Category theory and its applications - ITI9200

https://compose.ioc.ee

TODAY'S TOPIC: functors!

The list functor

An elementary example: consider the following snippet of Haskell

```
data List a
  = Nil
  | (:) a (List a)
  deriving Show

map :: ((a -> b)) -> List a -> List b
map f Nil = Nil
map f (a:as) = f a : (map f as)

instance Functor List where
  fmap = map
```

The list functor

An elementary example: consider the following snippet of Haskell

```haskell
data List a
  = Nil
  | (:) a (List a)
  deriving Show

map :: (a -> b) -> List a -> List b
map f Nil = Nil
map f (a:as) = f a : (map f as)

instance Functor List where
  fmap = map
```

```
map :: (a -> b)
```

The list functor

An elementary example: consider the following snippet of Haskell

```haskell
data List a
  = Nil
  | (:) a (List a)
  deriving Show

map :: (a -> b) -> List a -> List b
map f Nil = Nil
map f (a:as) = f a : (map f as)

instance Functor List where
  fmap = map
```

```
map :: (a -> b) -> List a
```

The list functor

An elementary example: consider the following snippet of Haskell

```haskell
data List a
  = Nil
  | (:) a (List a)
  deriving Show

map :: (a -> b) -> List a -> List b
map f Nil = Nil
map f (a:as) = f a : (map f as)

instance Functor List where
  fmap = map
```

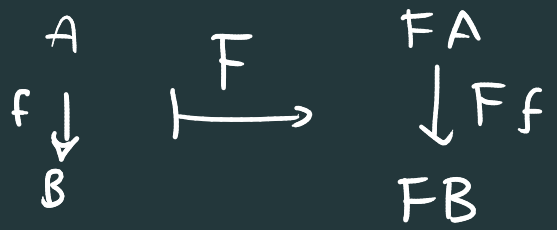
F sends objects to objects

morphism to morphism

```
                  f          ā          f ā
map :: (a -> b) -> List a -> List b
<$>               F          F
```

A functor is a correspondence $\mathcal{C} \xrightarrow{F} \mathcal{D}$ between categories sending objects of $\mathcal{C}$ to objects of $\mathcal{D}$

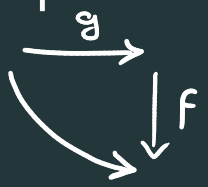
…like `List` does: `A ↦ List A` (type)



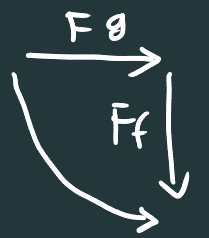
morphisms of $\mathcal{C}$ to morphisms of $\mathcal{D}$

…like `List` does: `map (f :: A -> B) (as :: List A) :: List B`

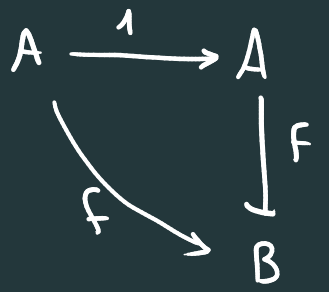
and preserving compositions and identities, i.e.



$F(g\,;f) = Fg\,;Ff$



$F(1_A) = 1_{FA}$




```
compLaw f g as =
  fmap (f . g) as :~: fmap f (fmap g as)
    F

idLaw as =
  fmap id as :~: as
```

Definition (Functor)
Let $\mathcal{C}, \mathcal{D}$ be two categories; a functor $F : \mathcal{C} \to \mathcal{D}$ consists of a pair (F_0, F_1) as follows:

- F_0 is a function $\mathcal{C}_o \to \mathcal{D}_o$ sending an object $C \in \mathcal{C}_o$ to an object $FC \in \mathcal{D}_o$;
- F_1 is a family of functions $F_{AB} : \mathcal{C}(A, B) \to \mathcal{D}(FA, FB)$ (set), one for each pair of objects $A, B \in \mathcal{C}_o$, sending each arrow $f : A \to B$ into an arrow $Ff : FA \to FB$, and such that:
  - $F_{AA}(1_A) = 1_{FA}$;
  - $F_{AC}(g \circ_{\mathcal{C}} f) = F_{BC}(g) \circ_{\mathcal{D}} F_{AB}(f)$.

One can translate these conditions into commutative diagrams

EQUATIONAL $\leftrightsquigarrow$ DIAGRAMS (THAT COMMUTE!)

$c : \mathcal{C}(A,B) \times \mathcal{C}(B,C) \to \mathcal{C}(A,C)$

$(A \xrightarrow{f} B)(B \to C) \mapsto g \cdot f$

$f ; g$

Examples

$H : \mathcal{C} \longrightarrow \mathbf{Set}$

$\mathcal{C}$ a category; functors with codomain Set are very useful and easy to define:

$h_A : \mathcal{C} \longrightarrow \mathbf{Set}$ $\quad h_{A,0} : X \mapsto \mathcal{C}(A,X)$ $\quad h_{A,1}$

- the covariant hom functor $\hom_{\mathcal{C}}(A, -)$ at an object A:

$(f : X \to Y) \longmapsto \mathcal{C}(A,X) \longrightarrow \mathcal{C}(A,Y)$

$(u : A \to X) \longmapsto (f \cdot u : A \xrightarrow{u} X \xrightarrow{f} Y)$

$g \cdot (f \cdot u) = (g \cdot f) \cdot u$

- the contravariant hom functor $\hom_{\mathcal{C}}(-, A)$ at an object A:

$h^A : X \longmapsto \mathcal{C}(X,A)$; $(f : X \to Y) \mapsto \mathcal{C}(X,A) \longleftarrow \mathcal{C}(Y,A)$

$(X \xrightarrow{f} Y \to A) \longleftarrow\!\shortmid\ v : Y \to A$

a functor whose domain is the opposite category $\mathcal{C}^{\mathrm{op}}$ is called contravariant. The arcane saying that, in a sense, these functors are points of $\mathcal{C}$ will appear clearer later.

Examples

Let M be a monoid; a functor $\underline{a} : \underline{M} \to \mathrm{Set}$ consists of the following data:

- A correspondence on objects, sending the unique object $\underline{\bullet}$ of M to a set A;

Examples

Let M be a monoid; a functor $a : M \to \mathrm{Set}$ consists of the following data:

- A correspondence on objects, sending the unique object $\bullet$ of M to a set A;
- A correspondence on morphisms, sending each $m \in M$ to a function $a(m) : A \to A$ with the property that
  - $a(m \cdot m') = a(m) \circ a(m')$ for every $m, m' \in M$;
  - $a(1)$ is the identity function of A.

Examples

Let M be a monoid; a functor $a : M \to \mathrm{Set}$ consists of the following data:

- A correspondence on objects, sending the unique object $\bullet$ of M to a set A;
- A correspondence on morphisms, sending each $m \in M$ to a function $a(m) : A \to A$ with the property that
  - $a(m \cdot m') = a(m) \circ a(m')$ for every $m, m' \in M$;
  - $a(1)$ is the identity function of A.

  This is just a well-known mathematical structure in disguise, a set equipped with an action of the monoid M:

$$\{\text{functors } M \to \mathrm{Set}\} \cong \{\text{sets with an } M \text{ action}\}$$

$\mathbb{N} = \{0, 1, 2 \ldots\}$ $\quad \bullet \overset{a_0}{\longmapsto} \mathbb{R} \quad x \overset{a_1}{\longmapsto} x+1$

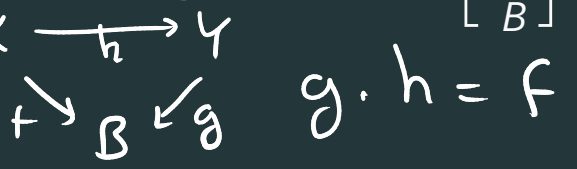
$(\mathbb{N}, +) \quad 1 \in \mathbb{N} \longmapsto \mathbb{R} \to \mathbb{R}$

} defines exactly a functor $\mathbb{N} \to$ Set or an action of $\mathbb{N}$ over $\mathbb{R}$

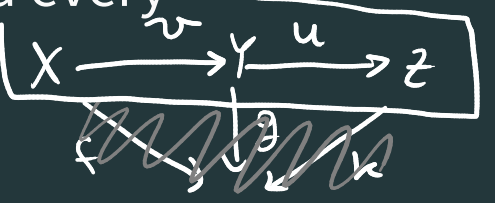
real numb

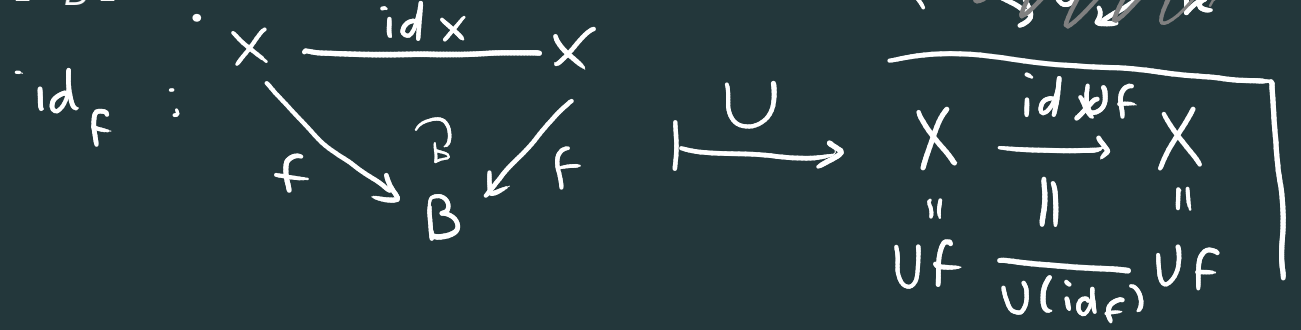
Examples

Recall the definition of the slice category of $\mathcal{C}$ over an object B: objects are morphisms $f : X \to B$, and morphisms $h : \begin{bmatrix} X \\ f\downarrow \\ B \end{bmatrix} \to \begin{bmatrix} Y \\ g\downarrow \\ B \end{bmatrix}$ are $h : X \to Y$ such that $gh = f$.

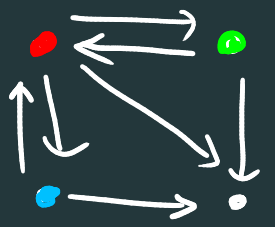


There exists a 'domain' functor U from the category $\mathcal{C}/B$ to $\mathcal{C}$, sending each object $\begin{bmatrix} X \\ f\downarrow \\ B \end{bmatrix}$ to its domain X, and every $h : \begin{bmatrix} X \\ f\downarrow \\ B \end{bmatrix} \to \begin{bmatrix} Y \\ g\downarrow \\ B \end{bmatrix}$ to itself as a morphism in $\mathcal{C}$.





Examples



Let K be the complete graph on a set K of colors (here, four: red, green, blue, white); the slice category $\mathbf{dGph}/K$ consists of (directed) graphs $\mathcal{G}$ with a choice of coloring from the set K, and homomorphisms are homomorphisms of digraphs that preserve colorings.

There is a functor $\mathbf{dGph}/K \xrightarrow{U} \mathbf{dGph}$ that forgets coloring.

There is a functor $\mathbf{dGph} \to \mathbf{dGph}/K$ that endows a graph with the trivial coloring.

Examples

A simple polynomial is a functor $F : \mathrm{Set} \to \mathrm{Set}$ that is defined from the following inductive rules:

$X \longmapsto FX$ "is a polynomial"

- the identity functor is a simple polynomial;

Examples

A simple polynomial is a functor $F : \mathrm{Set} \to \mathrm{Set}$ that is defined from the following inductive rules:

- the identity functor is a simple polynomial;
- every constant functor is a simple polynomial;

Examples

A simple polynomial is a functor $F : \mathrm{Set} \to \mathrm{Set}$ that is defined from the following inductive rules:

- the identity functor is a simple polynomial;
- every constant functor is a simple polynomial;
- the product $F \times G$ of two simple polynomials is simple;

$$(F \times G)(X) = FX \times GX$$

$$(u, v)$$

$$Ff \times Gf$$

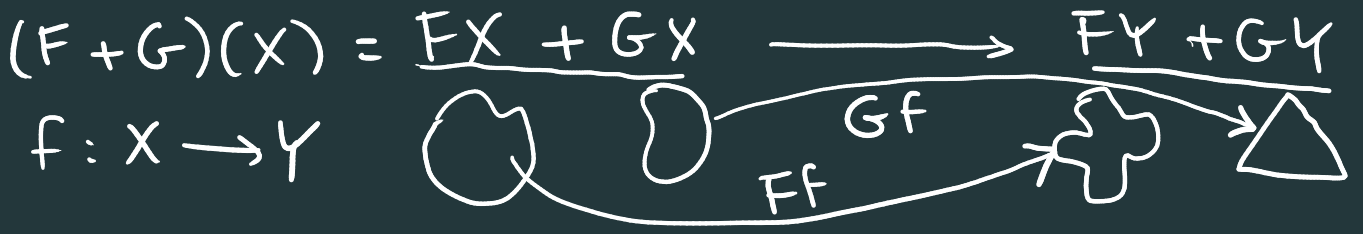
$$FY \times GY$$

$$(Ffu, Gfv)$$

Examples

A simple polynomial is a functor $F : \mathrm{Set} \to \mathrm{Set}$ that is defined from the following inductive rules:

- the identity functor is a simple polynomial;
- every constant functor is a simple polynomial;
- the product $F \times G$ of two simple polynomials is simple;
- the sum $F + G$ of an arbitrary number of simple polynomials is simple.



Examples

A simple polynomial is a functor $F : \mathrm{Set} \to \mathrm{Set}$ that is defined from the following inductive rules:

- the identity functor is a simple polynomial;
- every constant functor is a simple polynomial;
- the product $F \times G$ of two simple polynomials is simple;
- the sum $F + G$ of an arbitrary number of simple polynomials is simple.

An example of a polynomial functor is

$$\lambda X \,.\, A \times X^3 + B \times X^2 + X + 1,$$

where $\times$ denotes cartesian product, A, B are fixed, and $+$ denotes disjoint union.

Examples

Definition
Let A be a set; the set of streams of elements of A is the set of infinite lists $(a_1, a_2, a_3, \ldots, a_n, \ldots)$ of elements of A.

More formally, a stream is a function $a_\bullet : \mathbb{N} \to A$, and the infinite list $(a_1, a_2, a_3, \ldots, a_n, \ldots)$ is just the list of values that the function $a_\bullet$ takes on $n = 0, 1, 2, \ldots$.

The set A^∞ is defined to be the union $A^* \cup A^{\mathbb{N}}$: these are finite or infinite lists.

A closer look to the `Stream` functor

F has relation to A^∞

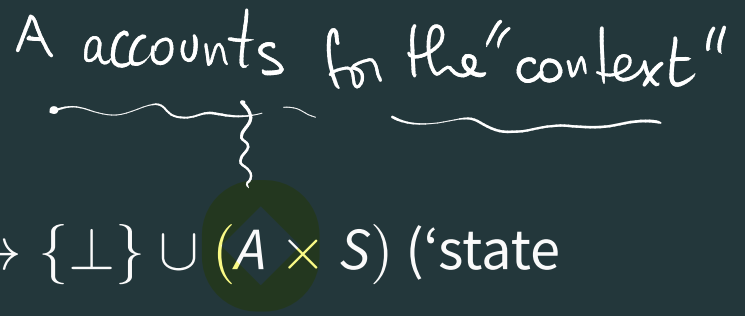
$S \longmapsto (A \times S) + 1$ Maybe $(A \times S)$

(a, s')

$\Sigma : \mathbf{Set} \longrightarrow \mathbf{Set}$

Definition (The stream functor)
Let A be a set; define a correspondence $S \mapsto \{\bot\} \cup (A \times S)$ ('state with possibility of failure').

A closer look to the `Stream` functor



Definition (The stream functor)
Let A be a set; define a correspondence $S \mapsto \{\bot\} \cup (A \times S)$ ('state with possibility of failure').

It is a functor (let's name it Σ); we call it the stream functor.

A closer look to the `Stream` functor

Definition (The stream functor)
Let A be a set; define a correspondence $S \mapsto \{\bot\} \cup (A \times S)$ ('state with possibility of failure').

It is a functor (let's name it Σ); we call it the stream functor.

How it acts on morphisms: given $f : S \to T$ a function between sets, we define

$$\begin{array}{ccc} (a,s) & \longmapsto & (a, fs) \\ \{\bot\} \cup (A \times S) & \longrightarrow & \{\bot\} \cup (A \times T) \\ \| & & \| \\ \Sigma S & \longrightarrow & \Sigma T \end{array}$$

sending $\bot$ to $\bot$, and (a, s) to (a, fs).

This is easily seen to preserve identity and composition.

Coalgebras

Given a functor $F : \mathbf{Set} \to \mathbf{Set}$, we define a coalgebra, or an F-coalgebra, as a pair

$$\left(X, X \xrightarrow{\xi} FX\right)$$

Coalgebras

objects morphisms

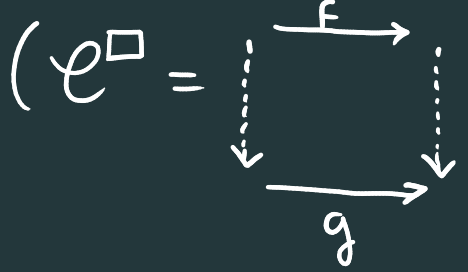
coalgebras hom. of coalg.

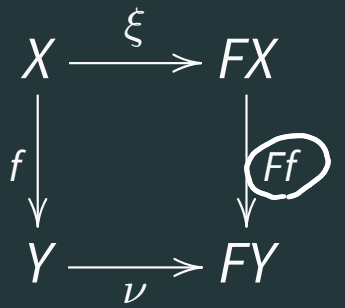
define the category of F-coalg

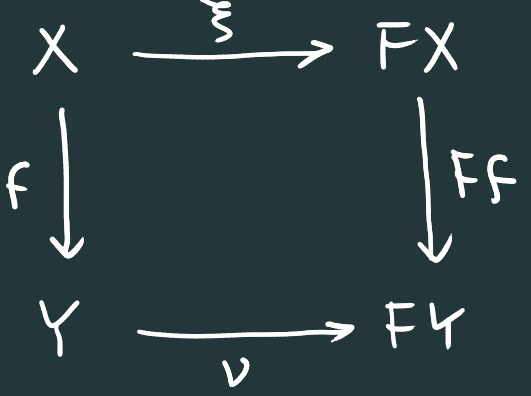
Given a functor $F : \mathbf{Set} \to \mathbf{Set}$, we define a coalgebra, or an F-coalgebra, as a pair

$$\left(X, X \xrightarrow{\xi} FX\right) \qquad (Y, \nu : Y \to FY)$$

and a homomorphism of F-coalgebras as a function $f : X \to Y$ with the property that





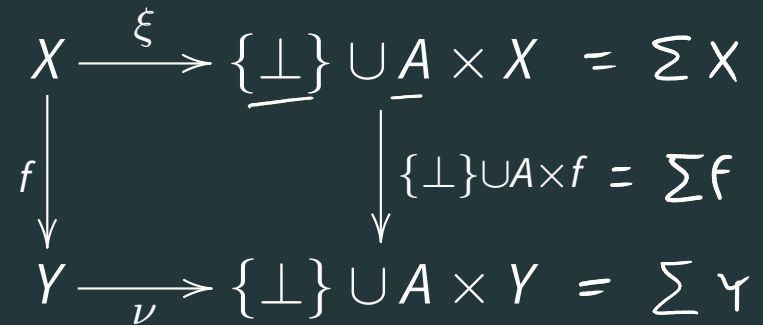


Coalgebras

In case of the stream functor, a Σ-coalgebra is

$$\left(X, X \xrightarrow{\xi} \{\bot\} \cup A \times X\right)$$

and a homomorphism of Σ-coalgebras is a function of sets $f : X \to Y$ such that



Coalgebras

In case of the stream functor, a Σ-coalgebra is

$$\left(X, X \xrightarrow{\xi} \{\bot\} \cup A \times X\right)$$

and a homomorphism of Σ-coalgebras is a function of sets $f : X \to Y$ such that

$$\begin{array}{ccc} X & \xrightarrow{\xi} & \{\bot\} \cup A \times X \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \{\bot\} \cup A \times f} \\ Y & \xrightarrow[\nu]{} & \{\bot\} \cup A \times Y \end{array}$$

Coalgebras

In case of the stream functor, a Σ-coalgebra is

$$\left(X, X \xrightarrow{\xi} \{\bot\} \cup A \times X\right)$$

and a homomorphism of Σ-coalgebras is a function of sets $f : X \to Y$ such that

$$\begin{array}{ccc} X & \xrightarrow{\xi} & \{\bot\} \cup A \times X \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \{\bot\} \cup A \times f} \\ Y & \xrightarrow[\nu]{} & \{\bot\} \cup A \times Y \end{array}$$

Coalgebras

In case of the stream functor, a Σ-coalgebra is

$$\left(X, X \xrightarrow{\xi} \{\bot\} \cup A \times X\right)$$

and a homomorphism of Σ-coalgebras is a function of sets $f : X \to Y$ such that

$$\begin{array}{ccc} X & \xrightarrow{\xi} & \{\bot\} \cup A \times X \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \{\bot\} \cup A \times f} \\ Y & \xrightarrow[\nu]{} & \{\bot\} \cup A \times Y \end{array}$$

Coalgebras

In case of the stream functor, a Σ-coalgebra is

$$\left(X, X \xrightarrow{\xi} \{\bot\} \cup A \times X\right)$$

and a homomorphism of Σ-coalgebras is a function of sets $f : X \to Y$ such that

$$\begin{array}{ccc} X & \xrightarrow{\xi} & \{\bot\} \cup A \times X \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \{\bot\} \cup A \times f} \\ Y & \xrightarrow[\nu]{} & \{\bot\} \cup A \times Y \end{array}$$

Coalgebras

In case of the stream functor, a Σ-coalgebra is

$$\left(X, X \xrightarrow{\xi} \{\bot\} \cup A \times X\right)$$

and a homomorphism of Σ-coalgebras is a function of sets $f : X \to Y$ such that

$$\begin{array}{ccc} X & \xrightarrow{\xi} & \{\bot\} \cup A \times X \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \{\bot\} \cup A \times f} \\ Y & \xrightarrow[\nu]{} & \{\bot\} \cup A \times Y \end{array}$$

Coalgebras

In case of the stream functor, a Σ-coalgebra is

$$\left(X, X \xrightarrow{\xi} \{\bot\} \cup A \times X\right)$$

and a homomorphism of Σ-coalgebras is a function of sets $f : X \to Y$ such that

$$\begin{array}{ccc} X & \xrightarrow{\xi} & \{\bot\} \cup A \times X \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \{\bot\} \cup A \times f} \\ Y & \xrightarrow[\nu]{} & \{\bot\} \cup A \times Y \end{array}$$

Coalgebras

A^* finite + $A^{\mathbb{N}}$ infinite

Theorem

A^∞ as defined above has a remarkable property with respect to the Stream functor:

Coalgebras

Theorem
A^∞ as defined above has a remarkable property with respect to the Stream functor:

$$A^\infty \longrightarrow \text{Maybe}\ (A \times A^\infty)$$

- *it's the domain of a Stream-coalgebra (A^∞, ξ_∞); moreover, ξ_∞ is a bijection.*

Coalgebras

Theorem
A^∞ as defined above has a remarkable property with respect to the Stream functor:

- *it's the domain of a Stream-coalgebra (A^∞, ξ_∞); moreover, ξ_∞ is a bijection.*
- *for every other Stream-coalgebra (S, σ) there exists a unique function $h : S \to A^\infty$ such that h is an homomorphism of coalgebras.*

$\sigma : S \longrightarrow \text{Maybe}(A \times S)$

Coalgebras

Theorem
A^∞ as defined above has a remarkable property with respect to the Stream functor:

- *it's the domain of a Stream-coalgebra (A^∞, ξ_∞); moreover, ξ_∞ is a bijection.*
- *for every other Stream-coalgebra (S, σ) there exists a unique function $h : S \to A^\infty$ such that h is an homomorphism of coalgebras.*

Stream-coalgebras and Stream-coalgebra homomorphisms form a category CoAlg_Σ; the object A^∞ has the property that there exists a unique morphism $(S, \sigma) \to (A^\infty, \xi_\infty)$ in CoAlg_Σ, for every (S, σ) in CoAlg_Σ.

Coalgebras

The proof has several pieces;

- Define a coalgebra structure on A^∞; it is the map

$$n : A^\infty \to \{\bot\} \cup A \times A^\infty$$

Coalgebras

The proof has several pieces;

- Define a coalgebra structure on A^∞; it is the map

$$n : A^\infty \to \{\bot\} \cup A \times A^\infty$$

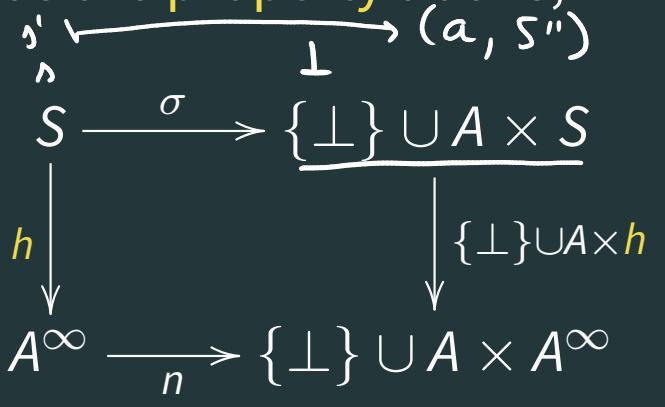
$A^* \cup A^{\mathbb{N}}$

sending $s = ()$ to $\bot$, and $s = (\underline{a} : \underline{as})$ to (a, as). This map is invertible:

$$n^{-1} : \{\bot\} \cup A \times A^\infty \to A^\infty \begin{cases} \{\bot\} & \mapsto () \\ (a, as) & \mapsto (\underline{a} : \underline{as}) \end{cases}$$

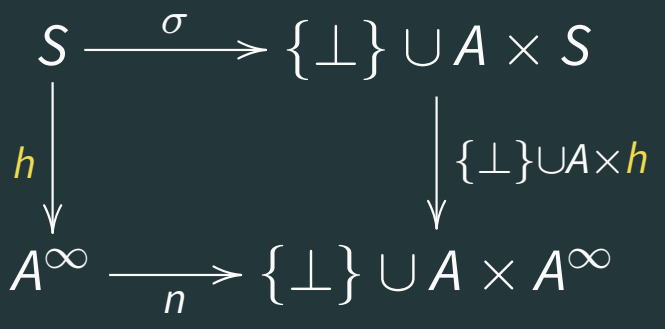
- To show that A^∞ has the property above,

- To show that A^∞ has the property above, in



  we shall specify its action on each finite and infinite list.

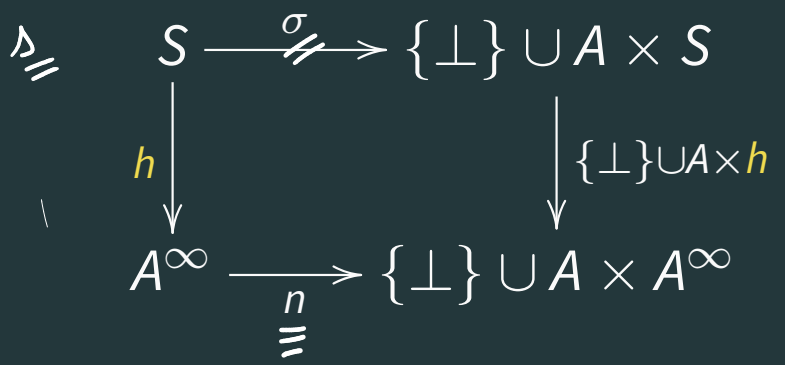
- To show that A^∞ has the property above, in



  we shall specify its action on each finite and infinite list. Define inductively

$$\begin{cases} \sigma^0(s) = \sigma(s) \\ \sigma^{n+1}(s) = y \ \texttt{if} \ \ \sigma^n(s) = (a, y) \ \ \texttt{else} \ \ \bot \end{cases}$$

- To show that A^∞ has the property above, in



we shall specify its action on each finite and infinite list. Define inductively

$$\vec{\sigma} = \begin{cases} \sigma^0(s) = \sigma(s) \\ \sigma^{n+1}(s) = y \ \texttt{if} \ \sigma^n(s) = (a, y) \ \texttt{else} \ \bot \end{cases}$$

"sequence of successive states" induced by σ

and

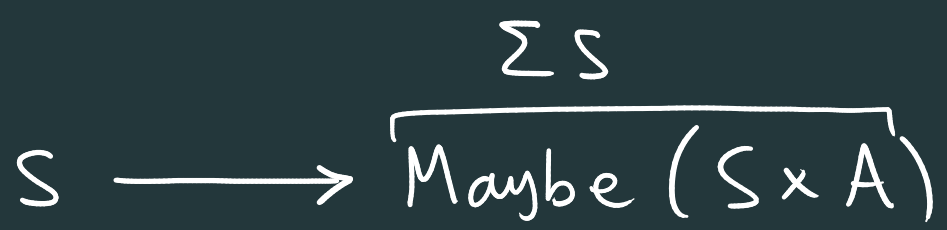
$$h(s) = \begin{cases} (a_0, a_1, a_2, \dots) & \text{if } \sigma^n(s) \neq \bot \ \forall n, \sigma^k(s) = (a_k, s_k) \\ (a_0, a_1, a_2, \dots, a_{m-1}) & \text{if } m = \min\{k \mid \sigma^k(s) = \bot\} \end{cases}$$

if the sequence $\vec{\sigma}$ "never stops" (never reaches failure)

if it stops after m steps

Why insist so much on this example?

- functors from Set to Set are already many and expressive;
- the theory of ‘Set to Set’ functors arising from ‘elementary set-theoretic operations’ is well-understood and has many applications
- …you already know the stream functor.



parsers are Σ-coalgebras!

```
newtype Parser a = Parser
  { runP :: String -> Maybe (String, a)
  }
```

"hello"

cartesian products

```
instance Functor Parser where
  -- <$> : (a -> b) -> Parser a -> Parser b
  f <$> (Parser p) =
    Parser $ \input -> do
      (input', x) <- p input
      return (input', f x)
```

```haskell
instance Applicative Parser where
  -- pure : a -> Parser a
  pure x =
    Parser $ \input -> Right (input, x)
  -- <*> : Parser (a -> b) -> Parser a -> Parser b
  (Parser p1) <*> (Parser p2) =
    Parser $ \input -> do
      (input', f) <- p1 input
      (input'', a) <- p2 input'
      return (input'', f a)
```

More info at

https://www.youtube.com/watch?v=N9RUqGYuGfw

tsoding

Throwing exceptions

We have seen how

$$S \longrightarrow \{\bot\} \cup (S \times A)$$

represents an operation that takes an input, and yields Nothing or a pair (s, a), where s is what we wanted to find and a what is left from the input that the parser consumes.

Throwing exceptions

We have seen how

$$S \xrightarrow{\sigma} \{\bot\} \cup S \cup (S \times E)$$

$$1 + S + S \times E \qquad (s, e)$$

represents an operation that takes an input, and yields Nothing or a pair (s, a), where s is what we wanted to find and a what is left from the input that the parser consumes.

This picture can be generalised to encompass the case where we want to raise an exception that has nothing to do with the impossibility of the computation to terminate.