technische universität dortmund

FLDIT

Lehrstuhl 1
Logik in der Informatik

Prof. Dr. Peter Padawitz

Jos Kusiek (*jos.kusiek@tu-dortmund.de*) Wintersemester 2016/2017

Übungen zu Funktionaler Programmierung
Übungsblatt 9

Ausgabe: 9.12.2016, **Abgabe:** 16.12.2016 - 12:00 Uhr

Aufgabe 9.1 (3 Punkte) Schreiben Sie Traversierungsfunktionen für `Bintree`.

1. Die Funktion `preorderB` soll die Knoten des Baumes als Liste in Hauptreihenfolge (pre-order) ausgeben.
2. Die Funktion `postorderB` soll die Knoten des Baumes als Liste in Nebenreihenfolge (post-order) ausgeben.

Lösungsvorschlag

```
preorderB :: Bintree a -> [a]
preorderB (Fork a l r) = a : preorderB l ++ preorderB r
preorderB Empty                = []

postorderB :: Bintree a -> [a]
postorderB (Fork a l r) = postorderB l ++ postorderB r ++ [a]
postorderB Empty        = []
```

Aufgabe 9.2 (3 Punkte) Bestimmen Sie die Typen der folgenden Ausdrücke. Die Aufgabe soll mithilfe von Typinferenzregeln gelöst werden.

1. `zipWith (+)`
2. `\x -> \f -> f x`

Lösungsvorschlag

1.
$$\frac{\texttt{zipWith :: (a->a->a) -> [a] -> [a] -> [(a,a)]}, \texttt{(+) :: Num a => a->a->a}}{\texttt{zipWith (+) :: Num a => [a] -> [a] -> [(a,a)]}}$$

2.
$$\frac{\texttt{x :: a,} \quad \dfrac{\texttt{f :: a -> b,} \quad \dfrac{\texttt{f :: a -> b,} \quad \texttt{\textbackslash x :: a}}{\texttt{f x :: b}}}{\texttt{\textbackslash f -> f x :: (a -> b) -> b}}}{\texttt{\textbackslash x -> \textbackslash f -> f x :: a -> (a -> b) -> b}}$$

Aufgabe 9.3 (6 Punkte)

1. Modellieren folgende Eigenschaften mit Datentypen.
   - Eine Bank führt eine Liste von Konten.

- Ein Konto hat einen Kontostand und einen Kunden als Besitzer.
- Für einen Kunden werden die Daten Vorname, Name und Adresse (`String`) gespeichert.

2. Legen Sie eine Beispielbank an mit mindestens zwei Konten.
3. Definieren Sie folgende Funktionen. Benutzen Sie für `ID` den Typ `Int` (`type ID = Int`).

   `credit :: Int -> ID -> Bank -> Bank` Addiert den angegebenen Betrag auf das angegebene Konto. Hinweis: Schauen Sie sich die Funktion `updList` an.

   `debit :: Int -> ID -> Bank -> Bank` Subtrahiert den angegebenen Betrag von dem angegebenen Konto.

   `transfer :: Int -> ID -> ID -> Bank -> Bank` Überweist den angegebenen Betrag vom ersten Konto auf das zweite.

Lösungsvorschlag

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

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

acc1, acc2 :: Account
acc1 = Account 100 own1
acc2 = Account 0 own2

bank :: Bank
bank = Bank [acc1, acc2]

credit :: Int -> ID -> Bank -> Bank
credit amount id (Bank ls)
  = Bank (updList ls id entry{ balance = oldBalance + amount})
  where
    entry = ls !! id
    oldBalance = balance entry

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

transfer :: Int -> ID -> ID -> Bank -> Bank
transfer amount id1 id2 = debit amount id1 . credit amount id2
```