1 = 1x(123 % 10)
3 = 3x (1x10) + (123 3 10|12 2
10+3 30 2x 30+ 32, 1
10|123 12 10 23 20 3
3
3 x 10 = ~~30~~
3x10 30x 10+3

M T W T F S S
Day: Formal Methods Date 08 02 2022

* Immutable => ~~con~~variable value cannot be changed
* State = set of particular values of the variable
when value is changed, state is also changed.
Your program output does not only depend on input, it also depends on state of program. e.g, ATM.

5 * -4 ✗ , 5 * (-4) ✓

(GHCI Commands)

not True || False = False
5 /= 5 = False
"hello" == "hell o" = False
succ 5 = 6
pred 5 = 4
min 4 5 = 4 , max 4 5 = 5 } only for two
succ 7 + min 2 3 + 2 = 12
doubleMe x = x + x } function defined
doubleMe 100 } function call
It is not fixed with data type (except string)
doubleMe' x y = x * 2 + y * 2
doubleMe' 3 3.5 = 13.0
doubleMe 5 + doubleMe' 5 6 = 32

lucky :: integral a => a → String
(first parameter)
lucky 7 = "Best of Luck"
lucky x = "You are out of Luck"

M T W T F S S
Day:

To compile: $5 \times 4 \times 3 \times 2$

cd address

ghci

:set prompt "ghci>"

lucky 3

let is used for List.

: to append at start of List. Find length of List?

++ is used to append List.

!! used for index in List.

:t (operator) to understand what an operator do

head return first element, tail return all elements except first, last return last element, init return all except last.

Length return length of List.

null return weather list is empty or not.

reverse reverses the List.

take no. takes the provided numbers and return

drop no. ~~takes~~ drops the provided numbers and return others.

minimum, maximum, sum, product, elem no., ['a'.. 'z'],

~~maximum~~ ['A'.. 'Z'], replicate, repeat, cycle

~~sum, product~~ [x * 2 | x <- [1..10]]

~~no. elem elem no.~~

Filters [x+2 | x <- [1..10], odd x, x /= 3]

fst, snd => order pairs, head

4×3×2×1×0

cin<< int num; —

count << "Hell";

init last

M T W T F S S
Day:☆☆☆☆☆☆☆

Date

[(x,y) | x ← [2, 5, 10], y ← [8, 10, 11]]

order pair

return x+y

sum = x+y

return sum

[1 | x ← ['a'..'z']]

"Eq" type class equality, "Ord" ordering

~~Sum~~ "Bounded" minimum or maximum ~~upper~~ bound

"Inov"

[(1,2), (3,4)]

(1,2)

1 2

h :: [a] → a h[] , h[3] ⇒ h 3:[]

h [] = ~~~~ h [1, 2, 3] ⇒ h 1:[2, 3]

h (x:xs) = x

Show simples st

read str → sin

Let a = 0

xs = [1, 2, 3, 4]

① 2 3 = 2 3, 1 : 1

② 3 = 3, 2 : 2

③ = [3] : 3

[] return x+y [3

Let [] = 0

fact n = n * fact (n-1)

0

length x:xs = 1 + length xs

length [] = 0

length x:xs = 1 + length

[1, 2, 3, 4] 4

sum [] = 0

2 3 4 3

sum x:xs = x + sum xs

3 4 2

4 1

last

1 (?:?) 2

guards, pattern, otherwise

x:xs ([x]:xs

where new cgppa

reverse (~~xs:x~~) = (~~xs++[x]~~)

① 2 3 4 1

reverse (x:xs) = ~~(xs:[x]~~ reverse xs : [x])

[]

1, 2, 3

2 3 [3]

Good Good Reve

M T W T F S S
Day:
Date 01 03 2022

myelem y [] = False
myelem y (x : xs)
  | y = = x = True
  | otherwise = myelem ~~xs~~ y xs

factorial 3 = 3 * factorial 2

max [x] = x
max (x : xs)
  | x > max xs = x ;
  | otherwise = ~~mx~~ xs ;

replicate n x xs = [ ]
replicat 3 5
n <= 0 [ ] ;
x : replicate (n-1) x
3-2

take 3 [2, 1]

Loop invarient

~~for (int i=0; i<~~
bubble (x : xs)
  | x > xs = buble xs ++ x
  | otherwise x : buble xs
  buble [ ] = [ ]

[6 9 4 5]
6 > [9 4 5]
9 > [4 5]
4 > [5]
5 > [ ]

[5]
5
[4 5]

x > buble xs = buble xs ++ x ;
otherwise

[4 5 9]
[6 4 5 9]

M T W T F S S
Day: ☆☆☆☆☆☆☆ Date

buble (x:xs)
~~t (x>~~ | ~~bublesort~~ (x>y | y:xs) = y ~~++ x ++ tail~~ xs   y ++ x ++ xs

~~where~~ min

buble (x:xs)
| x > head xs = [head xs] ++ (buble [x] ++ tail xs)
| otherwise = [x] ++ buble xs
~~bubble x:y:[ ]~~
~~| x>y = [y,x]~~
~~| otherwise = [x,y]~~

2,1,3,4

Buble sorting

[2, 4, 1, 3]
2 ++ [4, 1, 3] == temp

bubleup (x:xs)

buble sort (x:xs)
bubble sort = bubble sort (sort xs)

$\{x > y \mid y \leftarrow (y : xs)\}$

$\frac{6(6+1)}{2}$ , $\frac{4(4+1)}{2}$

M T W T F S S
Day: ☆ ☆ ☆ ☆ ☆ ☆ ☆ Date

Main purpose is formal verification to be easy

- Imperative Language

```
bubbleSort   xs  = bubbleSort (init bubbledUp)++ last bubbledUp
        | bubbledUp = bubbleUp xs
     quickSort [] = []
~~quickSort~~ quickSort x:xs
        let smaller = quickSort [y | y <- xs, y<x ]
            larger = quickSort [z | z <- xs; z>x]
        in  smaller ++ [x] ++ larger
```

- Equational Reasoning

```
          Let y = 2
     sort [] = []
     sort  x:xs                                  [: 1, 3,4,5]
      | y > x  = x: sort [~~x:~~ y++xs]
      | otherwise = y: sort [y: x++xs],

  1 : [3,4,5]                                    1 : [3, 4, 5]
2>1  -  1: sort [2, 3,4,5]                    1 : 2++ >[3,4,5]
2 : [3,4,5]                        [1,2,3,4,5]   [2,3,4,5]

     | y > x  = x : y++ sort [xs]
     | otherwise = y++ x++ xs
```

M T W T F S S

Day: ☆☆☆☆☆☆☆ Date

sort [] = []

Sort x : xs

| y > x = x : (y ++ sort xs)

| otherwise = y ++ (x ++ xs)

[3, 2, 1]

3-2 > 0

if x > 7   1-2 < 0

Assignment # 01

insert (x : xs)   1, [2, 5, 7]   [1, 2, 5, 7]

| 3 > 1 = 1 ++ (3 ++ (insert xs))   1-2 < 0   2-6 = -4   6-6 6-8

1 > max [2, 6, 8, 7, 4]   ← [1, 2, 6, 8, 7, 4]   < 0

2 6 8 [7, 4]   1-8 > 0   6-8

2-8 6 8-8 >= 0   8-7 = 1   7-4 =

1   x > max xs   (4, 5, 3, 2)   [1, 2, 6, 8, 7, 4]

1-2 > 0

1   1-8

~~split last~~ 1   2   [1, 2, 3, 2, 1]   2-6   6-6   ~~8-6~~   6-8

x >   x ++ [], x ++ []   8-7 = 1   7-4 >

7 > 8   8 > 6

x - max 3 = < 0   1   max = 8

[1, 2, 3]   x - max ≥ 0

1 [2, 3]   2-1 > 0

[a | a <- (1, 2, 3)]

1, 2

2 - 3,

① [a] → ([a], [a])

M T W T F S S
Day:
Date

splitList ~~[1, 2, 3~~ [1, 3, 4, 2]

1
1

fib 5 = ~~fib~~ (5 + fib(5-1))
= 5 + fib 4   16
fib 4 = 4 + fib 3   11
fib 3 = 3 + fib 2   7
fib 2 = 2 + fib 1   4
fib 1 = 1 + 1   2

[ ]   [ ]

fibo [1, 2]

[ x = 1, y = 2
xs ++ [y + x]

lord* lordly

② [1, 1]

[1, 1, 2, 3]

[ ] 1 : 1

fon [a b], n   y+x

5

fon a+b i+1   [1, 1]   n>0, n-1
4

x : y : 1

① 1   y : x :
5   x : (y : x : [xs])

(y : x : xs)   11

[x] y+x ++ [y]

xs ++ [y+x]
[y] ++ [xs]   ~~[1, 1, 2]~~

mulThree x y z = x * y * z
mulThree 2 3 4

mulThree' 2 = 2 * _ * _ = 2
mulThree'' 2 3 = 2 * 3 * _ = 6
mulThree''' 2 3 4 = 2 * 3 * 4 = 24

* λ calculus is base of functional language.
Curried function Partialy Applied Function

Lamda Function = Anonymous functions (without name)

Lamda language : just variables, no types

(λ x. x + 1) 100
((λx. λy. x+y) 2) 3)
True = λx. λy. x
False = λx. λy. y
Not = λb. b False True
Not True
(λb. b False True) True
True False True = False
Not False
(λb. b False True) False
False False True = True

Good Good

M T W T F S S
Day:

[ ] ++ [1 2 3] :-

Date

Induction Proof:-
i) Base case
ii) Induction hypothesis (Induction case)

$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

$\boxed{xs = [\,]}$

xs ys

Base case: n = 1 = 1
SumInts 1 = 1

Induction case: ~~Assume~~ Assume n = k
~~Assume~~ SumInts k = 1+2+....+k

SumInts (k+1) = ~~k+k~~ (k+1) + SumInts (k)
= (k+1) + (1+2+...+k)
= 1+2+....+k +(k+1)
= SumInts (k+1)

4 : 1 2 3
a : as [4 1 2 3]
[a] ++ as

∵ agar do lists main se sirf ek ke upar induction kerni ho to base case tab ho ga jab xs empty ho in (xs ++ ys).

(++) → [a] → [a] → [a]
(x : xs) ++ ys = x : (xs ++ ys) → A2
[ ] ++ ys = ys → A1

Structural Induction:-
Length [ ] = 0 → L1
Length (x : xs) = 1 + Length (xs) → L2

length (xs ++ ys) = length xs + length ys
Base case : xs = [ ]
length ([ ] ++ ys) = length [ ] + length ys ~~(This matches A1)~~
~~length ys~~ L.H.S
= length ([ ] ++ ys)
= length ys using A1

R.H.S:
0 + length ys
= length ys

You know nothing
False creation.

M T W T F S S
Day: ☆☆☆☆☆☆☆

Date

$= 0 + \text{length } ys$ using simple Maths
$= \text{length } [\,] + \text{length } ys$ using L1
$= R.H.S$

Induction step : $xs = zs$ (arbitrary)

(I.H) $\Rightarrow$ $\text{length}(zs ++ ys) = \text{length } zs + \text{length } ys$

To prove

$\text{length}(z:zs ++ ys) = \text{length } z:zs + \text{length } ys$

L.H.S

$= \text{length}((z:zs) ++ ys)$
$= \text{length}(z:(zs ++ ys))$ using A2
$= 1 + \text{length}(zs ++ ys)$ using L2
$= 1 + \text{length } zs + \text{length } ys$ Equational reasoning
$= (1 + \text{length } zs) + \text{length } ys$ using associative law
$= \text{length}(z:zs) + \text{length } ys$
$= R.H.S$

```
code :: (Integral a) => [a] -> [a] -> a [a]
code ([] ys) = ys
code (x:xs ys) =
    | x elem ys = code (xs ys)
    | otherwise = x : code (xs ys)
```

Day: M T W T F S S Date

Theorem: (map f . map g) xs = map (f . g) xs

Base case, xs = [ ]

(map f . map g) [ ] = map (f . g) [ ]

L.H.S = (map f . map g) [ ]

= map (f . g) [ ] = R.H.S

Induction case, xs = ~~z~~ zs

Induction hypothesis: (map f . map g) zs = map (f . g) zs

To prove

(map f . map g) z:zs = map (f . g) z:zs

L.H.S

Theorem: $2+4+6+\cdots+(2*n) = n*(n+1)$

Base case, n = 1

$\cancel{2+4+6+\cdots+}(2*1) = 1*(1+1)$

$2 = 2$

~~Induction case, Assume n = K-1~~

~~Induction hypothesis: $2+4+6\cdots+(2*(K-1)) = (K-1)((K-1)+1)$~~

~~To prove~~

~~n = K~~

~~$2+4+6+\cdots+(2*K) = K(K+1)$~~

$a(a+b)+c(a+b)$

$(a+b)(a+c)$

Induction case, Assume n = K

Induction hypothesis: $2+4+6+\cdots+(2*K) = K*(K+1)$

To prove, n = K+1

$2+4+6+\cdots+(2*K)+2*(K+1) = (K+1)*(K+1+1)$

L.H.S = $2+4+6+\cdots+(2*K)+2*(K+1)$

$= K*(K+1)+2*(K+1)$

$= (K+1)*(K+2)$

M T W T F S S
Day:

rev [] = []

[1, 2, 3] ++ [4 5 6]

3

[1 2 3 4 5 6]

$(map\ f\ .\ map\ g)\ [] = map\ (f.g)\ []$

$map\ f\ (map\ g\ []) = map\ f\ [] = []$

$map\ (f.g)\ [] = []$

1 2 ... K-1 K K+1

$x\ 'elem'\ rev\ xs = x\ 'elem'\ xs$

using code
I.H
Maths
already proven properties

Base case, $xs = []$

$x\ 'elem'\ rev\ [] = x\ 'elem'\ []$

L.H.S = $x\ 'elem'\ rev\ []$

$= x\ 'elem'\ []$ by R1

$=$ R.H.S

x : xs

a :

Induction case, $xs = as$

$x == a$

Induction hypothesis, Assume

$x\ 'elem'\ rev\ as = x\ 'elem'\ as$

To prove

[a]++

$x\ 'elem'\ rev\ (a:as) = x\ 'elem'\ a:as$

a!

L.H.S = $x\ 'elem'\ (rev\ (a:as))$

$= x\ 'elem'\ rev\ (as ++ [a])$ by R2

~~= x 'elem' (as++[a]) by hypothesis~~

property: $x\ 'elem'\ (xs ++ ys) = (x\ 'elem'\ xs)\ ||\ (x\ 'elem'\ ys)$

$= (x\ 'elem'\ rev\ as)\ ||\ (x\ 'elem'\ \cancel{rev}\ [a])$ by property

$= x\ 'elem'\ as\ ||\ x\ 'elem'\ [a]$ by hypothesis

$= x\ 'elem'\ [a]\ ||\ x\ 'elem'\ as$ by commutative ~~order reversing~~

$= x == a\ ||\ x\ 'elem'\ as$

$= x\ 'elem'\ (a : as)$

$= x\ 'elem'\ ([a] ++ as)$

x : y : xs

$= x\ 'elem'\ (a : as)$

CamScanner

2 /= 1    1 ++ 2 [2 3 4]

1 2 3 4    2 /= 2    2 [3 4]

Day: M T W T F S S    Date

rem w (xs ++ ys) = rem w xs ++ rem w ys

Base case, xs = [ ]

rem w ([ ] ++ ys) = rem w [ ] ++ rem w ys

L.H.S = rem w([ ] ++ ys)

= rem w ys

= [ ] ++ rem w ys

= rem w [ ] ++ rem w ys

= R.H.S

Induction case, xs = zs

Induction hypothesis, Assume

rem w (zs ++ ys) = rem w zs ++ rem w ys

To prove

rem w (z:zs ++ys) = rem w z:zs ++ rem w ys

L.H.S = rem w (z:zs ++ys)

= rem w z : (zs ++ys)

= x /= yz ++ rem w ys

~~= [y] ++ rem w ys~~

= [z] ++ rem w zs ++ rem w ys

= rem w z:zs ++ rem w ys

= R.H.S.

~~$k + \frac{(k-1)(k)}{2}$~~

(2:25)

M T W T F S S
Day:

$$\frac{K-1}{K-1+1} = \frac{K-1}{K}$$

Date

$$K + \frac{(K-1)(K)}{2} = \frac{2K + (K-1)(K)}{2} = \frac{2K + K^2 - K}{2}$$

$$= \cancel{K^2 + 2K - K} = \frac{K^2 + K}{2} = \frac{K(K+1)}{2}$$

$$\frac{n}{n+1} = \frac{K}{K+1}$$

$$= \frac{n}{n+1} = \frac{1}{2}$$

$$\frac{n}{n+1}$$

$$\frac{K}{K+1}$$

$$= \frac{K-1}{(K-1+1)} = \frac{K-1}{K}$$

$$= \frac{K}{K+1} + \frac{K-1}{K} = \frac{K^2 + K^2 - 1}{K(K+1)} = \cancel{\frac{2K^2 - 1}{K}}$$

$$= \cancel{\frac{\quad}{K(K+1)}} = \frac{2\cancel{(K+1)}(K-1)}{K\cancel{(K+1)}} = \frac{2K-1}{K}$$

$$\frac{K}{K+1} = K\left(\frac{1}{K(K+1)} + \frac{K-1}{K}\right) = \quad n \; 5, 14 \quad \frac{n}{n+1}$$

$$\frac{n}{n+1}$$

$$\text{Sumf } n = 1 \% (n * (n+1)) + \text{Sumf } (n-1) \qquad \frac{K-1}{K-1+1}$$

$$= \frac{1}{K^*(K+1))} + \frac{K-1}{K} \qquad 2\left(\frac{K-1}{K}\right)$$

$$= \frac{1}{K}\left(\frac{1}{K+1} + \frac{K-1}{1}\right) = \frac{1}{K}\left(\frac{1 + K^2 - 1}{K+1}\right)$$

$$= \frac{1}{\cancel{K}}\left(\frac{K^2}{K+1}\right) = \frac{K}{K+1}$$

CamScanner

f x
[x3] 10

M T W T F S S
Day: Date

~~Reverse (Reverse xs) = xs~~
~~Base case, xs = []~~ Reverse [] = []
Reverse (Reverse []) = [] reverse (x:xs) = rev xs ++ [x]

Reverse [] = []
Reverse (Reverse xs) = xs

Base case, xs = []
Reverse (Reverse []) = []
Reverse [] = []
[] = []

Reverse ~~xs ++ zs = Reverse xs ++ Reverse zs~~

Induction case, xs = zs
Induction hypothesis, Assume
Reverse (Reverse zs) = zs
To proof
Reverse (Reverse z:zs) = z : zs
~~L.H.S = Reverse~~
Reverse (~~rev xs~~ Reverse zs ++ [z]) = z : zs
Reverse (Reverse zs) ++ Reverse [z] = z : zs
zs ++ z = z : zs
z : zs = z : zs

Day: M T W T F S S

2 1 4 3

Date

~~SelectionSort x:xs = SelectionSort (minimum xs)~~

~~Selection xs = minimum xs : (Selection Sort x:xs)~~

~~Selection (2 1 4) = minimum (2 1 4) : selectionSort 1~~

SelectionSort xs = minimum xs : SelectionSort (RemoveMinimum xs)

[2 1 3] = xs    x:xs = 2:[1 3]

x = 2    y = 1

2 < 1
x < y    minimum (y : xs)

2 : xs    1 : 3

2 + xs

~~merge (x : xs) = [2 1 3]~~    [2 1 3] [4 3 2]

2, 1, 3    4

x : xs , y : ys    1 3    4 3 2

x < y = x : merge (xs  y:ys)

otherwise = y : merge (x:xs  ys)

f x y

flip :: (a -> b -> c) -> (b -> a -> c)

flip f x y = g y x

2 : 1

zip

map

10 .

M T W T F S S
Day: ☆☆☆☆☆☆☆ Date

~~9+25~~ $3^2 + 5^2$

filter :: (a → Bool) → [a] → [a]
filter _ [ ] = [ ]
filter p (x:xs)
  | P x == True = x: filter p xs
  | otherwise = filter p xs
filter (>3) [0,1,2,3,4] ⇒ [4]

~~quicksort [ ] = [ ]~~
~~quickSort (x:xs)~~
~~Bet~~ |

quickSort ~~f~~ [ ] = [ ]
quickSort f ~~l~~(x:xs)
filter (<x) xs
filter (≥ x) xs

Find the largest number under 100,000 that is divisible by 3829.
head filter ( 'mode' 3829) [100000, 99999, .. ]

~~takeWhile~~ Find the sum of all odd squares that are smaller than 10000.

sum (filter ('mod' 2) [10000, 99999..])

map (*) x | x =

map f (filter ( 'mode' 2 /= 0) [list]

3 : [4] ✓

3

map g a:as

M T W T F S S

Day:

Date

~~takeWhile (.~~

sum (takeWhile (< 10000) (filter (odd) (map (^2) [1..])))

① map g (xs++ys) = map g xs ++ map g ys

Base case: xs = []

map g ([]++ ys) = map g [] ++ map g ys

L.H.S = map g ([]++ys)

= map g ys From A1

= [] ++ map g ys by identity property ~~simple arithmetic~~

= map [] ++ map g ys from M1

= map g [] ++ map g ys by fold map ~~map~~

Induction case: xs = as

~~Base cas~~ Induction hypothesis: map g (as++ys) = map g as ++ map g ys

To prove

map g (a:as ++ys) = map g a:as ++ map g ys

L.H.S = map g (a:as ++ ys)

= map g a : (as++ys) from A2

~~= map g a:as ++ map g ys~~ map g a:as

= ~~map g (a:as ++ys)~~

= g a : map g (as++ys) from M2

= g a : map g as ++ map g ys from hypothesis

= map g a : as ++ map g ys by fold map ~~calculations~~

2) Base case: n = 1

$= \frac{3^1 - 2^1}{1} = \frac{1}{1} = 1$

Induction case: n = k-1

Induction hypothesis

[1, 2] ++ [3, 4]    [3,4] ++ [1, 2]
= [1, 2, 3, 4]    = [3, 4, 1, 2]

$x^2 - y^2$

M T W T F S S
Day: ☆☆☆☆☆☆☆    Date

~~listGenerat (K-1) = $\frac{3^{K-1} - 2^{K-1}}{2}$~~
~~To prove~~

~~listGenerat K = K + lististGenerate (K-1)~~

③ Base case: [] ys []
reverse ([] ++ ys) = (reverse ~~[]~~) ++ (reverse ~~xs~~)
L.H.S = reverse ([] ++ ys)
= reverse (reverse [] ++ ys) from R1
= reverse(reverse []) ++ reverse ys by unfold
= reverse [] ++ reverse ys from R1
= reverse ys ++ reverse [] by commutative
Induction case: xs = as
Induction hypothesis: Assume
reverse (as ++ ys) = (reverse ys) ++ (reverse as)
To prove: reverse (a:as ++ ys) = (reverse ys) ++ (reverse ~~as~~ a:as)
L.H.S = reverse (a: as ++ ys)
= reverse (a:as) ++ reverse ys by unfold
~~= ((reverse as)) ++ [a]) ++ reverse ys~~
= ((reverse as) ++ [a]) ++ reverse ys by R2
= reverse ys ++ ((reverse as) ++ [a]) by commutative
= reverse ys ++ reverse (a: as) by R2
= reverse as ++ [a] ++ reverse ys
= reverse as ++ reverse ys ++ [a] by assosiativity
= reverse (as ++ ys) ++ [a] by fold
= reverse ys ++ reverse as ++ [a] by hypothesis
= reverse ys ++ reverse a: as by R2

Day: M T W T F S S    $3 \in [3]$    $3 \,||$    Date

(4) x 'elem' (xs ++ ys) = (x 'elem' xs) || (x 'elem' ys)

Base case: xs = [ ]

x 'elem' ([] ++ ys) = (x 'elem' [ ]) || (x 'elem' ys)

L.H.S = x 'elem' ([] ++ ys)

= ~~x 'elem' [] ++~~ x 'elem' ys    by calculations

R.H.S = ~~False ++~~ (x 'elem' [ ]) || (x 'elem' ys)

= False || (x 'elem' ys)    by E1

= (x 'elem' ys)

Induction case: xs = as

Induction hypothesis: Assume

x 'elem' (as ++ ys) = (x 'elem' as) || (x 'elem' ys)

~~L.H.S~~ To prove: x 'elem' (a:as ++ ys) = (x 'elem' a:as) || (x 'elem' ys)

L.H.S = x 'elem' (a: as ++ ys)

= x 'elem' ([a] ++ as ++ ys)    ~~by~~ by unfold

~~= x 'elem' [a] ++ x 'elem' as ++ x 'elem' ys    by unfold~~

= x 'elem' [a] || x 'elem' as || x 'elem' ys    by hypothesis

= x == a || x 'elem' as || x 'elem' ys    by calculations

= True || x 'elem' as || x 'elem' ys    by E2

= a||x 'elem' as || x 'elem' ys    since condition is True

= x 'elem' a:as || x 'elem' ys    by folding

(5) ~~a ++ x 'elem' as~~

~~x 'elem' a:as~~

~~L.H.S = x 'elem' (a:as ++ ys)~~

~~= x 'elem' (a:as) || x 'elem' ys~~

~~=~~

Day: M T W T F S S

[1, 2, 3] [ ] Date

(5) concat (xss ++ yss) = concat xss ++ concat yss

Base case, xss = [ ]

concat ([ ] ++ yss) = concat [ ] ++ concat yss

L.H.S: concat yss

R.H.S = [ ] ++ concat yss → [ ]

= concat yss

Induction case: xss = ass

Induction hypothesis: Assume

concat (ass ++ yss) = concat ass ++ concat yss

To prove: concat (as:ass ++ yss) = concat as:ass ++ concat yss

L.H.S = concat (as:ass ++ yss)

= concat (as:ass) ++ concat yss    by unfold

= as ++ concat ass ++ concat yss    by c2

= as ++ concat (ass ++ yss)    by hypothesis

= ~~ass~~

(7) Base case: n = 1

listGenerate 1 = $3^1 - 2^1 = 1$

Base case: n = 2

listGenerate 2 = $3^2 - 2^2 = 9 - 4 = 5$

Induction case: n = K - 2

listGenerate (K-2) = $3^{k-2} - 3^{k-2}$

$= 3^k \cdot 3^{-2} - 2^k \cdot 2^{-2}$

$= \frac{3^k}{3^2} - \frac{2^k}{2^2} = \frac{3^k}{9} - \frac{2^k}{4}$

Induction case: n = K - 1

listGenerate (K-1) = $3^{k-1} - 2^{k-1}$

$= 3^k \cdot 3^{-1} - 2^k \cdot 2^{-1}$

$= \frac{3^k}{3} - \frac{2^k}{2}$

Day: M T W T F S S  Date

~~Induction hypothesis~~

To prove : $n = K$

$$\text{listGenerate } K = 5 * (\text{listGenerate } K-1) - 6 * (\text{listGenerate } K-2)$$

$$= 5 * \left(\frac{3^k}{3} - \frac{2^k}{2}\right) - 6 * \left(\frac{3^k}{9} - \frac{2^k}{4}\right)$$

$$= \frac{5 \cdot 3^k}{3} - \frac{5 \cdot 2^k}{2} - \frac{6 \cdot 3^k}{9} + \frac{6 \cdot 2^k}{4}$$

$$= 3^k\left(\frac{5}{3} - \frac{6}{9}\right) \mp 2^k\left(\frac{5}{2} \mp \frac{6}{4}\right)$$

$$= 3^k\left(\frac{15-6}{9}\right) - 2^k\left(\frac{10-6}{4}\right)$$

$$= 3^k(1) - 2^k(1) = 3^k - 2^k$$

Induction hypothesis : xs = as

x 'elem' (as ++ys) = (x 'elem' as) || (x 'elem' ys)

To prove:

x 'elem' (a:as ++ys) = (x 'elem' a:as) || (x 'elem ys)

(For a x == y)

L.H.S = x 'elem' (a:as ++ys)

M T W T F S S
Day:

Book Almassary'
C

Date

```
Select *salaries
From Faculty
Where dept no like ( Select deptno
                     From dept
                     Where deptname = 'ICS')
UNION
Select salaries
From Faculty
Where deptno Like ( select deptno
                    From dept
                    where deptname = 'COF')
```

```
Select *
From Lecturer
Where salary > ALL (Select Salary
                    From Lecturer
                    where deptno = (Select deptno
                                    From dept
                                    Where dname = 'COF'))
```

True

M T W T F S S
Day: Date

L.H.S =
x 'elem' (a:as ++ ys) = (x elem a:as) || (x elem ys)
when x == a
L.H.S
x elem (a:(as ++ ys))
True
R.H.S
(x elem a:as) || (x elem ys)
(True) || (x elem ys)
True
L.H.S x ≠ a

x elem (a:as ++ ys)
x elem (a:(as ++ ys))
x elem (as ++ ys)
x elem as || x elem ys => L.S.
R.H.S
(x elem a: as) || (x elem ys)
x elem as || x elem ys.

M T W T F S S

Day: ☆☆☆☆☆☆☆ x 'elem' (as ++ ys) = Date

x 'elem' (a:as ++ ys) = x 'elem' (a:as) || x 'elem' ys

L.H.S = x 'elem' (a:as ++ ys)

= x 'elem' (a:as) ++ x 'elem' ys

= ~~x 'elem'~~ True ++ x 'elem' ys

---

x 'elem' (a:as ++ys) = x 'elem' (a:as) || x 'elem' ys

L.H.S = x 'elem' (a:as ++ ys)

= x 'elem' ([a] ++ [as] ++ ys) by unfold

= x 'elem' [a] || x 'elem' as || x 'elem' ys by ~~hypothesis~~ property

= x == a || x 'elem' as || x 'elem' ys by ~~calculation~~/unfolding

= True || x 'elem' as || x 'elem' ys by E2

= x 'elem' as || x 'elem' ys → by simple OR rule

= x 'elem' (as ++ ys) by hypothesis

R.H.S = x 'elem' (a:as) || x 'elem' ys

---

L.H.S = x 'elem' (a:as ++ ys)

= x 'elem' (a:as) ++ x 'elem' ys → by unfold

= True ++ x 'elem' ys by E2

= x 'elem' ys

R.H.S = x 'elem' (a:as) || x 'elem' ys

= True || x 'elem' ys By E2

= x 'elem' ys

x /= y

L.H.S = x 'elem' (a:as ++ ys)

= x 'elem' (a:as) ++ x 'elem' ys by unfolding

= x 'elem' as ++ x 'elem' ys by E3

= x 'elem' (as ++ ys) by folding

Day: M T W T F S S

Date

= x 'elem' ~~as~~ as || x 'elem' ys

R.H.S = x 'elem' (a:as) || x 'elem' ys

= x 'elem' as || x 'elem ys by E3

(5) Base case : xss = []

concat ([] ++ yss) = concat [] ++ concat yss

L.HS. concat ([] ++ yss)

= concat ( concat [] ++ yss) by C1

= concat (concat ([]) ++ concat yss by unfold

= concat [] ++ concat yss by C1

= R.H.S

Induction hypothesis: concat (ass ++ yss) = concat ass ++ concat yss

To prove: concat ( as: ass ++ yss) = concat as:ass ++ concat yss

~~L.H.S: concat (as:ass ++ yss)~~

R.H.S = concat as:ass ++ concat yss

= as ++ concat ass ++ concat yss by C2

= as ++ concat (ass ++ yss) by hypothesis

= concat ( as : ass ++ yss ) by C2

Assalamu o Alaikum

Good. Good Good. Assalama I sit by myself

Talking to the Moon

M T W T F S S

Day:

Date

foldl $\boxed{\text{function}}$ $\boxed{\text{accumulator}}$ $\boxed{\text{list}}$

(a binary function) (any number)

elem' y ys = foldl (\acc x → y == x = acc) False ys

(\acc x → if y == x = True else acc)

foldr (\ x acc → ) That's it

No more change than

Max' ys = foldr (\y acc → if ~~acc~~ y > acc
then ~~acc~~ y
else acc ) 0 ys

foldl1 => assigns first value as acc

~~reverse' = foldl (~~

Reverse :-

reverse' xs = foldl (\acc x → x : acc) [ ] xs

Remove Occurance :-

Remove' xs y = foldl (\acc x → if y == x
then acc
else acc ++ [x] ~~x : acc~~ ) [ ] xs

Scan Max :-

It will maximum of two numbers at each step. If y is greater than acc then it

will store y as max in list otherwise it will again store acc value in list.
The new list will be:

[1, 2, 5, 5, 9, 9, 9]

map $ 3 [(4+), (2*), (^2)]

Sum $ filter (>10) $ map (*2) [2 .. 10]

(f o g) x = f(g(x))    ~~Comp~~ Function composition

map (negative . abs) [2, -6, 5, -1]

1. Curried function :- partial applied functions
generating new functions from old functions
• instead of using -4 as a function, use substract 4.
2 Some higher-orderism is in order :-
applyTwice (+3) 10 =
10+3 = 13 => ~~(+3)~~ => ~~3+13~~ 16
applyTwice (+3) 13
13+3 = 16
applyTwice (multiThree 2 2) 9
multiThree 2 2 = ~~multiThree~~ 4
applyTwice (multiThree 4) 9
multiThree 4*9 = 36
applyTwice (multiThree 4) 36
multiThree 4*36 = 144
3 31
Zipwith => takes a function and two lists
Flip => flip ~~the~~ the first two arguments of function
3. Maps and Filters :- Map takes a function and a list and
apply it to every element of list
Filter ek function or list leta hai or function ko apply
ker ke Boolean ki base per result deta hai
*(fun !! 4) 5
4. Lambdas :-

Path represents one concrete execution

Program Analysis: Program ko analyze karen or is ~~ke~~ tarhan ke sare paths execute hon.

∵ X Y => these are symbolic values

2 3 => these are concrete values

Path condition: $(X == Y) \land (Y+1) \leq 5 \land X != (Y+1)$ => TTT

$\neg(X == Y) \land \neg(Y <= 5)$ => FF

$(X == Y) \land ((Y+1) <= 5) \land \neg(X == Y)$ => TTF

$(X == Y) \land ((Y+1) > 5)$ => TF  we can remove this

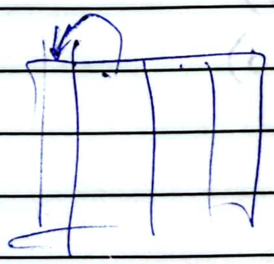
$\neg(X == Y) \land (Y <= 5) \land (X != Y)$ => FTT

$(X != Y) \land (Y <= 5) \land \neg(X != Y)$ => FTF

We use symbolic values, not concrete values.
This whole thing is called Symbolic Execution.

1. X = ~~8~~4, Y = 4  2. Path will never be followed
3. X = 5, Y = 5  4. X = 5, Y = 4
5. ~~X = 5, Y = 4~~ Infisible  6. X = 5, Y = 6

These are our test cases.

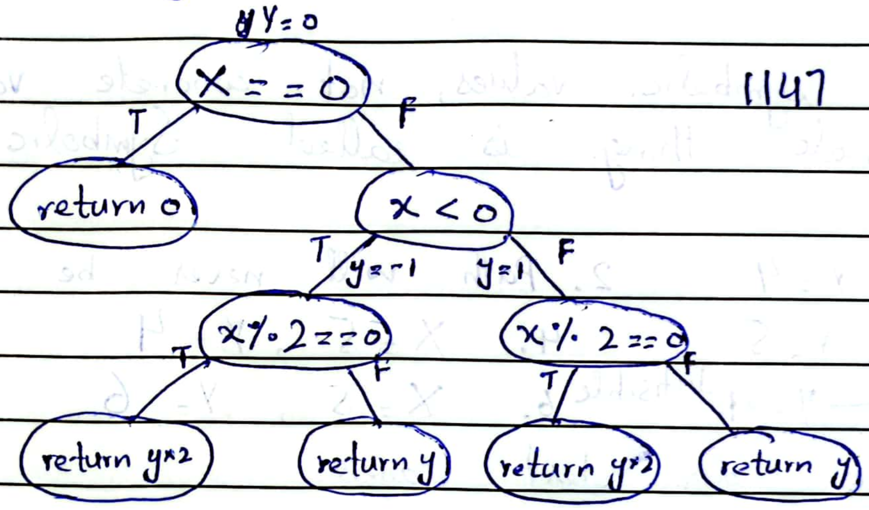


Day: M T W T F S S

Date

```
int ajeeb (int x) {
    int y=0;
    if (x==0)
        return 0;
    if (x<0)
        y = -1;
    else
        y = 1;
    if (x % 2 == 0)
        return y*2;
    return y;
}
```

Concrete Symbolic Execution
(Concolic execution)




1147

$(X != 0) \wedge !(X < 0) \wedge !(X \% 2 == 0)$ 1147

" $\wedge$ " $\wedge$ $(X \% 2 == 0)$ 2

$(X != 0) \wedge (X < 0) \wedge !(X \% 2 == 0)$ -1

M T W T F S S
Day: Date

foldr ( \ ~~acc~~

~~sumSequence n = foldr (\x acc → acc + if x <= n then acc acc + x² else~~

① SumSequence n = foldr (\x acc → if x <= n
then acc + (x * x)
else acc ) 0 [1,2,..]

② SumSequence n = ~~take~~ if x <= n
~~map (+) sqr x~~ then map (+) ~~(x)~~

13 /-

⑦ ["ate"]

① 1 + (2²) = 5 Ashar [1, 2]
5 + (3²) = 14

0 + 1 = 1
1 + 4

1 2 3 2 3 1
3 2 1

⑨ foldl (\acc xs → ~~x : x~~ (shift xs) : acc
shift (head acc) )

Prime Number
(Prime n = ~~foldl (\acc n → if (n % 2 == 0)~~
foldl (\acc x → if x % n ≠ 0
then ~~False~~ False True
else ~~acc~~ True ) False [1, 2 ..

The Hell is waiting for the lost souls, to put some cuffs upon their throats.

M T W T F S S
Day:

Hadies Master of Darkness Cool Super Excellent Fields Excellent Good Good Date

if count > 0 then True else False
where count = foldl (\acc x -> if x == True then acc + 1 else acc - 1) 0 xs

Alloy (Formal Specification) Analizer

Field:- sig A { f : e } => A×e
sig A { f : B -> C } => A×B×C
} Ordered pair

Jab ham A×e cartesian product nikalen ge to iska matlab hai ke A×e ke kuch instances is field main hon ge

Multiplicity =>

| | |
|---|---|
| Some | 1+ |
| set | 0+ |
| one | 1 |
| lone | 0/1 |

Multiplicity, Extends, Instances

sig => set create karta hai

Files (Files directories, shortcuts)

abstract sig Object { }
sig Dir extends Object {
contents : set Object
}

sig File extend Object {

}

$1/x$ is partially applied.
$x^2$ is total

sig Alias ~~extend~~ in ~~Object~~ File { (because alias are a ~~part~~ type of file)
to : one Object
}

Day: M T W T F S S    Date

Usama Rehman

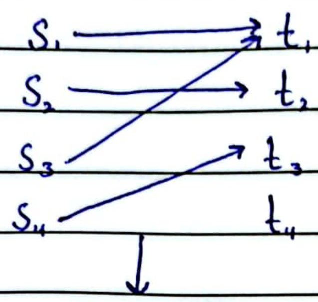
20F-0127

BSE-4A

① sig S { f : lone T }    (Partial Function)

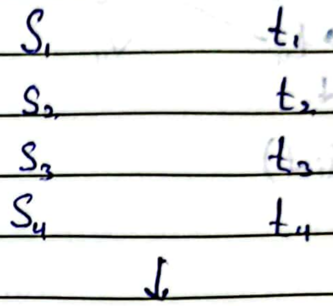
For each element s of S, f maps s to 0/1 value in T.

i)



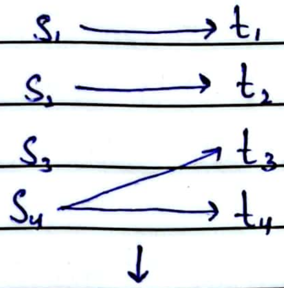

(valid)

ii)



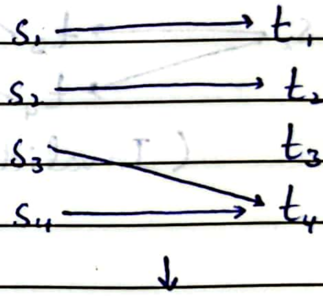

(valid)

iii)




(Invalid)

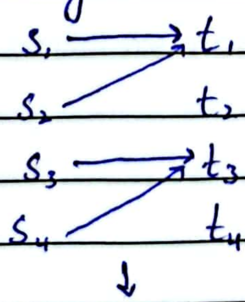
iv)




(valid)

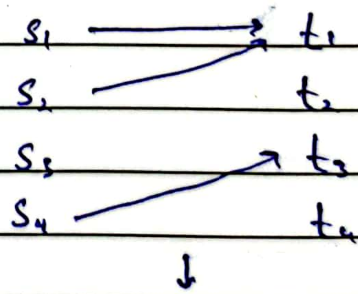
② sig S { f : one T }    (Total Function)

i)




(valid)

ii)



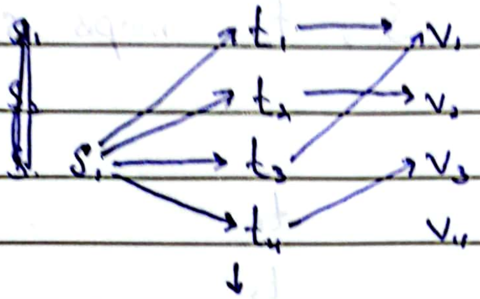

(Invalid)

Day: M T W T F S S                                Date

③ sig S { f : T → one v }

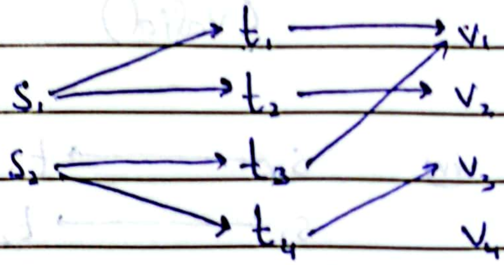
For each element s of S, over the tripels that starts with s, f maps T element to exactly one V element.

i)



(valid)

ii)



( Invalid)

M T W T F S S
Day: ☆☆☆☆☆☆☆ Date

Usama Rehman BSE-4A 20F-0127

Class Activity

i) sig University {
group member : student → one Prof }

Answer: It means that every student in university has exactly one professor in research group.

ii) sig University {
group member : Student → lone Prof
}

Answer: It means that every student in university have zero or at most one professor in ~~s~~ research group.

iii) For some

Answer: It means that every student in university has at least one or more than one research group.

iv) group member : Student some → Prof

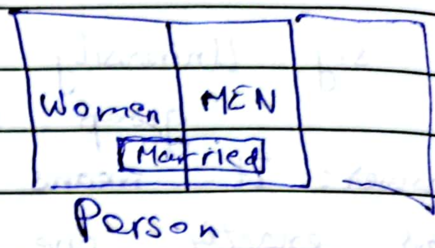
Answer: It means that atleast one student have a professor research group.

v) group member : Student + some → lone Prof

Answer: It means that some students have zero or at ~~leas~~ most one professor.

Day: M T W T F S S Date

abstract sig Person { }
sig Men extend Person{}
" women " " "
sig Married in Person {}




sig Forecast { weather: ~~[illegible]~~ City -> one weather

Day: M T W T F S S  Date

" ^ " this is closure.

```
sig Node {
    next : lone Node,
    prev : lone Node
}
one sig Head extends Node { }
fact NoSelfloop {
    all n : Node | {
        n.next != n
        n. prev != n  ,  n. prev != n.next
    }
}
fact RechableFromHead {
    Node = Head.^next + Head
  // Node = Head.*next
}
fact {
    all n : Node, x : n.next | x. prev = n
}
fact NoCycle {
    all n : Node | {
        n.^next & n = none
        n.^prev & n = none
    }
}
```

length (xs ++ ys) = length xs + length ys

Base case: xs = []

length ([] ++ ys) = length [] + length ys

L.H.S

= length ([] ++ ys)

= length ys → $A_1$

= 0 + length ys → by mathematical simplific

= length [] + length ys = R.H.S

Induction case: xs = as

Induction hypothesis: Assume

length (as ++ ys) = length as + length ys

To prove:

length (a: as ++ ys) = length a:as + length ys

L.H.S: length(a:as ++ ys)

= length a:(as ++ ys) → by $A_2$

= 1 + length(as ++ ys) → $A_4$

= 1 + length as + length ys → hypothesis

= length (a: as) + length ys → $A_4$

# Symbolic Execution

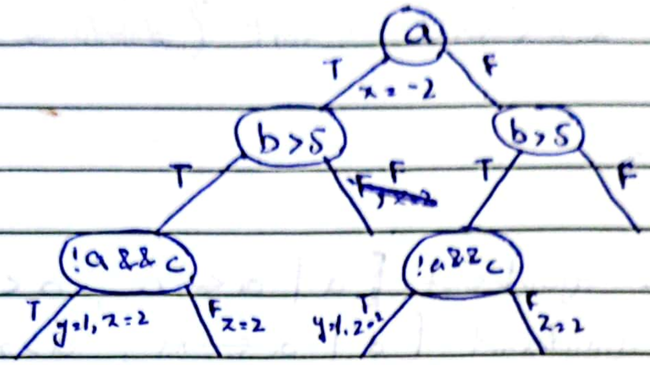
Concrete execution: symbolic values ko concrete values provide karna.

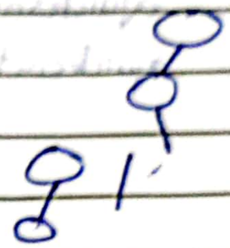
```
function f(x, y) {
    var s = "foo";
    if (x < y) {
        s += "bar";
        console.log(s);
    }
    if (y == 23) {
        console.log(s);
    }
}
```

```
int f(a, b, c) {
    var x = y = z = 0
    if (a == true) {
        x = -2;
    }
    if (b > 5) {
        if (!a && c) {
            y = 1;
        }
        z = 2;
    }
}
```

~~concrete execution~~ :- Symbolic execution

~~Let a=b=c=1~~



$\neg(x==0) \wedge (x<0) \wedge \neg(x\%2==0)$

T

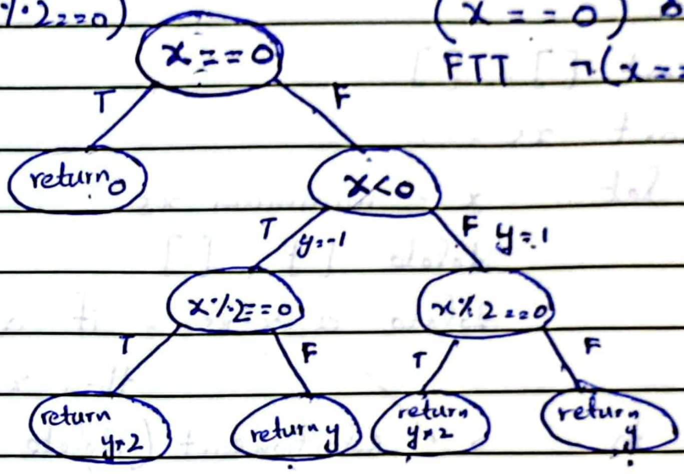
x=-2

T

F

z=2

ass




(x==0) ò T

FTT $\neg(x==0) \wedge (x<0) \wedge (x\%2==$

3

x= - Y 3

| | | |
|---|---|---|
| $\neg(x==0) \wedge (x>0) \wedge (x\%2 != 0)$. | FFF | 3 |
| $\neg(x==0) \wedge (x>0) \wedge (x\%2==0)$ | FFT | 2 |
| $\neg(x==0) \wedge (x<0) \wedge (x\%2 != 0)$ | FTF | -3 ~~infeasible~~ |
| $\neg(x==0) \wedge (x<0) \wedge (x\%2==0)$ | FTT | -2 |
| $(x==0)$ | T | 0 |

M T W T F S S

Day: ☆☆☆☆☆☆☆

Date

```
elem a [] = False
elem a (x:xs) =
    | a==x = True
    | otherwise = elem a xs

sort [] = []
Sort xs = let {x = minimum xs}
          in x : Sort (xs)

quicksort [] = []
quicksort x:xs =
      let smaller = quicksort [a | a <- xs , a <= x]
          bigger = quicksort [a | a <- xs , a > x]
      in  smaller ++ [x] ++ bigger

selectionsort [] = []
selectionsort xs =
      let  x = minimum xs
           delete [] = []
           delete a x:xs = if a==x
                           then xs else x:delete a xs
      in   x : selectionsort (delete x xs)

insertsort sorted [] = sorted
insertsort sorted (x:xs) =
      let insert a [] = [a]
          insert a x:xs = if a <= x
                          then a:(x: xs)
                          else x: (insert a xs)
      in  insertsort (insert x sorted) xs
```

M T W T F S S
Day: Date

\

Arbitrary

```
map _ [] = []
map f (x:xs) = f x : map f xs

filter _ [] = []
filter f (x:xs)
        | f x == true = x : filter f xs
        | otherwise = filter f xs

            filter (>10000)(sum (map (^2) ([1,2,..])))

sum (takeWhile (<10000)(filter odd (map (^2) [1,2,..])))
 sum (takeWhile(<10000)([n^2 | n <- [1,2,..] , odd (n^2)]))

    \ lambda
elem y ys = foldl (\acc x -> if x==y then True else False)
                                                      False ys
between' x y zs = filter (\z -> z>x && z<y) zs
```