Koprvoky

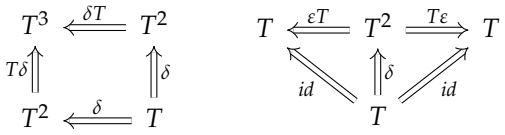
22. dubna 2022

Komonáda v kategorii $\mathcal{C}$ je monáda v $\mathcal{C}^{\mathrm{op}}$. Explicitně: Srovnání s monádami

Definice 1. *Komonáda v kategorii $\mathcal{C}$ je trojice $W = (T, \varepsilon, \delta)$, kde*

- $T\colon \mathcal{C} \to \mathcal{C}$, *(endofunktor)*
- $\varepsilon\colon T \Rightarrow \mathrm{Id}$, *(kojednotka, ko`return`, `extract`)* $\eta\colon \mathrm{Id} \Rightarrow T$
- $\delta\colon T \Rightarrow T^2$. *(konásobení, ko`join`, `duplicate`)* $\mu\colon T^2 \Rightarrow T$

takové, že následující diagramy komutují.



Haskell

```
class Functor w => Comonad w where
    extract   :: w a → a                    return :: a → m a
    extend    :: (w a → b) → w a → w b      bind :: (a → m b) → m a → m b
    duplicate :: w a → w (w a)              join :: m (m a) → m a
```

Jako u monád, z `extend` a `duplicate` stačí jedno:

```
  extend f  = fmap f · duplicate            bind = join · fmap f
  duplicate = extend id                     join = bind id
```

Skládání ko-Kleisliho šipek:

```
  (=<=) :: (w b → c) → (w a → b) → w a → c  (<=<)::(b → mc)→(a → mb)→ a → mc
  f =<= g = f · extend g                    f <=< g = bind f · g
```

Pravidla pro komonády v Haskellu:

```
     extend extract   ≡   id
  extract · extend f  ≡   f
extend f · extend g   ≡   extend (f · extend g)
```

Z adjunkcí

Pokud $F\colon \mathcal{C} \to \mathcal{D}$ a $G\colon \mathcal{D} \to \mathcal{C}$ jsou adjungované funktory ($F \dashv G$):

- $\eta\colon \mathrm{Id}_{\mathcal{C}} \Rightarrow GF$, (jednotka)
- $\varepsilon\colon FG \Rightarrow \mathrm{Id}_{\mathcal{D}}$, (kojednotka)

Tvrzení 1. *Každá adjunkce $(F, G, \eta, \varepsilon)$ dává zrod komonádě FG v $\mathcal{D}$.* GF je monáda v $\mathcal{C}$

$\mu\colon GFGF \Rightarrow GF$

$\mu = G\varepsilon F$

$$\delta\colon FG \Rightarrow FGFG \qquad \delta = F\eta G$$

Tvrzení 2. *Každá komonáda se rodí z adjunkce.*

(Podobně jako u monád – přes ko-Kleisliho kategorii, v níž identita odpovídá `extract` a skládání šipek realisuje operátor `=<=`.)

Příklady

Dvojice Funktor `(r,−)`. Šipky mají podobu `(r, a) → b`.
Kočtenář (též `Env`), chová se jako čtenář.

Monáda čtenáře:
funktor `(r → −)`
šipky `a → r → b`

Zaměřené grafy `extract` vrací vrchol; `duplicate` vyrábí graf grafů se stejným tvarem jako původní graf, v každém vrcholu je původní graf s jiným zaměřeným vrcholem.

Kořenové stromy `extract` vrací kořen; `duplicate` vyrábí strom stromů se stejným tvarem jako původní strom, v každém uzlu je příslušný podstrom původního stromu.

Strom je pěkná monáda, má-li hodnoty jen v listech.

Nekonečné seznamy `data Stream a = Cons a (Stream a)`.
Výsledek `duplicate xs` má na i-té posici nekonečný seznam odpovídající `drop i xs`.

Zippery Datová struktura representující stav procházení jiné datové struktury: zaměřený prvek + zatím neprošlá část + již prošlá část. Např. `data ListZipper a = Zip [a] a [a]`.

Funkce Funktor `(m → −)`. Šipky mají podobu `(m → a) → b`.
`m` je monoid. Kopísař (též `Traced`), chová se jinak než písař.

Monáda písaře:
funktor `(l,−)`, `l` je monoid
šipky `a → (l, b)`

Zaměřený slovník `data Store s a = Store (s → a) s`, kostav.
Např. celulární automat à la *Game of Life*: Typem indexu (`s`) jsou souřadnice. Výpočet typu `Store Coords Bool → Bool` implementuje pravidlo pro jednu buňku, `extend` ho aplikuje po celé ploše.

Adjunkce `(s,−)` ⊣ `(s → −)` plodí
monádu `State s` ≃ `s → (s,−)`
komonádu `Store s` ≃ `(s, s → −)`

Mimo **Hask**

Definice 2 (funkce uzávěru)**.** *Funkce $f\colon S \to S$ na částečně uspořádané množině $(S, \leq)$ je* funkcí uzávěru, *pokud pro všechna $x, y \in S$ splňuje:*

$$\begin{aligned} x &\leq f(x) \\ f(f(x)) &= f(x) \\ x \leq y &\Rightarrow f(x) \leq f(x) \end{aligned}$$

Tvrzení 3. *Funkce uzávěru jsou právě monády v kategorii odpovídající příslušnému posetu. Duální pojem (funkce vnitřku) jsou tamní komonády.*

Modální logika Modální operátory (možnost a nutnost, „někdy platí“/„vždy platí“)

$$\begin{aligned} A &\to \Diamond A & \Box A &\to A \\ \Diamond\Diamond A &\to \Diamond A & \Box A &\to \Box\Box A \end{aligned}$$

Úlohy na procvičení

1. Ukažte, že pravidla pro haskellové komonády odpovídají pravidlům pro skládání šipek v ko-Kleisliho kategoriích.

2. Navrhněte dvě různé platné instance `Comonad` pro stromy s libovolnou aritou: `data Tree a = Node a [Tree a]`.

3. Uvažme tuto implementaci zaveditelnou pro každý funktor:

   `duplicate wx = fmap (λx → wx) wx`.

   Rozhodněte, zda může vést při vhodné implementaci `extract` taková implementace k platné komonádě

   (a) u funktoru `Stream`,

   (b) u některého jiného funktoru,

   (c) obecně u všech funktorů.