Material adițional: Interpretor monadic parametrizat

```
module Interpreter where
type Name = String
data Term
    = Var Name | Con Integer | Term :+: Term
    | Lam Name Term | App Term Term | Out Term
    deriving (Show)

data Value m
    = Num Integer
    | Fun (Value m -> m (Value m))
    | Wrong
instance Show (Value m) where
    show (Num x) = show x
    show (Fun _) = "<function>"
    show Wrong   = "<wrong>"

type Environment m = [(Name, Value m)]

class (Monad m) => Interpreter m where
    get :: Name -> Environment m -> m (Value m)
    get x env =
        case lookup x env of
           Just v  -> return v
           Nothing -> return Wrong

    add :: Value m -> Value m -> m (Value m)
    add (Num i) (Num j) = return (Num $ i + j)
    add _ _             = return Wrong

    apply :: Value m -> Value m -> m (Value m)
    apply (Fun k) v = k v
    apply _ _       = return Wrong

    tell :: String -> m ()

interp :: Interpreter m => Term -> Environment m -> m (Value m)
interp (Var x) env = get x env
interp (Con i) _ = return $ Num i
interp (t1 :+: t2) env = do
    v1 <- interp t1 env
    v2 <- interp t2 env
    add v1 v2
interp (Lam x e) env = return $ Fun $ \ v -> interp e ((x,v):env)
interp (App t1 t2) env = do
    f <- interp t1 env
    v <- interp t2 env
    apply f v
interp (Out t) env = do
    v <- interp t env
    tell (show v ++ "; ")
    return v
```

```
module Writer where
import           Interpreter

newtype Writer a = Writer { runWriter :: (a, String) }
instance Functor Writer where
    fmap f (Writer (a, s)) = Writer (f a, s)
instance Applicative Writer where
    pure a = Writer (a, "")
    Writer (f, sf) <*> Writer (a, sa) = Writer (f a, sf ++ sa)
instance Monad Writer where
    return = pure
    Writer (a, sa) >>= k =
        let (b, sb) = runWriter (k a)
        in Writer (b, sa ++ sb)
instance Show a => Show (Writer a) where
    show (Writer (a, w)) = "Output: " ++ w ++ "Value: " ++ show a
instance Interpreter Writer where
    tell w  = Writer ((),w)

test :: Term -> String
test t = show $ interp t ([]::Environment Writer)
```

Examen Programare Declarativă, seria 33

2 februarie 2018

Citiți cu atenție programul Haskell de pe verso. Programul este foarte asemănător cu interpretorul monadic folosind monada **Writer** făcut la curs, cu câteva diferențe:

1. In loc de a fi un folosi un tip rigid, modificat de fiecare dată când schimbăm monada, parametrizăm toate tipurile implicate (**Value** și **Environment**) cu un tip **m** care este gândit ca tipul monadei în care va rula interpretorul.
2. Grupăm toate funcțiile dependente de interpretarea monadică (**add**, **get**, **apply**, **tell**) intr-o clasă **Interpreter**. **add**, **get** și **apply** au definiții implicite.
3. Monada **Writer** e definită intr-un modul separat, fiind făcută instanță a lui **Interpreter** (definind **tell**) pentru a putea fi folosită de **interp** și **Show** (pentru afișare).

Exercițiile care urmează se vor referi la elementele definite pe verso. Puteți cere lămuriri asupra oricărei definiții de acolo.

1 (0,5p). Scrieți rezultatul aplicării funcției **test** asupra programelor **pgm1** (0,1p), **pgm2** (0,2p), **pgm3** (0,3p).

```
pgm1 = Out (Con 1) :+: Out (Con 2)
pgm2 = App (Lam "x" (Out (Con 3))) (Out (Con 4))
pgm3 = App (Lam "x" (Out (Con 5)))
    ((Lam "x" (Out (Con 6))) :+: (Out (Con 7)))
```

Pentru restul exercițiilor, vrem să combinăm efectul de afișare cu acela de a întrerupe execuția la întâlnirea unei erori. Pentru aceasta avem nevoie de o monadă care combină monada **Writer** cu monada **Maybe**. Sunt posibile două astfel de combinații. Pentru astăzi vom alege posibilitatea care, în caz de eroare, ne permite să observăm output-ul generat până la apariția erorii.

2 (1p). Urmând exemplul modulului **Writer**, continuați modulul de mai jos pentru a obține un interpretor asemanator cu cel pentru **Writer**, dar care se oprește imediat după întâlnirea unei erori, fără a mai acumula output. Observație: pentru a obține acest efect, este necesară redefinirea funcțiilor implicite atunci când se crează instanța pentru clasa **Interpretor**.

```
module WriterMaybe where

import           Interpreter

newtype WM a = WM { runWM :: (Maybe a, String) }
```

3 (0,5p). Se consideră modificarea tipului de date **Term** prin adăugarea unei expresii **Halt**:

```
data Term = … | Halt String
```

semantica intuitivă a instrucțiunii este aceea că adaugă la output argumentul, apoi termină abrupt execuția, fără a mai evalua nimic.

a (0,25p) Modificați clasa **Interpreter** prin adăugarea unei funcții noi fără o definiție implicită și definiția funcției **interp** pentru a da semantică lui **Halt** folosind acea nouă funcție din **Interpreter**

b (0,25p) Definiți funcția de mai sus în instanța **Interpreter** a lui WM definită în modulul WriterMaybe.