Dr. Steffen Jost
Leah Neukirchen

Ludwig-Maximilians-Universität München
Institut für Informatik
3. Mai 2018

04. Übung zur Vorlesung

Programmierung und Modellierung

Hinweis: Aufgrund des Feiertages entfallen die Übungen am Donnerstag & Freitag (10./11.5.18); diese Übungsgruppen bearbeiten Blatt 5 dann am 17./18.5.18. Entsprechend wird Übungsblatt 5 am Dienstag, 15.5.18 herausgegeben.
Grundsätzlich gilt ab sofort: Alle Übungsgruppen finden gleich oft statt und bearbeiten alle Blätter nacheinander; einzige Ausnahme war das 3. Blatt, welches Dienstags nicht bearbeitet werden konnte.

A4-1 ***Termination*** Ein Riesenfass ist mit vielen Weißwürsten und Brezen gefüllt. Außerdem steht neben dem Fass ein Backofen, der beliebig viele Brezen erzeugen kann. Der folgende Vorgang soll nun solange ausgeführt werden, bis er sich nicht mehr wiederholen läßt:

1) Entnehmen Sie zwei beliebige Nahrungsmittel aus dem Fass.
2) Falls beide vom gleich sind, essen Sie sie beide Nahrungsmittel auf und füllen eine neue Breze aus dem Backofen in das Fass.
3) Haben Sie eine Breze und eine Weißwurst genommen, dürfen Sie nur die Breze verputzen. Die Weißwurst muss wieder in das Fass zurück.

Beantworten Sie folgende Fragen: Terminiert dieser Vorgang? Falls ja, wie viele Schmankerl verbleiben am Ende im Fass? Falls nein, wie entwickelt sich das Verhältnis von Weißwürsten zu Brezn? Falls jein, hängt die Termination davon ab, welche Kombination von Nahrungsmittel man in welcher Reihenfolge zieht? Begründen Sie Ihre Antwort!

A4-2 ***Induktion mit Listen I*** Wir bezeichnen mit $|l| \in \mathbb{N}$ die Länge einer Liste l. Für eine beliebige nicht-leere Liste $(x : xs)$ gilt $|(x : xs)| = 1 + |xs|$.

a) Beweisen Sie mit Induktion über die Länge der Liste, dass für alle ganzen Zahlen $z \in \mathbb{Z}$, für alle natürlichen Zahlen $n \in \mathbb{N}$, und alle Listen von ganzen Zahlen zs mit $|zs| = n$ die Gleichung $|\texttt{insert z zs}| = 1 + |zs|$ gilt. Die Funktion `insert` ist definiert wie auf Folie 3-29:

```
insert :: Int -> [Int] -> [Int]
insert x    [] = [x]
insert x (y:ys) | x <= y     = x : y : ys
                | otherwise = y : insert x ys
```

Da Haskell eine rein funktionale Sprache ist, dürfen wir die definierenden Gleichung einer Funktion beliebig von links-nach-rechts oder auch von rechts-nach-links einsetzen. Lediglich bei überlappenden Mustervegleichen und/oder Wächtern müssen wir die entsprechenden Seitenbedingungen beachten. So dürfen wir zum Beispiel in unseren Rechnungen den Ausdruck $y :$ `insert` $x\ ys$ nur dann durch `insert x` $(\texttt{y} : \texttt{ys})$ ersetzen, wenn $x > y$ angenommen werden darf. Das ist eigentlich klar, denn ansonsten würde ja bei der Ausführung der andere Zweig ausgewählt.

b) Beweisen Sie, dass für eine beliebige Liste l von ganzen Zahlen gilt

$$|l| = |\texttt{sort}\ l|$$

Dabei dürfen Sie die Aussage von Teilaufgabe a) ohne weiteres direkt verwenden. Die Funktion `sort` ist definiert wie auf Folie 3-29:

```
sort :: [Int] -> [Int]
sort     []  = []
sort (x:xs) = insert x (sort xs)
```

A4-3 *Datentypen Modellieren*

a) Geben Sie eine Datentypdeklaration für einen Typ `Akteur` an, um die Akteure eines Spiels zu verwalten: Es gibt Spieler, welche durch einen Namen (`String`) und ein Rating (`Double`) beschrieben werden; Computer, welche eine einstellbare Spielstärke (`Int`) besitzen; und Zuschauer, welche lediglich einen Namen (`String`) haben.

b) Implementieren Sie eine Funktion `anzeige :: Akteur -> String`, welche einen Akteur in einen String umwandelt, z.B. um diesen auf dem Bildschirm auszugeben. Bei der Darstellung als String soll ein Spieler nur durch seinen Namen repräsentiert werden, z.B. `"Steffen"`, d.h. das Rating des Spielers bleibt geheim, z.B. `"Steffen"`; bei einem Computer wird die Spielstärke mitangegeben, z.B. `"KI(42)"`; ein Zuschauer wird ebenfalls nur durch seinen Namen dargestellt, aber zur Unterscheidung soll dieser in Klammern eingefasst werden, z.B. `"[Martin]"`.

Hinweise: Folgende Bibliotheksfunktion könnten dabei nützlich sein: `show` zur Umwandlung von Zahlen in Strings; `(++)` zum Aneinanderhängen zweier Strings.

H4-1 *(Rekursion)* (2 Punkte; Datei `H4-1.hs` als Lösung abgeben)

Schreiben Sie eine Funktion `safeIndex :: [b] -> Integer -> Maybe b` welche das n-te Element einer Liste zurückliefert, falls dies existiert, und `Nothing` sonst.

```
> [1..10] `safeIndex`  3
Just 4
> [1..10] `safeIndex` 10
Nothing
```

H4-2 *Induktion mit Listen II* (2 Punkte; Abgabe: `H4-2.txt` oder `H4-2.pdf`)

Es sei vs und ws zwei beliebige Listen, weiterhin sei $n = |vs|$. Beweisen Sie mit Induktion über die Länge $n \in \mathbb{N}$ von vs, dass $|\texttt{zip}\ vs\ ws| = \min\big(|vs|, |ws|\big)$ gilt, wobei `zip` definiert ist wie auf Folie 03-30. *Hinweis:* Eine Induktion über n reicht aus. Eine Induktion über die Länge von ws ist nicht notwendig!

```
zip :: [a] -> [b] -> [(a,b)]
zip    []      _    = []
zip    _       []   = []
zip (x:xs) (y:ys) = (x,y) : zip xs ys
```

H4-3 ***Benutzerdefinierte Listen*** (2 Punkte; Datei `H4-3.hs` als Lösung abgeben)
Listen verstehen die meisten Teilnehmer sehr gut, so dass es Ihnen hoffentlich nicht mehr allzu viel Mühe bereitet, folgende Definition aus der Standardbibliothek zu verstehen:

```
(++) :: [a] -> [a] -> [a]
(++) []     ys = ys
(++) (x:xs) ys = x : xs ++ ys
```

Die Infix-Funktion `(++)` verkettet zwei Listen miteinander, also

```
> [1..3] ++ [4..5]
[1,2,3,4,5]
> (++) ['K','l'] ('a':'r':['o','!'])
"Klaro!"
```

Fragen Sie Ihren Assistenten oder Tutor, falls Sie mit dieser polymorphen, rekursiven Infix-Definition noch Schwierigkeiten haben!

Viele Teilnehmer haben jedoch Probleme mit benutzerdefinierten Datentypen, weshalb auf Vorlesungsfolien 4.19 und 4.35 gezeigt wurde, wie man gewöhnliche Listen ganz äquivalent "zu Fuss" deklarieren müsste, falls diese in Haskell nicht eingebaut wären:

```
data MyList a = Leer | Element a (MyList a)
  deriving Show
```

Ihre Aufgabe: Schreiben Sie eine Funktion `verketten :: MyList a -> MyList a -> MyList a`, welche zwei My-Listen mit einander verkettet. Sie müssen also lediglich den oben angegeben Code zur Benutzung dieses Datentyp `MyList a` umschreiben. Die Struktur des Codes (Fallunterscheidungen, Rekursion, etc.) bleiben gleich. *Beispiel:*

```
> verketten (Element 1 Leer) (Element 2 (Element 3 Leer))
Element 1 (Element 2 (Element 3 Leer))
```

Abgabe: Lösungen zu den Hausaufgaben können bis Samstag, den 12.5.18, mit UniWorX nur als `.zip` abgegeben werden. Abschreiben bei den Hausaufgaben gilt als Betrug und kann zum Ausschluss von der Klausur zur Vorlesung führen. Bis zu 3 Studierende können gemeinsam als Gruppe abgeben. Bitte beachten Sie auch die Hinweise zum Übungsbetrieb auf der Vorlesungshomepage (`www.tcs.ifi.lmu.de/lehre/ss-2018/promo/`).