Monad Transformers

Christoph Gonsior

10.12.2007

Der Inhalt

- Wiederholung - Monaden
  - IO Monad
  - State Monad
- Monad Transformers
  - Definition
  - Vorteile + Nutzen von Transformers
  - Die State Transformer Monad als Beispiel

Monaden

Monaden bestehen aus:

- einem Typkonstruktor('M', 'IO', ...)
- den Operationen
  - `return :: a -> M a`
  - `(>>=):: M a -> (a -> M b) -> M b`

Anmerkung:
"bind" ist " $>>=$ "
"return" wird auch "unit" genannt

Monaden

- helfen bei der Modellierung erwünschter Nebeneffekte
- sind eine Abstraktion, die gut strukturierte, modulare Programme ermöglicht
- können die Berechnungsreihenfolge festlegen(nur bei IO)
- ermöglichen die Integration imperativer Eigenschaften in eine rein funktionale Programmiersprache

IO Monad

- Verwendung zur Ein- und Ausgabe
- führt Ein- und Ausgaben aus, bevor ein Wert zurückgegeben wird

State Monad

- State Monad
  - ermöglicht es uns einer Funktion einen State zuzuordnen

```
newtype State st a = State (st -> (st, a))

instance Monad (State state) where
    return a = State (\state -> (state, a))
    State run >>= action = State run'
                           where run' st =
                                   let (st', a) = run st
                                       State run'' = action a
                                   in run'' st'
```

Operationen

```
getState :: State state state
getState = State (\state -> (state, state))
```

getState gibt den Zustand aus

```
putState :: state -> State state ()
putState new = State (\_ -> (new, ()))
```

putState ändert den Zustand

Motivation

Monad Transformers

Motivation

Monad Transformers

- helfen uns nun die Eigenschaften der einzelnen Monaden zu kombinieren, d.h. praktisch Verwendung mehrerer Monaden gleichzeitig
- z.B. Verknüpfung von State und beliebiger Monade

Monad Transformers - Definition

Monad Transformers

- ermöglichen die Kombination von Monaden
- ändern die Eigenschaften der verwendeten Monaden nicht
- ”umhüllen” eine innere Monade
- sind selbst Monaden

Formale Definition

```
class MonadTrans t where
lift :: Monad m  =>  m a  ->  t m a
```

Die lift-Operation ist die Funktion, die bei einem Monad Transformer den Zugriff auf Funktionen der inneren Monade ermöglicht.

Beispiel für einen Monad Transformer

State Monad Transformer

ist die Kombination einer State Monade mit einer beliebigen anderen Monade

```
newtype StateT state m a = StateT (state -> m (state, a))

instance MonadTrans (StateT state) where
lift m = StateT (\s -> do a <- m
                          return (s,a))

instance Monad m => Monad (StateT state m) where
return a = StateT (\s -> return (s,a))
StateT m >>= k = StateT (\s -> do (s', a) <- m s
                                  let StateT m' = k a
                                  m' s')
  fail s = StateT (\_ -> fail s)
```

Beispiel für einen Monad Transformer

State Monad Transformer

getT und putT entsprechen den Funktionen getState und putState aus der State Monad

Beispiele aus der Praxis

- Beim Durchlauf eines Graphen(u.U. zyklisch) wollen wir keine Knoten doppelt besuchen:
  Markieren von besuchten Knoten!
- Zählen von Funktionsaufrufen/Auswertungsschritten zur Programmanalyse

Fazit

Monad Transformers bringen

- Modularität
- Flexibilität
- die Möglichkeit selbst neue Monaden zusammenzustellen und sich ihrer Eigenschaften zu bedienen

Fragen?

Vielen Dank für Ihre Aufmerksamkeit!