Lecture 4

hylomorphisms

Cotree x (Nu)

```
build :: Ord a => [a] -> Tree a
build []     = Empty
build (x:xs) = Fork (build ys) x (build zs) where
    where (ys,zs) = (filter (<x) xs, filter (>=x) xs)
```

Mu

```
flatten :: Tree a -> [a]
flatten Empty        = []
flatten (Fork t x u) = flatten t ++ [x] ++ flatten u
```

Fokkinga & Meijer 1991

data Fix f a = In {out :: f a (Fix f a)}

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

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

hylo :: Bifunctor f ⇒ (f b c → c) → (a → f b a) → a → c

hylo phi psi = cata phi ∘ ana psi

= phi ∘ bimap id (cata phi) ∘ out ∘

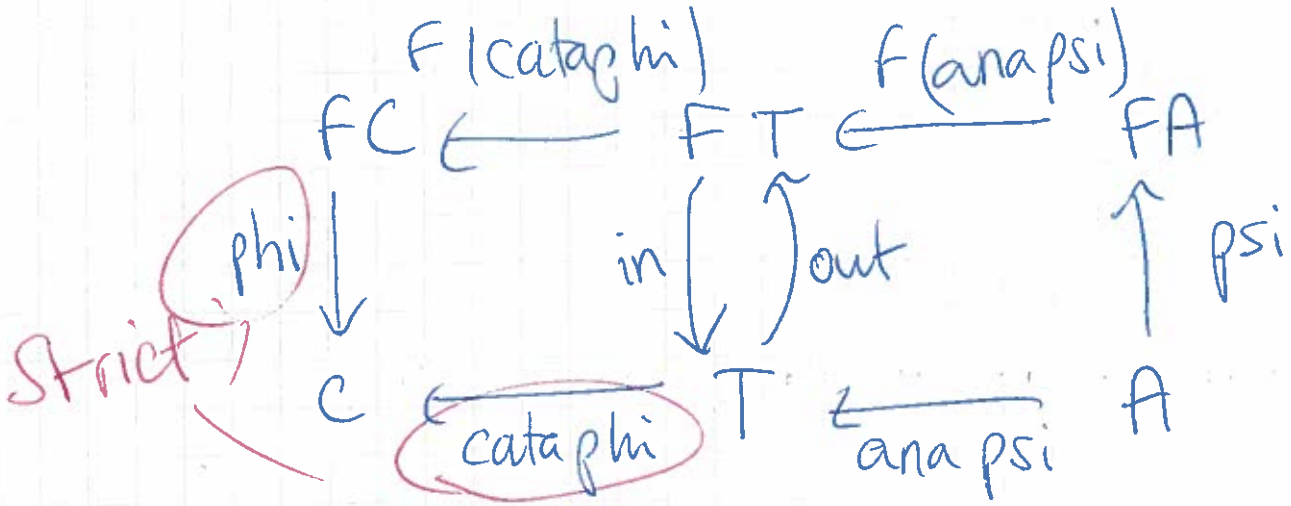
In ∘ bimap id (ana psi) ∘ psi

= phi ∘ bimap id (cata phi ∘ ana psi) ∘ psi

= phi ∘ bimap id (hylo phi psi) ∘ psi

bimap f g ∘ bimap h k
= bimap (f ∘ h) (g ∘ k)

Fokkinga & Meijer — polynomial functor F



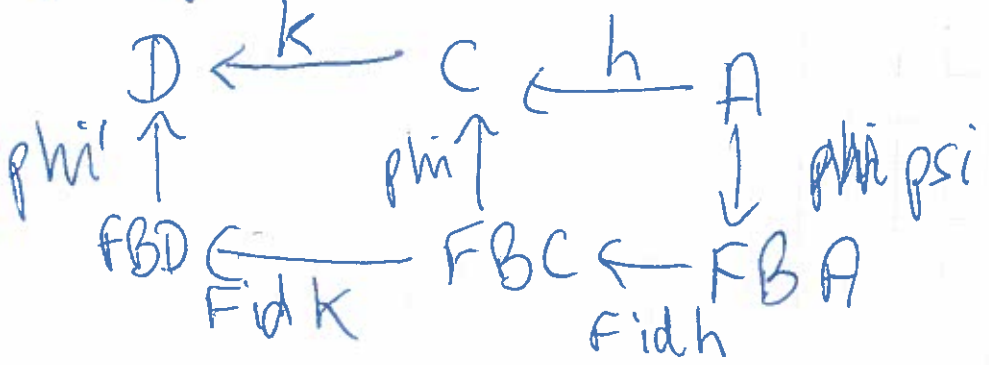
recursive coalgebras Uustalu, Vene, Capretta

$$h = phi \circ bimap\ id\ h \circ psi \quad (*)$$

"hylo equation"

psi is a recursive coalgebra
if (*) has a unique solution for each phi
then:
h = hylo phi psi (=) (*)

fusion



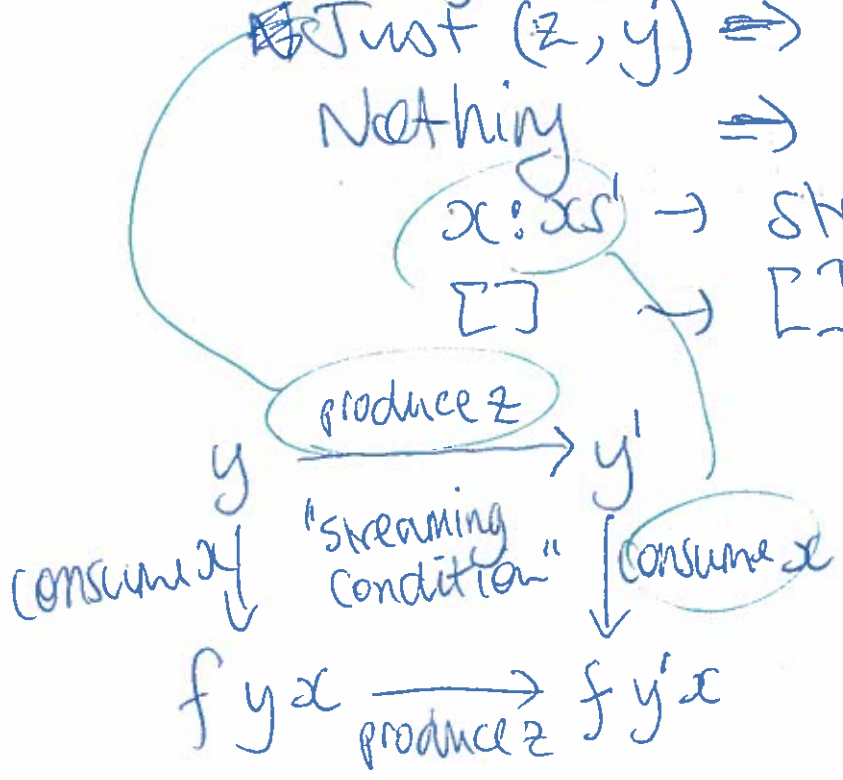
metamorphisms (J.G.)

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr f e []     = e
foldr f e (x:xs) = f x (foldr f e xs)

foldl :: (b -> a -> b) -> b -> [a] -> b
foldl f e []     = e
foldl f e (x:xs) = foldl f (f e x) xs

unfoldr :: (b -> Maybe (a,b)) -> b -> [a]
unfoldr f z = case f z of Nothing -> []
                         Just (x,z') -> x : unfoldr f z'
```

```
stream :: (b → a → b) → b → (b → Maybe (c, b)) → [a] → [c]
stream f e g = unfoldr g ∘ foldl f e
stream f y g xs = case g y of
    Just (z, y') ⇒ z : stream f y' g xs
    Nothing      ⇒ case xs of
        x : xs' → stream f (f y x) g xs'
        []      → []
```



if streaming condition holds,
stream = stream
on finite inputs

```
fstream :: (b -> a -> b) -> b -> (b -> Maybe (c, b)) -> (b -> [c]) ->
           [a] -> [c]
fstream f y g h xs =
    case g y of  Just (z, y') -> z : fstream f y' g h xs
                 Nothing ->
        case xs of  x : xs' -> fstream f (f y x) g h xs'
                    []      -> h y
```

if streaming condition holds, then

fstream f y g h = apo ... g ... h ... . foldl f y

on finite input

Converting from base 3 to base 7

```
fromBase3 :: [Int] → Rat          [0,1,2] ↦ 5/27
fromBase3 = foldr stepr 0 where stepr d x = (d+x)/3

toBase7 :: Rat → [Int]
toBase7 = unfoldr split where
    split x = let y = 7*x in Just (⌊y⌋, y - ⌊y⌋)

fromBase3 = extract . foldl stepl (0,1) where
    stepl (u,v) d = (d + u*3, v/3)

(u,v) ≈ represents (v*x) . (u+)
extract (u,v) = v*u
```

does the streaming condition hold?

extract7 toBase7 ∘ extract
= unfoldr split ∘ extract
= unfoldr split' where
split' (u,v) = let y = ⌊7×u×v⌋ in
Just (y, (u − y/(v*7), v*7))

$0.1_3 \approx 0.222_7$ $0.11_3 \approx 0.305_7$