Klausur: Programmierung und Modellierung Bry 2016

July 29, 2016

Gedankenprotokoll

Aufgabe 1 - Multiple Choice

Geben sie den allgemeinsten Typen für die folgenden Ausdrücke an:

a) `f x y = (3 * x :: Int, y)`

- ☐ `Int -> Int -> (Int, Int)`
- ☐ `Int -> a -> (Int, a)`
- ☐ `(Int, Int) -> (Int, Int)`
- ☐ Kein korrekter Haskell Ausdruck
- ☐ Nichts davon, sondern .....

b) `(\x y -> (x + y) `mod` y)`

- ☐ `Int -> Int -> Int`
- ☐ `Integer -> Integer -> Integer`
- ☐ `Integral a => a -> a -> a`
- ☐ Kein korrekter Haskell Ausdruck
- ☐ Nichts davon, sondern .....

c) `(\x -> (\y -> x ++ y))`

- ☐ `String`
- ☐ `Char -> String`
- ☐ `[a] -> [a] -> [a]`
- ☐ Kein korrekter Haskell Ausdruck
- ☐ Nichts davon, sondern .....

d) `(\x -> [x])`

- ☐ `Char -> String`
- ☐ `a -> [a]`
- ☐ `a -> a`
- ☐ Kein korrekter Haskell Ausdruck
- ☐ Nichts davon, sondern .....

Aufgabe 2 - Rekursion

Definieren sie für $n \in \mathbb{N} \setminus \{0\}$:

$\prod_{i=1}^{i=1} i = 1$ für $n = 1$

$\prod_{i=1}^{i=n} i = n * \prod_{i=1}^{i=n-1} i$ für $n \geq 1$

a) Eine rekursive Funktion wie oben beschrieben mit der folgenden Typsignatur. Sie soll für $n \leq 0$ nicht terminieren.

```
produkt :: Integral a => a -> a
```

b) Erweitern sie die Funktion aus *a)* sodass sie für Eingaben $n \leq 0$ eine 1 zurückgibt.

c) Schreiben sie eine Funktion die sich wie die Funktion aus *b)* verhält, bloß endrekursiv.

Aufgabe 3 - Auswertungsreihenfolgen

Umgebung soll nicht mit angegeben werden. Gegeben:

```
tail [1,2,3] = [2,3]
f = tail . tail
g = (\x -> 42)
h = (\x -> x * x)
```

a) Werten sie `g (h 3)` in applikativer Reihenfolge aus

b) Werten sie `g (h 3)` in normaler Reihenfolge aus

c) Werten sie `h (h 3)` in verzögerter Reihenfolge aus

d) Werten sie `f [1,2,3]` in verzögerter Reihenfolge aus

Aufgabe 4 - Binärbäume

Gegeben:

```
data BB a = L | K (BB a) a (BB a)
```

a) Geben sie einen ausgeglichenen Binärbaum vom Typ

```
BB Integer
```

für die Werte $0 - 6$ (eingeschlossen) an.

b) Füllen sie die Lücken aus. Gefragt ist eine rekursive Suchfunktion für Elemente in einem Baum

```
suche :: Eq a => a -> BB a -> Bool
suche .....  ......... = False
suche .....  ......... = ..........................
                         ..........................
```

Aufgabe 5 - Faltung

Gegeben:

```
data BB a = L | K (BB a) a (BB a)

tief :: b -> (b -> a -> a -> a) -> BB a -> b
tief fL fK L = fL
tief fL fK (K linkerBaum w rechterBaum) =
        fk (tief fL fK linkerBaum) w (tief fL fK rechterBaum)
```

Vervollständigen sie die Definitionen:

a) Eine Funktion die die Anzahl der Knoten in einem Binärbaum zurückgibt

```
anzahlKnoten :: BB a -> Int
anzahlKnoten baum =
    tief 0 (\left w right -> ........................... ) baum
```

b) Eine Funktion die die Tiefe eines Binärbaum zurückgibt

```
baumTiefe :: BB a -> Int
baumTiefe baum =
    tief 0 (\left w right -> ........................... ) baum
```

c) Eine Funktion die überprüft ob ein Element in einem Baum vorhanden ist. Ergänzen sie die Typsignatur.

```
istIn :: Eq a => a -> BB a -> ........................
istIn wert baum =
    tief False (\left w right -> ........................... ) baum
```

Aufgabe 6 - Datentypen

Gegeben (alle Namen sind `Strings`):

- Im Land L_1 besteht ein Personennamen aus
  - einem oder mehreren Vornamen und
  - einem Nachnamen
- Im Land L_2 besteht ein Personennamen aus
  - einem Vornamen und
  - einem Mittelnamen und
  - einem Nachnamen

a) Definieren sie einen rekursiven Datentyp `Vornamen` für Menschen aus L_1 mit:

- `EV` *(ein Vorname)*
- `MV` *(mehrere Vornamen)*

b) Stellen sie die Vornamen von einem Menschen "v1", "v2" und "v3" mithilfe von diesem Datentyp dar.

c) Definieren sie eine rekursive Funktion mit folgender Typsignatur die alle Vornamen zusammensetzt und zurückgibt.

```
vDruck :: Vorname -> String
```

d) Definieren sie einen Datentyp `Name1` mit Konstruktor `N1` für Personen aus L_1.

e) Definieren sie eine Funktion mit folgender Typsignatur die einen Personennamen `Name1` als `String` in der Form `Vorname(n) Nachname` zurückgibt.

```
druckt1 :: Name1 -> String
```

f) Definieren sie einen Datentyp `Name2` mit Konstruktor `N2` für Personen aus L_2.

g) Definieren sie eine Funktion mit folgender Typsignatur die einen Personennamen `Name2` als String in der Form `Vorname Mittelname Nachname` zurückgibt.

```
druckt2 :: Name2 -> String
```

h) Vervollständigen sie folgende Instanzen der Typklasse `Name`. Die Funktionen sollen analog zu `druckt1` und `druckt2` für die jeweiligen Datentypen funktionieren.

```
class Name a where

    druckt :: a -> String

instance Name Name1 where

    ....................................
    ....................................
    ....................................

instance Name Name2 where

    ....................................
    ....................................
    ....................................
```

Aufgabe 7

a) Definieren sie den Datentyp `Paar a` mit Konstruktor `P` und Typvariable `a` das ein Tupel aus Haskell simuliert. Sie dürfen den eingebauten Datentyp in Haskell jedoch **nicht** benutzen.

b) Ergänzen sie folgende Definition von der Addition von den Datentyp `Paar a`. Bei der Addition in Tupeln werden die Elemente nach dem Index addiert (`(x1,y1) + (x2,y2) = (x1+x2, y1+y2)`)

```
plus :: ............... => Paar a -> Paar a -> Paar a
plus ................ ................ = P (x1 + x2) (y1 + y2)
```

c) Ergänzen sie die Monoidinstanz für `Pair a` sodass `a` eine Instanz in der Typklasse `Integral` besitzt und eine Monoid bezöglich der Addition ist.

```
instance ............ => Monoid (Paar a) where

    mempty = ............................
    mappend p1 p2 = ....................
```

Aufgabe 8

a) Vervollständigen sie die folgende Funktion `minus` die die Differenz von $n_1, n_2 \in \mathbb{Z} : n_1 - n_2$ nur dann berechnet, wenn $n_1 \geq 0, n_2 \geq 0, n_1 \geq n_2$. Wenn die nicht gilt soll der Fehlerwert von `Maybe Int` zurückgegeben werden. Das erste Argument ist n_1 und das zweite n_2.

```
minus :: Maybe Int -> Maybe Int -> Maybe Int
minus Nothing _ = ....................
minus _ Nothing = ....................
minus (Just n)  _ | n < 0 = ....................
minus _  (Just n) | n < 0 = ....................
minus (Just n1) (Just n2) | ....................
    = Nothing
minus (Just n1) (Just n2) | ....................
    = ..........
```

b) Ist der Datentyp `Maybe Int` mit `mappend = minus` ein Monoid? Kreuzen sie an und vervollständigen sie wenn nötig.

☐ Monoid, mit Neutrum .....

☐ ein Monoid