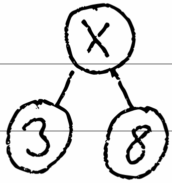
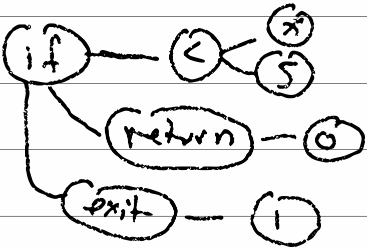
1-1 1 + 1 5 1+

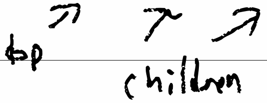
1 x 3 " $\underbrace{3 \times 8}$ "

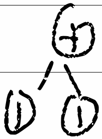


```
if (x < 5) {
  return 0; }
else {
  exit(1); }
```

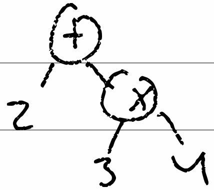


(+ 1 1)





(+ 2 (x 3 4))

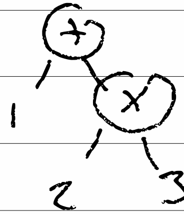


1-2) J_0 ⇒ e := v | (+ e e)
v := number | (* e e)

(+ 1 (+ 2 3)) ∈ 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 () ... {} }
```

(+ 1 (* 2 3)) → Sexpr =



```
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 : J₀ ⇒ string
   ② pp n = itos (n)
   ③ pp (+ eL eR) = "(" ++ pp(eL) ++ " + " ++ pp(eR)
                        ++ ")"
  ④ pp (x eL eR) = "(" ++ pp (eL) ++ " * " ++
                        pp(eR) ++ ")"


 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

$$\text{interp} : e \rightarrow v$$

```
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(); }}
```

$$(+\ 1\ \underbrace{2\ 3)}_{\text{desugar}} \xrightarrow{} (+\ 1\ (+\ 2\ 3))$$

```
se = empty | (cons se se) | string
              pair
(a b c) = (pair "a" (pair "b" (pair "c" mt)))
(+ 1 2) = (p "+" (p "1" (p "2" mt)))
(+ 1 (+ 2 3)) = (p "+" (p "1"
                   (p (p "+" (p "2"
                          (p "3" mt)))
                      mt)))
```

1-5/ desugar for J_0

("-" e) => (* -1 (desugar e))

("-" e₁ e₂) => (+ (d e₁) (de ("-" e₂)))

("+") => 0

("+" e₁ more ...) =>

(+ (d e₁) (d ("+" more ....)))

'*' =1

"x"

(x "x"

$$\begin{tikzcd}
se \arrow[r, Rightarrow, "desugar"'] & J_0 \arrow[r, Rightarrow, "interp"'] \arrow[dr, Rightarrow, "compile"'] & v & \\
 & & bc \arrow[r, Rightarrow, "vm"'] & v
\end{tikzcd}$$

2-1) b

desugarer

$(-\ e_1) \Rightarrow (\hat{*}\ -1\ e_1')$

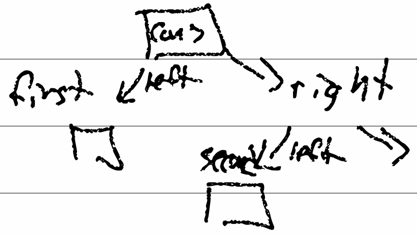
$(-\ e_1\ e_2) \Rightarrow (\hat{+}\ e_1'\ (-\ e_2))$

$(+) \Rightarrow 0$

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

$(\hat{+}\ e_1'\ (+\ e_2\ \ldots))$

```
def
 desugar ( se) :
   if  isList(se)  &&  length (se) = 2  &&
           first(se) == "-"  then

>  def  length (se) :
>     if is Null (se) :  return  0
>     else  is Cons (se) :  return  1+ length (right(se))
>     else  false
         new  JMult ( new JNum (-1), desugar(
                                            second (se)))
```



```
                                                      first
  if  isList(se)  &&  len(se) = 3  &&  fisf (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))))

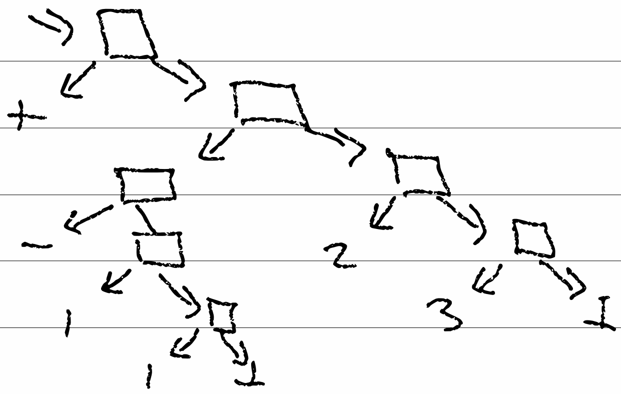
len = 3

(+ (- 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/ $J_0 \to J_1$

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

$v ::= b$

$b ::=$ 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 ..

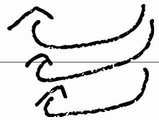
$\delta : b \ldots \to b$

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

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

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 e_t e_f) = e_t
step (if false e_t e_f) = e_f
step (p v_a ...) = δ(p, v_a ...)

step v = v

step (if e($\neq$v) e_t e_f) =
(if (step e) e_t e_f) or (if e (step e_t) ~~e_f~~
step (v_b .... e($\neq$v) e_a ...) =
(v_b ... (step e) e_a ...)

A context

C ::= hole | if0 C e e
| if1 e C e
| if2 e e C
| (e ... C e ...)

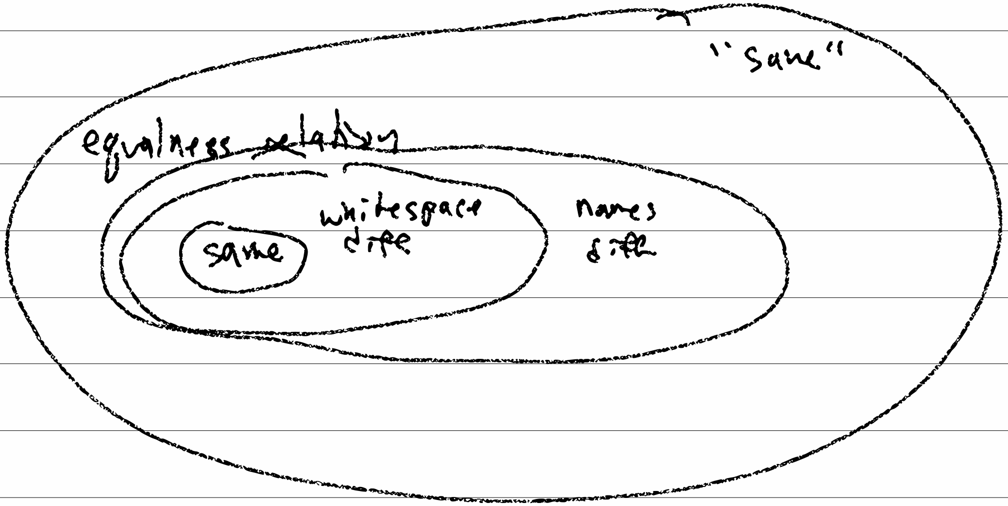
plug C e (C[e])
plug hole x = x
plug (if0 C e_1 e_2) x = if x e_1 e_2
plug (if1 e_1 C e_2) x = if e_1 x e_2
plug (e_1 ... C e_2 ..) x = (e_1 ... x e_2 ...)

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 ...)]


    "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)

$$
\begin{aligned}
C ::= \; & \text{hole} \mid \text{if } C \; e \; e \\
& \mid \text{if } e \; C \; e \\
& \mid \text{if } e \; e \; C \\
& \mid (e \ldots \; C \; e \ldots)
\end{aligned}
$$

$$
\begin{aligned}
E ::= \; & \text{hole} \mid \text{if } E \; e \; e \\
& \mid (v \ldots \; E \; e \ldots)
\end{aligned}
$$

"unique decomposition"

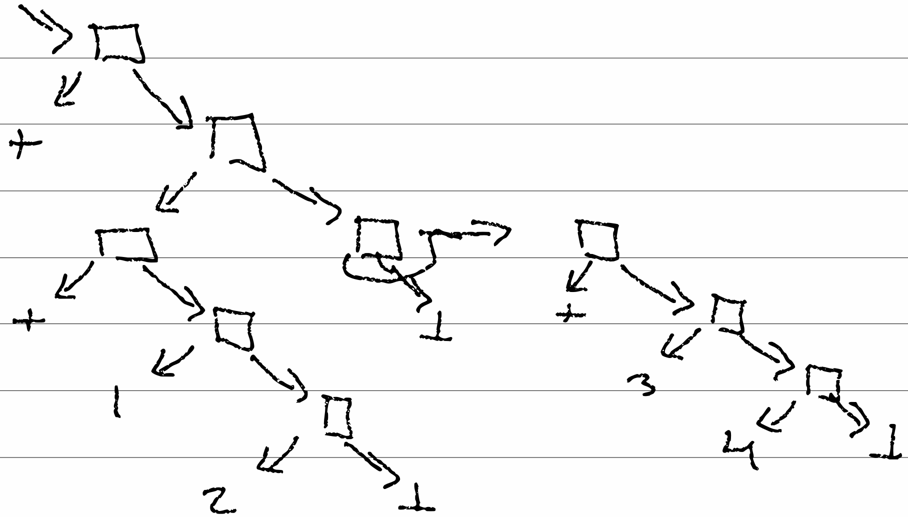
$$\forall e. \; e \in v \quad \text{or} \quad e = \overset{\downarrow \text{ unique } E}{E}[e'] \text{ where } e' \notin v$$

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

(+ (+ 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))