Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

28. August 2014

Aufgaben zur Klausur **Grundlagen der funktionalen Programmierung** im SS 2014 (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.

Nutzen Sie die Rückseiten der Klausur zur Entwicklung der Lösungen und übertragen die fertigen Lösungen in das Aufgabenblatt.

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 6 Seiten.

Aufgabe 1:

Entwickeln Sie eine rekursive Funktion `merge`, die zwei sortierte Listen zu einer sortierten Liste zusammen mischt. Zum Beispiel ergibt `merge [2,3,5,6] [1,3,4]` das Ergebnis `[1,2,3,4,5,6]`. Elemente, die in beiden Listen auftauchen, sind im Resultat also nur einmal enthalten.

$$merge :: Ord\ a \Rightarrow [a] \rightarrow [a] \rightarrow [a]$$

Eintwickeln Sie eine Function `msort`, die mit Hilfe der `merge`-Funktion und einer Funktion `halve` (halbieren) eine Liste sortiert. Die `halve`-Funktion spaltet eine Liste in zwei Teillisten auf, und zwar so, dass deren Länge sich nur um 1 unterscheidet. Tipp: Sehr kurze Listen sind immer sortiert.

Die `halve`-Funktion:

$$halve :: [a] \rightarrow ([a], [a])$$

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Die `msort`-Funktion:

$$msort :: Ord\ a \Rightarrow [a] \rightarrow [a]$$

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Aufgabe 2:

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

$$
\begin{aligned}
&prod && :: Num\ a \Rightarrow [a] \to a \\
&prod\ [\,] && = 1 \\
&prod\ (x : xs) && = x * prod\ xs \\
&reverse && :: [a] \to [a] \\
&reverse\ [\,] && = [\,] \\
&reverse\ (x : xs) && = reverse\ xs +\!\!+ [x]
\end{aligned}
$$

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

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

Implementieren Sie diese `foldr`-Funktion:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Reimplementieren sie `prod` mit Hilfe von `foldr`

. . . . . .

. . . . . .

Reimplementieren sie `reverse` mit Hilfe von `foldr`

. . . . . .

. . . . . .

. . . . . .

Aufgabe 3:

Die `filter`-Funktion dient zum Löschen aller Elemente einer Liste, die eine bestimmte Eigenschaft nicht erfüllen. Sie besitzt folgenden Typ:

$$\mathit{filter} :: (a \to \mathit{Bool}) \to [a] \to [a]$$

Definieren Sie `filter` mit Hilfe einer Listcomprehension

. . . . . .

. . . . . .

. . . . . .

Definieren Sie `filter` mit Hilfe einer rekursiven Funktion

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Definieren Sie `filter` mit Hilfe von `foldr`

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Aufgabe 4:

Es sei folgender rekursiver Datentyp gegeben:

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

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, und zwar so, dass für ein `Fork` erst das Element in `Fork`, dann die Elemente aus dem linken Teilbaum und anschießend die aus dem rechten Teilbaum in der Liste enthalten sind.

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

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .