Prof. Dr. Manfred Schmidt-Schauß
Künstliche Intelligenz/Softwaretechnologie

Fachbereich Informatik (20)
Johann Wolfgang Goethe-Universität Frankfurt am Main

Einführung in das funktionale Programmieren

Wintersemester 1995/96

Übungsblatt Nr. 10

Es bedurfte unendlicher Geduld, um sich in die Atmosphäre des Dienstagsmorgens zurückzuversetzen.
aus Simenon: Maigret und der Minister

Aufgabe 1

Wir wollen in dieser Aufgabe die Monaden noch etwas weiter anreichern. Wir definieren zwei weitere Monadenklassen:

```
class Monad m => Monad0 m where
 zero :: m a

class Monad0 c => MonadPlus c where
 (++) :: c a -> c a -> c a
```

Die Funktionen `zero` und `(++)` sollen für alle Monaden in ähnlicher Weise, wie für die Listenmonade definiert werden.

a) Schreiben Sie mit den monadischen Funktionen folgende Funktionen für Monaden, die auf der Listenmonade die bekannten Listenfunktionen ergeben sollen:

```
concat :: MonadPlus c =>  [c a] -> c a

filter :: Monad0 m => (a->Bool) -> m a -> m a

mfoldl :: Monad m => (a -> b -> m a) -> a -> [b] -> m a

mfoldl :: Monad m => (a -> b -> m b) -> b -> [a] -> m b

mapl   :: Monad m => (a -> m b) -> ([a] -> m [b])

mapr   :: Monad m => (a -> m b) -> ([a] -> m [b])
```

b) Definieren Sie für folgende Datentypen Instanzen von `Monad0` und `MonadPlus`.

```
data Tree a = TNil
             |TCons (Tree a) a (Tree a)

data Foo a = Eins a (Foo a) |List a |String
```

Aufgabe 2

Zeigen Sie, daß die monadischen Gesetze:

$$
\begin{aligned}
\texttt{(result a) `bind` k} &= \texttt{k a} \\
\texttt{m `bind` result} &= \texttt{m} \\
\texttt{m `bind` (\textbackslash a -> (k a) `bind` h)} &= \texttt{(m `bind` (\textbackslash a -> (k a)) `bind` h}
\end{aligned}
$$

Für die Listenmonade und die Nullmonade erfüllt sind:

```
instance Functor [] where mapp = map

instance Monad [] where
   result a = [a]
   [] `bind` k = []
   (x:xs) `bind` k = (k x) ++ (xs `bind` k)

type I a = a

instance Functor I where mapp f (I a) = (f a)

instance Monad I where
   result a = a
    m `bind` k = k m
```