Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

24. Januar 2017

Aufgaben zur Klausur **Grundlagen der funktionalen Programmierung** im WS 2016/17 (BInf)

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 7 Seiten.

Aufgabe 1:

`elem` und `filter` sind zwei häufig verwendete, vordefinierte Funktionen zum Suchen in und Selektieren aus Listen.

$$
\begin{aligned}
&elem :: Eq\ a \Rightarrow a \to [a] \to Bool \\
&elem\ x\ [\,] = False \\
&elem\ x\ (y : ys) = x\ ==\ y\ ||\ elem\ x\ ys
\end{aligned}
$$

$$
\begin{aligned}
&filter :: (a \to Bool) \to [a] \to [a] \\
&filter\ p\ [\,] = [\,] \\
&filter\ p\ (x : xs) \\
&\quad \mid p\ x = x : filter\ p\ xs \\
&\quad \mid otherwise = filter\ p\ xs
\end{aligned}
$$

Entwickeln Sie eine Funktion `elem1` mit gleicher Funktionalität wie `elem`, die aber die Berechnung mit Hilfe von `filter` durchführt. Nutzen Sie bei der Definition, wenn möglich, die Funltionskomposition `(.)` aus.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Entwickeln Sie eine Funktion `elem2` mit gleicher Funktionalität wie `elem`, die aber mit `foldr` arbeitet.

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

Der Code für `elem2`

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Entwickeln Sie eine Funktion `elem3` mit gleicher Funktionalität wie `elem`, die aber mit `foldl` arbeitet.

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

Der Code für `elem3`

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Welche der Funktionen `elem2` und `elem3` ist effizienter?

. . . . . .

Begründung:

. . . . . .

. . . . . .

Entwickeln Sie eine Funktion `filter1` mit gleicher Funktionalität wie `filter`, die aber mit `foldr` arbeitet.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Aufgabe 2:

Gegeben seien die folgenden Datentypen

$$
\begin{aligned}
&\textbf{data } T\ a \\
&\quad = T0\ a \\
&\quad \mid T1\ Op1\ (T\ a) \\
&\quad \mid T2\ Op2\ (T\ a)\ (T\ a) \\
&\textbf{data } Op1 = U1 \mid U2 \mid U3 \textbf{ deriving } Eq \\
&\textbf{data } Op2 = B1 \mid B2 \mid B3 \textbf{ deriving } Eq
\end{aligned}
$$

Ziel der ersten Teilaufgabe ist es, eine Gleichheitsfunktion für `T` zu definieren, also eine Instanz von `Eq` für `(T a)` zu entwickeln.

$$
\begin{aligned}
&\textbf{instance } Eq\ a \Rightarrow Eq\ (T\ a) \textbf{ where} \\
&\quad (==) = eqT
\end{aligned}
$$

Entwickeln Sie den Code für die Funktion `eqT`

$$eqT :: Eq\ a \Rightarrow T\ a \rightarrow T\ a \rightarrow Bool$$

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

`T` ist wie `[]` und `Maybe` ein Typkonstruktor mit einem Typparameter. Man kann also versuchen, für `T` eine Instanz von `Functor` zu definieren.

$$\begin{array}{l}\textbf{class } \mathit{Functor}\ f\ \textbf{where}\\ \quad \mathit{fmap} :: (a \to b) \to f\ a \to f\ b\\ \textbf{instance } \mathit{Functor}\ T\ \textbf{where}\\ \quad \mathit{fmap} = \mathit{mapT}\end{array}$$

Entwickeln Sie die Funktion `mapT`. Diese sollte gemäß der Gesetze für Funktoren die Struktur der `T`-Werte erhalten.

$$\mathit{mapT} :: (a \to b) \to T\ a \to T\ b$$

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Aufgabe 3:

Es gibt eine vordefinierte Funktion `zipWith` zur gleichzeitigen Verarbeitung zweier Listen.

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

Die Funktion arbeitet so, dass aus den i-ten Elementen der Argumentlisten das i-te Element des Resultats mit der als Parameter gegebenen Funktion berechnet wird. Dieses ist nur dann möglich, wenn das i-te Element in beiden Listen existiert.

Konstruieren sie `zipWith`

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Gegeben sei die folgende Datenstruktur `Tree` für 2-stelligen Bäume mit Information an den inneren Knoten.

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

Konstruieren Sie eine Funktion `zipTreeWith`, die analog zu `zipWith` zwei Bäume elementweise verarbeitet und daraus einen neuen Baum aufbaut. Dieses ist, wie bei Listen, nur dann möglich, wenn in beiden Bäumen an der gleichen Stelle ein Element existiert.

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

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .
