Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

17. Februar 2014

Aufgaben zur Klausur **Grundlagen der funktionalen Programmierung** im WS 2013/14 (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:

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

. . . . . .

. . . . . .

. . . . . .

Definieren Sie `map` mit Hilfe von `foldr`

. . . . . .

. . . . . .

. . . . . .

Aufgabe 2:

Entwickeln Sie eine Funktion `zipWith`, mit der zwei Listen verarbeitet werden, und zwar so, dass jedes Element der ersten Liste mit jedem Element der zweiten mit einer 2-stelligen Funktion verarbeitet wird.

Die Signatur von `zipWith`:

$$zipWith :: (a \to b \to c) \to [a] \to [b] \to [c]$$

Implementieren Sie die Funktion mit Hilfe einer rekursiven Funktion.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Aufgabe 3:

Im Haskell Prelude ist eine Funktion `takeWhile` vordefiniert. Diese nimmt ein Prädikat und eine Liste als Argumente und nimmt das längste Anfangsstück der Liste, für das die Elemente das Prädikat erfüllen.

Entwickeln Sie diese Funktion:

$takeWhile :: (a \to Bool) \to [a] \to [a]$

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Die duale Funktion zu `takeWhile` ist `dropWhile`. Für `dropWhile` gilt folgendes Gesetz: `takeWhile p xs ++ dropWhile p xs == xs`.

Entwickeln Sie diese Funktion:

$dropWhile :: (a \to Bool) \to [a] \to [a]$

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Aufgabe 4:

Es sei folgender rekursiver Datentyp gegeben:

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

Entwickeln Sie eine Funktion `mapTree`, mit der ein Baum in einen strukturgleichen Baum transformiert wird, indem alle Elemente mit einer Funktion verarbeitete werden.

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

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Gegeben sei ein Beispiel-Baum mit Zahlen als Elementen

$$\begin{aligned}&\textbf{type}\ \mathit{TreeOfInt} = \mathit{Tree}\ \mathit{Int}\\ &\mathit{t1} :: \mathit{TreeOfInt}\\ &\mathit{t1} = \mathit{Node}\ (\mathit{Leaf}\ 1)\ (\mathit{Node}\ (\mathit{Leaf}\ 3)\ \mathit{Nil})\end{aligned}$$

Entwickeln Sie eine Funktion, die solche `TreeOfInt`-Bäume verarbeitet, indem alle Werte an den Blättern um 1 erhöht werden.

. . . . . .

. . . . . .

Entwickeln Sie eine Funktion `sumTree` (einschließlich Typ), die die Elemente in einem `TreeOfInt`-Baum aufsummiert.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Entwickeln Sie eine Funktion `invTree` (einschließlich Typ), mit der überprüft werden kann, dass ein `Nil`–Baum nie Teilbaum eines `Node`–Baums ist.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .
