Programare declarativă

Monade

Ioana Leuștean
Traian Florin Șerbănuță

Departamentul de Informatică, FMI, UB
ioana@fmi.unibuc.ro
traian.serbanuta@unibuc.ro

12 ianuarie 2021

Clasa de tipuri **Monad**

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

    (>>) :: m a -> m b -> m b
    ma >> mb = ma >>= \_ -> mb
```

- m a este tipul comenzilor care produc rezultate de tip a (și au efecte laterale)
- a -> m b este tipul continuărilor / a funcțiilor cu efecte laterale
- >>= este operația de „secvențiere” a comenzilor

În Haskell, monada este o clasă de tipuri!

Ce este o monadă?

Există multe răspunsuri, variind între

- O monadă este o clasă de tipuri în Haskell.

Ce este o monadă?

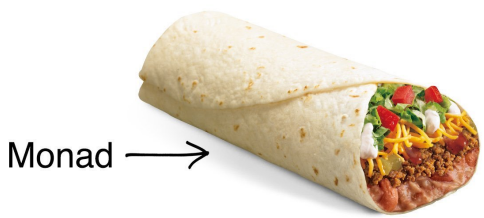
Există multe răspunsuri, variind între

- O monadă este o clasă de tipuri în Haskell.
- "All told, a monad in X is just a monoid in the category of endofunctors in X, with product x replaced by composition of endofunctors and unit set by the identity endofunctor."
  Saunders Mac Lane, Categories for the Working Mathematician, 1998.

Ce este o monadă?

Există multe răspunsuri, variind între

- O monadă este o clasă de tipuri în Haskell.
- "All told, a monad in X is just a monoid in the category of endofunctors in X, with product x replaced by composition of endofunctors and unit set by the identity endofunctor."
  Saunders Mac Lane, Categories for the Working Mathematician, 1998.
- O monadă este un burrito. https://byorgey.wordpress.com/2009/01/12/abstraction-intuition-and-the-monad-tutorial-fallacy/



https://twitter.com/monadburritos

https://chrisdone.com/posts/monads-are-burritos/

Funcții "îmbogățite" și programarea cu efecte

Funcții îmbogățite și efecte

- Funcție simplă: $x \mapsto y$

  știind x, obținem direct y

Funcții îmbogățite și efecte

- Funcție simplă: $x \mapsto y$

  știind x, obținem direct y

- Funcție îmbogățită: $x \mapsto$ 

  știind x, putem să extragem y și producem un efect

Funcții îmbogățite și efecte

- Funcție simplă: $x \mapsto y$

  știind x, obținem direct y

- Funcție îmbogățită: $x \mapsto$ 

  știind x, putem să extragem y și producem un efect

Referințe:

https://bartoszmilewski.com/2016/11/21/monads-programmers-definition/

https://bartoszmilewski.com/2016/11/30/monads-and-effects/

Compunerea funcțiilor cu efecte

Într-o monadă putem să compunem funcțiile cu efecte

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

(>=>) :: Monad m => (x -> m y) -> (y -> m z) -> (x -> m z)
kxy >=> kyz = \x -> kxy x >>= kbc
```

Exemple de functii cu efecte laterale

Parțialitate	Monada **Maybe**	a –> **Maybe** b
Excepții	Monada **Either** err	a –> **Either** err b
Nedeterminism	Monada [] (listă)	a –> [b]
Logging	Monada Writer mon	a –> (b, mon)
Stare	Monada State s	a –> (s –> (b, s))
Memorie read-only	Monada Reader mem	a –> (mem –> b)
I/O	Monada **IO**	a –> (world –> (b, world))

Notația **do** pentru monade

Notația cu operatori	Notația **do**
e >>= \x -> rest	x <- e rest
e >>= _ -> rest	e rest
e >> rest	e rest

Notația **do** pentru monade

Notația cu operatori	Notația **do**
e >>= \x -> rest	x <- e rest
e >>= _ -> rest	e rest
e >> rest	e rest

De exemplu

```
e1    >>=  \x1  ->
e2    >>  e3
```

devine

Notația **do** pentru monade

Notația cu operatori	Notația **do**
e >>= \x -> rest	x <- e rest
e >>= _ -> rest	e rest
e >> rest	e rest

De exemplu

```
e1    >>= \x1 ->
e2    >> e3
```

devine

```
do
    x1 <- e1
    e2
    e3
```

Legile clasei **Monad**

Functor și Applicative pot fi definiți cu **return** și >>=

```
instance Monad M where
  return a = ...
  ma >>= k = ...

instance Applicative M where
  pure = return
  mf <*> ma = do
      f <- mf
      a <- ma
      return (f a)

instance Functor M where
  fmap f ma = do
      a <- ma
      return (f a)
```

Compunerea funcțiilor cu efecte este monoid

Compunerea funcțiilor cu efecte

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

(>=>) :: Monad m => (x -> m y) -> (y -> m z) -> (x -> m z)
kxy >=> kyz = \x -> kxy x >>= kbc
```

Compunerea este asociativă cu element neutru return

- Pentru orice `k :: a -> m b`

  ```
  k >=> return == k
  return >=> k == k
  ```

- Pentru orice `k1 :: a -> m b, k2 :: b -> m c, k3 :: c -> m d`

  ```
  (k1 >=> k2) >=> k3 == k1 >=> (k2 >=> k3)
  ```