Programare funcțională

Introducere în programarea funcțională folosind Haskell
C11- Seriile 23 si 25

Ana Iova
Denisa Diaconescu

Departamentul de Informatică, FMI, UB

Recap

```
class Functor f where
  fmap :: (a -> b) -> f a -> f b

class Functor m => Applicative m where
  pure :: a -> m a
  (<*>) :: m (a -> b) -> m a -> m b

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

Functor și Applicative definiți cu return și >>=

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

Notația do pentru monade

```
(>>=)  :: m a -> (a -> m b) -> m b
(>>)   :: m a -> m b -> m b
```

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

```
binding' :: IO ()
binding' =
    getLine >>= putStrLn

binding :: IO ()
binding = do
    name <- getLine
    putStrLn name
```

Monada Writer (variantă simplificată)

```haskell
newtype Writer log a = Writer {runWriter :: (a, log)}
-- a este parametru de tip

tell :: log -> Writer log ()
tell msg = Writer ((), msg)

instance Monad (Writer String) where
  return va =  Writer (va, "")
  ma >>= k =   let (va, log1) = runWriter ma
                   (vb, log2) = runWriter (k va)
               in Writer (vb, log1 ++ log2)
```

Monada Writer (varianta generalizată)

```haskell
class Semigroup a where
  (<>) :: a -> a -> a

class Semigroup a => Monoid a where
  mempty :: a
  mappend :: a -> a -> a
  mappend = (<>)

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

instance Monoid log => Monad (Writer log) where
  return a = Writer (a, mempty)
  ma >>= k = let (va, log1) = runWriter ma
                 (vb, log2) = runWriter (k va)
             in Writer (vb, log1 `mappend` log2)
```

Monada Reader(stare nemodificabilă)

```haskell
newtype Reader env a = Reader {runReader :: env -> a}

ask :: Reader env env
ask = Reader id

instance Monad (Reader env) where
    return x = Reader (\_ -> x)
    -- return x = Reader (const x)
    ma >>= k = Reader f
        where
          f env = let va = runReader ma env
                  in  runReader (k va) env
```

Monada Reader- exemplu: mediu de evaluare

```
newtype Reader env a = Reader {runReader :: env -> a}
data Prop = Var String | Prop :&: Prop
type Env = [(String, Bool)]

var :: String -> Reader Env Bool
var x = do
      env <- ask
      return $ fromMaybe False (lookup x env)

eval :: Prop -> Reader Env Bool
eval (Var x) = var x
eval (p1 :&: p2) = do
    b1 <- eval p1
    b2 <- eval p2
    return (b1 && b2)

runEval prop env = runReader (eval prop) env
```

Monada State

```
newtype State state a = State{runState :: state->(a,
    state)}

instance Monad (State state) where
  return va = State (\s -> (va, s))
  -- return a = State f where f s = (a,s)

  ma  >>= k = State $ \s -> let
                (va, news) = runState ma s
                State h = k va
        in (h news)

  -- ma :: State state a
  -- runState ma :: state -> (a, state)
  -- k :: a -> State state b
  -- h :: state -> (b, state)
  -- ma >>= k :: State state b
```

Monada State

```
newtype State state a = State{runState :: state->(a,
    state)}

instance Monad (State state) where
    return va = State (\s -> (va, s))
    ma  >>= k = State $ \s -> let
                  (va, news) = runState ma s
                  State h = k va
                in (h news)
```

Funcții ajutătoare:

```
get :: State state state
get = State (\s -> (s, s))

modify :: (state -> state) -> State state ()
modify f = State (\s -> ((), f s))
```

Monada State - exemplu "random"

```
newtype State state a = State{runState :: state->(a,
    state)}

cMULTIPLIER, cINCREMENT :: Word32 -- 32-bit unsigned
    integer type
cMULTIPLIER = 1664525 ; cINCREMENT = 1013904223

rnd, rnd2 :: State Word32 Word32
rnd = do
    modify (\seed -> cMULTIPLIER * seed + cINCREMENT)
    get
rnd2 = do
    r1 <- rnd
    r2 <- rnd
    return (r1 + r2)

-- runState rnd2 0  = (2210339985,1196435762)
```

Pe data viitoare!