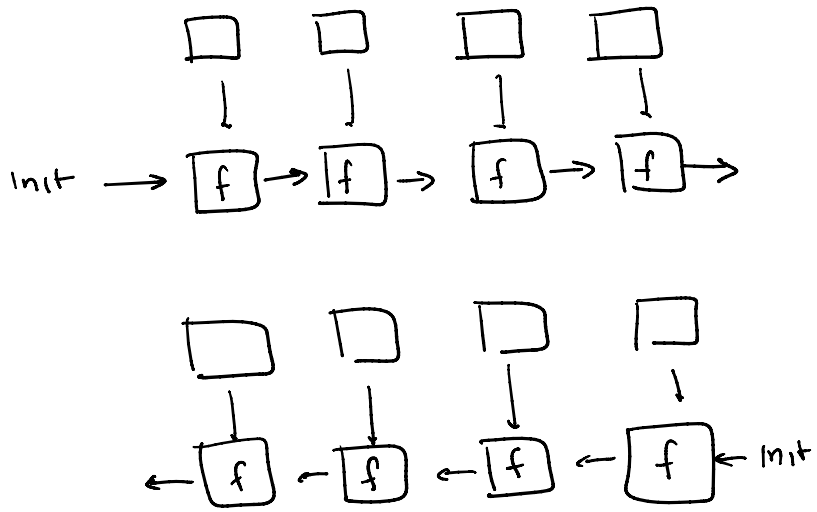
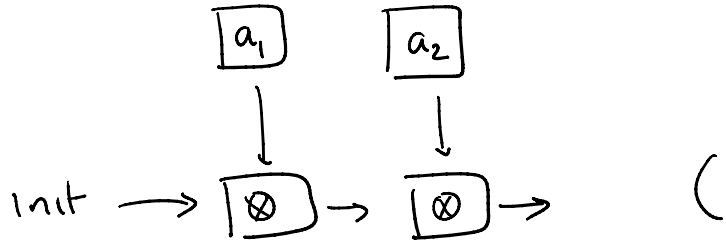
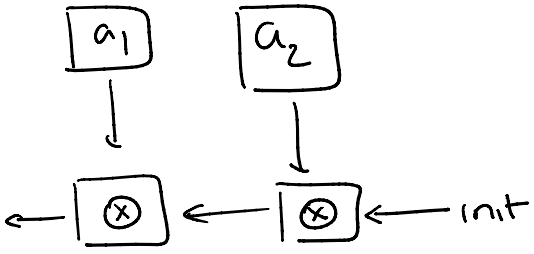
Lecture 8

Thursday, September 16, 2021 2:02 PM





$(init \otimes a_1) \otimes a_2$

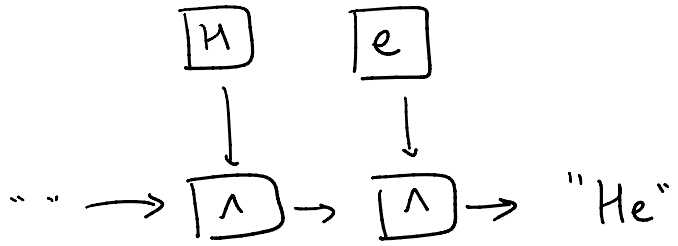


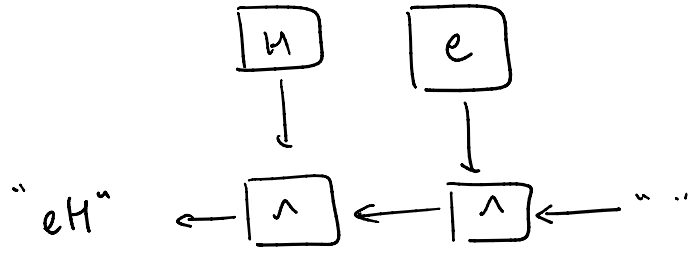
$a_1 \otimes (a_2 \otimes init)$

"Hello" $\frown$ "World" = "HelloWorld"

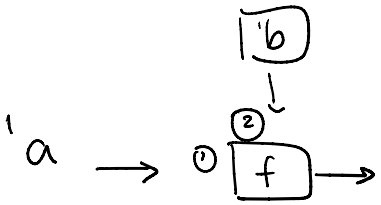
string concatenation

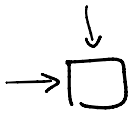
"World" $\frown$ "Hello" = "WorldHello"





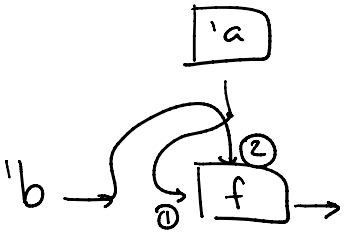

```
fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a
```





fold_left (fun acc le -> ___) ___ ___

```
fold_right : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b
```



fold_right (fun le acc -> ___) ___ ___

```
List.fold_left (fun acc le -> acc ^ le) "" ["H"; "e"; "l"; "l"; o"]
```

"Hello" . Therefore ⟶

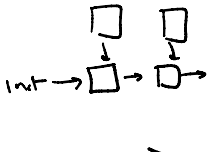
```
List.fold_right (fun le acc -> acc ^ le) ["H"; "e"; "l"; "l"; "o"] ""
```

"olleH" . Therefore ⟵

"stepper"

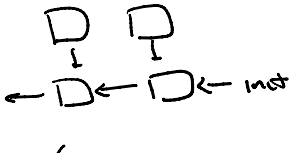
Initial value often chosen to be the identity element of the stepper.

```
let rec fold_left f init l =
  match l with
  | [] -> init
  | hd::tl -> let v = f init hd in fold_left f v tl
              fold_left f (f init hd) tl
```

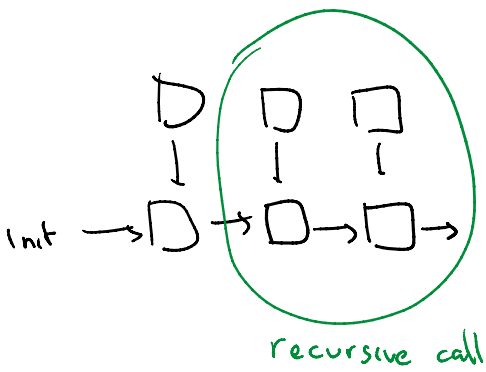


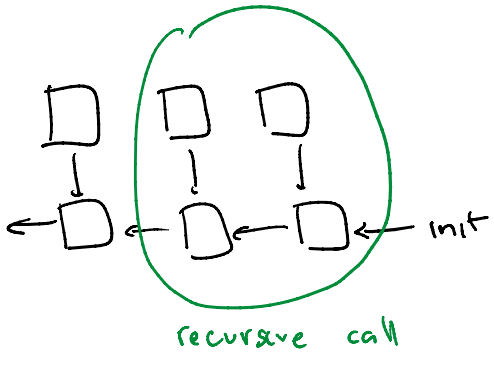
⟶

```
let rec fold_right f l init =
  match l with
  | [] -> init
  | hd::tl -> f hd (fold_right f tl init)
```



⟵





map filter fold group_by ...

Visitor Pattern

- Imagine yourself writing code for a compiler

```
abstract class Expr
class Sum extends Expr
class Diff extends Expr
class Prod extends Expr
  ⋮
```

```
type expr =
  | Sum of expr * expr
  | Diff of expr * expr
  | Prod of expr * expr
      ⋮
```

Expr. find Os

Expr. size

Expr. is_empty

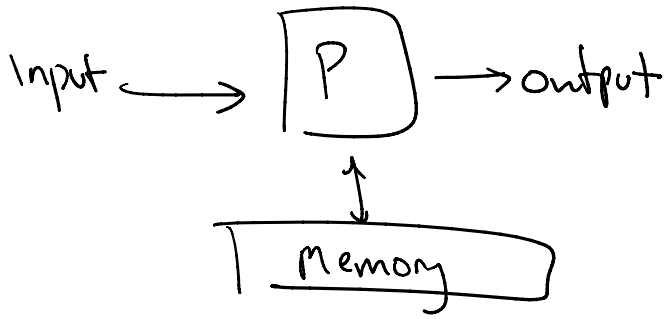
```
Expr . is-empty
    :

new function (Expr e) {
    if e is Sum ——
    else if e is Diff ——
        :
}
```

Introducing Mutable State

"change"

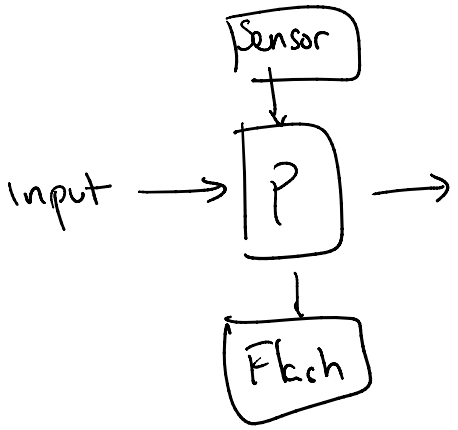
"memory that can change"



"Pure" functional programs

= Disallow changes to memory /

interacting with the outside world



"Identical inputs should evaluate to identical outputs, always" "Pure"

- Simon Peyton Jones + Phil Wadler

  IO Monad.

- Two approaches to mutable state

  1) let mount sessionStr = ~~~

① let print screencap str = ~~~~

in new screencap

World variable

② We can't print to the screen.

But what stops us from speaking about printing to the screen?

"Imagine printing 3 to the screen"

"Now imagine printing 5 to the screen"

⇓

"Imagine printing 3,5 to the screen"

Abstract vocabulary of actions