

# Programare funcțională

Introducere în programarea funcțională folosind Haskell  
C10

---

Ana Iova

Denisa Diaconescu

Departamentul de Informatică, FMI, UB

## **Monade - privire de ansamblu**

---

# Despre intenție și acțiune

[1] S. Peyton-Jones, *Tackling the Awkward Squad*: ...

- [1] A purely functional program implements a function; it has no side effect.
- [1] Yet the ultimate purpose of running a program is invariably to cause some side effect: a changed file, some new pixels on the screen, a message sent, ...

## Exemplu

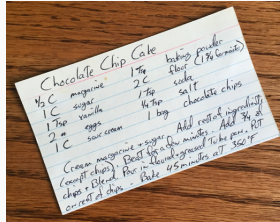
```
putChar :: Char -> IO ()  
Prelude> putChar '!'
```

reprezintă o comandă care, **dacă va fi executată**, va afișa un semn de exclamare.

# Mind-Body Problem - Rețetă vs Prăjitură



**c :: Cake**



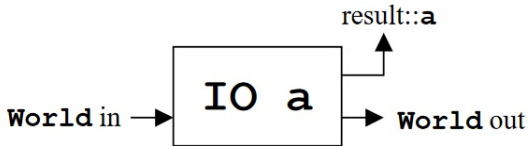
**r :: Recipe Cake**

**IO** este o rețetă care produce o valoare de tip **a**.

Motorul care citește și execută instrucțiunile IO se numește **Haskell Runtime System** (RTS). Acest sistem reprezintă legătura dintre programul scris și mediul în care va fi executat, împreună cu toate efectele și particularitățile acestuia.

# Comenzi în Haskell

```
type IO a = RealWorld -> (a, RealWorld)
```



S. Peyton-Jones, Tackling the Awkward Squad: ...

# 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  $\times$  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>

# 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/>

# Funcții îmbogățite și efecte

Funcție îmbogățită:  $x \mapsto$



Exemplu

Folosind tipul **Maybe** a

```
data Maybe a = Nothing | Just a
```

```
f :: Int -> Maybe Int
```

```
f x = if x < 0 then Nothing else (Just x+1)
```

Cum putem calcula  $f.f$ ?



## Funcții îmbogățite și efecte

Funcție îmbogățită:  $x \mapsto$



Exemplu

Folosind un tip care are ca efect un mesaj

```
newtype Writer log a = Writer {runWriter :: (a, log)}
```

```
f :: Int -> Writer String Int
```

```
f x = if x < 0 then (Writer (-x, "negativ"))  
      else (Writer (x, "pozitiv"))
```

Cum putem calcula  $f.f$  și să concatenăm mesajele?

# Clasa de tipuri Monad

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

În Haskell, monada este o clasă de tipuri!

Clasa **Monad** este o extensie a clasei **Applicative**!

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

## Asociativitate și element neutru

Operația  $>>=$  este asociativă și are element neutru **return**

- Element neutru (la dreapta):

$$(\mathbf{return} \ x) \ >>= \ g = g \ x$$

- Element neutru (la stânga):

$$x \ >>= \ \mathbf{return} = x$$

- Asociativitate:

$$(f \ >>= \ g) \ >>= \ h = f \ >>= \ (\backslash \ x \ \rightarrow \ (g \ x \ >>= \ h))$$

## Notăția `do` pentru monade

$(>>=) \quad :: m\ a \rightarrow (a \rightarrow m\ b) \rightarrow m\ b$

$(>>) \quad :: m\ a \rightarrow m\ b \rightarrow m\ b$

Notăția cu operatori	Notăția <b>do</b>
$e >>= \backslash x \rightarrow \text{rest}$	$x \leftarrow e$ $\text{rest}$
$e >>= \backslash \_ \rightarrow \text{rest}$	$e$ $\text{rest}$
$e >> \text{rest}$	$e$ $\text{rest}$

`binding' :: IO ()`

`binding' =`

`getLine >>= putStrLn`

`binding :: IO ()`

`binding = do`

`name <- getLine`

`putStrLn name`

## Notăția do pentru monade

$(>>=) \quad :: m\ a \rightarrow (a \rightarrow m\ b) \rightarrow m\ b$

$(>>) \quad :: m\ a \rightarrow m\ b \rightarrow m\ b$

Notăția cu operatori	Notăția <b>do</b>
$e >>= \backslash x \rightarrow \text{rest}$	$x \leftarrow e$ $\text{rest}$
$e >>= \backslash \_ \rightarrow \text{rest}$	$e$ $\text{rest}$
$e >> \text{rest}$	$e$ $\text{rest}$

De exemplu

$e1 >>= \backslash x1 \rightarrow e2 >> e3$

devine

**do**

$x1 \leftarrow e1$

$e2$

$e3$

## Notăția do pentru monade

$(>>=) \quad :: m\ a \rightarrow (a \rightarrow m\ b) \rightarrow m\ b$

$(>>) \quad :: m\ a \rightarrow m\ b \rightarrow m\ b$

De exemplu

$e1\ >>= \ \backslash x1 \rightarrow e2\ >>= \ \backslash x2 \rightarrow e3\ >>= \ \backslash\_ \rightarrow e4\ >>= \ \backslash x4 \rightarrow e5$

devine

**do**

$x1 \leftarrow e1$

$x2 \leftarrow e2$

$e3$

$x4 \leftarrow e4$

$e5$

## Functor și Applicative 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)
```

```
  -- mf >=> (\f -> ma >=> (\a -> return (f a)))
```

```
instance Functor M where
```

```
  fmap f ma = pure f <*> ma
```

```
  -- ma >=> \a -> return (f a)
```

```
  -- ma >=> (return . f)
```

## Exemple de efecte laterale

I/O	Monada <b>IO</b>
Parțialitate	Monada <b>Maybe</b>
Excepții	Monada <b>Either</b>
Nedeterminism	Monada [] (listă)
Logging	Monada Writer
Stare	Monada State
Memorie read-only	Monada Reader



**Pe săptămâna viitoare!**