Funktionale Programmierung
Sommersemester 2021

Prof. Dr. Jakob Rehof
TU Dortmund
LS XIV Software Engineering

Weitere Lektüre für Interessierte

- Grundlagen der Kategorientheorie
  - Saunders MacLane: *Categories for the Working Mathematician*. Springer Graduate Texts in Mathematics, 1971.
- Kategorientheorie in Haskell
  - Bartosz Milewski: *Category Theory for Programmers*. https://github.com/hmemcpy/milewski-ctfp-pdf/
  - Bartosz Milewski blog: https://bartoszmilewski.com/
- Getypter Lambda Kalkül
  - Sørensen, Urzyczyn: *Lectures on the Curry-Howard Isomorphism*. Lecture notes sind online verfügbar. Buch: Elsevier 1998.
- Formale Systeme der Typtheorie
  - Nederpelt, Geuvers: *Type Theory and Formal Proof*. Cambridge UP 2014.

Diese Vorlesung:

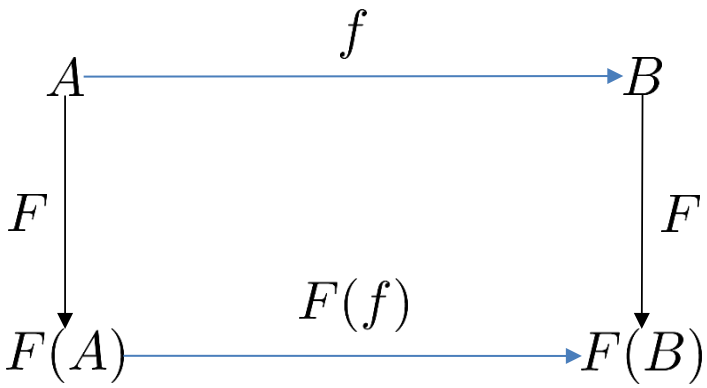
- P. Padawitz *„Modellieren und Implementieren in Haskell“* Folien, Abschnitt 7.2 – 7.8 (S. 214-235) und Abschnitt 7.11 (S. 253-255)
  - https://fldit-www.cs.tu-dortmund.de/~peter/Essen.pdf
  - Weitere Beispiele (Funktoren, Monaden)

- Natürliche Transformationen in Haskell
- Duale Kategorie und Kontravarianter Funktor

Funktor

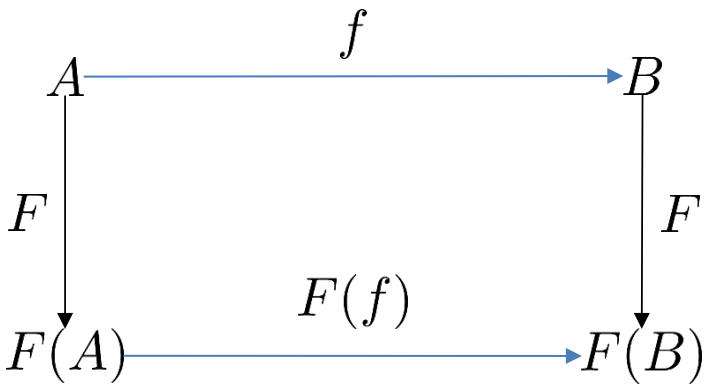
Seien $\mathcal{C}$ und $\mathcal{D}$ Kategorien. Ein *Funktor* F von $\mathcal{C}$ nach $\mathcal{D}$ ist eine Abbildung mit den Eigenschaften

- F bildet jedes Objekt $A \in ob(\mathcal{C})$ in ein Objekt $F(A) \in ob(\mathcal{D})$ ab
- F bildet jeden Morphismus $f : A \to B \in hom(\mathcal{C})$ in einen Morphismus $F(f) : F(A) \to F(B) \in hom(\mathcal{D})$ ab, wobei folgende Bedingungen gelten:
  - $F(1_A) = 1_{F(A)}$ für alle $A \in ob(\mathcal{C})$
  - $F(g \circ f) = F(g) \circ F(f)$ für alle $f : A \to B, g : B \to C \in hom(\mathcal{C})$



Haskell Funktor

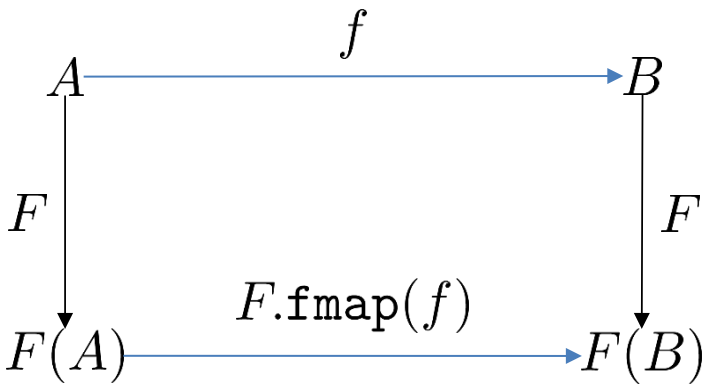
```
class Functor F where

    fmap :: (a -> b) -> F a -> F b
```



Haskell Funktor

- Objekte $\mapsto$ Typen
- Morphismen $\mapsto$ Funktionen
- Funktoren $\mapsto$ Typkonstruktoren $F :: * \to *$ mit Typklassen

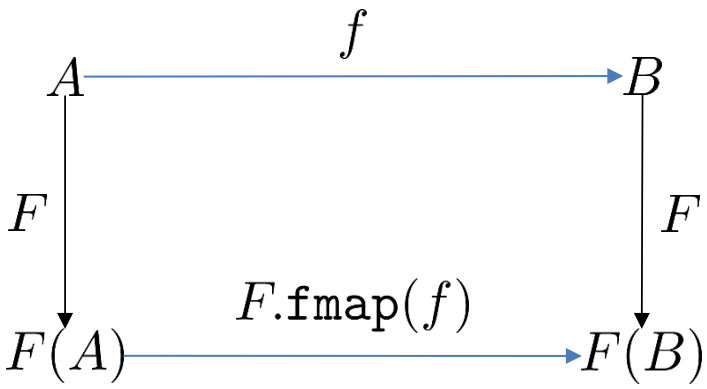
```
class Functor F where

    fmap :: (a -> b) -> F a -> F b
```



Haskell Funktor

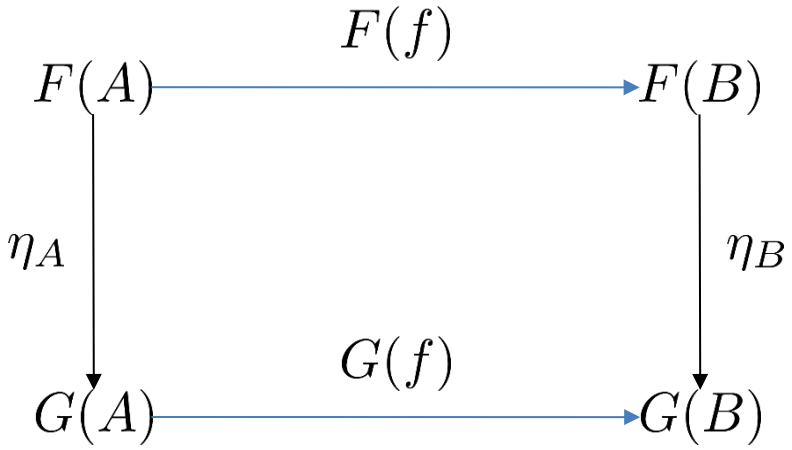
```
class Functor F where
    fmap :: (a -> b) -> F a -> F b
```

- `fmap id = id`
- `fmap (g . f) = (fmap g) . (fmap f)`



Natürliche Transformation

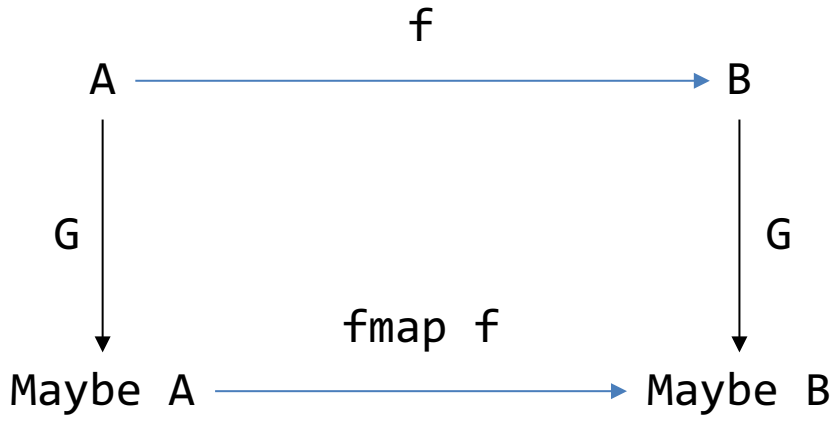
Seien F und G Funktoren von $\mathcal{C}$ nach $\mathcal{D}$ (Kategorien). Eine *natürliche Transformation* (eng. natural transformation) η von F nach G knüpft zu jedem Objekt A in $\mathcal{C}$ einen Morphismus $\eta_A : F(A) \to G(A)$ in $\mathcal{D}$, so daß es für jeden Morphismus $f : A \to B$ in $\mathcal{C}$ gilt: $\eta_B \circ F(f) = G(f) \circ \eta_A$. Mit anderen "Worten", das Diagramm kommutiert:



Na klar doch: seien $\mathcal{C}$ und $\mathcal{D}$ Kategorien, dann können wir die *Funktorkategorie* $\mathcal{D}^{\mathcal{C}}$ bilden: Objekte sind die Funktoren von $\mathcal{C}$ nach $\mathcal{D}$, Morphismen sind natürliche Transformationen zwischen solchen.

Funktoren in Haskell

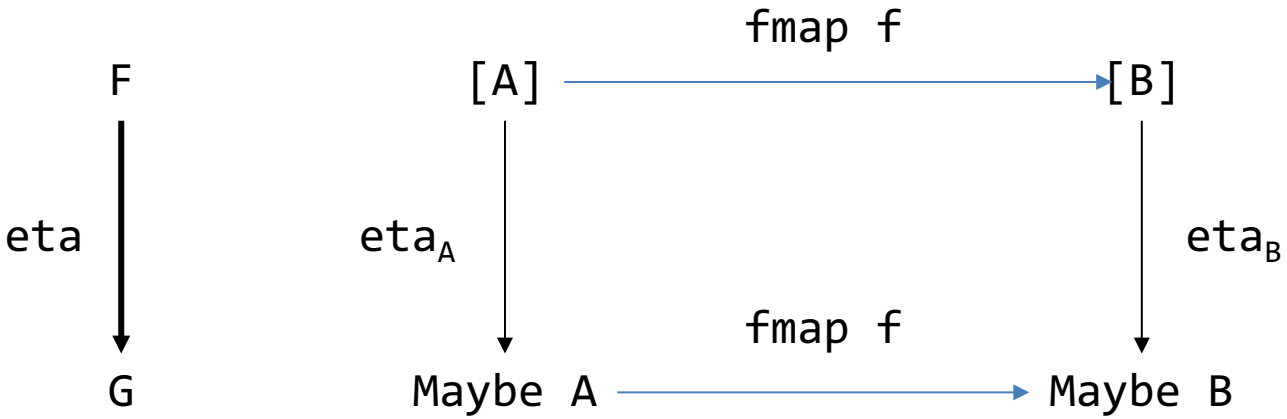
```
F = [], fmap f [] = [], fmap f (x:xs) = (f x): fmap f xs
G = Maybe, fmap f Nothing = Nothing, fmap f (Just x) = Just (f x)
```



Natürliche Transformation in Haskell

```
F = [], fmap f [] = [], fmap f (x:xs) = (f x): fmap f xs
G = Maybe, fmap f Nothing = Nothing, fmap f (Just x) = Just (f x)
```

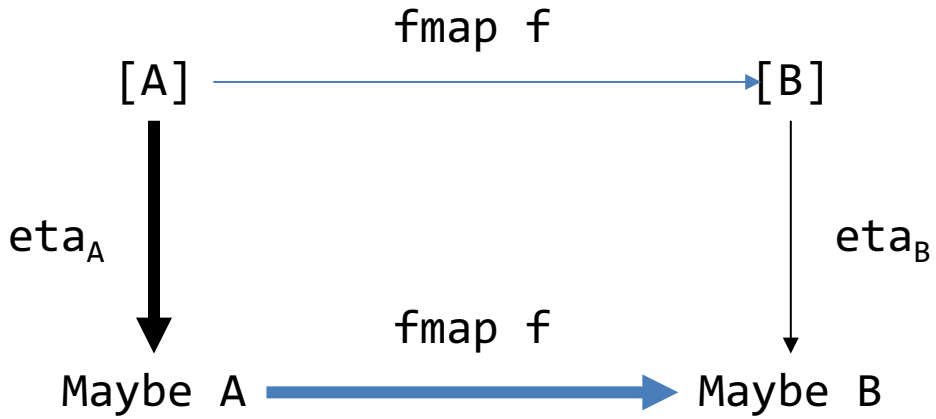
```
eta :: [a] -> Maybe a, eta [] = Nothing, eta (x:xs) = Just x
```



Natürliche Transformation in Haskell

```
F = [], fmap f [] = [], fmap f (x:xs) = (f x): fmap f xs
G = Maybe, fmap f Nothing = Nothing, fmap f (Just x) = Just (f x)
```

```
eta :: [a] -> Maybe a, eta [] = Nothing, eta (x:xs) = Just x
```

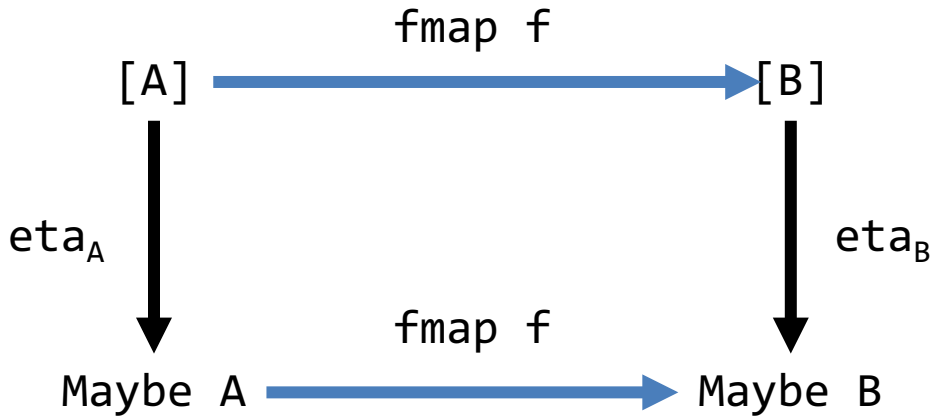


$\texttt{fmap f . eta}_A$

Natürliche Transformation in Haskell

```
F = [], fmap f [] = [], fmap f (x:xs) = (f x): fmap f xs
G = Maybe, fmap f Nothing = Nothing, fmap f (Just x) = Just (f x)
```

```
eta :: [a] -> Maybe a, eta [] = Nothing, eta (x:xs) = Just x
```

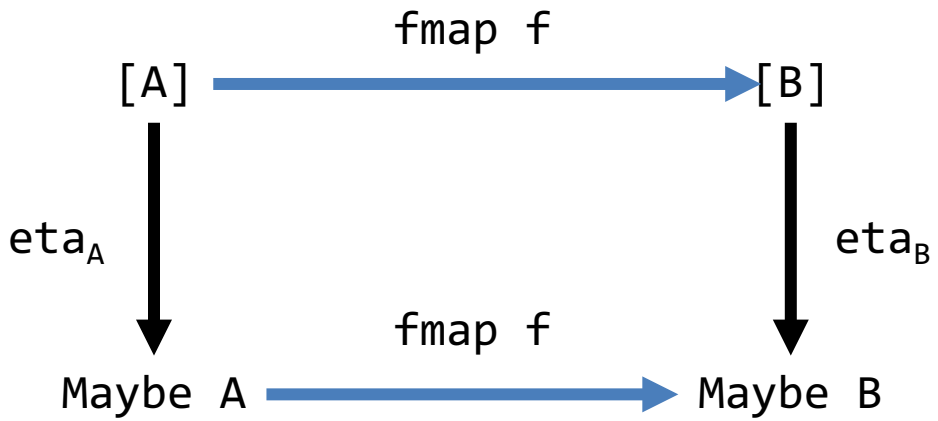


`fmap f . eta`$_A$ `eta`$_B$ `. fmap f`

Natürliche Transformation in Haskell

```
F = [], fmap f [] = [], fmap f (x:xs) = (f x): fmap f xs
G = Maybe, fmap f Nothing = Nothing, fmap f (Just x) = Just (f x)
```

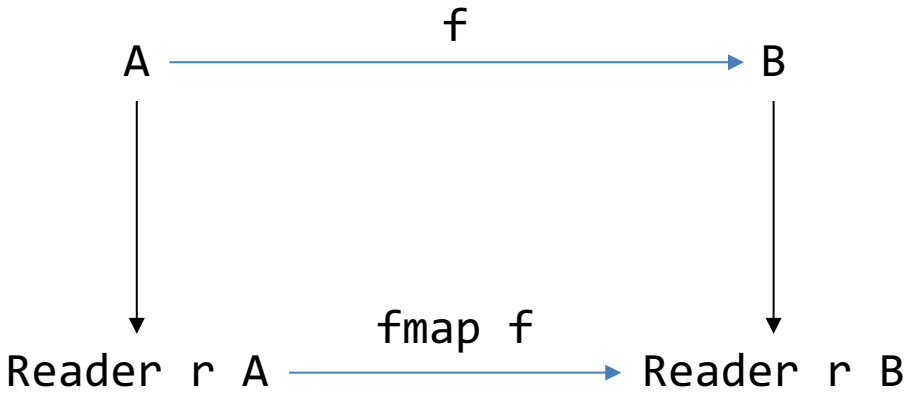
```
eta :: [a] -> Maybe a, eta [] = Nothing, eta (x:xs) = Just x
```



$$\text{fmap } f \cdot \text{eta}_A = \text{eta}_B \cdot \text{fmap } f$$

Reader-Funktor

```
(->) r :: * -> *

type Reader r a = r -> a, Reader r :: * -> *
```




```
f :: A -> B, g :: Reader r A

fmap f g = f . g

fmap :: (a -> b) -> Reader r a -> Reader r b

fmap :: (a -> b) -> (r -> a) -> (r -> b)
```

Reader-Funktor

```
type Reader r a = r -> a, Reader r :: * -> *
```

`Reader r` ist ein Funktor mit Parameter a im Zieltyp (codomain). Was passiert, wenn wir versuchen, einen Funktor im Argumenttyp (domain) zu machen?

Wir versuchen es:

```
type Op r a = a -> r, Op r :: * -> *
```

Wir müssen nun die Funktion `fmap` spezifizieren, mit dem Typ:

```
fmap :: (a -> b) -> Op r a -> Op r b
```

Also

```
fmap :: (a -> b) -> (a -> r) -> (b -> r)
```

Es gibt aber keine Funktion mit diesem Typ!

Kovarianz und Kontravarianz

Bei Funktionstypen spielen Argument (domain) und Ziel (codomain) unterschiedliche Rollen:

- Semantisch: domain = input, codomain = output
- Logisch: domain = Hypothese, codomain = Konklusion

Aus der Logik wissen wir: Eine Aussage mit stärkerer Konklusion ist eine stärkere Aussage („*Kovarianz*“, covariance). Eine Aussage mit stärkerer Hypothese ist eine schwächere Aussage („*Kontravarianz*“, contravariance).

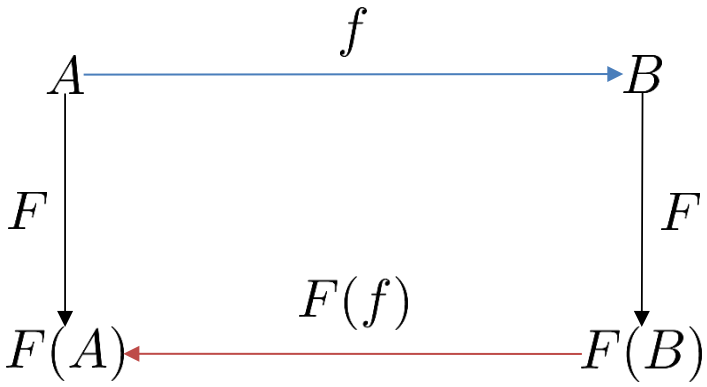
Mengentheoretisch (Subtypen):

$$C \subseteq A, B \subseteq D \quad \Rightarrow \quad A \to B \subseteq C \to D$$

Kontravarianter Funktor

Seien $\mathcal{C}$ und $\mathcal{D}$ Kategorien. Ein *kontravarianter Funktor* F von $\mathcal{C}$ nach $\mathcal{D}$ ist eine Abbildung mit den Eigenschaften

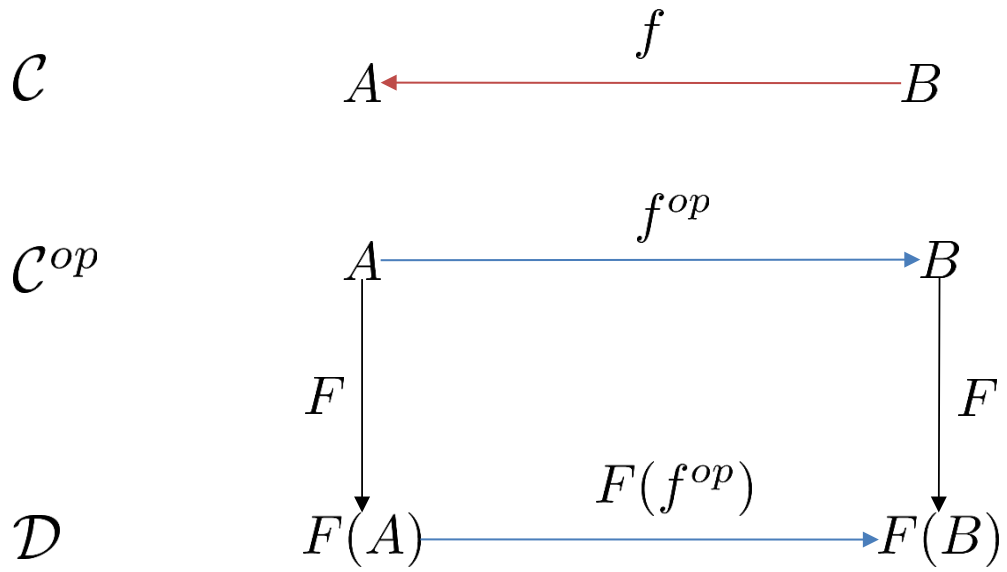
- F bildet jedes Objekt $A \in ob(\mathcal{C})$ in ein Objekt $F(A) \in ob(\mathcal{D})$ ab
- F bildet jeden Morphismus $f : A \to B \in hom(\mathcal{C})$ in einen Morphismus $F(f) : F(B) \to F(A) \in hom(\mathcal{D})$ ab, wobei folgende Bedingungen gelten:
  - $F(1_A) = 1_{F(A)}$ für alle $A \in ob(\mathcal{C})$
  - $F(g \circ f) = F(f) \circ F(g)$ für alle $f : A \to B, g : B \to C \in hom(\mathcal{C})$



Kontravarianter Funktor

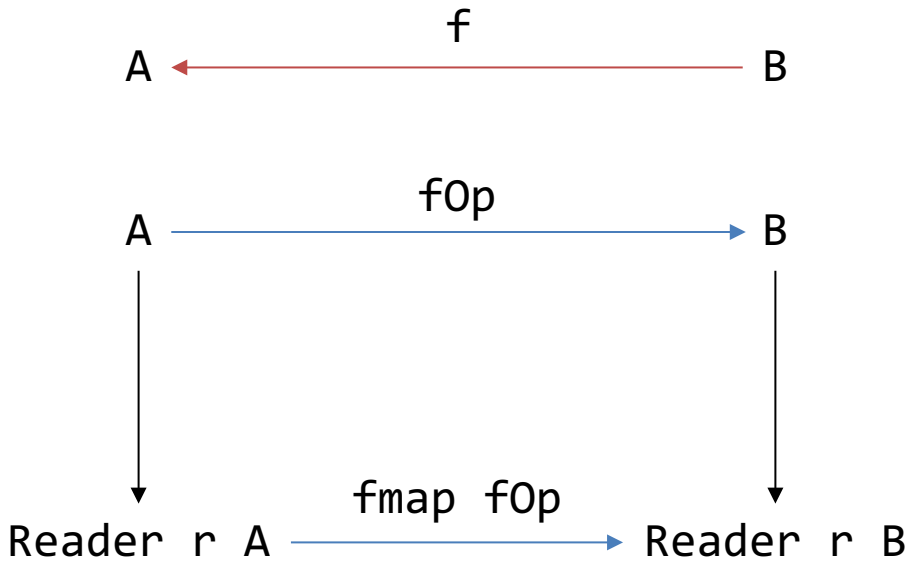
Ein *kontravarianter Funktor* $F : \mathcal{C} \to \mathcal{D}$ kann auch als *kovarianter Funktor* $F : \mathcal{C}^{op} \to \mathcal{D}$ in der *dualen Kategorie* aufgefasst werden.

- $\mathcal{C}^{op}$ ist die Kategorie, die aus $\mathcal{C}$ dadurch entsteht, dass alle Morphismen in $\mathcal{C}$ umgekehrt werden.



Kontravarianter Funktor

```
type Op r a = a -> r, Op r :: * -> *
```




```
class Contravariant cF where
   cmap :: (b -> a) -> g a -> g b
   cmap f g = g . f
```