

Prof. Dr. Jürgen Giesl
Peter Schneider-Kamp
Stephan Swiderski

Exercises *Functional Programming* – Sheet 5

Solutions will be collected until Wednesday, May 16, 2007 in the exercise course.

Exercises can be solved both in English and in German.

Exercise 1 (4 + 4 + 4 points)

We define a monad `Count` as follows:

```
data Count a = Counter Int a

instance Monad Count where
  return          = Counter 0
  (Counter i x) >>= f = addTo i (f x)

addTo :: Int -> Count a -> Count a
addTo i (Counter j x) = Counter (i+j) x

instance Show a => Show (Count a) where
  show (Counter i x) = show x++" : Count "++show i
```

- (a) Show that the three monad-laws hold for the `Count` monad.
- (b) In the following Haskell code the `Count` monad is used in the evaluation of terms (i.e., of objects from the data structure `Term`).

```
data Term = Con Float | Plus Term Term

countFlag :: Count ()
countFlag = Counter 1 ()

eval :: Term -> Count Float
eval (Con x)      = return x
eval (Plus t u) = do x <- eval t
                    y <- eval u
                    countFlag
                    return (x+y)
```

The function `eval` evaluates a term and also counts the needed evaluation steps. For example:

```
show (eval (Plus (Plus (Con 0.5) (Con 0)) (Con 1))) evaluates to
"1.5 : Count 2 "
```

Extend the data structure `Term` by a data constructor `Mul :: Term -> Term -> Term` for representing a multiplication. Extend the function `eval` to handle terms containing `Mul`, too. Multiplications with 0 should not be counted. For example:

```
show (eval (Plus (Con 0.5) (Mul (Con 0) (Con 1)))) evaluates to
"0.5 : Count 1 "
```

```
show (eval (Plus (Con 0.5) (Mul (Con 2) (Con 1)))) evaluates to
"2.5 : Count 2 "
```

- (c) Replace the data structure `Count` in the algorithm `eval` by a new data structure that contains two different counters, one for `Mul` and one for `Plus`. Again, multiplications with 0 should not be counted.

For Example, the evaluation of

```
show (eval (Mul (Plus (Con 0.5) (Con 0.4)) (Plus (Con 0.6) (Con 0.4))))
should yield the string
"0.9 : Mul 1 : Plus 2"
```

Make your new data structure an instance of the classes `Monad` and `Show` and change the implementation of `eval` as little as possible.