Lecture 8

```
data List a
  = Nil
  | Cons a (List a)
```

GADT style:

```
data List a where
  Nil :: List a
  Cons :: a -> List a -> List a
```

A data constructor starts with a capital letter, and creates data. eg. Nil, Cons.

```
data [] a
  = []
  | (:) a [a]
```

```
data [] a where
  [] :: [a]
  (:) :: a -> [a] -> [a]
```

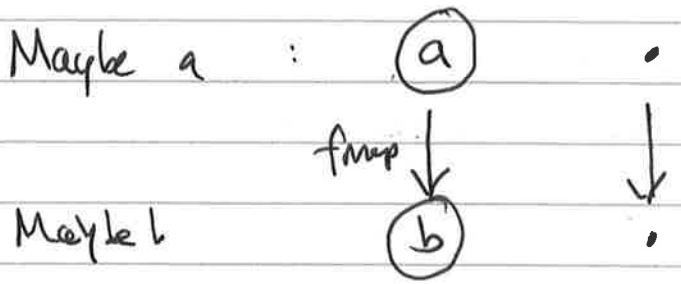
Note: `[] a = [a]` at the type level.

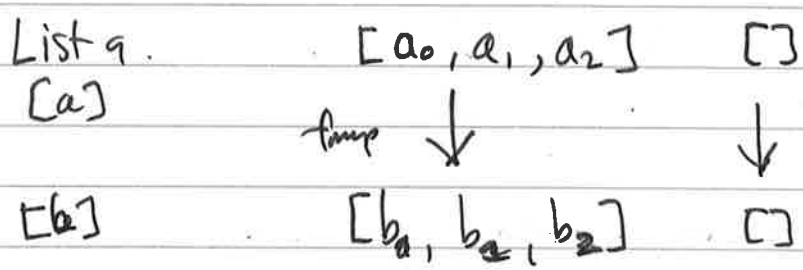
Note:

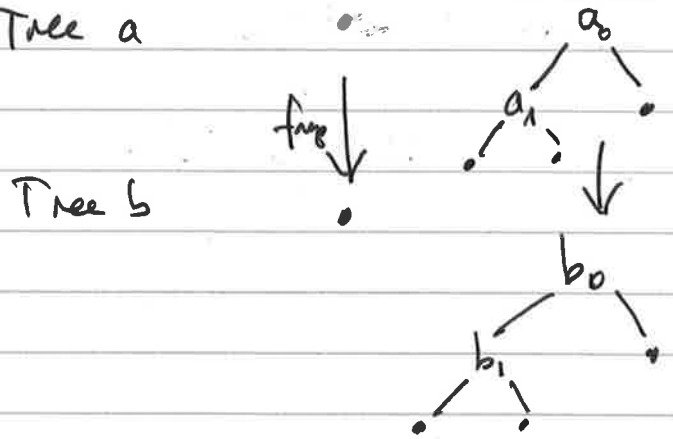
```
Cons x xs
     ↑  ↑
     a  [a]
[a]
```

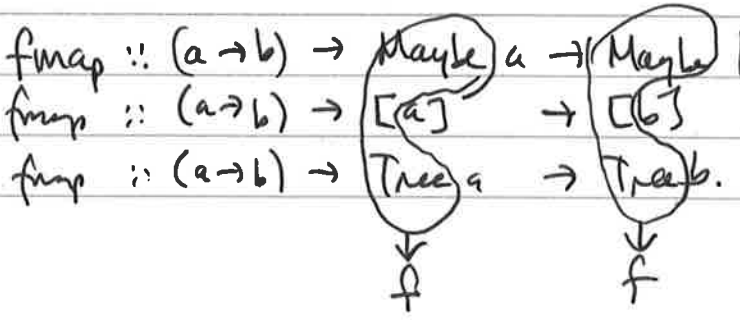
```
(:) x xs = x : xs
    ↑  ↑     ↑  ↑
    a  [a]   a  [a]
[a]          [a]
```

```
data Maybe a where
    Nothing :: Maybe a
    Just    :: a → Maybe a

head :: [a] → a
head []     = error "head empty!"
head (x:xs) = x

headMay :: [a] → Maybe a
headMay []     = Nothing
headMay (x:xs) = Just x

maximum :: [Integer] → Maybe Integer
```

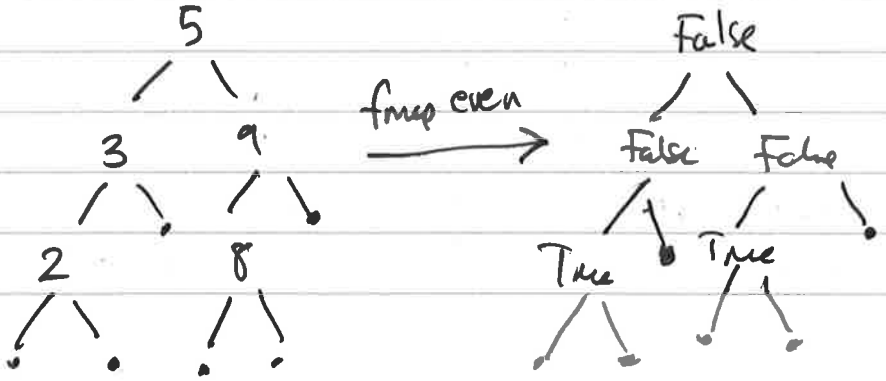

Maybe a
fmap
Maybe b


List a.
[a]
[a0, a1, a2]
[]
fmap
[b]
[b0, b1, b2]
[]


Tree a
fmap
Tree b
a0
a1
b0
b1


fmap :: (a → b) → Maybe a → Maybe b
fmap :: (a → b) → [a] → [b]
fmap :: (a → b) → Tree a → Tree b.
f
f

```
class Functor f where
    fmap :: (a -> b) -> f a -> f b

instance Functor Maybe where
    fmap :: (a -> b) -> Maybe a -> Maybe b
    fmap f Nothing = Nothing
    fmap f (Just x) = Just (f x)

instance Functor Tree where
    fmap :: (a -> b) -> Tree a -> Tree b
    fmap f Tip = Tip
    fmap f (Node l x r) =
            Node (fmap f l)
                 (f x)
                 (fmap f r)
```



Folding arbitrary structure.

```
foldr :: (a→b→b) → b → [a] → b

                        Maybe a → b

                        Tree a → b
```

```
  [] :: [a]
  (:) :: a → [a] → [a]                    } Lists
foldr :: (a→b→b) → b → [a] → b
foldr (:) [] xs = xs
foldr cons nil xs = ...
```

```
  Nothing :: Maybe a
  Just :: a → Maybe a
foldMaybe :: b → (a→b) → Maybe a → b
foldMaybe nothing just Nothing = nothing
foldMaybe nothing just (Just x) = just x
          b       a→b      a       b
```

Example

~~foldMaybeInt :: Maybe Int → Maybe Bool.~~

Example

```
isPositive :: Integer -> Maybe Bool.
isPositive x
    | x < 0   = Just False
    | x == 0  = Nothing
    | x > 0   = Just True .

examplePos :: Maybe Bool -> Integer
examplePos Nothing = 0
examplePos (Just True) = 5
examplePos (Just false) = -5

examplePos :: Maybe Bool -> Integer
examplePos x = foldMaybe nothing just x
    where
        nothing = 0
        just True = 5
        just false = -5.
```

```
Tip :: Tree a
Node :: Tree a → a → Tree a → Tree a

foldTree :: b → (b → a → b → b) → Tree a → b
foldTree tip node Tip = tip
foldTree tip node (Node l x r)
                   ↑ ↑ ↑
              Tree a = Tree a
  = node (foldTree tip node l) x
         (foldTree tip node r)
```