

Łukasz Czapliński

## Zadanie z transformatorów monad

22 października 2013

### 1. Do kiedy, komu?

Zadanie należy wysłać do 21.11.2013 r. na [eternal.kadir@gmail.com](mailto:eternal.kadir@gmail.com). Tam też należy kierować wszelkie pytania, prośby, podania i inne, związane z tematyką wykładu lub nie, sugestie. Oceniający zastrzega sobie prawo do zignorowania wszelkich otrzymanych dokumentów, za wyjątkiem zadań do oceny.

### 2. Co?

#### 2.1. Wstęp teoretyczny

Niech język termów będzie opisany jako

```
data Term = And Term Term | Or Term Term
          | Var Variable | Not Term
          | Let Variable Term Term | Const Value
          | Appl Function Term deriving
```

Wówczas możemy interpretować dany term poleceniem

```
interp :: InterpMonad m => Term -> m Value
```

gdzie InterpMonad jest na przykład następującą klasą:

```
class (MonadState m, MonadEnv m) =>
  InterpMonad m where
  start :: (Show a, InterpMonad m) => m a -> String
```

której instancja może być pewnym złożeniem transformatorów monad. Wówczas każdy transformator będzie spełniał swoje zadanie, a jedynym problemem jest odpowiednie "liftowanie" operacji. Zostało to omówione na wykładzie,

— kod: <https://github.com/scoiatael/MonadTransformerExample>,

— prezentacja: <https://github.com/scoiatael/MonadTransformerPresentation>.

#### 2.2. Do rzeczy

Niech dany będzie język, w którym termy są opisane w następujący sposób:

```
data Term = And Term Term | Or Term Term
          | Var Variable | Not Term
          | Let Variable Term Term | Const Value
          | Twofold Variable Term deriving (Read, Show, Ord, Eq)
```

Natomiast interp będzie zwracał monadę będącą instancją klasy

```

class (MonadState m, MonadWriter m, MonadList m, MonadError m) =>
  InterpMonad m where
    start :: (Show a, InterpMonad m) => m a -> String

class Monad m => MonadWriter m where
  tell :: String -> m ()

```

Zadaniem jest napisać interpreter tego języka. Interpreter ten powinien spełniać następujące kryteria:

- instrukcje let, twofold powinny zostawiać ślad w logu (zrealizowanym przez tell MonadWriter), aby możliwe było prześledzenie historii obliczeń,
- twofold powinno być interpretowane w dany sposób:

```

interp Twofold v t -> do
  mv <- getVar v
  if mv /= Nothing
    then err $ "Variable " ++ v ++ " already taken at this point."
    else return ()
  let v1 = branch v True t
  let v2 = branch v False t
  merge v1 v2

```

branch v X t oblicza wartość t przy zmiennej x równej X; innymi słowy: twofold powinno pozwalać na sprawdzenie wyników pewnego wyrażenia dla wszystkich możliwych wartości zmiennej.

- and, or powinny być leniwe (można to sprawdzić używając twofold, w którym jedna z gałęzi obliczeń wywołuje wyjątek),
- semantyka pozostałych wyrażeń powinna być taka, jak w przykładzie z wykładu.

Można napisać program od zera lub użyć znajdujących się pod adresem:

<https://github.com/scoiatael/MonadTransformerExample> (w katalogu Task) plików przykładowych (znajduje się tam gotowa definicja funkcji interp, mogą pojawiać się tam dodatkowe wskazówki).

Dodatkową pomocą może być lektura [1].

### 2.3. Ocena

Ocenie podlega:

- poprawność interpretera,
- możliwość jego rozszerzenia,
- styl, w jakim został napisany kod.

Dodatkowe punkty można zyskać za

- dobrą argumentację wybranego złożenia transformatorów (jaki ma wpływ na zachowanie interpretera),
- przysłaniem oprócz gotowego interpretera, także jego wersji rozszerzonej o obsługę funkcji (podobnie jak w przykładzie),
- dobre przedstawienie możliwości swojego interpretera (przykładowe programy, problemy jakie można nim rozwiązać).

### Literatura

- [1] Sheng Liang Paul Hudak and Mark P. Jones. Monad transformers and modular interpreters. 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, CA, January 1995.