Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

11. Februar 2013

Aufgaben zur Klausur **Grundlagen der funktionalen Programmierung** im WS 2012/13 (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 Funktion `zip` verarbeitet zwei Listen, indem Sie aus den Elementen der beiden Listen Paare bildet. Die Länge der Resultatliste ist das Minimum der Längen der beiden Argumentlisten. Entwickeln Sie diese Funktion.

$zip :: [a] \to [b] \to [(a, b)]$

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Verallgemeinern Sie diese Funktion zu einer `zipWith`-Funktion, bei der die Elemente mit Hilfe einer zweistelligen Funktion zu einem neuen Wert verknüpft werden.

Die Signatur (der Typ) von `zipWith`:

. . . . . .

Die Funktionsdefinition von `zipWith`:

. . . . . .

. . . . . .

. . . . . .

Reimplementieren sie `zip` mit `zipWith`:

$zip' :: [a] \to [b] \to [(a, b)]$

. . . . . .

Aufgabe 2:

Entwickeln Sie eine Funktion `combine`, 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 `combine`:

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

Implementieren Sie die Funktion mit Hilfe einer *List Comprehension*.

. . . . . .

. . . . . .

. . . . . .

Aufgabe 3:

Entwickeln Sie eine rekursive Funktion `delete2`, die aus einer Liste jedes zweite Element löscht. Implementieren Sie die Funktion direkt, d.h. ohne vordefinierte Hilfsfunktionen zu verwenden.

$delete2 :: [a] \to [a]$

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Gegeben sei die folgende Funktion `number` zum Durchnummerieren der Elemente einer Liste:

$number :: [a] \to [(Int, a)]$
$number\ xs = zip\ [0\,..\,length\ xs - 1]\ xs$

Reimplementieren Sie die Funktion `delete2` als nicht rekursive Funktion. Hilfreich dabei können die Funktionen `number`, `filter`, `map`, `even` und `odd` sein.

$delete2' :: [a] \to [a]$

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Aufgabe 4:

Es sei folgender rekursiver Datentyp gegeben:

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

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

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

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

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

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

. . . . . .

. . . . . .

. . . . . .

. . . . . .

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.

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

. . . . . .

. . . . . .

. . . . . .

. . . . . .

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 Implementierung der `flatten`-Operation.

$$flatten' :: Tree\ a \to [a]$$

. . . . . .

. . . . . .
