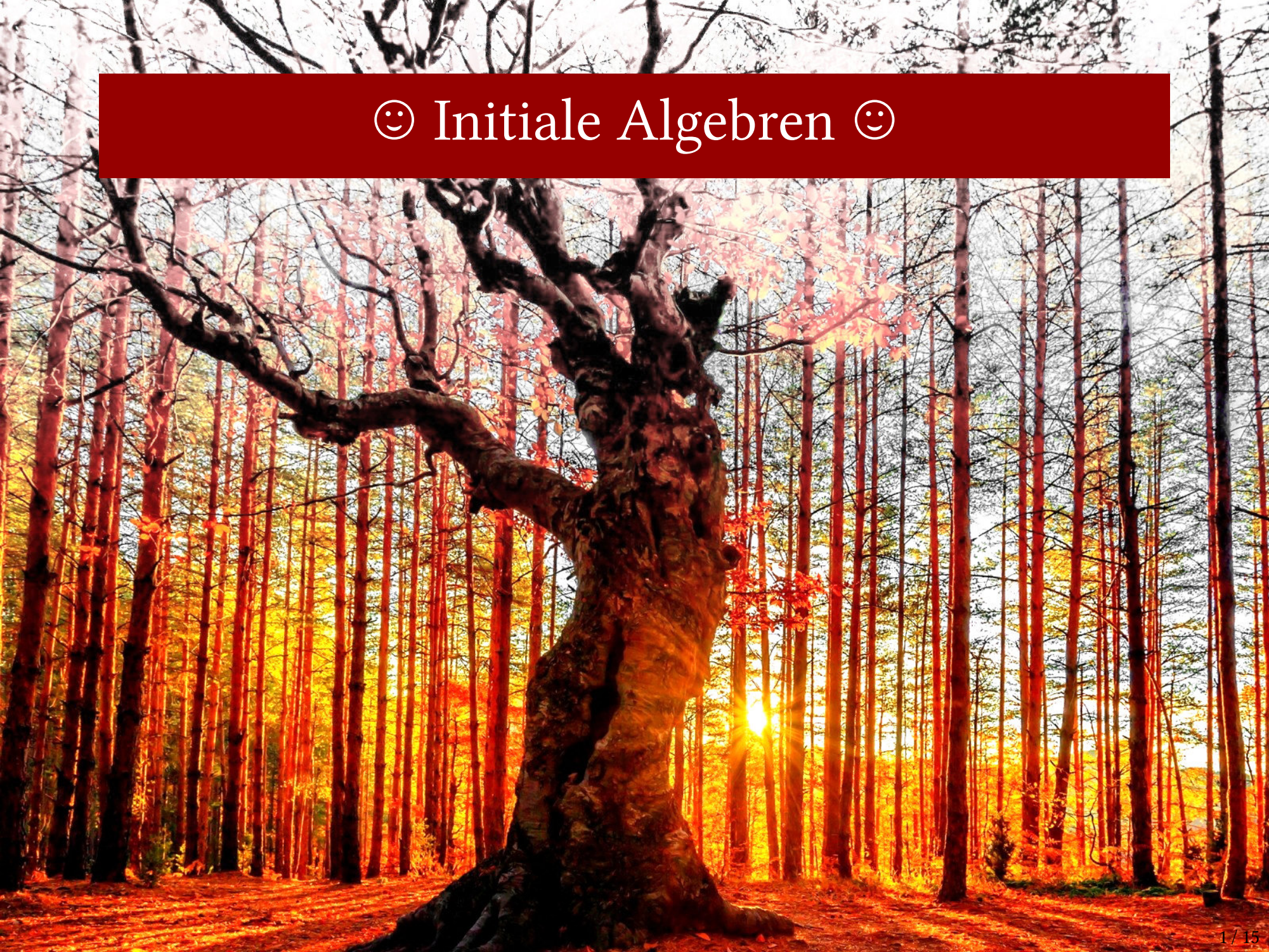
☺ Initiale Algebren ☺



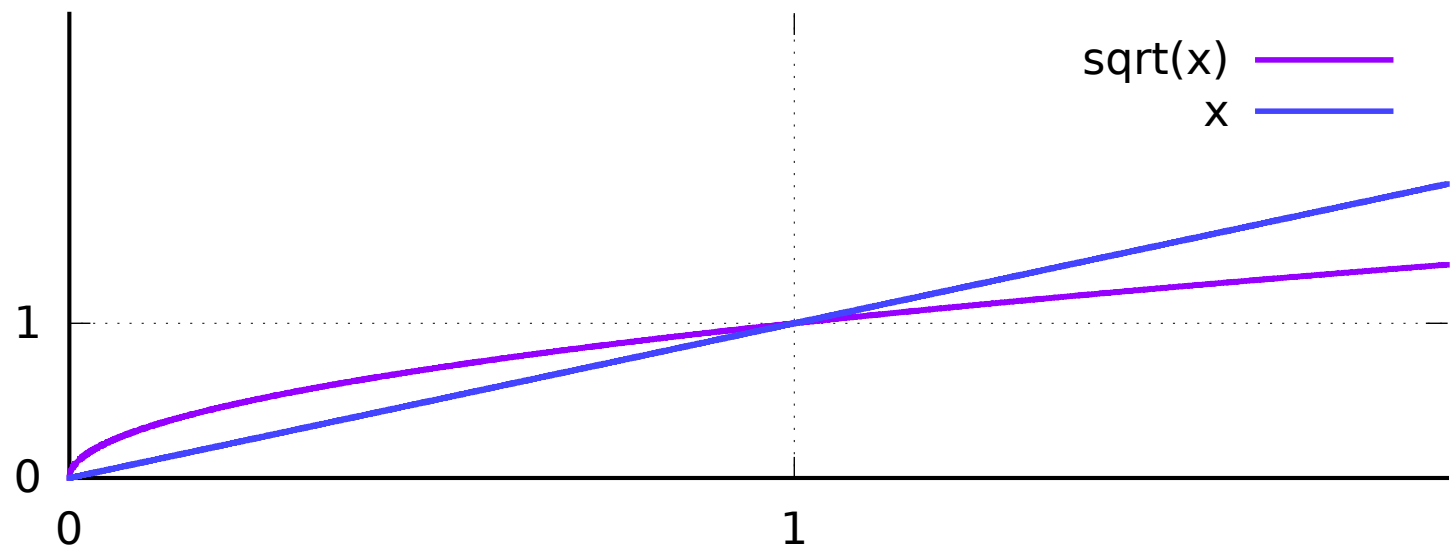
Motivation

In der Mathematik und theoretischen Informatik untersucht man oft **Fixpunktgleichungen**:

$$x = f(x)$$

Oft ist man am **kleinsten** oder **größten** Fixpunkt interessiert:

$$\mu f \qquad \text{oder} \qquad \nu f$$



Motivation

In der theoretischen Informatik benötigt man aber auch eine höhere Art von Fixpunkt„gleichungen“:

$$X \cong F(X)$$

Initiale Algebren verallgemeinern kleinste Fixpunkte,
terminale Koalgebren verallgemeinern größte Fixpunkte.

Wir klären heute folgende Frage: Was bedeutet

```
data Nat = Zero | Succ Nat
```

eigentlich wirklich?

Zunächst: „keine Bottoms, alles endlich“.

Algebren

Eine **Algebra** für einen Funktor $F : \mathcal{C} \to \mathcal{C}$ besteht aus

- einem Objekt $A \in \mathcal{C}$ und
- einem Morphismus $\alpha : F(A) \to A$ in $\mathcal{C}$.

```
data F a = Nil | Cons Int a   -- Beispielfunktor

instance Functor F where
    fmap f Nil         = Nil
    fmap f (Cons x r) = Cons x (f r)

productA :: F Int → Int   -- Beispielalgebra
productA Nil         = 1
productA (Cons x r) = x * r
```

Algebren sind nicht rar!

```
data F a = Nil | Cons Int a

productA :: F Int → Int
productA Nil         = 1
productA (Cons x r) = x * r

lengthA :: F Int → Int
lengthA Nil         = 0
lengthA (Cons _ r) = 1 + r

allNonzeroA :: F Bool → Bool
allNonzeroA Nil         = True
allNonzeroA (Cons x r) = x /= 0 && r
```

Eine Algebra für einen Funktor F kann man sich vorstellen wie die Anleitung für einen F-förmigen Rekursionsschritt. Die Rekursion selbst wird aber nicht ausgeführt. Algebren machen Ideen, wie eine Rekursion zu führen sei, zu first-class values.

Eine Algebra muss keinerlei Axiome erfüllen (anders als bei Monoiden, Gruppen oder Ringen). Das erklärt ihr universelles Vorkommen.

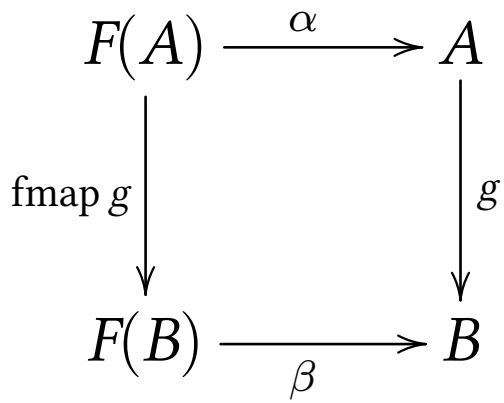
Ist der verwendete Funktor F der zugrundeliegende Funktor einer Monade, so kann kann man Axiome an die Algebra stellen. Dann spricht man von „Algebren für Monaden“. In gewisser Hinsicht sind sie für die Mathematik, Logik und Informatik noch wichtiger als Algebren für Funktoren. Sie sind aber nicht Gegenstück dieses Vortrags.

Ein besonderes Beispiel

```
data F a = Nil | Cons Int a

productA :: F Int → Int
productA Nil         = 1
productA (Cons x r) = x * r

allNonzeroA :: F Bool → Bool
allNonzeroA Nil         = True
allNonzeroA (Cons x r) = x /= 0 && r

initialA :: F [Int] → [Int]
initialA Nil         = []
initialA (Cons x r) = x : r
```

Morphismen zwischen Algebren

Ein **Morphismus** zwischen F-Algebren $\alpha : F(A) \to A$ und $\beta : F(B) \to B$ ist ein Morphismus $g : A \to B$ sodass das folgende Diagramm kommutiert.




```haskell
data F a = Nil | Cons Int a

g :: Int → Bool
g x = x /= 0
-- g . productA = allNonzeroA . fmap g
```

Initiale Algebren

Die „besondere Beispielalgebra“ hat eine **universelle Eigenschaft**: Sie ist die **initiale** *F*-Algebra.

```
data F a = Nil | Cons Int a

initialA :: F [Int] → [Int]
initialA Nil         = []
initialA (Cons x r) = x : r

cata :: (F a → a) → [Int] → a
cata beta []     = beta Nil
cata beta (x:xs) = beta (Cons x (cata f xs))
-- cata beta . initialA = beta . fmap (cata beta)

product :: [Int] → Int
product = cata productA
```

Viele (aber nicht alle) Datentypen „von endlichen Dingen“ in Haskell sind Beispiele für initiale Algebren:

```haskell
data NatF a = Zero | Succ a
type Nat    = Mu NatF
-- Typ der (endlichen) Peano-Zahlen

data IntListF a = Nil | Cons Int a
type IntList    = Mu IntListF
-- Typ der (endlichen) Listen von Ints

data IntTreeF a = Nil | Fork Int a a
type IntTree    = Mu IntTreeF
-- Typ der (endlichen) binären Bäume von Ints
```

Entsprechend sind viele (struturell-)rekursive Funktionen Beispiele für Katamorphismen – also Morphismen, die uns die universelle Eigenschaft der initialen Algebren schenken.

Gibt es immer initiale Algebren?

Sei $F : \mathcal{C} \to \mathcal{C}$ ein Funktor. Gibt es eine initiale F-Algebra?

Gibt es immer initiale Algebren?

Sei $F : \mathcal{C} \to \mathcal{C}$ ein Funktor. Gibt es eine initiale F-Algebra?
Antwort: **Manchmal** (hängt von $\mathcal{C}$ und F ab). Aber in der Kategorie der Haskell-Typen und Haskell-Funktionen immer (zumindest moralisch), denn man kann sie explizit konstruieren:

```
data Mu f = MkMu { outF :: f (Mu f) }
-- mit sozialer Vereinbarung, nur "endliche" Werte zu
-- betrachten. Der Morphismenanteil der initialen
-- Algebra ist MkMu :: f (Mu f) → Mu f.

cata :: (Functor f) ⇒ (f a → a) → (Mu f → a)
cata g (MkMu r) = g (fmap (cata g r))
```

Gibt es immer initiale Algebren?

Sei $F : \mathcal{C} \to \mathcal{C}$ ein Funktor. Gibt es eine initiale F-Algebra?
Antwort: **Manchmal** (hängt von $\mathcal{C}$ und F ab). Aber in der Kategorie der Haskell-Typen und Haskell-Funktionen immer (zumindest moralisch), denn man kann sie explizit konstruieren:

```
data Mu f = MkMu { outF :: f (Mu f) }
-- mit sozialer Vereinbarung, nur "endliche" Werte zu
-- betrachten. Der Morphismenanteil der initialen
-- Algebra ist MkMu :: f (Mu f) → Mu f.

cata :: (Functor f) ⇒ (f a → a) → (Mu f → a)
cata g (MkMu r) = g (fmap (cata g r))
```

Initiale Algebren modellieren Datentypen, für die man Funktionen heraus durch Rekursion angeben kann.

Kategorielles Fun Fact: Ist eine Kategorie sowohl vollständig (besitzt für jedes kleine Diagramm eines Limes) als auch „algebraisch vollständig“ (besitzt für jeden Endofunktor eine initiale Algebra), so ist sie schon dünn (je zwei parallele Morphismen sind gleich), kommt also von einer Quasiordnung her.

Das ist ein Theorem von Freyd.

Endlichkeit

Initiale Algebren modellieren Datentypen, für die jeder Wert „endlich“ ist.

Tatsächlich kann man in vielen Kategorien die initale Algebra eines Funktors F gewinnen als

$$\mu F = \operatorname{colim}(\emptyset \to F(\emptyset) \to F(F(\emptyset)) \to \cdots).$$

Dabei ist $\emptyset$ das initiale Objekt (der leere Datentyp `Void`).

Ist F der Funktor der vorherigen Folien, so realisiert die gezeigte Formel für μF als (Ko-)Limes der Datentypen der leeren Listen, der höchstens einelementigen Listen, der höchstens zweielementigen Listen, und so weiter.

Lambeks Lemma

Sei $\alpha : F(A) \to A$ eine initiale Algebra. Dann ist α ein Isomorphismus (besitzt einen Umkehrmorphismus).

In diesem Sinn löst A die Fixpunkt„gleichung“

$$X \cong F(X).$$

Anschaulich: Mit α konstruiert man neue Werte aus alten.
Die Isomorphie bedeutet, dass jeder Wert aus anderen Werten konstruierbar ist.

Lambeks Lemma

Sei $\alpha : F(A) \to A$ eine initiale Algebra. Dann ist α ein Isomorphismus (besitzt einen Umkehrmorphismus).

In diesem Sinn löst A die Fixpunkt„gleichung“

$$X \cong F(X).$$

Anschaulich: Mit α konstruiert man neue Werte aus alten.
Die Isomorphie bedeutet, dass jeder Wert aus anderen Werten konstruierbar ist.

Übungsaufgabe!

Wie viele Hilfsmorphismen benötigst du?

Eine Art Umkehrung von Lambeks Lemma gilt nicht: Ist der Strukturmorphismus $F(A) \to A$ einer Algebra zufälligerweise ein Isomorphismus, so heißt das noch nicht, dass sie eine initiale Algebra ist.

Ist die Basiskategorie $\mathcal{C}$ nämlich eine Partialordnung, so sind initiale Algebren dasselbe wie kleinste Fixpunkte und terminale Koalgebren dasselbe wie größte Fixpunkte. Aber nicht jeder Fixpunkt ist ein kleinster.

Terminale Koalgebren

Eine **Algebra** für einen Funktor $F : \mathcal{C} \to \mathcal{C}$ besteht aus

- einem Objekt $A \in \mathcal{C}$ und
- einem Morphismus $\alpha : F(A) \to A$ in $\mathcal{C}$.

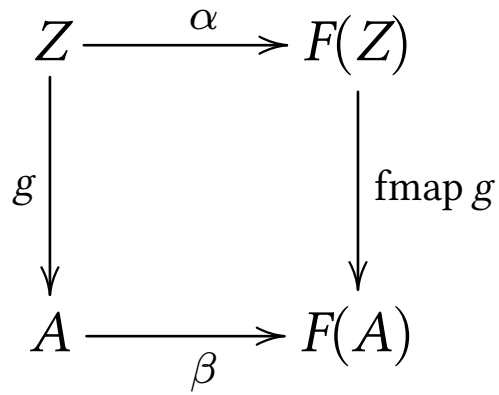
Eine **Koalgebra** für einen Funktor $F : \mathcal{C} \to \mathcal{C}$ besteht aus

- einem Objekt $A \in \mathcal{C}$ und
- einem Morphismus $\alpha : A \to F(A)$ in $\mathcal{C}$.

```
data Nu f = MkNu { outF :: f (Nu f) }
-- ohne sozialen Vertrag!

ana :: (Functor f) ⇒ (a → f a) → (a → Nu f)
ana alpha x = MkNu (fmap (ana alpha) (alpha x))
```

Eine Koalgebra für einen Funktor F kann man sich vorstellen wie die Anleitung für eine F-förmige Beobachtung. Die Beobachtung wird aber nicht „bis zum Ende“, verschachtelt, korekursiv ausgeführt, sondern nur für einen Schritt lang. Koalgebren machen also Ideen, wie eine Korekursion zu führen sei, zu first-class values.

Eine *terminale Koalgebra* ist eine Koalgebra $\beta : A \to F(A)$, sodass für jede Koalgebra $\alpha : Z \to F(Z)$ genau ein Morphismus $g : Z \to A$ mit $\beta \circ g = \text{fmap}\, g \circ \alpha$ existiert.



Wir haben bereits gesehen, dass viele Datentypen „von endlichen Dingen“ in Haskell initiale Algebren sind. Viele Datentypen „von nicht notwendigerweise endlichen Dingen“ sind terminale Koalgebren:

```haskell
data NatF a = Zero | Succ a
type Nat    = Nu NatF
-- Typ der Peano-Zahlen, mit einem Element für +infty:
-- MkNu (Succ (MkNu (Succ (...))))

data IntListF a = Nil | Cons Int a
type IntList    = Nu IntListF
-- Typ der (endlichen und unendlichen) Listen von Ints

data IntTreeF a = Nil | Fork Int a a
type IntTree    = Nu IntTreeF
-- Typ der (endlichen und unendlichen) binären Int-Bäume
```

Entsprechend sind viele (struturell-)korekursive Funktionen Beispiele für Anamorphismen – also Morphismen, die uns die universelle Eigenschaft der terminalen Koalgebren schenken.

Übungsaufgabe: Wieso spricht niemand über terminale Algebren und initiale Koalgebren?

Vergleich

Initiale Algebren modellieren Datentypen, für die man Funktionen heraus durch Rekursion angeben kann.

- Konstruktion von Werten mittels $F(A) \to A$
- `cata :: (F a → a) → (Mu F → a)`
- „endlich“

Terminale Koalgebren modellieren Datentypen, für die man Funktionen hinein durch Korekursion angeben kann.

- Beobachtung von Werten mittels $A \to F(A)$
- `ana :: (a → F a) → (a → Nu F)`
- „endlich oder unendlich“

Das Wesentliche an einer initialen Algebra A ist, dass man vermöge der mitgegebenen Funktion $F(A) \to A$ Werte von A *konstruieren* kann. Wir stellen uns $F(A)$ als den Datentyp der Konstruktionsbeschreibungen für Werte von A vor; die Funktion $F(A) \to A$ nimmt eine solche Beschreibung und führt sie aus.

Beispiel: `initialA (Cons x xs)` konstruiert die verlängerte Liste `x:xs`.

Das Wesentliche an einer terminalen Koalgebra A ist, dass man vermöge der mitgegebenen Funktion $A \to F(A)$ Werte von A *beobachten* kann. Wir stellen uns $F(A)$ als den Datentyp der möglichen Beobachtungen über Werte von A vor.

Beispiel: `terminalCoA xs` beobachtet, ob die übergebene Liste `xs` leer ist oder eine Cons-Zelle ist (aus einem vorderen Element `x` und einer Restliste `xs'` besteht).

Lambeks Lemma garantiert, dass die zu einer initialen Algebra gehörige Funktion $F(A) \to A$ (bzw. die zu einer terminalen Koalgebra gehörige Funktion $A \to F(A)$) umkehrbar ist. Daher kann man aus einer initialen Algebra eine Koalgebra (welche nur in pathologischen Fällen terminal sein wird) und dual aus einer terminalen Koalgebra eine Algebra machen.

Beispiel: Das Wesentliche vom Datentyp der endlichen Listen ist, dass man seine Werte aus Grundbestandteilen (`Nil` und `Cons x`) sukzessive kontruieren kann. Trotzdem kann man seine Werte auch beobachten (prüfen, ob ein Wert `Nil` ist oder eine Cons-Zelle ist).

Beispiel: Das Wesentliche vom Datentyp der nicht notwendigerweise endlichen Listen ist, dass man seine Werte beobachten kann (ist eine solche Liste leer oder eine Cons-Zelle?). Trotzdem kann man aber auch seine Werte konstruieren (etwa an eine gegebene beliebig lange Liste vorne ein Element anfügen).

Ausblick

- Behandlung von Bottoms durch Wechsel der Kategorie – nicht die Kategorie der Mengen, sondern die Kategorie der **Domänen** (domains)

Ausblick

- Behandlung von Bottoms durch Wechsel der Kategorie – nicht die Kategorie der Mengen, sondern die Kategorie der **Domänen** (domains)
- Haskell: boldly going where no functor has gone before.

```
data Seltsam = MkSeltsam (Seltsam → Bool)
```

Die Theorie der initialen Algebren und terminalen Koalgebren ist nur der Anfang. Für Datentypen, die durch eine kompliziertere Rekursionsgleichung gegeben sind (etwa, wo der verwendete Funktor kontra- statt kovariant ist; wo der verwendete Funktor in einem Argument ko- und in einem anderen kontravariant ist; wo es gar keinen erkennbaren Funktor mehr gibt), benötigt man eine leistungsfähigere Theorie.

Die Kategorie der Domänen hat sich als sehr leistungsfähig erwiesen. In ihr gibt es viel mehr als nur initiale Algebren und terminale Koalgebren.

Mehr zum seltsamen Typ hat Dan Piponi zu berichten. Der Typ ist durchaus beachtenswert, liefert er doch ein Gegenbeispiel zu einer (internen Variante von) Cantors Theorem:

In früheren Curry-Club-Vorträgen haben wir gesehen, dass keine Menge X isomorph zu ihrer Potenzmenge $\mathcal{P}(X)$ sein kann. („Isomorph“ heißt hier „gleich mächtig“ und die Potenzmenge ist die Menge aller Teilmengen von X, verallgemeinerbarer formuliert als Menge aller Abbildungen von X nach Bool.)

Eine Domäne X kann aber durchaus isomorph zu ihrer „Potenzdomäne“ $(X \to \text{Bool})$ sein. Das ist etwa bei `Seltsam` der Fall.