

Übungen zu Funktionaler Programmierung

Übungsblatt 11

Ausgabe: 12.2.2017, **Abgabe:** 19.2.2017 – 16:00 Uhr, **Block:** 6

Lesen Sie bitte selbstständig die Abschnitte *Lesermonaden* und *Schreibermonaden* auf den Folien 205–208 und den Abschnitt *Zustandsmonaden* auf den Folien 213–215.

Aufgabe 11.1 (4 Punkte) *Lesermonade*

Schreiben Sie die Funktion `bexp2store` so um, dass sie Gebrauch von der Lesermonade macht. Verwenden Sie die `do`-Notation.

```
type BStore x = x -> Bool

bexp2store :: BExp x -> Store x -> BStore x -> Bool
bexp2store True_ _ _ = True
bexp2store False_ _ _ = False
bexp2store (BVar x) _ bst = bst x
bexp2store (Or bs) st bst = or $ map (\x -> bexp2store x st bst) bs
bexp2store (And bs) st bst = and $ map (\x -> bexp2store x st bst) bs
bexp2store (Not b) st bst = not $ bexp2store b st bst
bexp2store (e1 := e2) st _ = exp2store e1 st == exp2store e2 st
bexp2store (e1 <= e2) st _ = exp2store e1 st <= exp2store e2 st
```

Aufgabe 11.2 (4 Punkte) *Schreibermonade*

Gegeben sei folgende Vorlage.

```
type ID = Int
type Bank = [(ID,Account)]
data Account = Account { balance :: Int, owner :: Client } deriving Show
data Client = Client
  { name :: String
  , surname :: String
  , address :: String
  } deriving Show

own1, own2, own3 :: Client
own1 = Client "Max" "Mustermann" "Musterhausen"
own2 = Client "John" "Doe" "Somewhere"
own3 = Client "Erika" "Mustermann" "Musterhausen"

acc1, acc2, acc3 :: Account
acc1 = Account 100 own1
acc2 = Account 0 own2
acc3 = Account 50 own3

bank :: Bank
bank = [(1,acc1), (2,acc2), (3,acc3)]

credit :: Int -> ID -> Bank -> Bank
credit amount id ls
  = updRel ls id entry{ balance = oldBalance + amount} where
    Just entry = lookup id ls
    oldBalance = balance entry

debit :: Int -> ID -> Bank -> Bank
debit amount = credit (-amount)
```

Erweitern Sie die Vorlage um eine Transferfunktion mit Protokollierung (Log). Machen Sie hierfür Gebrauch von einer Schreibermonade mit `String` für die Protokollierung (`(,) String`).

- Eine Loggerfunktion `logMsg`, welche einen beliebigen `String` an das Protokoll anhängt. Geben Sie auch den Typ der Funktion an.
- `transferLog :: Int -> ID -> ID -> Bank -> (String,Bank)` – Überweist den angegebenen Betrag vom ersten Konto auf das zweite und schreibt einen Eintrag in das Protokoll. Der Eintrag soll folgendes Format besitzen.

"Der Betrag *<ammount>* wurde von Konto *<id1>* auf Konto *<id2>* übertragen."

Verwenden Sie die `do`-Notation. Der Tupelkonstruktor `(,)` darf *nicht* benutzt werden.

- c) `transactions :: Bank -> (String,Bank)` – Soll Transaktionen durchführen, welche zu folgenden Protokolleinträgen passen.
`putStrLn $ fst $ transactions bank ~>`

Der Betrag 50 wurde von Konto 1 auf Konto 2 übertragen.

Der Betrag 25 wurde von Konto 1 auf Konto 3 übertragen.

Der Betrag 25 wurde von Konto 2 auf Konto 3 übertragen.

Verwenden Sie die `do`-Notation. Der Tupelkonstruktor `(,)` darf *nicht* benutzt werden.

Aufgabe 11.3 (4 Punkte) Zustandsmonade

Benutzen Sie die Vorlage aus Aufgabe 11.2 und erweitern Sie diese um die folgenden Funktionen.

- a) `putAccount :: ID -> Account -> State Bank ()` – Erstellt ein neues Konto mit angegebener ID. Ist bereits ein Konto mit der ID vorhanden, wird dieses überschrieben.
- b) `getAccount :: ID -> State Bank (Maybe Account)` – Falls vorhanden, wird das Konto mit der ID ausgegeben.
- c) `creditS :: Int -> ID -> State Bank ()` – Addiert den angegebenen Betrag auf das angegebene Konto. Verwenden Sie die `do`-Notation. Konstruktor und Destruktor von `State` (`State`, `runS`) dürfen *nicht* verwendet werden.
- d) `debitS :: Int -> ID -> State Bank ()` – Subtrahiert den angegebenen Betrag von dem angegebenen Konto. Verwenden Sie die `do`-Notation. Konstruktor und Destruktor von `State` dürfen *nicht* verwendet werden.
- e) `transferS :: Int -> ID -> ID -> State Bank ()` – Überweist den angegebenen Betrag vom ersten Konto auf das zweite. Verwenden Sie die `do`-Notation. Konstruktor und Destruktor von `State` dürfen *nicht* verwendet werden.

Beispiel: Mit `transferS 25 1 3` werden 25 Geldeinheiten von Konto 1 auf Konto 3 übertragen.

```
map (fmap balance) $ snd $ runS (transferS 25 1 3) bank  
~>[(1,75),(2,0),(3,75)]
```