1 compose

Type of compose?

compose : $(a \to b) \to (b \to c) \to (a \to c)$

g f

$f \cdot g$ | $g = (2*)$.

$f = (+2)$.

$\equiv$ fog = compose . g . f

g ; f = f . g

((2*) ; (+2)) . 5

(+2) . ((2*) . 5)

12

f3 : (Int → Int) → (Int → Int) → Int → Int

2 filter

myfilter : (a → Bool) → [a] → [a]

predicate
e.g. (< 7) (a :: b :: s(s))

---

4 zip

zip : [a] → [b] → [(a,b)]

→ zip . [ ] . (2) = [ ]

→ zip . [1] . [ ] = [ ]

~~zip . [ ] . [ ] = [ ]~~

zip . (x :: xs) . (y :: ys)

[4,5] []

zip . [1,2,7,3,5] . ['b','c','p']

Atharva → [(1,'b'),(2,'c'),(7,'p')]

Another possibility → [(1,'b'),(2,'c'),(7,'p'),(3, ud/null),(5, ud/null)]

convert → lambda exp.

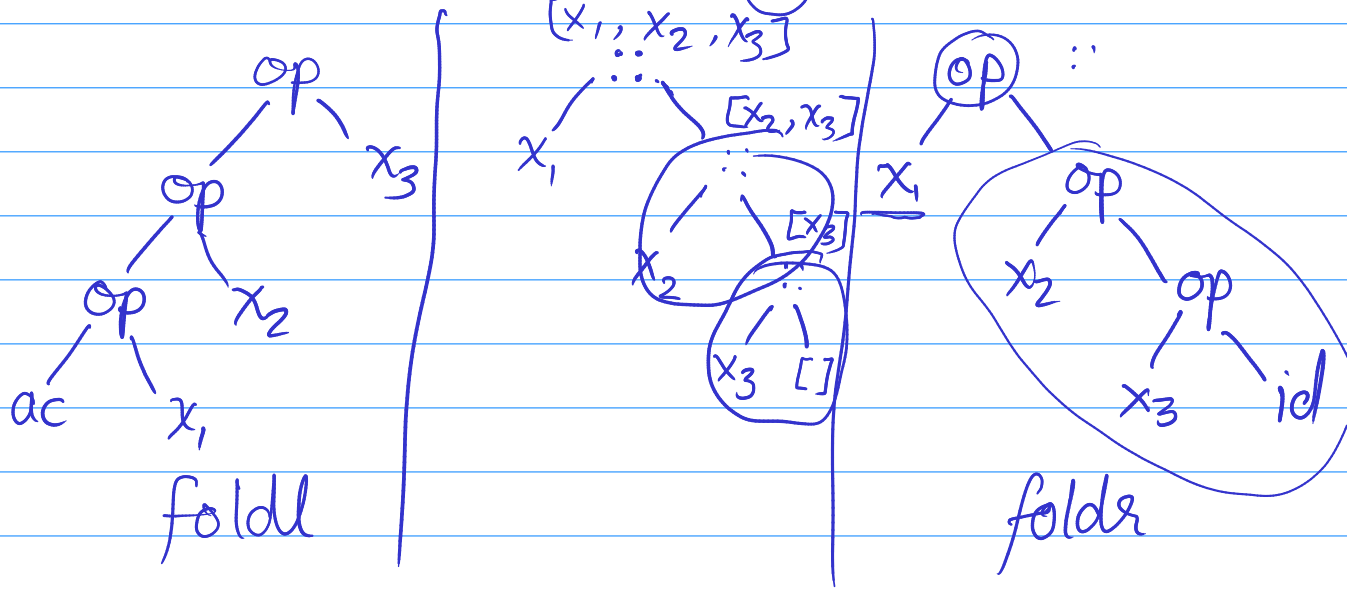
$cons \cdot x \cdot y = x :: y$

$cons : a \to [a] \to [a]$

$cons \cdot x = \lambda y \to x :: y$

~~$\lambda x \to$~~ $\lambda y \to x :: y$

$cons \cdot x \cdot y = foldl \cdot op \cdot id \cdot ac$



$cons \cdot x \cdot ls = foldr \cdot (::) \cdot [] \cdot (x :: ls)$

~~$(x :: ls)$~~

$cons = foldr \cdot (::) \cdot []$

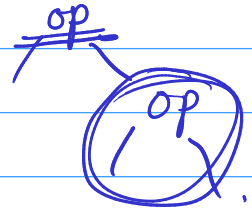
a $\xrightarrow[gen]{>}$ b a = ...b b = ...a

~~map~~:

map : (a → b) → [a] → [b]

reduce : (a → b → b) → b → [a] → b

(a → [c] → [c]) → [c] → [a] → [c]



shapeshifting

type b : [c]

~~[b]~~ list ← map

val → non-list, list ← reduce

map . f1 . ls = reduce . f2 . $\frac{id}{dc}$. ls

map . f1 = reduce . f2 . $\frac{\_}{foldr}$

map . f1 . k = foldr . f2 . $\frac{\_}{op}$. k

where f2 . x . y = (f1 . x) :: y

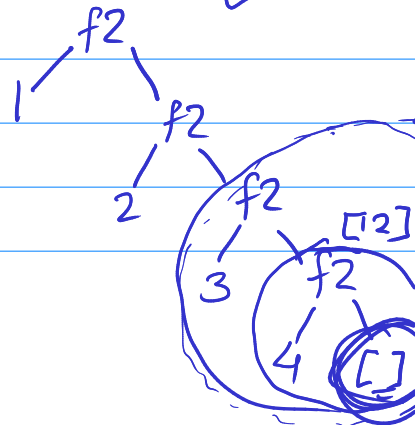
4 [] → [12]

f2 . 3 . [12] = (f1 . 3) :: [12]

= [9, 12]

k = [1, 2, 3, 4]

f1 (*3) [3, 6, 9, 12]



map . f1 . ls = foldr . f2 . [] . ls
where f2 . x . rl = (f1 . x) :: rl

f2 . x . rl = ((f1 . x) ::) . rl

3 [12]
|
g :: [12] → [9,12]

map . f1 = foldr . f2 . []
where f2 . x = ((f1 . x) ::)

f2 = λx → ((f1 . x) ::)

map . f1 = foldr . (λx → (f1 . x ::)) . []

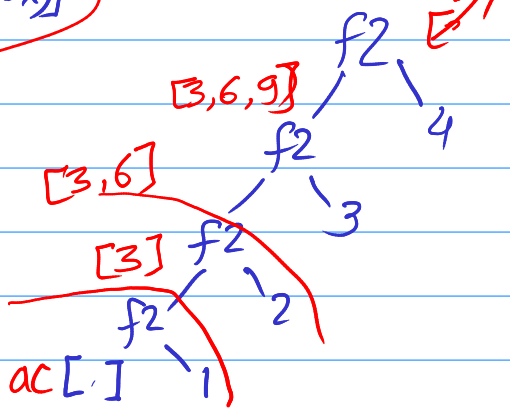
foldl ?

map . f1 . ls = foldl . f2 . [] . ls
where f2 . ac . x = ac ++ [f1 . x]

l = [1,2,3,4]

f1 = (* 3)

out = [3,6,9,12]



foldr

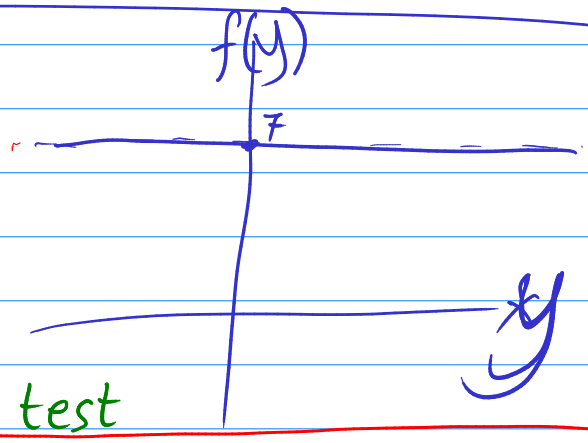
3 :: (6 :: (9 :: [12]))

(((([] ++ [3]) ++ [6]) ++ [9]) ++ [12])

```
map . f1      = foldr . f2 . []
  where f2 x = ((f1 x) ::)
```

```
map . f1      = foldl . f2 . []
  where f2 x y = x ++ [f1 y]
```

```
const x y = x
```



end of test

```
lu_table = [('a',5),('b',9),('c',3),('d',4)]
```

Exercise

```
lookup : [(Char, Int)] -> Char -> Int
lookup : [(a,b)] -> a -> b
```

action
Shift
Reduce
S&

char -> Int

assoc [(Char, Int

```
error : String -> a

lookup . table . x =
  lookup . ((a,b) :: abs) . x
    | a == x  = b
    | otherwise = lookup . abs . x

lookup . [] . x = []
```

errorval