Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

26. August 2013

Aufgaben zur Klausur **Grundlagen der funktionalen Programmierung** im SS 2013 (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:

Im Haskell Prelude ist eine Funktion `splitAt` vordefiniert. Diese nimmt eine ganze Zahl und eine Liste als Argumente und teilt die Liste in ein Anfangs- und ein Endstück.

Sei i die Zahl und n die Länge der Liste. Das Anfangsstück hat höchstens die Länge i. Wenn $i > n$ ist, so ist das Endstück leer, wenn $i \leq 0$ so ist das Anfangsstück leer. Anfangs- und Endstück konkateniert ergibt wieder die Ausgangsliste.

Entwickeln Sie diese Funktion:

$$splitAt :: Int \to [a] \to ([a], [a])$$

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Aufgabe 2:

Gegeben sei die folgende Funktion zur Verarbeitung von Listen:

$$mapZip :: (a \to b) \to [a] \to [(a, b)]$$
$$mapZip\ f\ xs = [(x, f\ x) \mid x \leftarrow xs]$$

Welches Ergebnis liefert ein Aufruf von
`mapZip (*2) [1..5]`

. . . . . .

Entwickeln Sie eine zweite Version von `mapZip`, die nicht mit einer Listcomprehension arbeitet.

. . . . . .

. . . . . .

. . . . . .

Weiter sei die folgende Funktion `rems` gegeben:

$$rems :: Integer \to [(Integer, Integer)]$$
$$rems\ n = [(i, n\ `mod`\ i) \mid i \leftarrow [2\,..\,(n-1)]]$$

Welches Resultat liefert ein Aufruf von `rems 6`

. . . . . .

Entwickeln Sie eine zweite Version von `rems`, die nicht mehr mit einer Listcomprehension arbeitet, sondern `mapZip` nutzt.

. . . . . .

. . . . . .

Die Funktionen `all` und `any` entsprechen den All- und Existenzquantoren. Sie sind wie folgt definiert:

$$
\begin{aligned}
&all && :: (a \to Bool) \to [a] \to Bool \\
&all\ p\ [\,] && = True \\
&all\ p\ (x : xs) && = p\ x \wedge all\ p\ xs \\
&any && :: (a \to Bool) \to [a] \to Bool \\
&any\ p\ [\,] && = False \\
&any\ p\ (x : xs) && = p\ x \vee any\ p\ xs
\end{aligned}
$$

Nutzen Sie diese (`all` oder `any`) und `rems` um eine Funktion
`isPrime :: Integer -> Bool`
zu entwickeln, die einen Primzahltest durchführt.

. . . . . .

. . . . . .

. . . . . .

Nutzen Sie `rems` und die vordefinierten Funktionen `map` und `filter`, um die Liste aller echten Teiler einer Zahl mit einer Funktion `factors` zu berechnen.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Entwickeln Sie eine zweite Version von `isPrime`, die mit `factors` arbeitet.

. . . . . .

. . . . . .

Aufgabe 3:

Es sei folgender rekursiver Datentyp gegeben:

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

Entwickeln Sie eine Funktion `mapTree`, mit der alle Elemente eines Baumes verarbeitet werden können, ohne die Struktur des Baumes zu verändern.

$$\mathit{mapTree} :: (a \to b) \to \mathit{Tree}\ a \to \mathit{Tree}\ b$$

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Gegeben sei eine Funktion

$$\begin{aligned}&\mathit{flatten} :: \mathit{Tree}\ a \to [a]\\ &\mathit{flatten}\ (\mathit{Leaf}\ x) = [x]\\ &\mathit{flatten}\ (\mathit{Node}\ l\ r) = \mathit{flatten}\ l +\!\!+ \mathit{flatten}\ r\end{aligned}$$

Entwickeln sie eine Funktion `reverseTree`, die den Baum so umstrukturiert, das gilt:
`flatten . reverseTree == reverse . flatten`

$$\mathit{reverseTree} :: \mathit{Tree}\ a \to \mathit{Tree}\ a$$

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Entwickeln Sie eine Funktion `foldTree`, mit der ein Baum zu einem Wert *zusammengefaltet* werden kann. Tipp: Orientieren Sie sich bei der Entwicklung der Funktion an der Signatur.

$$foldTree :: (b \to b \to b) \to (a \to b) \to Tree\ a \to b$$

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Nutzen Sie die Funktion `foldTree` zum Aufsummieren aller Elemente eines Baums, in dem Zahlen gespeichert sind.

$$sumTree :: Num\ a \Rightarrow Tree\ a \to a$$

. . . . . .

Nutzen Sie die Funktion `foldTree` zur Reimplementierung der `flatten`-Operation.

. . . . . .

. . . . . .

. . . . . .

Nutzen Sie die Funktion `foldTree` zur Reimplementierung der `mapTree`-Operation.

. . . . . .

. . . . . .

. . . . . .