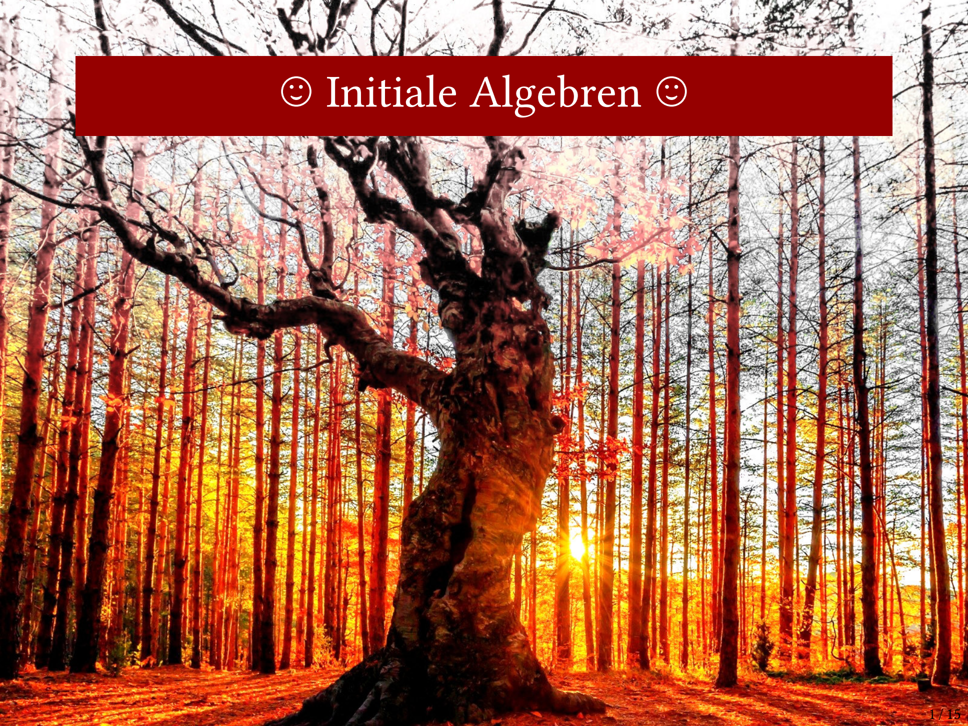
☺ Initiale Algebren ☺



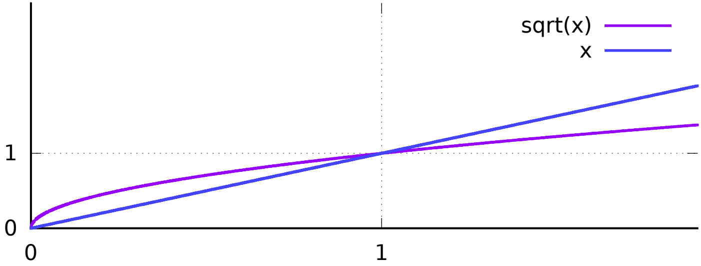
Motivation

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 $\mathcal{C}$.

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

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

-- Beispielalgebra
productA :: F Int → Int
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
```

Ein besonderes Beispiel

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

prodA :: F Int → Int
prodA Nil          = 1
prodA (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.

$$\begin{tikzcd}
F(A) \arrow[r, "\alpha"] \arrow[d, "\text{fmap } g"'] & A \arrow[d, "g"] \\
F(B) \arrow[r, "\beta"'] & B
\end{tikzcd}$$

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

g :: Int → Bool
g x = x /= 0
-- g . prodA = 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 g []      = g Nil
cata g (x:xs) = g (Cons x (cata f xs))
-- cata g . initA = g . fmap (cata g)

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

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.**

```
data Mu f = MkMu { outF :: f (Mu f) }
-- mit sozialer Vereinbarung
-- MkMu :: f (Mu f) → Mu f
-- outF :: Mu f      → 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.**

```haskell
data Mu f = MkMu { outF :: f (Mu f) }
-- mit sozialer Vereinbarung
-- MkMu :: f (Mu f) → Mu f
-- outF :: Mu f     → 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.

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

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

Lambeks Lemma

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

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

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) }

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

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“

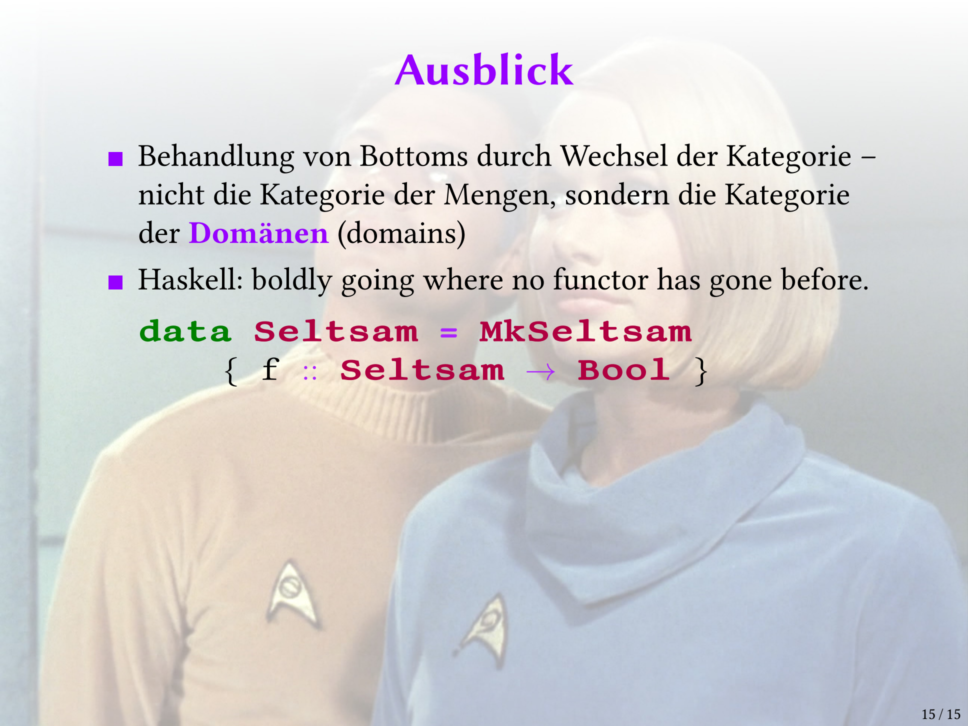
Terminale Koalgebren modellieren Datentypen, für die man Funktionen hinein durch Korekursion angeben kann.

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

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
     { f :: Seltsam → Bool }
```