More Category Theory

Haskell and Cryptocurrencies

Dr. Andres Löh, Well-Typed LLP
Dr. Lars Brünjes, IOHK

2017-09-01

Goals

- Adjunctions
- Exponentials
- Monoids
- Monads
- (Co-)Algebras

Adjunctions

Reminder: Natural transformation

- Let $F, G : \mathcal{C} \to \mathcal{D}$ be functors. A natural transformation $\varphi : F \to G$ is given by the following data:
- For each object $X \in \mathrm{Ob}(\mathcal{C})$, a morphism $\varphi_X : FX \to GX$ in $\mathcal{D}$.
- For each morphism $f : X \to Y$ in $\mathcal{C}$, the following diagram must commute (i.e. $\varphi_Y\, Ff = Gf\, \varphi_X$):

$$\begin{tikzcd}
X \arrow[d, dashed, "f"'] & FX \arrow[r, "\varphi_X"] \arrow[d, "Ff"'] & GX \arrow[d, "Gf"] \\
Y & FY \arrow[r, "\varphi_Y"'] & GY
\end{tikzcd}$$

Category of functors

- Let $\mathcal{C}$ and $\mathcal{D}$ be categories. Then we can consider the category $\mathcal{D}^{\mathcal{C}}$ of (covariant) functors from $\mathcal{C}$ to $\mathcal{D}$:
- Objects of $\mathcal{D}^{\mathcal{C}}$ are (covariant) functors from $\mathcal{C}$ to $\mathcal{D}$
- Morphisms from $F : \mathcal{C} \to \mathcal{D}$ to $G : \mathcal{C} \to \mathcal{D}$ are *natural transformations* from F to G.
- The composition of two natural transformations $\varphi : F \to G$ and $\psi : E \to F$ is given by $\varphi_X \psi_X : EX \to GX$ for $X \in \mathrm{Ob}(\mathcal{C})$.
- The identity $F \to F : \mathcal{C} \to \mathcal{D}$ is the *identity transformation* given by $\varphi_X = 1_{FX} : FX \to FX$ for $X \in \mathrm{Ob}(\mathcal{C})$.

Category of functors

- Let $\mathcal{C}$ and $\mathcal{D}$ be categories. Then we can consider the category $\mathcal{D}^{\mathcal{C}}$ of (covariant) functors from $\mathcal{C}$ to $\mathcal{D}$:
- Objects of $\mathcal{D}^{\mathcal{C}}$ are (covariant) functors from $\mathcal{C}$ to $\mathcal{D}$
- Morphisms from $F : \mathcal{C} \to \mathcal{D}$ to $G : \mathcal{C} \to \mathcal{D}$ are *natural transformations* from F to G.
- The composition of two natural transformations $\varphi : F \to G$ and $\psi : E \to F$ is given by $\varphi_X \psi_X : EX \to GX$ for $X \in \mathrm{Ob}(\mathcal{C})$.
- The identity $F \to F : \mathcal{C} \to \mathcal{D}$ is the *identity transformation* given by $\varphi_X = 1_{FX} : FX \to FX$ for $X \in \mathrm{Ob}(\mathcal{C})$.

Note

This in particular explains when two functors $F, G : \mathcal{C} \to \mathcal{D}$ are *isomorphic*, namely iff there are natural transformations $\varphi : F \to G$ and $\psi : G \to F$ with $\varphi\psi = 1_G$ and $\psi\varphi = 1_F$.

Category of functors

- Let $\mathcal{C}$ and $\mathcal{D}$ be categories. Then we can consider the category $\mathcal{D}^{\mathcal{C}}$ of (covariant) functors from $\mathcal{C}$ to $\mathcal{D}$:
- Objects of $\mathcal{D}^{\mathcal{C}}$ are (covariant) functors from $\mathcal{C}$ to $\mathcal{D}$
- Morphisms from $F : \mathcal{C} \to \mathcal{D}$ to $G : \mathcal{C} \to \mathcal{D}$ are *natural transformations* from F to G.
- The composition of two natural transformations $\varphi : F \to G$ and $\psi : E \to F$ is given by $\varphi_X \psi_X : EX \to GX$ for $X \in \mathrm{Ob}(\mathcal{C})$.
- The identity $F \to F : \mathcal{C} \to \mathcal{D}$ is the *identity transformation* given by $\varphi_X = 1_{FX} : FX \to FX$ for $X \in \mathrm{Ob}(\mathcal{C})$.

Note

So two functors $F, G : \mathcal{C} \to \mathcal{D}$ are *isomorphic* if for each $X \in \mathrm{Ob}(\mathcal{C})$, we have an isomorphism $\varphi_X : FX \to GX$ which is ”natural” in X.

Uniqueness of adjoints

If the (left or right) adjoint to a given functor exists, it is uniquely determined up to *unique isomorphism* of functors.

This means that we can *define* a functor by stating that it is (left or right) adjoint to a given functor, provided we know the adjoint exists.

Adjoint functors

- Let $\mathcal{C}$ and $\mathcal{D}$ be categories, and consider two functors $F : \mathcal{C} \leftarrow \mathcal{D}$ and $G : \mathcal{C} \to \mathcal{D}$.
- We say that F is left adjoint to G and that G is right adjoint to F, written $F \dashv G : \mathcal{C} \to \mathcal{D}$, if the functors $\mathrm{Mor}_{\mathcal{C}}(F\cdot, \cdot)$ and $\mathrm{Mor}_{\mathcal{D}}(\cdot, G\cdot)$ from $\mathcal{D}^{\mathrm{op}} \times \mathcal{C}$ to $\underline{\mathrm{Set}}$ are isomorphic.
- Explicitly, this means that for all objects Y in $\mathcal{D}$ and X in $\mathcal{C}$ we have an isomorphism

$$\varphi_{(Y,X)} : \mathrm{Mor}_{\mathcal{C}}(FY, X) \xrightarrow{\sim} \mathrm{Mor}_{\mathcal{D}}(Y, GX)$$

which is "natural" in Y and X, i.e. for all morphisms $g : Y' \to Y$ in $\mathcal{D}$ and $f : X \to X'$ in $\mathcal{C}$, the following diagram commutes:

$$\begin{tikzcd}
\mathrm{Mor}_{\mathcal{C}}(FY, X) \arrow[r, "\varphi_{(Y,X)}", "\sim"'] \arrow[d, "Fg^* f_*"'] & \mathrm{Mor}_{\mathcal{D}}(Y, GX) \arrow[d, "g^* Gf_*"] \\
\mathrm{Mor}_{\mathcal{C}}(FY', X') \arrow[r, "\sim", "\varphi_{(Y',X')}"'] & \mathrm{Mor}_{\mathcal{D}}(Y', GX')
\end{tikzcd}$$

Adjunction example: currying

- Let S be a set, and consider the functors $F, G : \underline{\mathrm{Set}} \to \underline{\mathrm{Set}}$ given by $FY := Y \times S$ and $GX := X^S := \mathrm{Mor}_{\underline{\mathrm{Set}}}(S, X)$.
- For sets Y and X due to *currying* we have

$$X^{FY} = X^{Y\times S} \overset{\text{curry}}{\cong} (X^S)^Y = (GX)^Y$$

- For functions $g : Y' \to Y$ and $f : X \to X'$, the following diagram commutes:

$$\begin{tikzcd}
X^{Y\times S} \arrow[r, "\mathrm{curry}", "\sim"'] \arrow[d, "Fg^* f_*"'] & (X^S)^Y \arrow[d, "g^* Gf_*"] \\
{X'}^{Y'\times S} \arrow[r, "\sim", "\mathrm{curry}"'] & ({X'}^S)^{Y'}
\end{tikzcd}$$

- Consequently, we get an adjunction $(\cdot \times S) \dashv (\cdot^S) : \underline{\mathrm{Set}} \to \underline{\mathrm{Set}}$.

Exponentials

- Let $\mathcal{C}$ be a category with finite products. If for each object S of $\mathcal{C}$, the functor $F := \cdot \times S$ has a right adjoint G (so $(\cdot \times S) \dashv G : \mathcal{C} \to \mathcal{C}$), we say that $\mathcal{C}$ has exponentials, and for an object Y of $\mathcal{C}$, we denote GY by Y^S.
- As we have seen on the last slide, the category $\underline{\mathrm{Set}}$ of sets has exponentials: For sets S and Y, Y^S is just the set of functions $\mathrm{Mor}_{\underline{\mathrm{Set}}}(S, Y)$ from S to Y.
- The category $\underline{\mathrm{Hask}}$ has exponentials: For types `s` and `y`, y^s is the *function type* `s -> y`.
- A category with (finite) sums and products and exponentials is called bicartesian closed. We have seen that both $\underline{\mathrm{Set}}$ and $\underline{\mathrm{Hask}}$ are bicartesian closed.

Adjunction example: partial functions

- Let $\underline{\mathrm{Prtl}}$ be the category of partial functions: Objects are sets, and a morphism between sets A and B is a partial function $f : A \dashrightarrow B$. This means f might be undefined for some $a \in A$.
- Obviously, a *partial* function $f : A \dashrightarrow B$ is just a *total* function $f : A \to B \uplus \{*\}$, where $f(a) = *$ means that f is undefined in a.
- We have an obvious functor $F : \underline{\mathrm{Set}} \to \underline{\mathrm{Prtl}}$ by considering a total function as partial. This functor sends a set to itself and a total function $f : A \to B$ to the total function $Ff : A \to B \uplus \{*\}$ with $Ff(a) = f(a)$ for all $a \in A$.
- In the other direction, consider the functor $G : \underline{\mathrm{Prtl}} \to \underline{\mathrm{Set}}$, which sends a set X to the set $X \uplus \{*\}$ and a partial function $X \dashrightarrow X'$, given by the total function $f : X \to X' \uplus \{*\}$, to the total function $Gf : X \uplus \{*\} \to X' \uplus \{*\}$ with $Gf(x) = f(x)$ for $x \in X$ and $Gf(*) = *$.

Adjunction example: partial functions (cntd.)

- For sets Y and X, we have

$$\begin{aligned}\mathrm{Mor}_{\underline{\mathrm{Prtl}}}(FY, X) &= \mathrm{Mor}_{\underline{\mathrm{Prtl}}}(Y, X) \\ &= \mathrm{Mor}_{\underline{\mathrm{Set}}}(Y, X \uplus \{*\}) = \mathrm{Mor}_{\underline{\mathrm{Set}}}(Y, GX).\end{aligned}$$

- For a total function $g : Y' \to Y$ and a partial function $f : X \dashrightarrow X'$, we can easily check that the following diagram commutes:

$$\begin{tikzcd}
\mathrm{Mor}_{\underline{\mathrm{Prtl}}}(FY, X) \arrow[r, "\varphi_{(Y,X)}", "\sim"'] \arrow[d, "Fg^* f_*"'] & \mathrm{Mor}_{\underline{\mathrm{Set}}}(Y, GX) \arrow[d, "g^* Gf_*"] \\
\mathrm{Mor}_{\underline{\mathrm{Prtl}}}(FY', X') \arrow[r, "\sim", "\varphi_{(Y',X')}"'] & \mathrm{Mor}_{\underline{\mathrm{Set}}}(Y', GX')
\end{tikzcd}$$

- As a consequence, we get an adjunction $F \dashv G : \underline{\mathrm{Prtl}} \to \underline{\mathrm{Set}}$.

Galois connections

- As a special case, consider two partially ordered sets $(C, \leq_C)$ and $(D, \leq_D)$ and their associated categories $\underline{C}$ and $\underline{D}$.
- We have seen last time that functors $F : \underline{C} \leftarrow \underline{D}$ and $G : \underline{C} \rightarrow \underline{D}$ are just *monotonic functions* $f : C \leftarrow D$ and $g : C \rightarrow D$.
- An adjunction $f \dashv g : C \rightarrow D$ in this context is called a Galois connection, and is given by a "natural" equivalence

$$\varphi_{(y,x)} : f(y) \leq_C x \iff y \leq_D g(x)$$

for elements $y \in D$ and $x \in C$.

- Naturality in this case translates to the following diagram, where $y' \leq_D y$ and $x \leq_C x'$:

$$\begin{tikzcd}
f(y) \leq_C x \arrow[r, Leftrightarrow, "\varphi_{(Y,X)}"] \arrow[d, Rightarrow] & y \leq_D g(x) \arrow[d, Rightarrow] \\
f(y') \leq_C x' \arrow[r, Leftrightarrow, "\varphi_{(Y',X')}"'] & y' \leq_D g(x')
\end{tikzcd}$$

Galois connection example: images and preimages

- Let $f : C \leftarrow D$ be a function between sets, and let $(\mathfrak{P}(C), \subseteq)$ and $(\mathfrak{P}(D), \subseteq)$ be the *powersets* of C and D, partially ordered by inclusion.
- We have monotonic maps $f_* : \mathfrak{P}(C) \leftarrow \mathfrak{P}(D)$ and $f^* : \mathfrak{P}(C) \rightarrow \mathfrak{P}(D)$, mapping a subset Y of D to its *image* $f(Y) \subseteq C$ and a subset X of C to its *preimage* $f^{-1}(X) \subseteq D$.
- Then $f_* \dashv f^* : \mathfrak{P}(C) \rightarrow \mathfrak{P}(D)$ is an adjunction/ Galois connection, because for subsets $Y \subseteq D$ and $X \subseteq C$ we have

$$f(Y) \subseteq X \iff \big(\forall y \in Y : f(y) \in X\big) \iff Y \subseteq f^{-1}(X).$$

Galois connection example: division

- Let $C = D = (\mathbb{N}, \geq)$, and for $n \geq 1$ consider the monotonic maps $f, g : \mathbb{N} \to \mathbb{N}$ given by $f(y) = ny$ and $g(x) = \lceil \frac{x}{n} \rceil$.
- For $y, x \in \mathbb{N}$, we have

$$f(y) \geq x \iff ny \geq x \iff y \geq_{\mathbb{Q}} \frac{x}{n} \iff y \geq \left\lceil \frac{x}{n} \right\rceil \iff y \geq g(x)$$

- As a consequence, we have an adjunction/ Galois connection

$$(\cdot n) \dashv \left\lceil \frac{\cdot}{n} \right\rceil : (\mathbb{N}, \geq) \to (\mathbb{N}, \geq).$$

Adjunction example: polynomial rings

- Let $\mathcal{C} = \underline{\mathrm{Ring}}$ be the category of rings, let $\mathcal{D} = \underline{\mathrm{Set}}$ be the category of sets, let $F : \underline{\mathrm{Ring}} \leftarrow \underline{\mathrm{Set}}$ be the functor $S \mapsto \mathbb{Z}[S]$, sending a set to the polynomial ring with variables given by the set elements, and let $G : \underline{\mathrm{Ring}} \to \underline{\mathrm{Set}}$ be the forgetful functor. Then $F \dashv G$:
- By the definition of polynomial rings, for each set S and ring R, we have

$$\varphi_{(S,R)} : \mathrm{Mor}_{\underline{\mathrm{Ring}}}(\mathbb{Z}[S], R) \xrightarrow{\sim} \mathrm{Mor}_{\underline{\mathrm{Set}}}(S, R),$$

- For a function $g : S' \to S$ and a ring homomorphism $f : R \to R'$, the following diagram commutes:

$$\begin{tikzcd}
\mathrm{Mor}_{\underline{\mathrm{Ring}}}(\mathbb{Z}[S], R) \arrow[r, "\varphi_{(S,R)}", "\sim"'] \arrow[d, "Fg^* f_*"'] & \mathrm{Mor}_{\underline{\mathrm{Set}}}(S, R) \arrow[d, "g^* Gf_*"] \\
\mathrm{Mor}_{\underline{\mathrm{Ring}}}(\mathbb{Z}[S'], R') \arrow[r, "\sim", "\varphi_{(S',R')}"'] & \mathrm{Mor}_{\underline{\mathrm{Set}}}(S', R')
\end{tikzcd}$$

Interlude: Monoids

- Don’t worry, monoids are a much easier concept than monads...
- ...however, there is the (in-)famous Haskell saying: *”A monad is simply a monoid in the category of endofunctors.”*

Interlude: Monoids

A monoid is a pair $(M, \cdot)$, where M is a set and

$$\cdot : M \times M \longrightarrow M,\ \ (m, n) \mapsto m \cdot n$$

is an *associative* operation possessing a left- and right-neutral element $1_M \in M$.

Interlude: Monoids

- A monoid homomorphism $\varphi : (M, \cdot) \to (N, \cdot)$ is a function from M to N which ”respects” the operation:

$$\forall m, m' \in M : \varphi(m \cdot m') = \varphi(m) \cdot \varphi(m').$$

- The identity function is a monoid homomorphism, and the composition of two monoid homorphisms is again a monoid homomorphism, so we get the category of monoids $\underline{\mathrm{Mnd}}$.

Examples of monoids

- If $(G, \cdot)$ is a *group*, then by ”forgetting” the existence of inverses, we get a monoid. In particular, we get a forgetful functor $\underline{\mathrm{Grp}} \to \underline{\mathrm{Mnd}}$.

Examples of monoids

- If $(G, \cdot)$ is a *group*, then by ”forgetting” the existence of inverses, we get a monoid. In particular, we get a forgetful functor $\underline{\text{Grp}} \to \underline{\text{Mnd}}$.
- If $(R, +, \cdot)$ is a ring, then by ”forgetting” multiplication, we get a group (and hence in particular a monoid) $(R, +)$, and by ”forgetting” addition, we get a monoid $(R, \cdot)$. In particular, we have forgetful functors $\underline{\text{Ring}} \to \underline{\text{Grp}} \to \underline{\text{Mnd}}$ and $\underline{\text{Ring}} \to \underline{\text{Mnd}}$.

Examples of monoids

- If $(G, \cdot)$ is a *group*, then by "forgetting" the existence of inverses, we get a monoid. In particular, we get a forgetful functor $\underline{\mathrm{Grp}} \to \underline{\mathrm{Mnd}}$.
- If $(R, +, \cdot)$ is a ring, then by "forgetting" multiplication, we get a group (and hence in particular a monoid) $(R, +)$, and by "forgetting" addition, we get a monoid $(R, \cdot)$. In particular, we have forgetful functors $\underline{\mathrm{Ring}} \to \underline{\mathrm{Grp}} \to \underline{\mathrm{Mnd}}$ and $\underline{\mathrm{Ring}} \to \underline{\mathrm{Mnd}}$.
- $(\mathbb{N}, +)$ – the natural numbers with addition – are a monoid. The neutral element is $0 \in \mathbb{N}$.

Examples of monoids

- If $(G, \cdot)$ is a *group*, then by ”forgetting” the existence of inverses, we get a monoid. In particular, we get a forgetful functor $\underline{\mathrm{Grp}} \to \underline{\mathrm{Mnd}}$.
- If $(R, +, \cdot)$ is a ring, then by ”forgetting” multiplication, we get a group (and hence in particular a monoid) $(R, +)$, and by ”forgetting” addition, we get a monoid $(R, \cdot)$. In particular, we have forgetful functors $\underline{\mathrm{Ring}} \to \underline{\mathrm{Grp}} \to \underline{\mathrm{Mnd}}$ and $\underline{\mathrm{Ring}} \to \underline{\mathrm{Mnd}}$.
- $(\mathbb{N}, +)$ – the natural numbers with addition – are a monoid. The neutral element is $0 \in \mathbb{N}$.
- For an object X in a category $\mathcal{C}$, the set of endomorphisms $\mathrm{End}_{\mathcal{C}}(X) := \mathrm{Mor}_{\mathcal{C}}(X, X)$, paired with composition as operation, is a monoid. The neutral element is the identity 1_X. (Square matrices with multiplication!)

Monoids in Haskell

Monoids play quite a prominent role in Haskell. From `Data.Monoid`:

```haskell
class Monoid a where
  mempty  :: a
  mappend :: a -> a -> a
  mconcat :: [a] -> a   -- = foldr mappend mempty
```

```haskell
(<>) :: Monoid a => a -> a -> a
(<>) = mappend
```

Examples of monoids in Haskell – lists

```haskell
instance Monoid [a] where
  mempty  = []
  mappend = (++)
```

Examples of monoids in Haskell – Sum

```haskell
newtype Sum a = Sum {getSum :: a}
  deriving Show
```

```haskell
instance Num a => Monoid (Sum a) where
  mempty = Sum 0
  Sum x `mappend` Sum y = Sum $ x + y
```

```
GHCi> mconcat (map Sum [1..10])
Sum {getSum = 55}
```

Examples of monoids in Haskell – `Product`

```haskell
newtype Product a = Product {getProduct :: a}
  deriving Show
```

```haskell
instance Num a => Monoid (Product a) where
  mempty = Product 1
  Product x `mappend` Product y = Product $ x * y
```

```
GHCi> mconcat (map Product [1..5])
Product {getProduct = 120}
```

Examples of monoids in Haskell – pairs

```haskell
instance (Monoid a, Monoid b)
   => Monoid (a, b) where
  mempty = (mempty, mempty)
  (a, b) `mappend` (a', b') = (a <> a', b <> b')
```

```
GHCi> (Sum 1, Product 2) <> (Sum 2, Product 3)
(Sum {getSum = 3}, Product {getProduct = 6})
```

Examples of monoids in Haskell – functions

```
instance Monoid b => Monoid (a -> b) where
  mempty         = const mempty
  mappend f g x = f x <> g x
```

```
GHCi> (show <> show) 42
"4242"
```

Examples of monoids in Haskell – `Ordering`

```haskell
instance Monoid Ordering where
  mempty = EQ
  LT `mappend` _ = LT
  GT `mappend` _ = GT
  EQ `mappend` x = x
```

```
GHCi> compare 1 2 `mappend` compare 'G' 'A'
LT
GHCi> compare 2 2 `mappend` compare 'G' 'A'
GT
```

This is very useful for lexicographical ordering!

Examples of monoids in Haskell – `First`

```haskell
newtype First a = First {getFirst :: Maybe a}
  deriving Show
```

```haskell
instance Monoid (First a) where
  mempty = First Nothing
  First x `mappend` First y = First $ case (x, y) of
    (Just a, _)   -> Just a
    (Nothing, m) -> m
```

Examples of monoids in Haskell – `First`

```haskell
newtype First a = First {getFirst :: Maybe a}
  deriving Show
```

```haskell
instance Monoid (First a) where
  mempty = First Nothing
  First x `mappend` First y = First $ case (x, y) of
    (Just a, _)   -> Just a
    (Nothing, m) -> m
```

```
GHCi> mconcat
        (map First [Nothing, Just 1, Just 2])
First {getFirst = Just 1}
```

Examples of monoids in Haskell – `Last`

```haskell
newtype Last a = Last {getLast :: Maybe a}
  deriving Show
```

```haskell
instance Monoid (Last a) where
  mempty = Last Nothing
  Last x `mappend` Last y = Last $ case (x, y) of
    (_, Just a)   -> Just a
    (m, Nothing) -> m
```

Examples of monoids in Haskell – `Last`

```haskell
newtype Last a = Last {getLast :: Maybe a}
  deriving Show
```

```haskell
instance Monoid (Last a) where
  mempty = Last Nothing
  Last x `mappend` Last y = Last $ case (x, y) of
    (_, Just a)   -> Just a
    (m, Nothing) -> m
```

```
GHCi> mconcat
        (map Last [Nothing, Just 1, Just 2])
Last {getLast = Just 2}
```

Examples of monoids in Haskell – `Endo`

```haskell
newtype Endo a = Endo {appEndo :: a -> a}
```

```haskell
instance Monoid (Endo a) where
  mempty = Endo id
  Endo f `mappend` Endo g = Endo (f . g)
```

Examples of monoids in Haskell – `Endo`

```haskell
newtype Endo a = Endo {appEndo :: a -> a}
```

```haskell
instance Monoid (Endo a) where
  mempty = Endo id
  Endo f `mappend` Endo g = Endo (f . g)
```

```
GHCi> appEndo (mconcat (map Endo [succ, succ])) 3
5
```

Adjunction example: monoids

- Let $\mathcal{C} = \underline{\mathrm{Mnd}}$ be the category of monoids, let $\mathcal{D} = \underline{\mathrm{Set}}$ be the category of sets, let $F : \underline{\mathrm{Mnd}} \leftarrow \underline{\mathrm{Set}}$ be the functor $S \mapsto [S]$, sending a set to the monoid of *lists* of elements of S, and let $G : \underline{\mathrm{Mnd}} \rightarrow \underline{\mathrm{Set}}$ be the forgetful functor. Then $F \dashv G : \underline{\mathrm{Mnd}} \rightarrow \underline{\mathrm{Set}}$:
- For each set S and monoid M, we have

$$\varphi_{(S,M)} : \mathrm{Mor}_{\underline{\mathrm{Mnd}}}([S], M) \xrightarrow{\sim} \mathrm{Mor}_{\underline{\mathrm{Set}}}(S, M),\ \alpha \mapsto \big(s \mapsto \alpha([s])\big),$$

where the inverse $\psi_{(S,M)}$ maps a function $g : S \rightarrow M$ to the monoid homomorphism

$$[s_1, s_2, \ldots, s_n] \mapsto g(s_1) \cdot g(s_2) \cdot \ldots \cdot g(s_n).$$

- For a function $g : S' \rightarrow S$ and a monoid homomorphism $f : M \rightarrow M'$, the following diagram commutes:

$$\begin{tikzcd}
\mathrm{Mor}_{\underline{\mathrm{Mnd}}}([S], M) \arrow[r, "\varphi_{(S,M)}", "\sim"'] \arrow[d, "Fg^* f_*"'] & \mathrm{Mor}_{\underline{\mathrm{Set}}}(S, M) \arrow[d, "g^* Gf_*"] \\
\mathrm{Mor}_{\underline{\mathrm{Mnd}}}([S'], M') \arrow[r, "\sim", "\varphi_{(S',R')}"'] & \mathrm{Mor}_{\underline{\mathrm{Set}}}(S', M')
\end{tikzcd}$$

Adjunction example: monoids (cntd.)

The inverse ψ of the last example is called `foldMap` in Haskell:

```haskell
foldMap :: Monoid m => (s -> m) -> [s] -> m
foldMap f = foldl' (\ m s -> m <> f s) mempty
```

It is actually more general, defined for arbitrary `Foldable`s, and a method of class `Foldable`.

```
GHCi> foldMap Sum [1 . . 10]
Sum {getSum = 55}
```

The meaning of ”free”

- We have seen two examples of adjunctions $F \dashv G$ with G a ”forgetful” functor.
- We have seen a third example of the same kind last Friday, the functor sending a set to itself, considered as a topological space with the *discrete topology*. That example, too, is an adjunction.
- Whenever we have a left-adjoint F to a forgetful functor G, we call objects of type FX free.
- So polynomial rings are ”free rings”, lists are ”free monoids”, and discrete topological spaces are ”free topological spaces”.
- There are many more examples: We have ”free groups”, ”free vector spaces”, ”free Abeliean groups” and so on.

Free monads in Haskell

Let $\mathcal{C}$ be the category whose objects are *Haskell monads* and whose morphisms are polymorphic functions `f :: m a -> n a` satisfying `f (return x) = return x` and

```
f (m >>= k) = f m >>= (f . k)
```

Let $\mathcal{D}$ be the category whose objects are *Haskell functors* and whose morphisms are polymorphic functions `h :: f a -> g a`. Then the functor `Free` from $\mathcal{D}$ to $\mathcal{C}$, sending a functor `f` to the *free monad* `Free f`, is left-adjoint to the forgetful functor from $\mathcal{C}$ to $\mathcal{D}$.

Free monads in Haskell (cntds.)

For a functor `f` and a monad `m`, the inverse of the isomorphism is given by the following Haskell function:

```
foldFree :: Monad m => (forall x . f x -> m x)
                    -> Free f a -> m a
foldFree _ (Return a) = return a
foldFree g (Wrap x)   = g x >>= foldFree g
```

This is yet another example of *rank-2 polymorphism*!

Unit

- Let $F \dashv G : \mathcal{C} \to \mathcal{D}$ be an adjunction.
- For an object $Y \in \mathrm{Ob}(\mathcal{D})$, the adjunction gives us an isomorphism

$$\varphi_{(Y,FY)} : \mathrm{Mor}_{\mathcal{C}}(FY, FY) \xrightarrow{\sim} \mathrm{Mor}_{\mathcal{D}}(Y, GFY).$$

- The image of the identity 1_{FY} under $\varphi_{(Y,FY)}$ in $\mathrm{Mor}_{\mathcal{D}}(Y, GFY)$ is denoted by $\eta_Y : Y \to GFY$.
- By naturality, the η_Y define a natural transformation $\eta : 1_{\mathcal{D}} \to GF$ between endofunctors on $\mathcal{D}$. This natural transformation is called the unit of the adjunction.

Counit

- Let $F \dashv G : \mathcal{C} \to \mathcal{D}$ be an adjunction.
- For an object $X \in \mathrm{Ob}(\mathcal{C})$, the adjunction gives us an isomorphism

$$\varphi_{(GX,X)} : \mathrm{Mor}_{\mathcal{C}}(FGX, X) \xrightarrow{\sim} \mathrm{Mor}_{\mathcal{D}}(GX, GX).$$

- The preimage of the identity 1_{GX} under $\varphi_{(GX,X)}$ in $\mathrm{Mor}_{\mathcal{C}}(FGX, X)$ is denoted by $\epsilon : FGX \to X$.
- By naturality, the ϵ_X define a natural transformation $\epsilon : FG_{\mathcal{C}} \to 1_{\mathcal{C}}$ between endofunctors on $\mathcal{C}$. This natural transformation is called the counit of the adjunction.

Counit-unit equations

- Let $F \dashv G : \mathcal{C} \to \mathcal{D}$ be an adjunction with unit $\eta : 1_{\mathcal{D}} \to GF$ and counit $\epsilon : FG \to 1_{\mathcal{C}}$.
- Then for objects X in $\mathcal{C}$ and Y in $\mathcal{D}$, the following equations hold:

$$1_{FY} = \epsilon_{FY} \circ F(\eta_Y) : FY \to FGFY \to FY$$
$$1_{GX} = G(\epsilon_X) \circ \eta_{GX} : GX \to GFGX \to GX.$$

- These equations are called the counit-unit equations.

(Co-)unit example: currying

- For a set S, Consider the adjunction $(\cdot \times S) \dashv (\cdot^S) : \underline{\mathrm{Set}} \to \underline{\mathrm{Set}}$.
- The *unit* $\eta : 1_{\underline{\mathrm{Set}}} \to (\cdot^S)(\cdot \times S)$ of this adjunction sends a set Y to the function $\eta_Y : Y \to (Y \times S)^S$ given by $\eta_Y(y)(s) = (y, s)$.
- The *counit* $\epsilon : (\cdot \times S)(\cdot^S) \to 1_{\underline{\mathrm{Set}}}$ of this adjunction sends a set X to the function $\epsilon_X : X^S \times S \to X$, given by *evaluation*: $\epsilon_X(\alpha, s) = \alpha(s)$.

(Co-)unit example: partial functions

- Consider the adjunction $F \dashv G : \underline{\mathrm{Prtl}} \to \underline{\mathrm{Set}}$.
- The *unit* $\eta : 1_{\underline{\mathrm{Set}}} \to GF$ of this adjunction sends a set Y to the total function $\eta_Y : Y \to Y \uplus \{*\}$ given by $\eta_Y(y) = y$.
- The *counit* $\epsilon : FG \to 1_{\underline{\mathrm{Prtl}}}$ of this adjunction sends a set X to a partial function $\epsilon_X : X \uplus \{*\} \dashrightarrow X$, given by the identity when considered as a total function from $X \uplus \{*\}$ to itself.

(Co-)unit example: lists

- Consider the adjunction $F \dashv G : \underline{\mathrm{Mnd}} \to \underline{\mathrm{Set}}$.
- The *unit* $\eta : 1_{\mathcal{D}} \to GF$ of this adjunction sends a *set* Y to the *function* $\eta_Y : Y \to [Y]$, given by mapping $y \in Y$ to the singleton list $[y] \in [Y]$.
- The *counit* $\epsilon : FG \to 1_{\mathcal{C}}$ of this adjunction sends a *monoid* $(X, \cdot)$ to the *monoid homomorphism* $\epsilon_X : [X] \to X$, given by mapping a list $[x_1, x_2, \ldots, x_n]$ to $x_1 \cdot x_2 \cdot \ldots \cdot x_n \in X$.

(Co-)unit example: (pre-)images

- Consider the adjunction $f_* \dashv f^* : (\mathfrak{P}(C), \subseteq) \to (\mathfrak{P}(D), \subseteq)$ given by a function $f : C \leftarrow D$.
- The *unit* $\eta : 1_{\mathfrak{P}(D)} \to f^* f_*$ of this adjunction sends a subset $Y \subseteq D$ to the *statement* $\eta_Y : Y \subseteq f^{-1}(f(Y))$.
- The *counit* $\epsilon : f_* f^* \to 1_{\mathfrak{P}(C)}$ of this adjunction sends a subset $X \subseteq C$ to the *statement* $\epsilon_X : f(f^{-1}(X)) \subseteq X$.

(Co-)unit example: division

- For $n \geq 1$, consider the adjunction $(\cdot n) \dashv \lceil \frac{\cdot}{n} \rceil : (\mathbb{N}, \geq) \to (\mathbb{N}, \geq)$.
- The *unit* $\eta : 1_{\mathbb{N}} \to \lceil \frac{\cdot}{n} \rceil (\cdot n)$ of this adjunction sends a $y \in \mathbb{N}$ to the *statement* $\eta_y : y \geq \lceil \frac{ny}{n} \rceil = \lceil y \rceil = y$.
- The *counit* $\epsilon : (\cdot n) \lceil \frac{\cdot}{n} \rceil \to 1_{\mathbb{N}}$ of this adjunction sends an $x \in \mathbb{N}$ to the *statement* $\epsilon_x : n \cdot \lceil \frac{x}{n} \rceil \geq x$.

(Co-)unit example: free monads

Consider the adjunction $F \dashv G : \mathcal{C} \to \mathcal{D}$ from Haskell monads to Haskell functors.

The *unit* $\eta : 1_{\mathcal{D}} \to GF$ of this adjunction is given by

```
eta :: Functor y => y a -> Free y a
eta = Wrap . fmap return
```

The *counit* $\epsilon : FG \to 1_{\mathcal{C}}$ of this adjunction is

```
epsilon :: Monad x => Free x a -> x a
epsilon (Return a) = return a
epsilon (Wrap u)   = u >>= epsilon
```

Monads

Monad

Let $\mathcal{C}$ be a category. A monad in $\mathcal{C}$ is an endofunctor $M : \mathcal{C} \to \mathcal{C}$, together with natural transformations $\eta : 1_{\mathcal{C}} \to M$ and $\mu : MM \to M$, such that the following coherence conditions are satisfied:

- $\mu \circ M\mu = \mu \circ \mu M : MMM \to M$,
- $\mu \circ M\eta = \mu \circ \eta M = 1_M : M \to M$.

$$\begin{tikzcd}
MMM \arrow[r, "M\mu"] \arrow[d, "\mu M"'] & MM \arrow[d, "\mu"] \\
MM \arrow[r, "\mu"'] & M
\end{tikzcd}
\qquad
\begin{tikzcd}
M \arrow[r, "\eta M"] \arrow[d, "M\eta"'] \arrow[dr, equal] & MM \arrow[d, "\mu"] \\
MM \arrow[r, "\mu"'] & M
\end{tikzcd}$$

Note

η is `return`, μ is `join`, the coherence conditions are associativity of `join` and neutrality of `return`.

Monads from adjunctions

- One versatile way to construct monads is from adjunctions.
- Consider an adjunction $F \dashv G : \mathcal{C} \to \mathcal{D}$ with unit $\eta : 1_{\mathcal{D}} \to GF$ and counit $\epsilon : FG \to 1_{\mathcal{C}}$.
- Then the counit-unit equations imply that $(GF, \eta, G\epsilon F)$ is a monad in $\mathcal{D}$.

Monad example: monoid

- Consider the adjunction $F \dashv G : \underline{\mathrm{Mnd}} \to \underline{\mathrm{Set}}$ with unit $\eta : 1_{\mathcal{D}} \to GF$ and counit $\epsilon : FG \to 1_{\mathcal{C}}$.
- Remember that for a set Y, the unit $Y \to [Y]$ sends $y \in Y$ to the singleton list $[y]$ (this is `return`).
- Remember also that for a monoid $(X, \cdot)$, the counit $[X] \to X$ sends a list $[x_1, x_2, \ldots, x_n]$ to $x_1 \cdot x_s \cdot \ldots \cdot x_n$.
- Therefore for a set Y, $\mu = G\epsilon F : [[Y]] \to [Y]$ sends a list of lists $[l_1, l_2, \ldots, l_n]$ to $l_1 \mathtt{++} l_2 \mathtt{++} \ldots \mathtt{++} l_n$.
- We have rediscovered the *list monad*!

Monad example: currying

- Consider the adjunction $(\cdot \times S) \dashv (\cdot^S) : \underline{\mathrm{Set}} \to \underline{\mathrm{Set}}$.
- Remember that the unit $\eta : 1_{\underline{\mathrm{Set}}} \to (\cdot^S)(\cdot \times S)$ is given by $\eta_Y(y)(s) = (y, s)$.
- Remeber that the counit $\epsilon : (\cdot \times S)(\cdot^S) \to 1_{\underline{\mathrm{Set}}}$ is given by *evaluation*.
- Therefore for a set Y,

$$\mu = (\cdot^S)\epsilon(\cdot \times S) : \left((Y \times S)^S \times S\right)^S \to (Y \times S)^S$$

is given by

```
mu :: (s -> (s -> (y, s), s)) -> s -> (y, s)
mu f s = let (g, s') = f s in g s'
```

- We get the *state monad* $Y \mapsto (Y \times S)^S$!
- This would work exactly the same in any category with products and exponentials.

Monad example: partial functions

- Consider the adjunction $F \dashv G : \underline{\mathrm{Prtl}} \to \underline{\mathrm{Set}}$ with unit $\eta : 1_{\mathcal{D}} \to GF$ and counit $\epsilon : FG \to 1_{\mathcal{C}}$.
- Remember that for a set Y, the unit $Y \to Y \uplus \{*\}$ sends $y \in Y$ to itself.
- Remember also that for a set X, the counit $X \uplus \{*\} \dashrightarrow X$ is the identity when considered as a total function.
- Therefore for a set Y,

$$\mu = G\epsilon F : (Y \uplus \{*\}) \uplus \{*\} \to Y \uplus \{*\}$$

sends an $y \in Y$ to itself and both $*$'s to $*$.
- We have rediscovered the *Maybe monad*!

Algebras and Coalgebras

F-Algebras

- Let $\mathcal{C}$ be a category, and let $F : \mathcal{C} \to \mathcal{C}$ be an endofunctor.
- An F-algebra is a morphism $\alpha : FX \to X$ in $\mathcal{C}$ for some object X in $\mathcal{C}$.
- A morphism of F-algebras $\alpha : FX \to X$ and $\beta : FY \to Y$ is a morphism $f : X \to Y$ in $\mathcal{C}$ such that the following diagram commutes:

$$\begin{tikzcd} FX \arrow[r, "Ff"] \arrow[d, "\alpha"'] & FY \arrow[d, "\beta"] \\ X \arrow[r, "f"'] & Y \end{tikzcd}$$

- The functor laws imply that identities in $\mathcal{C}$ are F-algebra morphisms and that the composition of two F-algebra morphisms is again an F-algebra morphism, so we get the category of F-algebras $\underline{\mathrm{Alg}}(F)$.

Initial F-Algebras

- Let $\mathcal{C}$ be a category, and let $F : \mathcal{C} \to \mathcal{C}$ be an endofunctor.
- If the category $\underline{\mathrm{Alg}}(F)$ of F-algebras has an initial object, it is denoted by $in : F(\mu F) \to \mu F$ and called the initial F-algebra.
- In that case, let $\alpha : FX \to X$ be any F-algebra. By the definition of "initial object", there is a unique F-algebra morphism $(\!|\alpha|\!) : in \to \alpha$, called a fold or catamorphism:

$$\begin{tikzcd}
F(\mu F) \arrow[r, "F(\!|\alpha|\!)"] \arrow[d, "in"'] & FX \arrow[d, "\alpha"] \\
\mu F \arrow[r, "\exists!", "(\!|\alpha|\!)"'] & X
\end{tikzcd}$$

Initial algebra example: $\mathbb{N}$

- In the category $\underline{\mathrm{Set}}$ of sets, consider the endofunctor $(\cdot \uplus \{*\})$.
- Let $\mathbb{B}$ be the set $\{T, F\}$, then

$$F\mathbb{B} = \mathbb{B} \uplus \{*\} \longrightarrow \mathbb{B},\ x \mapsto \begin{cases} T & \text{if } x = * \\ F & \text{otherwise} \end{cases}$$

  is an F-algebra.
- The initial F-algebra exists and is given by

$$in : F\mathbb{N} = \mathbb{N} \uplus \{*\} \longrightarrow \mathbb{N},\ x \mapsto \begin{cases} 0 & \text{if } x = * \\ x + 1 & \text{otherwise.} \end{cases}$$

- The catamorphism $(\!|z|\!) : \mathbb{N} \to \mathbb{B}$ sends 0 to T and everything else to F:

$$\begin{tikzcd} \mathbb{N} \uplus \{*\} \arrow[r, "(\!|z|\!)+1_*"] \arrow[d, "in"'] & \mathbb{B} \uplus \{*\} \arrow[d, "z"] \\ \mathbb{N} \arrow[r, "\exists!", "(\!|z|\!)"'] & \mathbb{B} \end{tikzcd}$$

- In this case, catamorphisms correspond to *primitive recursion on natural numbers.*

F-Coalgebras

- Let $\mathcal{C}$ be a category, and let $F : \mathcal{C} \to \mathcal{C}$ be an endofunctor.
- An F-coalgebra is a morphism $\alpha : X \to FX$ in $\mathcal{C}$ for some object X in $\mathcal{C}$.
- A morphism of F-coalgebras $\alpha : X \to FX$ and $\beta : Y \to FY$ is a morphism $f : X \to Y$ in $\mathcal{C}$ such that the following diagram commutes:

$$\begin{tikzcd}
FX \arrow[r, "Ff"] & FY \\
X \arrow[u, "\alpha"] \arrow[r, "f"'] & Y \arrow[u, "\beta"']
\end{tikzcd}$$

- The functor laws imply that identities in $\mathcal{C}$ are F-coalgebra morphisms and that the composition of two F-coalgebra morphisms is again an F-coalgebra morphism, so we get the category of F-coalgebras $\underline{\mathrm{Coalg}}(F)$.

Final F-Coalgebras

- Let $\mathcal{C}$ be a category, and let $F : \mathcal{C} \to \mathcal{C}$ be an endofunctor.
- If the category $\underline{\mathrm{Coalg}}(F)$ of F-coalgebras has a final object, it is denoted by $out : \nu F \to F(\nu F)$ and called the final F-coalgebra.
- In that case, let $\alpha : X \to FX$ be any F-coalgebra. By the definition of "final object", there is a unique F-coalgebra morphism $[\![\alpha]\!] : \alpha \to out$, called an unfold or anamorphism:

$$\begin{tikzcd}
FX \arrow[r, "F[\![\alpha]\!]"] & F(\nu F) \\
X \arrow[u, "\alpha"] \arrow[r, "\exists!", "{[\![\alpha]\!]}"'] & \nu F \arrow[u, "out"']
\end{tikzcd}$$

Final coalgebra example: sequences

- In the category $\underline{\mathrm{Set}}$ of sets, consider the endofunctor $(\mathbb{N} \times \cdot)$.
- Then
$$b : \mathbb{N} \longrightarrow \mathbb{N} \times \mathbb{N} = F\mathbb{N},\ n \mapsto (n, n+1)$$
is an F-coalgebra.
- The final F-coalgebra exists and is given by
$$out : \mathbb{N}^{\mathbb{N}} \longrightarrow \mathbb{N} \times \mathbb{N}^{\mathbb{N}} = F\mathbb{N},\ f \mapsto \big(f(0), n \mapsto f(n+1)\big)$$
(Note that $\mathbb{N}^{\mathbb{N}}$ is the set of sequences of natural numbers!)
- The anamorphism $[\![b]\!] : \mathbb{N} \to \mathbb{N}^{\mathbb{N}}$ sends $n \in \mathbb{N}$ to the sequence $n, n+2, n+2, \ldots$:
$$\begin{tikzcd}
\mathbb{N} \times \mathbb{N} \arrow[r, "1_{\mathbb{N}} \times [\![b]\!]"] & \mathbb{N} \times \mathbb{N}^{\mathbb{N}} \\
\mathbb{N} \arrow[u, "b"] \arrow[r, "\exists!", "{[\![b]\!]}"'] & \mathbb{N}^{\mathbb{N}} \arrow[u, "out"']
\end{tikzcd}$$
- In this case, anamorphisms correspond to *corecursion*.

Algebras and Coalgebras in Haskell

Recall the following definition:

```haskell
newtype Fix f = In {out :: f (Fix f)}
```

If `f` is a Haskell functor, then `In` is the initial *f*-algebra, and `out` is the final *f*-coalgebra.

```haskell
cata :: Functor f => (f a -> a) -> Fix f -> a
cata g = g . fmap (cata g) . out
```

```haskell
ana :: Functor f => (a -> f a) -> a -> Fix f
ana g = In . fmap (ana g) . g
```

Example: lists

```
data ListF a b = Nil | Cons a b
  deriving (Show, Functor)
```

```
prod :: Num a => ListF a a -> a
prod Nil            = 1
prod (Cons a acc) = a * acc
```

```
GHCi> cata prod $ In $ Cons 2 $ In $ Cons 3 $ In Nil
6
```

Example: lists

```haskell
data ListF a b = Nil | Cons a b
  deriving (Show, Functor)
```

```haskell
sh :: Show a => ListF a String -> String
sh Nil = "Nil"
sh (Cons a acc) = show a ++ ": " ++ acc
```

```
GHCi> cata sh $ In $ Cons 2 $ In $ Cons 3 $ In Nil
"2: 3: Nil"
```

Catamorphisms tear down data structures.

Example: lists

```haskell
data ListF a b = Nil | Cons a b
  deriving (Show, Functor)
```

```haskell
build :: Int -> ListF Int Int
build 0 = Nil
build n = Cons n (n - 1)
```

```
GHCi> cata sh $ ana build $ 4
"4: 3: 2: 1: Nil"
GHCi> cata prod $ ana build $ 5
120
```

Anamorphisms build up data structures.

Hylomorphisms

- Let $\mathcal{C}$ be a category, and let $F : \mathcal{C} \to \mathcal{C}$ be an endofunctor for which both the initial algebra and the final coalgebra exist and ”coincide” in the following sense: The initial algebra $in : F(\mu F) \to \mu F$ is an isomorphism, and $out := in^{-1} : \mu F \to F(\mu F)$ is the final coalgebra.
- For example, we have this situation in $\underline{\mathrm{Hask}}$ for any Haskell functor `f`.
- The composition $(\!|\alpha|\!)[\![\beta]\!]$ of a catamorphism α with an anamorphism β (both for F) is called a hylomorphism:

$$\begin{tikzcd}
FX \arrow[r, "F[\![\beta]\!]"] & F(\nu F) \arrow[r, equal] & F(\mu F) \arrow[r, "F(\!|\alpha|\!)"] \arrow[d, "in"'] & FY \arrow[d, "\alpha"] \\
X \arrow[u, "\beta"] \arrow[r, "[\![\beta]\!]"'] & \nu F \arrow[u, "out"'] \arrow[r, equal] & \mu F \arrow[r, "(\!|\alpha|\!)"'] & Y
\end{tikzcd}$$

Hylomorphisms

- Let $\mathcal{C}$ be a category, and let $F : \mathcal{C} \to \mathcal{C}$ be an endofunctor for which both the initial algebra and the final coalgebra exist and "coincide" in the following sense: The initial algebra $in : F(\mu F) \to \mu F$ is an isomorphism, and $out := in^{-1} : \mu F \to F(\mu F)$ is the final coalgebra.
- For example, we have this situation in $\underline{\mathbf{Hask}}$ for any Haskell functor `f`.
- We do not need the intermediate part and can "fuse it away" – a process known as deforestation.

$$\begin{tikzcd}
FX \arrow[r, "F\left(\llparenthesis\alpha\rrparenthesis\llbracket\beta\rrbracket\right)"] & FY \arrow[d, "\alpha"] \\
X \arrow[u, "\beta"] \arrow[r, "\llparenthesis\alpha\rrparenthesis\llbracket\beta\rrbracket"'] & Y
\end{tikzcd}$$

Hylomorphisms in Haskell

We can define hylomorphisms in Haskell:

```
hylo :: Functor f => (f a -> a) -> (b -> f b) -> b -> a
hylo g h = g . fmap (hylo g h) . h
```

```
GHCi> hylo prod build 5
120
```

This version has ”fused away” the intermediate list and is more efficient.