Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

30. Januar 2012

Aufgaben zur Klausur **Grundlagen der funktionalen Programmierung** im WS 2011/12 (BInf 13b)

Zeit: 90 Minuten erlaubte Hilfsmittel: keine

Bitte tragen Sie Ihre Antworten und fertigen Lösungen ausschließlich an den gekennzeichneten Stellen in das Aufgabenblatt ein. Ist ihre Lösung wesentlich umfangreicher, so überprüfen Sie bitte nochmals Ihren Lösungsweg.

Sollten Unklarheiten oder Mehrdeutigkeiten bei der Aufgabenstellung auftreten, so notieren Sie bitte, wie Sie die Aufgabe interpretiert haben.

Viel Erfolg!

Diese Klausur besteht einschließlich dieses Deckblattes aus 5 Seiten.

Aufgabe 1:

Die `map`-Funktion dient zur elementweisen Verarbeitung von Listen. Sie besitzt folgenden Typ

$$map :: (a \to b) \to [a] \to [b]$$

Definieren Sie `map` mit Hilfe einer Listcomprehension

. . . . . .

. . . . . .

. . . . . .

Definieren Sie `map` mit Hilfe einer rekursiven Funktion

. . . . . .

. . . . . .

. . . . . .

Aufgabe 2:

Entwickeln Sie eine Funktion `factors` mit folgendem Typ

$$factors :: Int \rightarrow [Int]$$

Diese Funktion soll alle Teiler einer Zahl in einer Liste sammeln. Die Liste soll aufsteigend sortiert sein. Hinweis: Es brauchen für die Lösung nur positive Zahlen betrachtet werden. Jeder Teiler soll in der Liste nur einmal vorkommen. Zum Beispiel ergibt sich für `n = 24` folgendes Resultat `[1,2,3,4,6,8,12,24]`.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Entwickeln Sie eine zweite Funktion `primefactors` mit gleichem Typ wie `factors`. Diese Funktion soll die Liste aller Primfaktoren für eine Zahl berechnen. Diese Liste soll wieder sortiert sein. Die Funktion soll folgende Eigenschaft erfüllen:

$$product\ (primefactors\ n) = n$$

für `n = 24` ergibt sich also das Ergebnis `[2,2,2,3]`, für `n = 23` das Ergebnis `[23]`.

Hinweis: Implementieren Sie die Funktion mit einer rekursiven Hilfsfunktion mit einem zusätzlichen Parameter für die möglichen Teiler

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Aufgabe 3:

Gegeben seien die folgenden rekursiven Funktionen. Diese besitzen eine ähnliche Struktur.

$$
\begin{array}{ll}
sum & :: Num\ a \Rightarrow [a] \rightarrow a \\
sum\ [\,] & = 0 \\
sum\ (x : xs) & = x + sum\ xs \\
reverse & :: [a] \rightarrow [a] \\
reverse\ [\,] & = [\,] \\
reverse\ (x : xs) & = reverse\ xs +\!\!+ [x]
\end{array}
$$

Um solche ähnlich strukturierten Funktionen einfacher zu implementieren, gibt es die so genannten `fold`-Funktionen. `foldr` ist eine dieser Funktionen mit folgendem Typ:

$$foldr :: (a \rightarrow b \rightarrow b) \rightarrow b \rightarrow [a] \rightarrow b$$

Implementieren Sie diese `foldr`-Funktion:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Reimplementieren sie `sum` mit Hilfe von `foldr`

. . . . . .

. . . . . .

Reimplementieren sie `reverse` mit Hilfe von `foldr`

. . . . . .

. . . . . .

. . . . . .

Aufgabe 4:

Es sei folgender rekursiver Datentyp gegeben:

$$\begin{aligned}\textbf{data}\ \mathit{Tree}\ a = {} & \mathit{Nil} \\ \mid {} & \mathit{Leaf}\ \ a \\ \mid {} & \mathit{Node}\ (\mathit{Tree}\ a)\ a\ (\mathit{Tree}\ a)\end{aligned}$$

Entwickeln Sie eine Funktion `find`, mit der ein Element in einem Baum gesucht werden kann.

$$\mathit{find} :: \mathit{Eq}\ a \Rightarrow a \rightarrow \mathit{Tree}\ a \rightarrow \mathit{Bool}$$

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Entwickeln sie eine Funktion `flatten`, die die in einem Baum gespeicherten Elemente in einer Liste aufsammelt.

$$\mathit{flatten} :: \mathit{Tree}\ a \rightarrow [a]$$

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .