8. Übung zur Vorlesung "Fortgeschrittene funktionale Programmierung"

Monaden

Boris Dudelsack

29. November 2016

Aufgabe 1: Monad-Instanzen

```haskell
class Monad m where
  return :: a → m a
  (>≥) :: m a → (a → m b) → m b
  (>>) :: m a → m b → m b
  m1 >> m2 = m1 >≥ \_ → m2

instance Monad Maybe where
  return = Just
  Nothing >≥ _ = Nothing
  Just x >≥ f = f x

data Identity a = Identity { runIdentity :: a }

instance Monad Identity where
  return = Identity
  Identity x >≥ f = f x

instance Monad (Either a) where
  return = Right
  Right x >≥ f = f x
  Left x >≥ _ = Left x
```

Aufgabe 2: Listen-Monade

```haskell
instance Monad [] where
  return x = [x]
  [] >≥ _ = []
  xs >≥ f = foldr (\y ys → f y ++ ys) [] xs

cross :: [a] → [b] → [(a,b)]
cross as bs = do
  x ← as
  y ← bs
  return (x,y)
```

Aufgabe 3: State-Monade

```haskell
data Tree a = Empty | Node (Tree a) a (Tree a) deriving (Show)

numberTree' :: Tree a → Int → (Tree Int, Int)
numberTree' Empty v = (Empty, v)
numberTree' (Node x _ y) v = (Node t1 v2 t2, v2+1)
  where
    (t1, v1) = numberTree' x v
    (t2, v2) = numberTree' y v1

numberTree :: Tree a → Tree Int
numberTree t = t1
  where
    (t1,_) = numberTree' t 1

data State s a = State { runState :: s → (a, s) }
```