Lecture 2

$$\mathbb{N}X = 1 + X$$
$$L_A X = \underline{1} + A \times X$$
$$T_A X = A + X \times X$$

least fp of functor F is
$(X, in: FX \to X)$ st.
$\forall\ (A, f: FA \to A) \cdot$
$\exists!\ h: X \to A \cdot\ h \circ in = f \circ Fh$

$$\begin{tikzcd}
F\overset{\mu F}{X} \arrow[r, "Fh"] \arrow[d, "in"'] & FA \arrow[d, "f"] \\
\overset{\mu F}{X} \arrow[r, dashed, "\underset{\text{cata} f}{h}"'] & A
\end{tikzcd}$$

write μF for X
cata f for h

List $A = \mu(L_A)$

```
data L a b = Nil
           | Cons a b

data Mu f a = In (f a (Mu f a))
type List a = Mu (L a)

In :: f a (Mu f a) -> Mu f a
in° :: Mu f a -> f a (Mu f a)
In ∘ in° = id        in° ∘ In = id
```

```
class Bifunctor f where
    bimap :: (a→c) → (b→d) → f a b → f c d

instance Bifunctor L where
    bimap f g Nil        = Nil
    bimap f g (Cons x y) = Cons (f x) (g y)

cata :: Bifunctor f ⇒ (f a b → b) → Mu f a → b
```

$$\text{cata phi (In } \underset{f\ a\ (\mathrm{Mu}\ f\ a)}{x}\text{)} = \underbrace{\text{phi } (\underbrace{\text{bimap id (cata phi) } x}_{f\ a\ b})}_{b}$$

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

instance Bifunctor f => Functor (Mu f) where
    fmap f (In x) = In (bimap f (fmap f) x)
```

a→b   Mu f a   f a (Mu f a)   Mu f b   f b (Mu f b)

```
fmap f = cata ( ... )
```

dualize — functor F

least fp ("initial algebra") is

$(X, in : FX \to X)$ s.t. μF

$\forall (A, phi : FA \to A)$,

$\exists! h : X \to A$, cata phi

$h \circ in = phi \circ Fh$

$$\begin{tikzcd} FX \arrow[r, "Fh"] \arrow[d, "in"'] & FA \arrow[d, "phi"] \\ X \arrow[r, dashed, "h"'] & A \end{tikzcd}$$

greatest fp ("final coalgebra") is

$(X, out : X \to FX)$ s.t. νF

$\forall (A, phi : A \to FA)$

$\exists! h : A \to X$, ana phi

$out \circ h = Fh \circ phi$

$$\begin{tikzcd} FX & FA \arrow[l, "Fh"'] \\ X \arrow[u, "out"] & A \arrow[l, dashed, "h"] \arrow[u, "phi"'] \end{tikzcd}$$

cata :: Bifunctor f ⇒ (f a b → b) → (Mu f a → a)

ana :: Bifunctor f ⇒ (b → f a b) → (b → Nu f a)

out (ana phi z) = out ~~(Out°~~ (bimap id (ana phi) (phi z)))

fa(Nu fa) fab b

data Mu f a = In (f a (Mu f a))

<u>data</u> Nu f a = Out° (f a (Nu f a))

out :: Nu f a → f a (Nu f a)

out (Out° x) = x

<u>data</u> ~~T~~ Nu f a = Out° {out :: f a (Nu f a)}

eg data L a b = Nil | Cons a b

type Colist a = Nu L a

range :: (~~Int~~ Integer, ~~Int~~) → Colist ~~Int~~

range = ana next where

next :: (Int,Int) → L Int (Int,Int)

next (m,n)

| m==n = Nil

| otherwise = Cons m (m+1,n)

range(0,3) = [0,1,2]

range(0,0) = []

range(3,2) = [3,4,5...]

```
data U a b = Empty | Fork b a b

type Cotree a = Nu U a

build :: [a] → Cotree a
build = ana next where
    next :: [a] → U a [a]
    next []     = Empty
    next (x:xs) = fork ys x zs where
        ys = [y | y ← xs, y < x]
        zs = [z | z ← xs, z ≥ x]
```

~~data~~

```
                    Out° (fmap (ana' phi) (phi z))

type Conat = Nu' Maybe
ana' :: (b -> Maybe b) -> b -> Conat
ana' phi z = Out°' (fmap (ana' phi) (phi z))
```

```
unfoldConat :: (b -> Maybe b) -> b -> Conat
unfoldConat phi z = case phi z of Nothing -> 0
                                  Just z' -> 1 + unfoldConat* z'

data Nu' f = Out°' { out' :: f (Nu' f) }
ana' :: Functor f => (b -> f b) -> b -> Nu' f
```