The aim is to understand this:

\textit{In category theory, the concept of catamorphism denotes the unique homomorphism from an initial algebra into some other algebra.}

To better understand this:

\textit{In functional programming, catamorphisms provide generalizations of folds of lists to arbitrary algebraic data types, which can be described as initial algebras. The dual concept is that of anamorphism that generalize unfolds. A hylomorphism is the composition of an anamorphism followed by a catamorphism.}
The outline

- Basic category theory
Basic category theory
Haskell as a category (if you squint)
The outline

- Basic category theory
- Haskell as a category (if you squint)
- Category theoretic functors, Haskell Functors endofunctors
Basic category theory
Haskell as a category (if you squint)
Category theoretic functors, Haskell Functors endofunctors
Algebras over an endofunctor
The outline

- Basic category theory
- Haskell as a category (if you squint)
- Category theoretic functors, Haskell Functors endofunctors
- Algebras over an endofunctor
- F-Algebra homomorphisms, or arrows in the category of F-algebras
The outline

- Basic category theory
- Haskell as a category (if you squint)
- Category theoretic functors, Haskell Functors endofunctors
- Algebras over an endofunctor
- F-Algebra homomorphisms, or arrows in the category of F-algebras
- Initial objects
The outline

- Basic category theory
- Haskell as a category (if you squint)
- Category theoretic functors, Haskell Functors endofunctors
- Algebras over an endofunctor
- F-Algebra homomorphisms, or arrows in the category of F-algebras
- Initial objects
- Catamorphisms as unique homomorphisms from initial objects
The outline

- Basic category theory
- Haskell as a category (if you squint)
- Category theoretic functors, Haskell Functors endofunctors
- Algebras over an endofunctor
- F-Algebra homomorphisms, or arrows in the category of F-algebras
- Initial objects
- Catamorphisms as unique homomorphisms from initial objects
- Flip the arrows
What’s a category?

Pick some things:

- Objects (X,Y,Z)

Assert some properties:
What’s a category?

Pick some things:

- Objects \((X,Y,Z)\)
- Arrows between objects \((f,g,g \circ f)\)

Assert some properties:
What’s a category?

Pick some things:
- Objects \((X,Y,Z)\)
- Arrows between objects \((f,g,g \circ f)\)

Assert some properties:
- All arrows compose associatively
What’s a category?

Pick some things:

- Objects (X, Y, Z)
- Arrows between objects (f, g, g ∘ f)

Assert some properties:

- All arrows compose associatively
- Every object has an identity arrow
What’s a category?

Pick some things:

- Objects \((X,Y,Z)\)
- Arrows between objects \((f,g,g \circ f)\)

Assert some properties:

- All arrows compose associatively
- Every object has an identity arrow
What’s a category?

Pick some things:

- Objects \((X,Y,Z)\)
- Arrows between objects \((f,g,g \circ f)\)

Assert some properties:

- All arrows compose associatively
- Every object has an identity arrow

Figure 1:
Haskell as sort of a category

Pick some things:

- Objects are types (not values!)

Assert some properties:

\[
(\cdot) :: (b \to c) \to (a \to b) \to a \to c
\]
\[
id :: a \to a
\]

NB: it's lies, all lies!

\[
\text{seq } \text{undefined } () = \text{undefined}
\]
\[
\text{seq } (\text{undefined } \cdot \text{id}) () = ()
\]
Haskell as sort of a category

Pick some things:

- Objects are types (not values!)
- Arrows are functions between types

Assert some properties:

\[
(\cdot) :: (b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c
\]

\[
id :: a \rightarrow a
\]

NB: it's lies, all lies!

\[
\text{seq undefined} \; () = \text{undefined}
\]

\[
\text{seq (undefined . id)} \; () = \; ()
\]
Haskell as sort of a category

Pick some things:

- Objects are types (not values!)
- Arrows are functions between types

Assert some properties:

- All arrows compose associatively

\[(\cdot) :: (b \to c) \to (a \to b) \to a \to c\]
\[\text{id} :: a \to a\]

NB: it's lies, all lies!

\[\text{seq } \text{undefined }() = \text{undefined}\]
\[\text{seq } (\text{undefined} \cdot \text{id}) () = ()\]
Haskell as sort of a category

Pick some things:

- Objects are types (not values!)
- Arrows are functions between types

Assert some properties:

- All arrows compose associatively
- Every object has an identity arrow

\[
(\cdot) :: (b \to c) \to (a \to b) \to a \to c
\]
\[
id :: a \to a
\]

NB: it's lies, all lies!

\[
\text{seq } \text{undefined } () = \text{undefined}
\]
\[
\text{seq } (\text{undefined } \cdot \text{id}) () = ()
\]
Functors, endofunctors

A functor is a mapping between categories that sends objects to objects (types to types) and arrows to arrows (terms to terms), preserving identity arrows and composition, possibly across categories.

$$fmap \ id = \ id$$
$$fmap \ f \ . \ fmap \ g = fmap (f \ . \ g)$$

Endofunctors map from a category to the same category.

In the case of Hask:

```haskell
class Functor (f :: * -> *) where
    fmap :: (a -> b) -> f a -> f b
```
For a category $C$ and endofunctor $F$ an algebra of $F$ is an object $X$ in $C$ and a morphism:

$$\text{alg} : F(X) \rightarrow X$$

$X$ is called the “carrier” of the algebra.

```haskell
-- For a category and endofunctor
data F a = Zero | Succ a

instance Functor F where
  fmap _ Zero = Zero
  fmap f (Succ a) = Succ (f a)

-- An algebra of F is an X in C
type X = Natural

-- And a morphism
alg :: F X -> X
alg Zero = 0
alg (Succ n) = n + 1
```
data F a = Zero | Succ a

alg :: F Natural -> Natural
alg Zero = 0
alg (Succ n) = n + 1

> alg Zero
0
> alg $ Succ $ alg Zero
1
> alg $ Succ $ alg $ Succ $ alg Zero
2
An alternate algebra, same F and C, different X

alg' :: F String -> String
alg' Zero = "!"
alg' (Succ s) = "QUACK" ++ s

> alg' $ Succ $ alg' $ Succ $ alg' Zero
"QUACKQUACK!"
An initial object of a category $C$ is an object $I$ in $C$ such that for every object $X$ in $C$, there exists precisely one morphism $I \to X$. - Wikipedia

(up to isomorphism)
An initial algebra for an endofunctor $F$ on a category $C$ is an initial object in the category of algebras of $F$. 
An initial algebra for an endofunctor $F$ on a category $C$ is an initial object in the category of algebras of $F$.

Category of algebras of $F$:

- Objects: $\text{alg}$, $\text{alg}'$, ...
An initial algebra for an endofunctor $F$ on a category $C$ is an initial object in the category of algebras of $F$.

Category of algebras of $F$:

- **Objects**: $\text{alg, alg}', \ldots$
- **Arrows**: structure preserving maps (homomorphisms) from an algebra to another
Homomorphisms between two algebras.

An arrow in the category of F-algebras of a given endofunctor e.g. between (Natural, alg) and (String, alg’) is a function mapping the carrier in the underlying category (Hask, hom : Natural -> String), such that the following square commutes:

![Diagram](image)

Figure 2:
That is to say that

\[ f_{\text{Nat}} :: F \text{ Natural} \]
\[ f_{\text{Nat}} = \text{Succ } 1 \]

\[ \text{hom} :: \text{Natural} \to \text{String} \]
\[ \text{hom } n = \text{timesN } n \ "\text{QUACK}" \ ++ \ "!" \]

\[
\begin{align*}
> \text{alg } f_{\text{Nat}} & \quad \text{-- 2} \\
> \text{fmap hom } f_{\text{Nat}} & \quad \text{-- Succ } "\text{QUACK}!" \\
> \text{hom } $ \text{alg } f_{\text{Nat}} & \quad \text{-- } "\text{QUACKQUACK}!" \\
> \text{alg'} $ \text{fmap hom } f_{\text{Nat}} & \quad \text{-- } "\text{QUACKQUACK}!" 
\end{align*}
\]

\[ \text{hom } \circ \text{alg} \equiv \text{alg'} \circ \text{fmap hom} \]

\[
\begin{array}{cccc}
F(\text{Natural}) & F(\text{hom}) & F(\text{String}) \\
\text{alg} & \downarrow & \text{alg'} \\
\text{Natural} & \overset{\text{hom}}{\longrightarrow} & \text{String} \\
\end{array}
\]

Figure 3:
An initial algebra for an endofunctor $F$ on a category $C$ is an initial object in the category of algebras of $F$.

Category of algebras of $F$:

- **Objects:** $\text{alg}, \text{alg}', \ldots$
An initial algebra for an endofunctor $F$ on a category $C$ is an initial object in the category of algebras of $F$.

Category of algebras of $F$:

- **Objects**: $\text{alg}$, $\text{alg}'$, $\ldots$
- **Arrows**: structure preserving maps (homomorphisms) from an algebra to another
An initial algebra for an endofunctor $F$ on a category $C$ is an initial object in
the category of algebras of $F$.

Category of algebras of $F$:

- **Objects**: $\text{alg}$, $\text{alg'}$, $\ldots$
- **Arrows**: structure preserving maps (homomorphisms) from an algebra to another
An initial algebra for an endofunctor $F$ on a category $C$ is an initial object in the category of algebras of $F$.

Category of algebras of $F$:

- **Objects**: $\text{alg}$, $\text{alg}'$, $\ldots$
- **Arrows**: structure preserving maps (homo)morphisms from an algebra to another

**Initial**:

- There is a unique morphism from the initial algebra to all other algebras.
The carrier must not “lose” any information, or there is some algebra that it cannot map to.

\[ X \text{ initial} \implies F(X) \cong X \]

\[
\begin{array}{ccc}
F \text{ Init}F & \xrightarrow{F(\text{hom})} & F(\text{String}) \\
\downarrow \text{alg} & & \downarrow \text{alg}' \\
\text{Init}F & \xrightarrow{\text{hom}} & \text{String}
\end{array}
\]

Figure 4:
What fits?

- The carrier must not “lose” any information, or there is some algebra that it cannot map to.
- The carrier can’t add information, or the morphism won’t be unique.

\[ X \text{ initial } \Longrightarrow F(X) \cong X \]

\[ F \text{ InitF } \xrightarrow{F(\text{hom})} F(String) \]

\[ \text{alg} \]

\[ \text{InitF} \xrightarrow{\text{hom}} \text{String} \]

\[ \text{alg'} \]

Figure 4:
What fits?

- The carrier must not “lose” any information, or there is some algebra that it cannot map to.
- The carrier can’t add information, or the morphism won’t be unique.
- The algebra must have type: $F\text{InitF} \rightarrow \text{InitF}$

$X_{\text{initial}} \Rightarrow F(X) \cong X$

![Diagram](image)

Figure 4:
What fits?

- The carrier must not “lose” any information, or there is some algebra that it cannot map to.
- The carrier can’t add information, or the morphism won’t be unique.
- The algebra must have type: $F \text{InitF} \rightarrow \text{InitF}$
- Lambek’s theorem says that if there is an initial object, it is isomorphic to the carrier via the algebra

$X \text{ initial} \implies F(X) \cong X$

![Diagram](image)

Figure 4:
What fits?

- The carrier must not “lose” any information, or there is some algebra that it cannot map to.
- The carrier can’t add information, or the morphism won’t be unique.
- The algebra must have type: \( F \text{InitF} \rightarrow \text{InitF} \)
- Lambek’s theorem says that if there is an initial object, it is isomorphic to the carrier via the algebra
- data \( \text{InitF} = \text{InitF} (F \text{InitF}) \)

\[
X \text{ initial} \implies F(X) \cong X
\]

\[
\begin{array}{c}
F \text{InitF} \\
\downarrow \text{alg} \\
\text{InitF}
\end{array} \xrightarrow{F(\text{hom})} \begin{array}{c}
F(\text{String}) \\
\downarrow \text{alg'} \\
\text{String}
\end{array}
\]

Figure 4:
More generally...

data Fix f = Roll { unRoll :: f (Fix f) }

type InitF = Fix F

Roll :: F InitF -> InitF

unRoll :: InitF -> F InitF

fix2 :: InitF

fix2 = Roll $ Succ $ Roll $ Succ $ Roll Zero

If this is the initial object in the category of algebras, there must be a unique arrow from InitF to every algebra:

∀ algebras \exists hom : InitF \rightarrow carrier of algebra
data Fix f = Roll { unRoll :: f (Fix f) }

type InitF = Fix F

Roll :: F InitF -> InitF
unRoll :: InitF -> F InitF

If this is the initial object in the category of algebras, there must be a unique arrow from InitF to every algebra:

\[ \forall \text{algebras} \; \exists \text{hom}: \text{InitF} \rightarrow \text{carrier of algebra} \]
More generally...

data Fix f = Roll { unRoll :: f (Fix f) }

type InitF = Fix F
Roll :: F InitF -> InitF
unRoll :: InitF -> F InitF

fix2 :: InitF
fix2 = Roll $ Succ $ Roll $ Succ $ Roll Zero
More generally...

data Fix f = Roll { unRoll :: f (Fix f) }
type InitF = Fix F
Roll :: F InitF -> InitF
unRoll :: InitF -> F InitF

fix2 :: InitF
fix2 = Roll $ Succ $ Roll $ Succ $ Roll Zero

If this is the initial object in the category of algebras, there must be a unique arrow from InitF to every algebra:

\[ \forall \text{algebras} \ \exists \text{hom} : \text{InitF} \rightarrow \text{carrier of algebra} \]
The unique homomorphism

\[ \forall \text{algebras } \exists \text{hom} : \text{InitF} \to \text{carrier of algebra} \]

\[
\begin{align*}
F \text{InitF} & \longrightarrow F(\text{Natural}) \\
\text{Roll} & \\
\text{InitF} & \longrightarrow \text{Natural}
\end{align*}
\]

*Figure 5:*
The unique homomorphism

∀ algebras ∃ hom : InitF → carrier of algebra

Figure 5:
The unique homomorphism

∀ algebras ∃ hom : InitF → carrier of algebra

Figure 5:
The unique homomorphism

\[ \forall \text{algebras } \exists \text{hom} : \text{InitF} \to \text{carrier of algebra} \]

\[
\begin{align*}
F \text{ InitF} & \xrightarrow{F(\text{hom})} F(\text{Natural}) \\
\text{unRoll} & \downarrow \quad \text{alg} \\
\text{InitF} & \xrightarrow{\text{hom}} \text{Natural}
\end{align*}
\]

unRoll :: InitF \to F \text{ InitF}

Figure 6:
The unique homomorphism

\[ \forall \text{algebras } \exists \text{hom } : \text{InitF } \rightarrow \text{carrier of algebra} \]

**Figure 6:**

\[
\begin{array}{ccc}
F \text{InitF} & \xrightarrow{F(\text{hom})} & F(\text{Natural}) \\
\downarrow \text{unRoll} & & \downarrow \text{alg} \\
\text{InitF} & \xrightarrow{\text{hom}} & \text{Natural}
\end{array}
\]

\[ \text{unRoll} :: \text{InitF } \rightarrow F \text{InitF} \]

\[ \text{hom} :: \text{InitF } \rightarrow \text{Natural} \]

\[ \text{hom} = \text{alg } \cdot \text{fmap hom } \cdot \text{unRoll} \]
The unique homomorphism

∀ algebras ∃ hom : InitF → carrier of algebra

\[ F \text{ InitF} \xrightarrow{F(hom)} F(\text{Natural}) \]

\[ \text{unRoll} \]

\[ \text{alg} \]

\[ F \text{ InitF} \xrightarrow{\text{hom}} \text{Natural} \]

**Figure 6:**

\[ \text{unRoll} :: \text{InitF} \rightarrow F \text{ InitF} \]

\[ \text{hom} :: \text{InitF} \rightarrow \text{Natural} \]

\[ \text{hom} = \text{alg . fmap hom . unRoll} \]

\[ \text{cata} :: \text{Functor } f \Rightarrow (f \ a \rightarrow a) \rightarrow \text{Fix } f \rightarrow a \]

\[ \text{cata alg} = \]

\[ \text{alg . fmap (cata alg) . unRoll} \]

\[ \text{hom} = \text{cata alg} \]
**Evaluation of cata**

cata :: Functor f
    => (f a -> a) -> Fix f -> a

cata alg =
    alg . fmap (cata alg) . unRoll

alg :: F Natural -> Natural
alg Zero = 0
alg (Succ n) = n + 1
evaluation of cata

cata :: Functor f
  => (f a -> a) -> Fix f -> a
cata alg =
  alg . fmap (cata alg) . unRoll

cata alg (Roll $ Succ $ Roll Zero)

alg :: F Natural -> Natural
alg Zero = 0
alg (Succ n) = n + 1
Evaluation of cata

cata :: Functor f
    => (f a -> a) -> Fix f -> a
cata alg =
    alg . fmap (cata alg) . unRoll

alg :: F Natural -> Natural
alg Zero = 0
alg (Succ n) = n + 1

cata alg (Roll $ Succ $ Roll Zero)

alg $ fmap (cata alg) (Succ $ Roll Zero)
Evaluation of cata

cata :: Functor f  
    => (f a -> a) -> Fix f -> a

cata alg =
    alg . fmap (cata alg) . unRoll

alg :: F Natural -> Natural
alg Zero = 0
alg (Succ n) = n + 1

cata alg (Roll $ Succ $ Roll Zero)

alg $ fmap (cata alg) (Succ $ Roll Zero)

alg $ Succ $ cata alg $ Roll Zero
Evaluation of cata

cata :: Functor f
    => (f a -> a) -> Fix f -> a
cata alg =
    alg . fmap (cata alg) . unRoll

alg :: F Natural -> Natural
alg Zero = 0
alg (Succ n) = n + 1

cata alg (Roll $ Succ $ Roll Zero)

alg $ fmap (cata alg) (Succ $ Roll Zero)

alg $ Succ $ cata alg $ Roll Zero

alg $ Succ $ alg $ fmap (cata alg) Zero
**Evaluation of cata**

\[
\text{cata} :: \text{Functor } f \\
\quad \Rightarrow (f \ a \rightarrow a) \rightarrow \text{Fix } f \rightarrow a \\
\text{cata alg =} \\
\quad \text{alg} \cdot \text{fmap (cata alg)} \cdot \text{unRoll}
\]

\[
\text{alg} :: F \ \text{Natural} \rightarrow \text{Natural} \\
\quad \text{alg Zero} = 0 \\
\quad \text{alg (Succ } n) = n + 1
\]

\[
\text{cata alg (Roll}$ $\$ Succ $ Roll Zero)$
\]

\[
\text{alg}$ $\text{fmap (cata alg)} (\text{Succ}$ $\$ Roll Zero)$
\]

\[
\text{alg}$ $\text{Succ}$ $\text{alg}$ $\text{Roll Zero}$
\]

\[
\text{alg}$ $\text{Succ}$ $\text{alg}$ $\text{fmap (cata alg)}$ Zero
\]

\[
\text{alg}$ $\text{Succ}$ $\text{alg}$ $\text{Zero}$
\]
This is recursion in a general sense

data Nat a = Succ a | Zero
This is recursion in a general sense

```haskell
data Nat a = Succ a | Zero

data String a = Cons Char a | End
```
This is recursion in a general sense

```haskell
data Nat a = Succ a | Zero

data String a = Cons Char a | End

data BinaryTree a = Branch a a | Tip
```
This is recursion in a general sense

```haskell
data Nat a = Succ a | Zero

data String a = Cons Char a | End

data BinaryTree a = Branch a a | Tip

data RoseTree a = Branches [a] | Tip
```
This is recursion in a general sense

data Nat a = Succ a | Zero

data String a = Cons Char a | End

data BinaryTree a = Branch a a | Tip

data RoseTree a = Branches [a] | Tip

data Group a = Action a a | Inv a | Unit
class Calculator a where
    lit :: Int -> a
    add :: a -> a -> a
    mult :: a -> a -> a

instance Calculator Int where
    lit = id
    add = (+)
    mult = (*)

instance Calculator String where
    lit = show
    add s1 s2 = s1 ++ " + " ++ s2
    mult s1 s2 = s1 ++ " x " ++ s2
Hutton’s razor - F algebra

data Calculator a = Lit Int | Add a a | Mult a a deriving Functor

evalAlg :: Calculator Int -> Int
evalAlg (Lit i) = i
evalAlg (Add i1 i2) = i1 + i2
evalAlg (Mult i1 i2) = i1 * i2

ppAlg :: Calculator String -> String
ppAlg (Lit i) = show i
ppAlg (Add s1 s2) = s1 ++ " + " ++ s2
ppAlg (Mult s1 s2) = s1 ++ " x " ++ s2

pp :: Fix Calculator -> String
pp = cata ppAlg
alg :: F Natural -> Natural
alg Zero = 0
alg (Succ n) = n + 1
Damn the torpedos, flip the arrows

```
alg :: F Natural -> Natural
alg Zero = 0
alg (Succ n) = n + 1

coalg :: Natural -> F Natural
coalg 0 = Zero
coalg n = Succ (n - 1)
```
Damn the torpedos, flip the arrows

\[
\text{alg} :: \text{F Natural} \rightarrow \text{Natural} \\
\text{alg Zero} = 0 \\
\text{alg (Succ n)} = n + 1
\]

\[
\text{coalg} :: \text{Natural} \rightarrow \text{F Natural} \\
\text{coalg 0} = \text{Zero} \\
\text{coalg n} = \text{Succ (n - 1)}
\]

For a category C and endofunctor F a co-algebra of F is an object X in C and a morphism:

\[ \text{coalg} : X \rightarrow F(X) \]
Morphisms on coalgebras: co all of the things!

Figure 7:

\[ F(\text{Natural}) \xrightarrow{F(\text{hom})} F(\text{String}) \]

\[ \downarrow \text{alg} \quad \downarrow \text{alg'} \]

\[ \text{Natural} \xrightarrow{\text{hom}} \text{String} \]

Figure 8:
Morphisms on coalgebras: co all of the things!

Figure 7:

\[ F(\text{Natural}) \xrightarrow{F(\text{hom})} F(\text{String}) \]

\[ \text{alg} \quad \downarrow \quad \text{alg}' \]

\[ \text{Natural} \xrightarrow{\text{hom}} \text{String} \]

Figure 8:

\[ F(\text{Natural}) \rightleftarrows F(\text{String}) \]

\[ \text{coalg} \quad \downarrow \quad \text{coalg}' \]

\[ \text{Natural} \xleftarrow{\text{hom}'} \text{String} \]
E.g.

\[
\begin{align*}
F(Natural) & \xrightarrow{\text{coalg}} F(String) \\
\text{coalg} & : \text{Natural} \to F \text{ Natural} \\
\text{coalg} 0 & = \text{Zero} \\
\text{coalg} n & = \text{Succ} (n - 1)
\end{align*}
\]

Figure 9:
E.g.,

\[ F(\text{Natural}) \leftrightarrow F(\text{String}) \]

**Figure 9:**

\[
\begin{align*}
\text{coalg} & : \text{Natural} \to F(\text{Natural}) \\
\text{coalg} \ 0 & = \text{Zero} \\
\text{coalg} \ n & = \text{Succ} \ (n - 1) \\
\\
\text{coalg}' & : \text{String} \to F(\text{String}) \\
\text{coalg}' \ "!" & = \text{Zero} \\
\text{coalg}' \ ('Q': 'U': 'A': 'C': 'K': \text{xs}) & = \text{Succ} \ \text{xs}
\end{align*}
\]
E.g.

\( F(\text{Natural}) \leftarrow F(\text{String}) \)

\[ \text{hom} :: \text{Natural} \rightarrow \text{String} \]
\[ \text{hom} n = \text{timesN} n \ "QUACK" ++ "!" \]

Figure 10:
E.g.

```
F(Natural) \rightarrow\leftarrow F(String)
```

- **F(\text{hom}')**: \text{coalg}'
- **F(\text{hom})**: \text{coalg}

\text{hom} :: \text{Natural} \rightarrow \text{String}
\text{hom} \ n = \ \text{timesN} \ n \ "\text{QUACK}" \ ++ \ "!"

\text{hom'} :: \text{String} \rightarrow \text{Natural}
\text{hom'} \ \text{str} =
(\text{fromIntegral} \ (\text{length} \ \text{str}) - 1)
`\div` 5

**Figure 10:**
E.g.

\[
\begin{array}{c}
F(\text{Natural}) & \xleftarrow{F(\text{hom}')}& F(\text{String}) \\
& \uparrow & \\
\text{hom} & \downarrow & \text{coalg}' \\
\text{Natural} & \leftarrow & \text{String}
\end{array}
\]

**Figure 10:**

\[
\begin{align*}
\text{hom} & : \text{Natural} \rightarrow \text{String} \\
\text{hom} \ n & = \text{timesN} \ n \ "\text{QUACK}" \ +\ + \ !
\end{align*}
\]

\[
\begin{align*}
\text{hom}' & : \text{String} \rightarrow \text{Natural} \\
\text{hom}' \ \text{str} & = \\
& (\text{fromIntegral} \ (\text{length} \ \text{str}) - 1) \div 5
\end{align*}
\]

\[
\begin{align*}
> (\text{hom'} \ "\text{QUACKQUACK}!", \text{coalg'} \ "\text{QUACKQUACK}!") \\
= (2, \text{Succ} \ "\text{QUACK}!")
\end{align*}
\]

\[
\begin{align*}
> (\text{coalg} \ \$ \ \text{hom'} \ "\text{QUACKQUACK}!", \ \text{fmap} \ \text{hom'} \ \$ \ \text{coalg'} \ "\text{QUACKQUACK}!") \\
= (\text{Succ} \ 1, \text{Succ} \ 1)
\end{align*}
\]
The unique homomorphism

\[ \forall \text{algebras } \exists \text{hom} : \text{carrier of algebra } \rightarrow \text{TermF} \]

\[
\begin{array}{ccc}
F & \text{TermF} & F(\text{hom'}) \\
\text{TermF} & \text{Natural} & \text{Natural} \\
\end{array}
\]

\[ \text{unRoll} \quad \text{hom'} \quad \text{coalg} \]

**Figure 11:**
The unique homomorphism

∀ algebras ∃ hom : carrier of algebra → TermF

Figure 11:

type TermF = InitF
unRoll :: TermF → F TermF
The unique homomorphism

∀algebras ∃hom : carrier of algebra → TermF

Figure 11:

type TermF = InitF
unRoll :: TermF → F TermF

hom' :: Natural → TermF
The unique homomorphism

\[ \forall \text{algebras } \exists \text{hom } : \text{carrier of algebra } \rightarrow \text{TermF} \]

**Figure 12:**

\[
\begin{array}{c}
F \text{TermF} \\
\text{Roll} \\
\downarrow \\
\text{TermF}
\end{array}
\quad
\begin{array}{c}
F(\text{hom}') \\
\text{coalg} \\
\uparrow \\
\text{Natural}
\end{array}
\quad
\begin{array}{c}
F(\text{Natural}) \\
\text{Roll} \\
\downarrow \\
\text{F TermF}
\end{array}
\]

**Roll** :: F TermF \(\rightarrow\) TermF
The unique homomorphism

\[ \forall \text{algebras} \ \exists \text{hom} : \text{carrier of algebra} \to \text{TermF} \]

Roll :: F \text{TermF} \to \text{TermF}

hom' :: \text{Natural} \to \text{TermF}

Figure 12:
The unique homomorphism

∀ algebras ∃ hom : carrier of algebra → TermF

\[
F \text{ Term}F \leftarrow F(\text{Natural})
\]

Roll :: F TermF → TermF

\text{hom}' :: \text{Natural} → \text{TermF}

\text{hom}' = \text{Roll . fmap hom' . coalg}
The unique homomorphism

∀ algebras ∃ hom : carrier of algebra → TermF

Roll :: F TermF → TermF

hom' :: Natural → TermF

hom' = Roll . fmap hom' . coalg

ana :: Functor f
    => (a → f a) → a → Fix f
ana coalg = Roll . fmap (ana coalg) . f
Whilst we’re here

cata :: Functor f => (f a -> a) -> InitF -> a
   cata alg = alg . fmap (cata alg) . unRoll

ana :: Functor f => (a -> f a) -> a -> TermF
   ana coalg = Roll . fmap (ana coalg) . coalg
Whilst we’re here

cata :: Functor f => (f a -> a) -> InitF -> a
cata alg = alg . fmap (cata alg) . unRoll

ana :: Functor f => (a -> f a) -> a -> TermF
ana coalg = Roll . fmap (ana coalg) . coalg

cata alg' $ ana coalg' $ hom 3
> "QUACKQUACKQUACK!"
Whilst we’re here

cata :: Functor f => (f a -> a) -> InitF -> a
    cata alg = alg . fmap (cata alg) . unRoll

ana :: Functor f => (a -> f a) -> a -> TermF
    ana coalg = Roll . fmap (ana coalg) . coalg

    cata alg' $ ana coalg' $ hom 3
> "QUACKQUACKQUACK!"

hylo :: Functor f => (f b -> b) -> (a -> f a) -> a -> b
    hylo alg coalg = alg . fmap (hylo alg coalg) . coalg
Whilst we’re here

```haskell
cata :: Functor f => (f a -> a) -> InitF -> a
cata alg = alg . fmap (cata alg) . unRoll

ana :: Functor f => (a -> f a) -> a -> TermF
ana coalg = Roll . fmap (ana coalg) . coalg
```

```haskell
hylo :: Functor f => (f b -> b) -> (a -> f a) -> a -> b
hylo alg coalg = alg . fmap (hylo alg coalg) . coalg
```

```haskell
cata alg' $ ana coalg' $ hom 3
> "QUACKQUACKQUACK!"
```

```haskell
hylo alg' coalg' $ hom 3
> "QUACKQUACKQUACK!"
```