Vorlesung
Programmierung und Modellierung mit Haskell

Probeklausur 1 – Aufgaben

François Bry

23.5.2016

Ablauf der Probeklausur

1. Bearbeitungsdauer: 1 Stunde
2. 4 Aufgaben
3. Jede Aufgabe wird 15 Minuten lang auf der Leinwand angezeigt.
4. Auf Backstage kann zu jeder Zeit jede Aufgabe angesehen werden.
5. Ausschließlich die Programmersprache Haskell soll verwendet werden.

Ablauf der Korrektur

Unmittelbar nach Ablauf der Probeklausur:

1. Die Lösung jeder Aufgabe wird gegeben.
2. Die Bewertung der Lösung wird erläutert.
3. Jeder Studierende
   - bewertet selbst seine Lösung,
   - kann seine Bewertung über Backstage mitteilen.

Aufgabe 1 – Teil 1

Wählen sie die korrekten Antworte aus oder geben Sie die korrekten Antworte an, falls keine der angegebenen Antworte korrekt ist:

1. Sei die folgende Definition: `f x = (x + 1 :: Int)`. Der Typ von `f` ist:
   - 1.1 `Int -> Int`
   - 1.2 `Int`
   - 1.3 `Integer`
   - 1.4 nichts davon, sondern ....
   - 1.5 Dieser Ausdruck ist kein Haskell-Ausdruck und hat folglich keinen Typ.
2. Der Typ von `(\x -> "b"++ x)` ist:
   - 2.1 `String -> String` (oder `[Char] -> [Char]`)
   - 2.2 `Char -> String` (oder `Char -> [Char]`)
   - 2.3 `Char -> Char`
   - 2.4 nichts davon, sondern ....
   - 2.5 Dieser Ausdruck ist kein Haskell-Ausdruck und hat folglich keinen Typ.

Aufgabe 1 – Teil 2

Wählen sie die korrekten Antworte aus oder geben Sie die korrekten Antworte an, falls keine der angegebenen Antworte korrekt ist:

3. Sei die folgende Definition: `g x = x ++ x`. Der Typ von `g` ist:
   - 3.1 `String -> String` (oder `[Char] -> [Char]`)
   - 3.2 `[a] -> [a]`
   - 3.3 `Integer -> Integer`
   - 3.4 nichts davon, sondern ....
   - 3.5 Dieser Ausdruck ist kein Haskell-Ausdruck und hat folglich keinen Typ.
4. Sei die folgende Definition:

```
h []       = []
h (x:xs) = x ++ x ++ (h xs)
```

   - 4.1 Der Typ von `h ["b", "c"]` ist
   - 4.2 `String (oder [Char])`
   - 4.3 `[String]` (oder `[[Char]]`)
   - 4.4 `[[String]]` (oder `[[[Char]]]`)
   - 4.5 nichts davon, sondern ....
   - 4.6 Dieser Ausdruck ist kein Haskell-Ausdruck und hat folglich keinen Typ.

Aufgabe 2

Die Summe der ersten natürlichen Zahlen kann wie folgt definiert werden:

$$\sum_{i=0}^{i=0} i = 0$$

$$\sum_{i=0}^{i=n} i = \left(\sum_{i=0}^{i=n-1} \right) + n \quad \texttt{für} \;\; n \geq 1$$

1. Geben Sie eine rekursive Funktion `summe` an, die dieser Definition unmittelbar entspricht. Die Funktion `summe` soll nicht terminieren, wenn sie auf negative ganze Zahlen angewandt wird.
2. Geben sie eine rekursive Funktion `summe'` an, die angewandt auf nicht-negative ganze Zahlen sich wie `summe` verhält und angewandt auf negative ganze Zahlen `0` liefert.
3. Geben Sie eine weitere rekursive Funktion `summe''` an, die endrekursiv ist, und sich wie `summe'` verhält.

Aufgabe 3

Seien die folgenden Definitionen:

```
f n = if n == 0 then 1 else n * (f (n-1))
doppelt x = x + x
null x = 0

hd :: [Int] -> Int
hd (x:xs) = x
```

Es ist *nicht* nötig, bei der Lösung der folgenden Aufgaben die Umgebung anzugeben. Pro Zeilet soll nur einen Auswertungsschritt angegeben werden.

1. Geben Sie die Auswertung von `f 1` in applikativer Reihenfolge.
2. Geben Sie die Auswertung von `f 1` in normaler Reihenfolge.
3. Geben Sie die verzögerte Auswertung von `null (doppelt 1)`.
4. Geben Sie die verzögerte Auswertung von `hd [4..]`.

Aufgabe 4

Sei die folgende Definition:

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

wobei `BB` für Binärbaum steht, `L` für leer und `K` für Knoten.

1. Geben Sie einen ausgeglichenen Baum vom Typ `Num a => BB a` für die Werte `0`, `1`, `2`, `3`, `4`, `5`, und `6` an.
2. Geben Sie eine rekursive Suchfunktion suche vom Typ `Eq a => a -> BB a -> Bool` für Binärbäume vom Typ `BB` an.