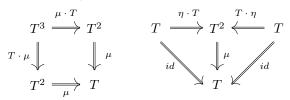
Seminář 7: Monády

Monáda

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

$$\begin{split} T:C \to C \\ \eta:Id &\Longrightarrow T \\ \mu:T^2 &\Longrightarrow T \end{split} \tag{jednotka}$$
 (násobení)

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



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 η_a

Monády a adjunkce

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

$$\eta: I_D \to R \circ L$$

$$\epsilon: L \circ R \to I_C$$

Tedy $R \circ L$ je endofunktor. Jednotce odpovídá η , 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 \mbox{join}:: \mbox{m (m a) -> m a} \\ \mbox{return}:: \mbox{a -> m a} \\ \mbox{join} \cong \mu \\ \mbox{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')
```