# Seminář 7: Monády

## Monáda

- Monáda $M = (T, \eta, \mu)$ – endofunktor $T$ a dvě přirozené transformace $\eta$ a $\mu$

$$\begin{aligned} &T : C \to C \\ &\eta : Id \implies T && \text{(jednotka)} \\ &\mu : T^2 \implies T && \text{(násobení)} \end{aligned}$$

- Tyto dva diagramy komutují (čtverec asociativity a dva jednotkové trojúhelníky)

$$\begin{tikzcd} T^3 \arrow[r, Rightarrow, "\mu \cdot T"] \arrow[d, Rightarrow, "T \cdot \mu"'] & T^2 \arrow[d, Rightarrow, "\mu"] \\ T^2 \arrow[r, Rightarrow, "\mu"'] & T \end{tikzcd} \qquad \begin{tikzcd} T \arrow[r, Rightarrow, "\eta \cdot T"] \arrow[dr, Rightarrow, "id"'] & T^2 \arrow[d, Rightarrow, "\mu"] & T \arrow[l, Rightarrow, "T \cdot \eta"'] \arrow[dl, Rightarrow, "id"] \\ & T & \end{tikzcd}$$

## Kleisliho kategorie

Monáda nad kategorií tvoří novou kategorii nazvanou Kleisliho kategorie. $C$ je kategorie a $M = (T, \eta, \mu)$ je monáda nad ní. Kleisliho kategorie $C_T$ je potom následující:

- Objekty z $C_T$ přímo odpovídají objektům z $C$
- Morfismy z $C_T$ jsou ve formě $f : a \to Tb$, tedy: $Hom_{C_T}(a, b) \simeq Hom_C(a, Tb)$
- Kompozice morfismů $f_T : a \to Tb$, $g_T : b \to Tc$ definována: $g_T \circ_T f_T \equiv \mu_c \circ (Tg) \circ f$
- Identity $id_{aT}$ jsou rovny $\eta_a$

## Monády a adjunkce

Mějme adjunkci $L \dashv R$ pro kategorie $C, D$.

$$\begin{aligned} \eta &: I_D \to R \circ L \\ \epsilon &: L \circ R \to I_C \end{aligned}$$

Tedy $R \circ L$ je endofunktor. Jednotce odpovídá $\eta$, násobení se definuje takto:

$$\mu = R \circ \epsilon \circ L$$

**Platí:** adjunkce $R \dashv L \implies$ uvnitř je monáda $(R \circ L, \eta, \mu)$

**Opačně pouze:** monáda nad C $\implies \exists$ adjunkce $F \dashv G$, kde $F : C \to C_T$ a $G : C_T \to C$

## Haskell

Pro názornost si zavedeme následující definici monády v Haskellu:

```
class Functor m => Monad m where
    join :: m (m a) -> m a
    return :: a -> m a
```

$$\texttt{join} \cong \mu$$
$$\texttt{return} \cong \eta$$

Skutečná definice je ekvivalentní. Funkci `bind` je možné napsat pomocí `join` a `fmap`.

```
class Applicative m => Monad m where
    return :: a -> m a
    (>>=) :: m a -> (a -> m b) -> m b
```

```
a >>= f = join (fmap f a)
```

Pro připomenutí:

```
fmap :: Functor f => (a -> b) -> f a -> f b
```

### Kleisliho rybí operátory

Operátor pro zřetězení monádových funkcí (obdoba `(.)`):

```
(<=<) :: Monad m => (b -> m c) -> (a -> m b) -> a -> m c
g <=< f = join . fmap g . f
```

Verze plující na druhou stranu:

```
(>=>) = flip (<=<)
```

## Příklady

```
instance Monad Maybe where
    join (Just (Just x)) = Just x
    join _               = Nothing
    return a = Just a
```

```
instance Monad [] where
    join = concat
    return x = [x]
```

```
instance Writer Str where
    join (Writer ((Writer (a, s')), s)) = Writer (a, s ++ s')
```