$)	$x = f(3)$
`console.log(x);` "42"	
`const y = f(3);` $y = x$	$y = f(3)$
`console.log(y);` "??"	
	$x = y$?
	$f(3) = f(3)$ ✓

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

```
let c = 0; const f = (x) => x + 39 + c++;
```

```
if p     f()
....
if q     c()
...
f (3)
```



12-1

Goal: add mutation

e ::= .... | box e
| unbox e
| set-box! e e

E = ... | box E
| unbox E
| set-box E e
| set-box v E

```
struct box { void *p; }
box (int x) { ip = malloc(int)
              box b = { p = ip }
              *ip = x;
              ret b; }
```

let b = box 6 in
(set-box! b 8;
unbox b) + (unbox b)

b σ_0

(set-box! (box 6) 8;
unbox (box 6)) + (unbox (box 6))

$\emptyset[\sigma_0 \mapsto 6]$ (set-box! σ_0 8;
unbox σ_0) + (unbox σ_0)

$\Rightarrow \emptyset[\sigma_0 \mapsto 8]$ (unbox σ_0 + ub σ_0) $\Rightarrow \emptyset[\sigma_0 \mapsto 8]$ / $(8 + 8) = 16$

12-2/ small step : $e \to e$

$\Sigma \times e \to \Sigma \times e$

$(M, S, pc) \to (M', S', pc')$

Σ = store (memory / heap)

ptrs $\Rightarrow$ vals

$\sigma \Rightarrow v$

$\Sigma / E[\text{if } T \text{ et } ef]$

inject $e = \emptyset / e$ $\quad \to \Sigma / E[et]$

extract $\Sigma / v = v$

$v ::= \ldots \mid \sigma$

$\Sigma / E[\text{box } v] \to \Sigma[\sigma \mapsto v] / E[\sigma]$

where σ d.n.o.i. Σ (σ = malloc)

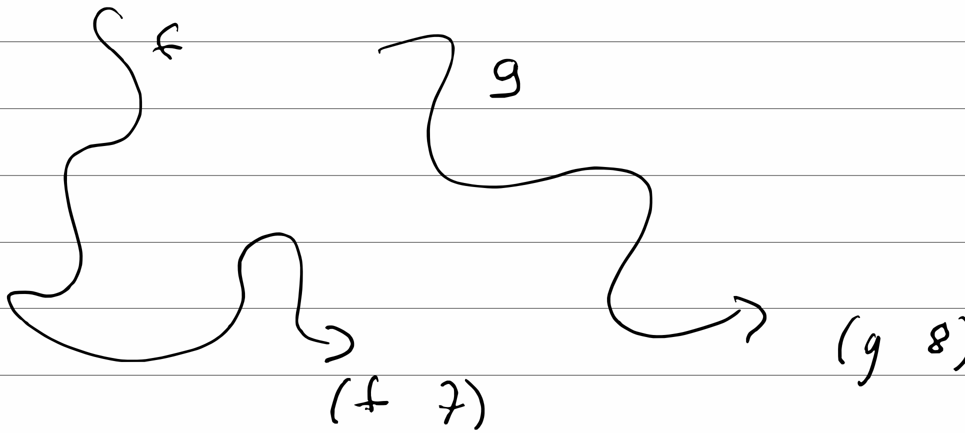
$\Sigma / E[\text{unbox } \sigma] \to \Sigma / E[\Sigma(\sigma)]$

$\Sigma / E[\text{set-box! } \sigma\ v] \to \Sigma[\sigma \to v] / E[v]$ C

$E[\text{unit}]$ void

12-3

let b = box 0 in
let f = λx. set-box! b (x+1);
x · 2 in
let g = λy. y + unbox b in



CEK$_3$ ↙ CESK$_0$ + store

CEK$_3$ ⟹ CEK$_4$

v = ... | ptr n
p = ... | box
| unbox
| set-box

⟨v, env, sto, kapp(clo(λx.e, env'), k)⟩
↦ ⟨e, env'[x ↦ v], sto, k⟩

⟨if ec et ef, env, ~~sto~~ sto, k⟩
↦ ⟨ec, env, sto, kif(env, et, ef, k)⟩

⟨box e, env, sto, k⟩
↦ ⟨e, env, sto, kbox(k)⟩

⟨v, env, sto, kbox(k)⟩
↦ ⟨σ, env, sto[σ ↦ v], k⟩
where σ = malloc(sto)

12-4/ $v = \ldots \mid \text{pair } v \;\; v$
$\quad\quad\quad\quad \mid \text{box } \sigma$

Question: should we add

set-fst : pair A B × A → ()

set-snd : pair A B × B → ()

mpair a b = pair (box a) (box b)

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

mpair-fst p = unbox (fst p)

---

let f xi =
let x = box xi in
set! x 5;
x in

let v = (box 7) in
set! v 8
v + v

e ::= .... | set! x e

(1) (2)

12-5/ Always store vars as pointers

OLD:

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

NEW:

$$\Sigma \,/\, E[(\lambda x.\, e)\ v] \to \Sigma[\sigma \mapsto v] \,/\, E[e[x \leftarrow \text{unbox } \sigma]]$$

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

$$\Sigma \,/\, E[\text{set!}\ (\text{unbox } \sigma)\ v] \to \Sigma[\sigma \mapsto v] \,/\, E[v]$$

desugar $(e_1 ; e_2)$ =
  let _nggmncihannim = e_1 in
    e_2

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

12-6/ Fancy desugar

desugar ($\lambda x.$ x

$\quad$ set! x $\underbrace{}_{}^{e}$

$\quad$ . . .)

$= \lambda x_i$. let x = box x_i in

$\quad$ . . . . (unbox x)

$\quad$ . . . . set-box! x e

$\quad$ . . .

desugar M ($\lambda x. e$) =

$\quad$ if $\underline{\text{modified } x \ e}$ ${}^{x \notin M}$ then

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

o.w.

$\quad\quad \lambda x$. desugar M e

desugar M x =

$\quad$ if $x \in M$ then unbox x

$\quad\quad$ o.w. x

12-7

eval e

=

$$\widehat{\text{eval}}\ (\text{let } \overbrace{\ldots\ldots}^{\text{stdlib}} \text{ in } e)$$

→ prints 11

→ compile

→ runs

desugar (map) = λf. . . . . . .)

15-1/ 1 / 0 — δ is undefined (partial)

(5 1) — fun app needs a λ or p

set-box! 7 2 — set-box needs a box

$x \to y \to z \to (\div\ 1\ 0) \not\to$

$eval(p) = v$ iff $p \to^* v$

partial

$v = \ldots \mid$ bad bad bad

$E[(v_0\ v \ldots)] \to$ bad bad bad

where $v_0 \notin p,\ \notin \lambda \ldots$

$v = \ldots \mid err_1 \mid \ldots \mid err_{255} \mid ok$

$E[(p\ v \ldots)] \to err_{17}$

$\delta(p, \vec{v}) = \perp$

$E[a] \to E[b]$

$e = \ldots \mid$ abort e

$E' = E$

$E[$abort $e] \to e$

$\langle$abort e, env, $k\rangle \longmapsto \langle e$, env, kret$\rangle$

J9:

15-2 $e = \ldots \mid \text{throw } e$
$\mid \text{try } e \text{ with catch } e$

try (+ 1 (throw 2))
with catch (λx. (- x 1)) $\Rightarrow$ 1

$E = \ldots \mid \text{try } e \text{ with catch } E$
$\mid \text{try } E \text{ with catch } v$

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

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

try (+ 1
(try (+ 2 (throw 3))
with catch (λ (v) (~~*~~ v 2))))
with catch (λ (x) (* x 3)) $\Rightarrow$ 7

15-3/ $K = \ldots \mid \text{preTryK} \; e \; env \; k$

$\mid \text{TryK} \; v \; k$

$$\langle \text{try } e_b \text{ with catch } e_h, \; env, \; k \rangle$$
$$\mapsto \langle e_h, \; env, \; \text{preTryK} \; e_b \; env \; k \rangle$$
$$\langle v_h, \; \text{—}, \; \text{preTryK} \; e_b \; env \; k \rangle$$
$$\mapsto \langle e_b, \; env, \; \text{TryK} \; v_h \; k \rangle$$
$$\langle v_{ans}, \; \text{—}, \; \text{TryK} \; vh \; k \rangle$$
$$\mapsto \langle v_{ans}, \; \text{—}, \; k \rangle$$
$$\langle \text{throw } e, \; env, \; \text{TryK} \; vh \; k \rangle$$
$$\mapsto \langle vh \; e, \; env, \; k \rangle$$
$$\mapsto \langle e, \; env, \; \text{kapp} \; (vh) \; \text{—} \; () \; k \rangle$$
$$\langle \text{throw } e, \; env, \; \text{kapp} \; (v \ldots) \; env' \; (e \ldots) \; k \rangle$$
$$\mapsto \langle \text{throw } e, \; env, \; k \rangle$$
$$\text{preTryK} \; e \; env' \; k$$

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

15-4/

```
(  a  b  c)  →
     (let ([av  a])
      (if (function? av)
        (if (= (function-arity av) 2)
             unsafe-apply
            (av  b  c)
            (throw "wrong num args"))
       (throw "not fun"))

     (+  2      (throw  o) )
            L  [(throw  e)]
  try  ......
   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

$e = \ldots \mid \text{callcc } e$

$E = \ldots \mid \text{callcc } E$

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

cek $\quad v = \ldots \mid \text{kont } k$

$\quad k = \ldots \mid \text{kcallcc } k$

$\langle v, \_, \text{kcallcc } k \rangle$

$\quad \mapsto \langle v\ (\overset{\text{kont}}{k}), \_, k \rangle$

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

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

$\langle v, \_, \text{kapp } (\text{kont } k)\ \_\ ()\ \_ \rangle$

$\quad \mapsto \langle v, \_, k \rangle$

15-6

```
f = (λ (x)
      (callcc (λ (return)
        (if (zero? x)
            (return 2)) ;

        (print x)
        (/ 2 x)))))
```

```
int f(int x){
  if (x == 0)
    return 2;
  printf("%d\n", x)
  return 2/x;}
```

```
(+ 1 (f 7))
(+ 1 (f 0))
```

return = λx. abort (+ 1 x)

```
(λ (x ...) b)  =>  (λ (x ...)
                     (callcc (λ (return)
                       b)))
```

15-7

```
(define last-handler
   (box (λ (x) (abort x))))
(define throw
   (λ (v) ((unbox last-handler) v)))
desugar (try e1 with catch e2)
   = (try-catch* (λ () e1) e2)

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

17-1 (f10) (5 3)

unsafe — just do something

.... (f x)

$((f10*) f) \rightarrow \underbrace{code\_ptr}(x)$

↓

jump (f+8)

(define (f x) (f (scanf))

(x 13))

unsafe = the language doesn't protect its abstractions

safe = DOES

C no abs, safe

C++ *(scanf()) (void*) 0 [2]

Java

JS intend to be safe

Py but loopholes

Racket

17-2

```
(define L
  (load "libOpenGL"))        → dlload
(define glDraw
  (extract L "glDraw"))      → dlsym
(glDraw .....)
```

desugar (define (f x ...) body) ; more
⇒
(let ([f (λ f (x ...) body)])
  (desugar more)

Assume we want safety

unsafe kernel (vm)	safe kernel	un
safe program	un	k n
unsafe compiler	un	safe

safe kernel

(f x) ⇒ if (obj_tag(f) == CLO) {
  ((clo*) f) -> code_ptr (x) ;
} else { error }

safe program ⇒ if (function? f) {
  (f x) }
} else { error }

17-3] p = ... ~~+~~ ... unsafe+

```
stdlib =
  ....
  (define (+ x y)
    (if (and (number? x)
             (number? y))
        (unsafe+ x y)
        (error)))
```

```
desugar (f x) =
  (if (fun? f) (if (= (arity f) 1)
                   (f x)
      error )      error )
```

```
p = .... unsafe-apply
apply f (list x y z) = (f x y z)

desugar (f x) = safe-apply f (list x)
```

17-4/ safety violation:

- what we wanted — ctc
- what we got — val
- who gave] — blame — pos
- who got — neg

```
(protect ctc val pos neg)

(+
 (protect num? v p n) 7)
 => (if (num? v) v
        (error "expected num, got " v "from
               pos at neg"))

desugar (+ x y) =>
   (unsafe+ (protect num? x "line 27" "stdlib")
           ... )
```

17-5/ (define (map f l)
  (if (empty? l)
      empty
      (cons (f (first l))
            (map f (rest l)))))

map : (Num -> Str) x (List Num) -> (List Str)

protect ((listof p) v pos neg =>
  checkall p v pos neg

protect (Num -> Str) f pos neg => function proxy
  λ x.
  protect Str (f (protect x Num neg pos))
      pos neg

18-1 Macro Systems

Word Macros

C Macros

Excel Macros

```
#define DEBUG 1

#define MAX(x, y)
        ((x) > (y) ? (x) : (y))

#define MAX(z, x, y)
  do {
     z = (x) > (y) ? (x) : (y); }
  while (0) ; ;;
```

F2 ⇒ Qx721

```
(let  xt = (x)
      yt = (y)
  xt > yt ? xt : yt)
```

C macros are textual not, expression-oriented

```
MAX( 1 ? 2 : 3 ,  .... )
MAX( a++ ,  .... )
MAX( 7 ,  xt )
```

purely substitutional

```
int db[32] = { ∀i : f(i) }
```

known at compile

```
{ F(0), F(1), F(2) ...

#define F(x)   (x) * 2 + 1
```

18-2/ The language kernel should be simple
and flexible ...
features should be added on top of
old if possible

call/cc ⇒ generator, nondet, threads, try/catch
set-box! ⇒ set! and arbitrary ds mut
λ ⇒ let
(let x = e in b)
⇒ ((λ (x) b) e)

Great languages have big desugarers

```
(define-desugar-rules
  [(let ([x xe]) be)
   ((λ (x) be) xe)]
  [(let () be)
   be] )
```

pattern

template

syntax

18-3) (define-~~desugar~~-rules
[(define-desugar-rule pat tem)
(define-desugar-rules
[pat tem])])

dsrs : id × List(pair(pat, template))

let := "let", [< (let ([x xe]) be),
((λ (x) be) xe) >]

pattern-match : pat × se → env
transcribe : tem × env → se

pm (let ([x xe]) be) (let ([foo (+ 1 2)])
(+ foo foo))

= [$x \mapsto$ foo, $xe \mapsto$ (+ 1 2)
$be \mapsto$ (+ foo foo)]

!

tr ((λ (x) be) xe) =
((λ (foo) (+ foo foo)) (+ 1 2))

18-4/ pm '() '() = ∅

pm (cons pa pd) (cons ia id) =

pm pa ia ⊎ pm pd id

pm var(x) in = [x ↦ in]

pm const(n) const(n) = ∅

tr '() env = '()

tr (cons ta td) env = (cons tr(ta, env)
tr(td, env))

tr var(x) env = env[x]

tr const(n) env = n

18-5/ (let* ([x * 1]
[y * 2]
[z * x + y]
[u * z + 3])
~~m~~
u)
=> (let ([x 1])
(let ([y 2])
(let ([z (+ x y)])
(let ([u (+ z 3)])
u))))

dsrs

① (let* () be) => be

② (let* ([x_0 xe_0] more ...) be) =>
(let ([x_0 xe_0]) (let* (more ...) be))

pm ② (let* ↑) =
[$x_0 \mapsto x$, $xe_0 \mapsto 1$, be $\mapsto$ u
more $\mapsto$ MANY([y 2], [z (+ x y)], [u (+ z 3)])

tr (list tmp '...) env = map (tr tmp) (pulloutmany env)

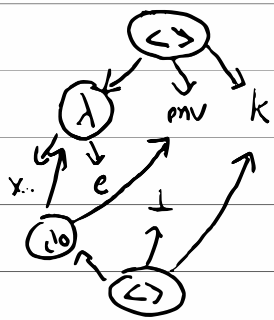
pm (list pat '...) in =
mergeintomany (map (pm pat) in)

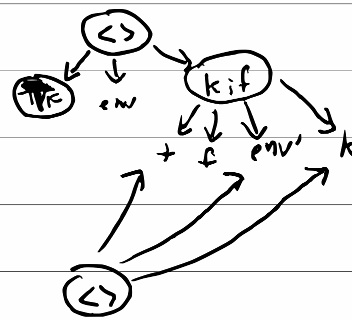
```
                [x ↦ a++ , y ↦ 7]

18-b/  dsp  (or  x  y)  =>
     (let ([tmp  x])
       (if tmp  tmp          (if x  x
          y))                    y)

        a--
(or  a--  7)

(let ([tmp  y])              (let ([tmp  y])
  (or #false  tmp))  =>        (let [(tmp FALSE)]
                                 (if tmp tmp tmp)))
```

19-1] Memory Management

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

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





19-2)

$$e = num \mid (op \; e \; e)$$

$$op = + \mid - \mid \times \mid \div$$

$$k = ret \mid L(op \; k \; e) \mid R(op \; num \; k)$$

$$\langle \text{app}(op, e_L, e_R), k \rangle \mapsto \langle e_L, L(op, e_R, k) \rangle$$

$$\langle num, L(op, e_R, k) \rangle \mapsto \langle e_R, R(op, num, k) \rangle$$

$$\langle num_R, R(op, num_L, k) \rangle \mapsto \langle \delta(op, num_L, num_R), k \rangle$$

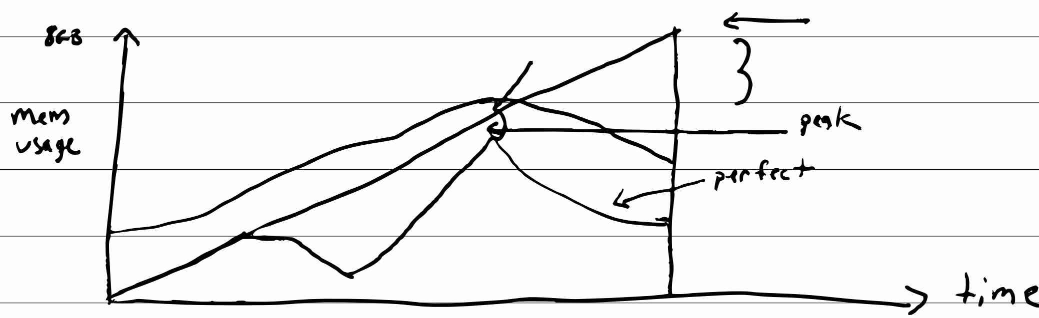
19-3/ What should a MM do?

- know when to call free()

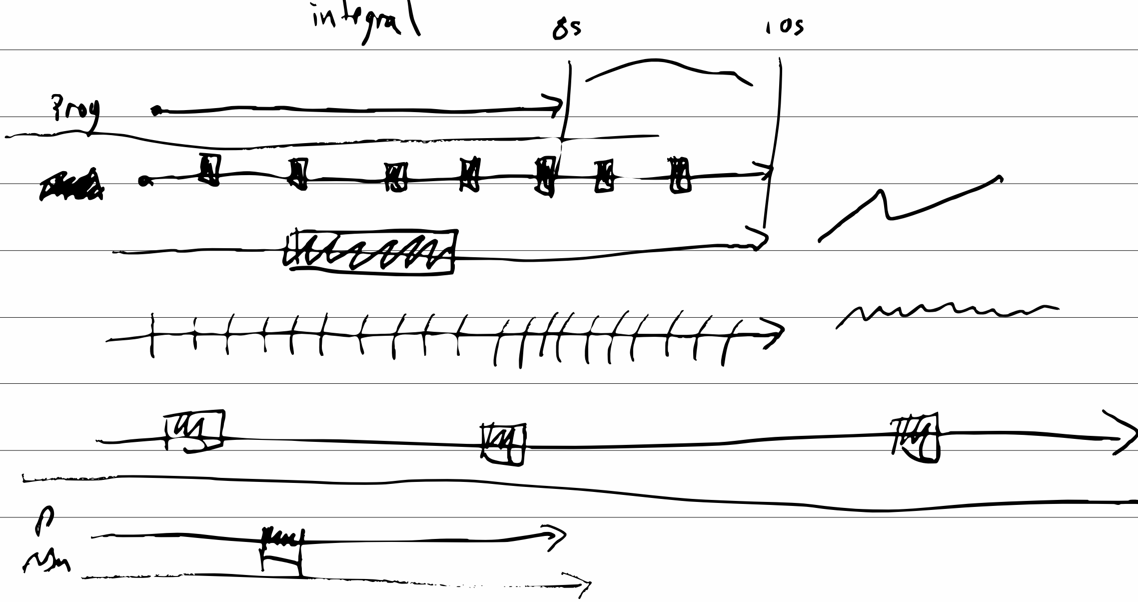
- wait to free to end (never free in between)
- free-ing ~~[illegible]~~ "active" objects

$f(x) = a$ but $= b$

Soundness = MM preserves same answer



peak overhead (min, median, avg, max)
integral



19-4/ How do you get sound MM in C?

unsound comes from aliases

one pointer   two vars/fields

```
f() {
  char * c = ... ;

  return; }
```

free c

p

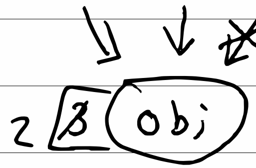
g (p)   x = p

free(p)   free(p)

- always pass ownership
- always make copies

(Alias)

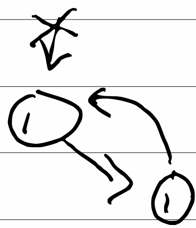
19-5 / Reference Counting / Smart Pointers




```
mkref(p) :=
    p.count++

rmref(p) :=
if(!--p.count)
    free(p);
```

64-bit counts

8-bit



&p   *p = NULL

fails on cycles (leaks)

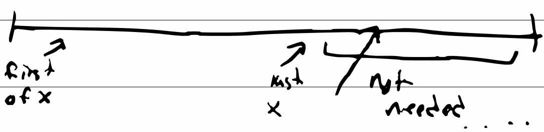
20-1/ When is an obj free()-able?

scope of var X

last use of obj

```
{ int * x = malloc(...)
    ⋮
    g = x
    ⋮
  ret }
```

← x is unbound




```
{ ....
  .....
  while () {
    if (scanf(...))
      .... X .. ..     ← last
  ..... }
  no xs
  .... }
```

```
{ int* x = m()
  ....
  ... x      ← free()?
  if ( f(7) ) {
     use x }
  o.w never use
  ⋮ }
```

Truth (Does mach HALT?) I don't x

vs

Provability (Do I know?) I know I don't x

20-2 | Reachability

Suppose o is an obj in memory

$$
\begin{aligned}
reach(o) \quad \text{iff} \quad & var(o) \\
\vee \quad & ptr(p, o) \wedge reach(p) \\
\vee \quad & field(o', o) \wedge reach(o') \\
\vee \quad & reg(o) \\
\vee \quad & stack(o)
\end{aligned}
$$

$\times \xrightarrow{o.f} \xrightarrow{*} \xrightarrow{[3]} \xrightarrow{.m} \xrightarrow{.x} \xrightarrow{*} O \swarrow$

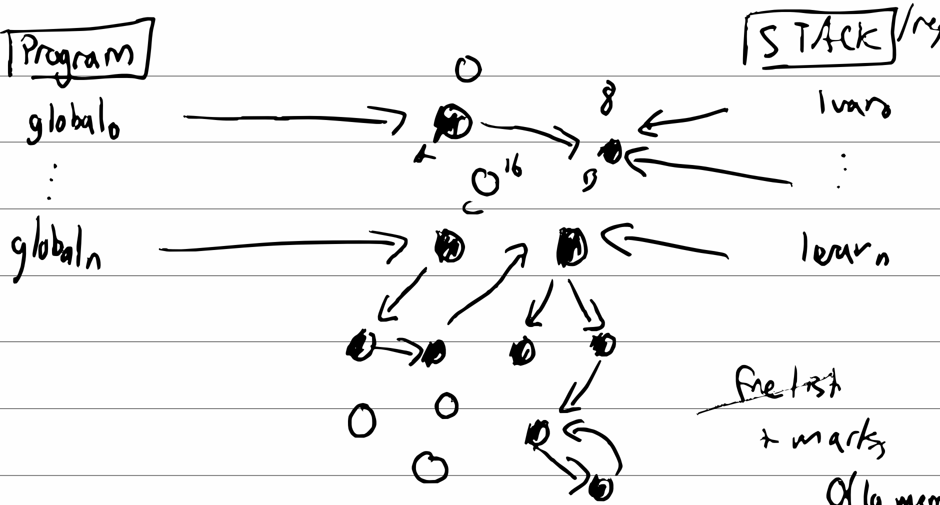
$*((*o.f)[3].m.x)$

$o = NULL$

$\bullet \to \perp \; X \to \; \to \; \to$

unreachable objects may be freed

20-3/



~~free list~~
+ marks
O(lg mem)

A B C D E F

O(1)	malloc		O(n lg n)
	free	—	
O(live)	gc	O(live)	+ O(mem)
		mark	sweep

Constraints:

- know size of objs
- all known pointers from malloc
- know obj layout

tricks:

- mark in tag

SB;B$_0$P

John McCarthy 1969
LISP

21-1

htdp.org

BSL	Py	Jav
ISL		
ASL		

count * prize = sum;

not an l-value
cannot cast int to int*

Mark and Sweep

Time — malloc — O(n lg n)
free — X
gc — O(live) + O(mem)
Space — overhead — mark bits = O(lg mem)
latency — could do tri-color — arb small pause

Stop and Copy

Time — malloc — O(1)
free — X
gc — O(live)
Space — overhead — x2
latency — long

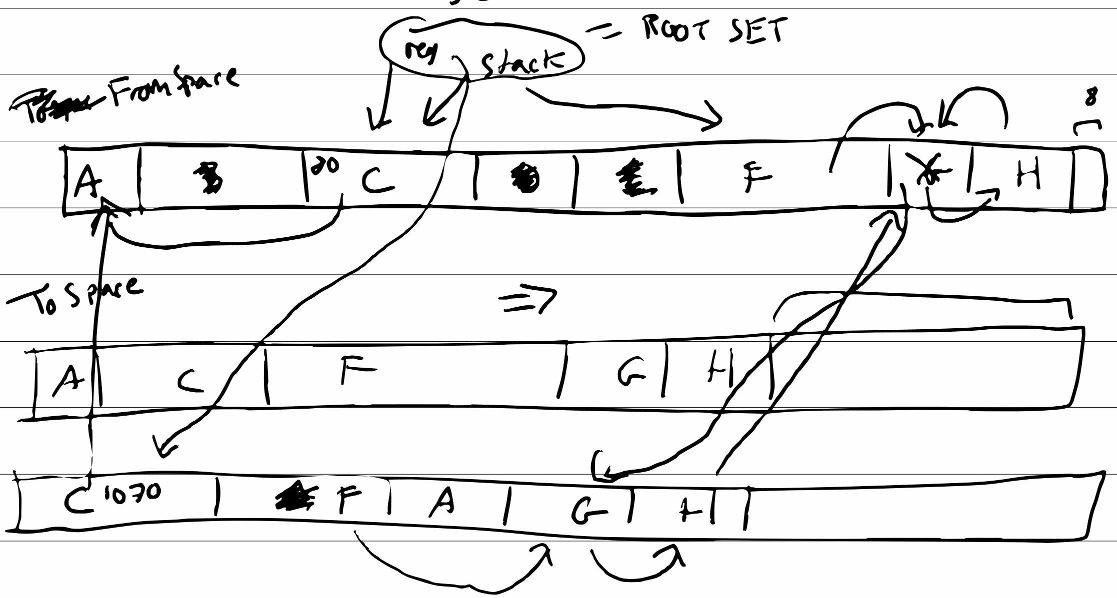
$O(n \lg n)$ | $O(1)$

21-2 malloc | searching a tree | free ptr

bool sectry? | (bump Ptr)

```
malloc(size_t  sz) {
    if (free_p + sz  <  last_free_spot){
        free_p += sz;
        ret  freep - sz;  }
    else {   gc();   malloc(sz); } }
```

gc(← sz) ; malloc(sz, ← true)



forwarding pointer: new tag contains a pointer

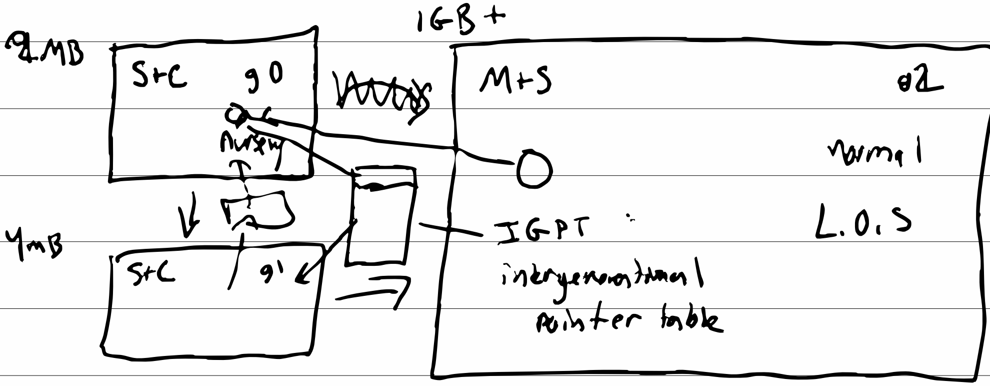
22-1 | M&S – $O(\lg n)$ malloc $O(\text{live}) + O(n)$

$O(\lg n)$ overhead time

S&C – $O(1)$ malloc $O(\text{live})$

~~O~~ x2 overhead time

Generational Collection



Generational Hypothesis

Hypothesis: Objects live a very long time or a very short time

"Most objects die young"

write barrier

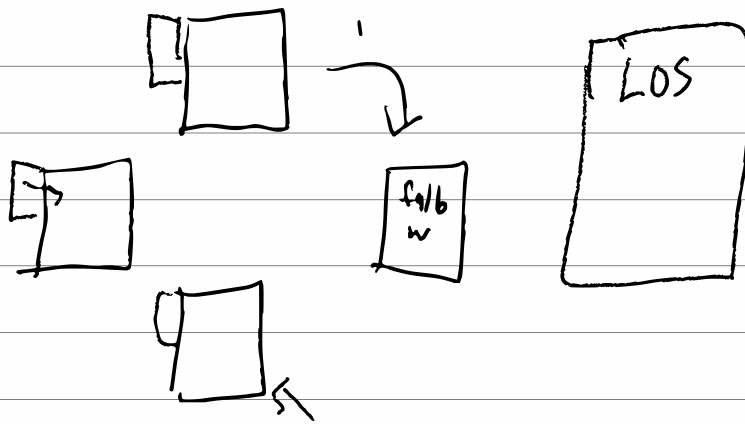
Large object exception $o.f = n$ then add it to a table IGPT

old new

22-2/ Radioactive Decay

"half life"

Pick any number $1/N$ $(N=4)$



22-3 / Type Systems

syntactic — return 1 + ;

logic — return 41;

partial fun — 1 / 0;

first (NULL)

type errors — "four" + 1;

⇒ "five" (PHP)

---

types provide safety

unsafe kernel < safe kernel

safe compiler (inserts checks)

V

unsafe k

unsafe compile

safe type system

```
x = read_a_bool
o = NULL
if f(x):
    o = new Cat
else
    o = new Dog
... (o and x unrelated
if x:
    o.purr()
else:
    o.bark()
```

(X ⇒ CAT)

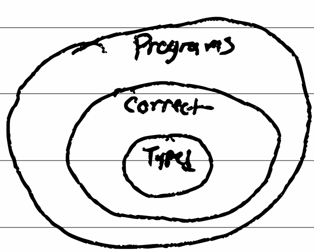
∧ (¬X ⇒ DOG)

occurrence typing

/ Typed Racket

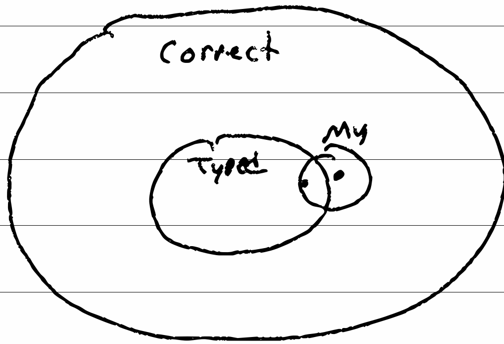
TypeScript

Hack



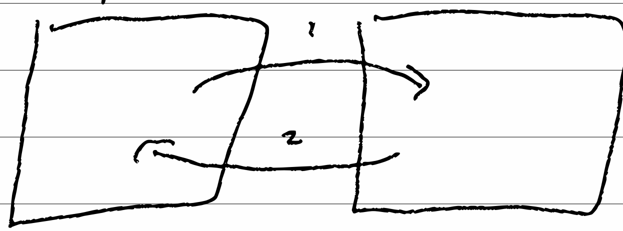
Coq

23-1/ Gradual Typing



Typed Racket
TypeScript
Hack
F#?
Reticulated Python
CoffeeScript

My program



1. T ⇒ UT : Always safe
2. UT ⇒ T : Unsafe

(typed e) ⟹ e

(untyped e) ⟹ (contract e supposed-to-be 'untyped 'typed)

(typed (λ(x). (untyped (+ x 1)))) → ok

(λ(x) (typed (+ x 1))) → illegal

(λ (x) (typed (+ (untyped x) 1))) ⇒ (λx. (+ (contract x num?) 1))

23-2 Type Systems make predictions

Theory – prediction

Model – real phenomenon

"$X \vdash P$" "X says P"

"Gravity $\vdash$ the pen will drop" – Theory stmt

"Univers $\vdash$ the pen will drop" – Model stmt/expr

○ $\forall P.$ Theory $\vdash P \Rightarrow$ Model $\vdash P$ soundness

X $\forall P.$ Model $\vdash P \Rightarrow$ Theory $\vdash P$ completeness

23-3/ $\forall A.\quad \underbrace{\text{Theory} \vdash P}_{} \Rightarrow \underbrace{\text{Model} \vdash P}_{}$

$$e \quad : \quad T$$

$$\begin{tikzcd}
e \arrow[r, Rightarrow] \arrow[d] & \Rightarrow\Rightarrow\Rightarrow e' \Rightarrow\Rightarrow\Rightarrow\Rightarrow v \arrow[d] \\
e \arrow[r, Rightarrow, "*"] & v
\end{tikzcd}$$

$e = v \mid (+\ e\ e) \mid (<\ e\ e) \mid (\text{not}\ e)$

$v = \text{true} \mid \text{false} \mid N \ \ldots$

$T = \text{Bool} \mid \text{Nat}$ "$\vdash e : T$"

$\vdash \text{true} : \text{Bool}$ $\qquad \vdash \text{false} : \text{Bool}$

$\vdash N : \text{Nat}$ $\qquad \dfrac{\vdash e_1 : \text{Bool}}{\vdash (\text{not}\ e_1) : \text{Bool}}$

$$\frac{\vdash e_1 : \text{Nat} \qquad \vdash e_2 : \text{Nat}}{\vdash (+\ e_1\ e_2) : \text{Nat}} \qquad \frac{\vdash e_1 : \text{Nat} \qquad \vdash e_2 : \text{Nat}}{\vdash (<\ e_1\ e_2) : \text{Bool}}$$

$$\dfrac{\dfrac{\dfrac{\overline{\vdash 1 : \text{Nat}} \quad \overline{\vdash 1 : \text{Nat}}}{\vdash (+\ 1\ 1) : \text{Nat}} \qquad \overline{\vdash 1 : \text{Nat}}}{\vdash (<\ (+\ 1\ 1)\ \ 1) : \text{Bool}}}{\vdash (\text{not}\ (<\ (+\ 1\ 1)\ \ 1)) : \text{Bool}}$$

$$\frac{\vDash A}{\vDash A \vee B} \qquad \frac{\vDash B}{\vDash A \vee B}$$

$$\frac{\vDash A \quad \vDash B}{\vDash A \wedge B}$$

23-4

$$\forall e, T.\quad \vdash e : T \implies \exists v.\ e \Rightarrow^* v \quad \vdash v : T$$

Soundness Theorem

$\implies$ Strong Normalization "all programs finish"

$e = \ldots \mid x \mid \text{let } x = e \text{ in } e$

"$\vdash e : T$"

$\vdash x : \_\_$

$$\frac{\vdash e_2[x \leftarrow e_1] : T}{\vdash \text{let } x = e_1 \text{ in } e_2 : T}$$

"$\Gamma \vdash e : T$" $\quad x \to T$

$$\frac{\Gamma(x) = T}{\Gamma \vdash x : T}$$

$$\frac{\Gamma \vdash e_1 : T_1 \qquad \Gamma[x \mapsto T_1] \vdash e_2 : T_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : T_2}$$

$e = \ldots \mid e\ e \mid \lambda \cancel{x}.\ e \mid \lambda x{:}T.\ e$

$T = \ldots \mid T \to T$

$$\frac{\Gamma[x \mapsto T_1] \vdash e : T_2}{\Gamma \vdash \lambda x{:}T_1.\ e : T_1 \to T_2}$$

$$\frac{\Gamma \vdash e_1 : T_1 \to T_2 \qquad \Gamma \vdash e_2 : T_1}{\Gamma \vdash (e_1\ e_2) : T_2}$$

$$\frac{\Gamma[x \mapsto T_1] \vdash e : T_2}{\Gamma \vdash \lambda x.\ e : T_1 \to T_2}$$

= (Var → Type)

23-5/ typeof : Gamma → Expr → Type

typeof g (Bool b) = TBool
typeof g (Num n) = TNum
typeof g (Add l r) =
    if (typeof g l) == TNum
       ∧ (typeof g r) == TNum
    then TNum
    ow error
typeof g (Var x) = g x
typeof g (App e₁ e₂) =
  case (typeof g e₁) of
    TArrow dom rng ⇒
      if typeof g e₂ == dom then
         rng
      ow error
    _ ⇒ error
typeof g (Lam x t₁ e) = TArrow t₁ t₂
  ~~where t₂ = typeof g' e~~
        ~~g' = g \`ext'\` x t₁~~
= TArrow t₁ (typeof (ext g x t₁) e))

23-6/ Progress:

$$\forall e, T, \Gamma.\quad \Gamma \vdash e : T$$
$$\to \exists e',\quad e \Rightarrow e'$$
$$\text{or}\quad e \in V$$

Preservation

$$\forall e, T, \Gamma, e'.\quad \Gamma \vdash e : T \ \wedge\ e \Rightarrow e'$$
$$\to \Gamma \vdash e' : T$$

$$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x) \Rightarrow \text{itself}$$

$$\frac{\dfrac{\dfrac{x : T_x \to T_x \qquad x : T_x}{[x{:}T_x] \vdash x\ x : T_r}}{(\lambda x{:}T_x.\ x\ x) : T_x \to T_r} \qquad (\lambda y{:}T_y.\ y\ y) : T_x}{(\lambda x{:}T_x.\ x\ x)\ (\lambda y{:}T_y.\ y\ y)}$$

$(T = Bool \mid Nat \mid T \to T)$

$$T_x = T_x \to T_x$$

no

25-1/ struct inl_t { obj tag; obj* v; }

inr_t { obj tag; obj* v; }

TAG = { ... INL ... INR } ...

case e_0 of [inl x_1 => e_1]
[inr x_2 => e_2]

===

if ($e_0^{\text{tag()}}$ == INL)
run e_1 with $x_1 \mapsto e_0 \to v$
o.w. run e_2 with $x_2 \mapsto e_0 \to v$

(inl e) = pair T e
(inr e) = pair F e
~~case~~ case s f_L f_R = if fst s then
f_L (snd s)
o.w. f_R (snd s)

25-2 $e = x \mid (e\ e) \mid v$

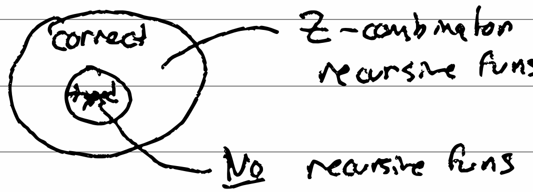
$v = \lambda x{:}T.\ e \mid p \mid c$

$T = T \to T \mid B$

B = "base types" one for every kind of c

num $\in C \to$ Num $\in B$

$\Delta(p) = T$ $\qquad \Delta(+) = N \to N \to N$



$v = \ \ldots \mid$ ~~λ f(x:T). e~~

$\mid \lambda Q\, f(x{:}T).\ e$

$\vdash : (\Gamma, e)$

$\Gamma \vdash e :$

$$\frac{\Gamma[x \mapsto T][f \mapsto T \to Q] \vdash e : Q}{\Gamma \vdash \lambda f(x{:}T).\ e : T \to Q}$$

"_ proves _ has type"

"g prove x has s type"

25-3

true := $\lambda x. \lambda y. x$

false := $\lambda x. \lambda y. y$

if := $\lambda c. \lambda x. \lambda y.\ c\ x\ y$

$\lambda f(x : \_\_).\ \lambda g(y : \_\_).\ x$

if $\in L_2$

$e = \ldots \mid \text{if } e\ e\ e$ $\quad L_3 = L_2 \cup \{\text{if}\}$

$$\frac{\Gamma \vdash e_c : \text{Bool} \quad \Gamma \vdash e_t : T_1 \quad \Gamma \vdash e_f : T_2}{\Gamma \vdash \text{if } e_c\ e_t\ e_f :\ ?}$$

T_1

T_2

$\boxed{T_1 \cup T_2}$ — Typed Racket

$T_1 \cap T_2$

$T_1 = T_2 \Rightarrow T_1$

~~number? : Any → Bool~~

$y = \text{if } \cancel{\ldots}\ x > 0 \text{ then ``four'' o.w. } 4$

$\{x \text{ is a num}\}$

$\Gamma \vdash e : T\ ;\ P_T\ ;\ P_F$

$$\frac{\begin{array}{c}\Gamma \vdash e_c : \text{Bool}\ ;\ P_{TC}\ ;\ P_{FC} \\ \Gamma \cup P_{TC} \vdash e_t : T_1\ ;\ P_{TT}\ ;\ P_{FT} \\ \Gamma \cup P_{FC} \vdash e_f : T_2\ ;\ P_{TF}\ ;\ P_{FF} \\ P_T = P_{TT} \cap P_{TF} \quad P_F = P_{FT} \cap P_{FF}\end{array}}{\Gamma \vdash \text{if } e_c\ e_t\ e_f : T_1 \cup T_2\ ;\ P_T\ ;\ P_F}$$

25-4/ Pairs

$$e = \ldots \mid \text{pair } e_1\ e_2 \mid \text{fst } e \mid \text{snd } e$$

$$\tau = \ldots \mid \tau \times \tau$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \text{pair } e_1\ e_2 : \tau_1 \times \tau_2}$$

$$e = \ldots \mid \text{inl } e \mid \text{inr } e \mid \text{case } \ldots$$

$$\tau = \ldots \mid \tau + \tau$$

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \text{inl}_{\tau_2}\ e : \tau_1 + \tau_2} \qquad \frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \text{inr } \tau_1\ e : \tau_1 + \tau_2}$$

$$e = \ldots \mid \text{box } e \mid \text{unbox } e \mid \text{setbox } e\ e$$

$$\tau = \ldots \mid \text{Box}(\tau)$$

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \text{box } e : \text{Box}(\tau)} \qquad \frac{\Gamma \vdash e : \text{Box}(\tau)}{\Gamma \vdash \text{unbox } e : \tau} \qquad \frac{\Gamma \vdash e_1 : \text{Box}(\tau) \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{setbox } e_1\ e_2 : V}$$

25-5/

```
template <class X>
size_t f( X x) {
  return 2 * sizeof(x); }
```

ArrayList<int>
ArrayList<Dog>

```
f<int>(5)
f<Dog>(Mickey)
```

$e = \dots \mid \Lambda X.\, e \mid e\langle T\rangle$

$T = \dots \mid \forall X.\, T \mid X$

untyped

$\lambda x.\, x$

typed

$\Lambda\alpha.\, \lambda\, f(x:\alpha).\, x = id$

$: \forall\alpha.\, \alpha \Rightarrow \alpha$

$$\frac{\Gamma, X \vdash e : T}{\Gamma \vdash \Lambda X.\, e : \forall X.\, T} \qquad \frac{\Gamma \vdash e : \forall X.\, T_2}{\Gamma \vdash e\langle T_1\rangle : T_2[X \leftarrow T_1]}$$

$\underbrace{id\langle num\rangle}_{num \to num}(5) : num$

Java: pretend don't exist
X = Object

Fw

$\Lambda X.\, e \implies$

$e\langle T\rangle \implies$

C++:

$C[(\Lambda X.\, e)\langle T\rangle] \to e[X \leftarrow T]$

26-1/ (define length
(λ rlen (l)
(case l of
[inl x => 0]
[inr y => (+ 1 (rlen (snd y)))]])))

true := λx. λy. x   false := λx. λy. y

samp env (c (λz. 5) (λz. 7)) 0

inl := λ x. pair true x
inr := λy. pair false y
$\hat{\text{case}}$:= λ s. λ lc. λ rc.
let nc := if (~~fst~~ s) then lc else rc in
nc (snd s)

(case S of [inr x => re]
[inl y => le]) =>

($\hat{\text{case}}$ s (λx. re) (λy. le))

2~~8~~-2/

(v-kont k)

$$\langle \text{call/cc } e,\ env,\ k\rangle \mapsto \langle e,\ env,\ \text{kcallcc } k\rangle$$

$$\langle v,\ env,\ \text{kcallcc } k\rangle \mapsto \langle \underbrace{v,\ k}_{\text{app}(v,\ [k])},\ env,\ k\rangle$$

$$\langle v_0,\ \text{—},\ \text{kapp}(\ (k'),\ \text{—},\ (),\ k)\rangle$$

$$\mapsto \langle v_0,\ \text{—},\ k'\rangle$$

$$\langle \text{call/cc } e,\ env,\ k\rangle \mapsto \langle e\ k,\ env,\ k\rangle$$

26-3

$T ::= \ldots \mid X \mid \forall X.\, T$

$e ::= \ldots \mid e\langle T\rangle$

$\mid \Lambda X.\, e$

C++ Java

Haskell ML

$(\text{if } c\ \Lambda X.\, e_1$

$\Lambda Y.\, e_2)\langle T_1\rangle$

$id = \Lambda X.\, \lambda a{:}X.\, a \quad : \forall X.\, X \to X$

$(id\langle int\rangle == id\langle bool\rangle) \Rightarrow$ false / true ?

functional extensionality

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

$f : \forall X.\, X \to X$

$f := \Lambda X.\, \lambda a{:}X.\, a$

add1 : Num → Num

add1 := $\lambda n.\, 1 + n$

$\Lambda X.\, \lambda a{:}X.\, rf\ a$

~~let f = λ...~~

$\Lambda X.\, \lambda (a{:}X).$ let $r := \lambda\, r().\, r(\text{add}\,h)$

in $r\ 7$

$\Lambda X.\, \lambda a{:}X.$ let $n = 1+2$ in a

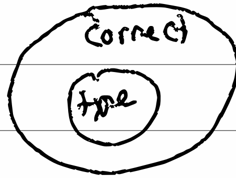
26-4)

```
template <class A>
  A id (A a) {
    if (dynamic_cast <dog> (a)){
      return ((dog) a). mate ; }
    else {
      return a; }
template <class Cat>
Cat id (Cat a) {
  return Garfield; }
```

map : $\forall A. \forall B.\ (A \to B) \to List(A)$
$\to List(B)$

parametric polymorphism

26-5




```
distFromOrigin ( posn p) {
    √(p.x^2 - p.y^2); }
~~shift~~
posn2d ( int x , int y ) {
    { x = x ; y = y }; }
posn3d ( x , y , z ) {
    { x = x ; y = y ; z = z ; } ; }
distFromOrigin( posn3d( 1 , 2 , 3 ) )
                posn2d( 1 , 2 )
```

$T ::= \ldots \mid \{ f : T , \ldots \}$

$e ::= \ldots \mid \{ f = e , \ldots \} \mid e.f$

$posn = \{ x : int , y : int \}$

$\{ x : int , y : int , z : int \}$

$T_1 = T_2$

$$\frac{T_1 = T_2[Y \leftarrow X]}{\forall X. T_1 = \forall Y. T_2}$$

$Int = Int$

$$\frac{T_1 = T_3 \qquad T_2 = T_4}{T_1 \to T_2 = T_3 \to T_4}$$

26-6 A typed program shouldn't crash.

$e.f \implies e$ must have an f field

$$\frac{\Gamma \vdash e : \{f_0 : T_0, \ldots, f_i : T_i, \ldots f_n : T_n\}}{\Gamma \vdash e.f_i : T_i}$$

$$\frac{\Gamma \vdash f : X \to R \qquad \Gamma \vdash a : Y}{\Gamma \vdash f\ a : R}$$

$Y <: X$

compatible (ie are subtypes)

"the types match"

$$\frac{}{T <: T}$$

$$\frac{\{f_0, \ldots f_n\} \supseteq \{g_0, \ldots, g_m\}}{\{f_0 : T_0, \ldots, f_n : T_n\} <: \{g_0 : T_0', \ldots, g_m : T_m'\}}$$

$T_0 = T_0'$

$T_0 <: T_0'$

"subtype relation" = $<:$

$$\frac{F <: X \qquad G <: Y}{(F \times G) <: (X \times Y)}$$

26-7

$$\frac{X <: A \qquad B <: Y}{X \Rightarrow Y \;<:\; A \Rightarrow B}$$

Animal ⇒ Animal

f : X ⇒ Y

h(f)

cat ⇏ cat

h (B g(A)) {

w = g(some : A)

w. do a b thing

w.purr()

Liskov - substitution principle

26-8 Java / C++

```
class Posn {
  int x; int y }

class Posn3d {
  int x; int y; int z; }

static int distance (Posn p) {
  p.x ... p.y ... ; }

distance ( new Posn3d (1, 2, 3))
```

$$\frac{X = Y}{X <: Y} \qquad \frac{X \text{ inherits from } Y}{X <: Y}$$

structural subtyping	nominal subtyping
Theory	C++ Java
Haskell Python*	Typed Racket
ML Go Racket*	
Coq Swift C*	

"duck typing"

<u>27-1</u>/ $\overset{\text{vm}}{/\underbrace{\text{interp}}_{\text{Prog}'}}$: Prog → Ans

$\underbrace{\text{compiler}}_{\text{Prog}_0}$: $\text{Prog}_1 \to \text{Prog}_2$

interp : $\text{Prog}_2 \to \text{Ans}$

$\underbrace{\text{cpus}}_{\text{universe}}$: $X86 \to \text{Ans}$

debug : $\text{Prog}_1 \times (\overset{\text{State} \to \text{State}}{\qquad}) \to \text{Ans}$

opt : $\text{Prog}_1 \to \text{Prog}_1$

JIT – "just-in-time compilation"

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

$$\text{where } f = \text{clo}(\lambda x_0 \ldots x_n.\ e,\ env) + (n, \_)$$

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

$$\mapsto\mapsto\mapsto\mapsto\mapsto \langle F(v_0 \ldots v_n), -, k \rangle$$

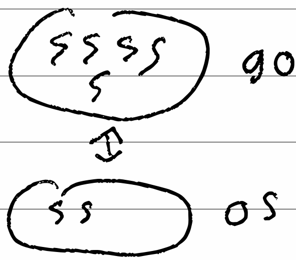
$(\to a) \to \text{thread}$

27-2

Thread — internal concurrency
— modeling + I/O

Futures — $(\to a) \to F\ a$ — fork
$F\ a \to a$ — wait

Places — $(\hat{a} \to \hat{b}) \to \text{place}\ \hat{b}$
$\text{place}\ \hat{b} \to \hat{b}$



$\forall A.\ T$ — forall possible values of A, T is a type

$\exists A.\ T$ — there's some type A (that you don't know,) where this value is
Q $\qquad T[A \leftarrow Q]$

$\text{hide}\ [Q]\ e \Leftarrow e :: T[A \leftarrow Q]$
$:: \exists A.\ T$

$\text{open}\ [A = Q]\ e$

27-3/ Stack inspection

~~eval : P A → A~~

eval in sandbox : P A × Permissions → A

F(a)
G(b)
H(c)
F(a')
...
→

void deleteAllMyStuff () {
  am I in a sandbox?
    does perms contain deleteAll?
      do it
    o.w. error
  o.w do it

kperm (perm , k)

$\langle v, env, kperm(-, k) \rangle$
  $\mapsto \langle v, env, k \rangle$

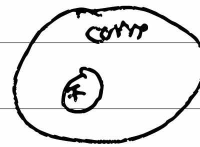
$\langle \text{withperm } p \; e, env, k \rangle$
  $\mapsto \langle e, env, kperm(p, k) \rangle$

$\langle \text{readperm}, env, k \rangle$
  $\mapsto \langle \text{read}(k), env, k \rangle$

27-4

$$f(x)$$
$$\downarrow$$
$$f(\text{int } x)$$
$$\downarrow$$
$$\text{int } f(\text{int } x)$$
$$\downarrow$$
$$\text{auto } f(\text{auto } x)$$
$$\downarrow$$
$$f(x)$$

$$\frac{\Gamma[x \mapsto P] \vdash e : Q}{\Gamma \vdash \lambda x.\, e : P \Rightarrow Q}$$



$$\Gamma \vdash e : \tau \,;\, C \,;\, V$$

$$\frac{\Gamma[x \mapsto X] \vdash e : Q \,;\, C \,;\, V}{\Gamma \vdash \lambda x.\, e : X \Rightarrow Q \,;\, C \,;\, V \cup \{X\}}$$

$T = \text{int} \Rightarrow X$

$V = \{X, Y, Z\}$

$$C = \{\; Y = \text{int},\;\; \text{int} = \text{int},\;\; Z = Y \Rightarrow X,\;\; Z = X \Rightarrow \text{int} \;\}$$

$$T = X \mid B \mid T \Rightarrow T$$

$C = EQ^*$ $\qquad$ $EQ = T = T$

unification

$$T \Rightarrow S = P \Rightarrow Q \implies T = P,\; S = Q$$