Dr. Steffen Jost
Leah Neukirchen

Ludwig-Maximilians-Universität München
Institut für Informatik
26. April 2018

03. Übung zur Vorlesung

Programmierung und Modellierung

Hinweis: Die Vorlesung am 30.4.18 entfällt planmäßig. Am Dienstag, den 1.5.18, entfallen aufgrund des Feiertags alle Übungen ersatzlos.

A3-1 ***Endrekursive Fakultät*** Definieren Sie Fakultätsfunktion von Folie 3.3 erneut in einer endrekursiven Version durch Einführung eines Akkumulators!

```
fakultät 0 = 1
fakultät n = n * fakultät (n-1)
```

A3-2 ***Quicksort*** Das Quicksort-Verfahren ist ein rekursiver Sortier-Algorithmus:

Schritt 1: Ist die Liste leer, so ist diese fertig sortiert. Ansonsten wähle man irgendein Element der sortierenden Liste, z.B. das erste Element.

Schritt 2: Teile die Restliste in zwei Teillisten auf: eine Teilliste enthalte alle Elemente, welche kleiner oder gleich sind als das gewählte Element; die andere alle größeren Elemente.

Schritt 3: Sortiere beide Teillisten durch rekursive Verwendung des Quicksort-Verfahrens.

Schritt 4: Füge die beiden sortierten Teillisten mit `(++) :: [a]->[a]->[a]` wieder zusammen, wobei das anfangs ausgewählte Element in die Mitte dazwischen gesteckt wird.

Implementieren Sie diesen Algorithmus in Haskell, indem Sie folgende Funktionen definieren:

```
quicksort :: [Int] -> [Int]
splitBy   :: Int -> [Int] -> ([Int],[Int])
```

Die Funktion `quicksort` sortiert eine Liste. Die Funktion `splitBy` teilt eine Liste von Zahlen wie angegeben in zwei Teillisten auf. *Beispiele:*

```
> splitBy 6 [3,9,4,2,6,8,5,1,7,2,2]
([3,4,2,6,5,1,2,2],[9,8,7])

> quicksort [3,9,4,2,6,8,5,1,7,2,2]
[1,2,2,2,3,4,5,6,7,8,9]
```

a) Implementieren Sie die Funktion `splitBy` und testen Sie diese anschließend.

   Wem nicht einfällt, wie man hier beginnen könnte, kann einfach zwei List-Comprehensions verwenden. Das funktioniert, ist aber nicht besonders effizient, da die Eingabeliste dabei zwei Mal durchlaufen werden muss.

   Schöner ist es, wenn die Eingabeliste nur einmal durchlaufen wird, z.B. durch eine Rekursion über die Eingabeliste mit anschließendem Anfügen des aktuellen Elementes in die richtige Ergebnisliste. Da die Reihenfolge der Elemente in den beiden Ergebnislisten nicht spezifiziert wurde, läßt sich dann auch schnell eine endrekursive Version schreiben.

b) Implementieren Sie nun die Funktion `quicksort` gemäß des angegebenen Algorithmus.

A3-3 ***Abstiegsfunktion*** Zeigen Sie jeweils mithilfe einer geeigneten Abstiegsfunktion, dass jede der die folgende Funktionsdefinition für alle ganzen Zahlen terminiert!

a)
```
foo x = if x > 1 then 2*x + foo(x-2) else x `div` 2
```

b) Gegeben ist folgende Funktionsdefinition:

```
bar :: Integer -> Integer -> Integer
bar x y | x+y < 1 = 1
        | odd  y  = (x+1) + (bar (x-1) y)
        | even y  = (y+1) * (bar x (y-1))
```

i) Warum ist $f(x, y) = \max(x, 0)$ hier keine geeignete Abstiegsfunktion?

ii) Zeigen Sie mithilfe einer geeigneten Abstiegsfunktion, dass `bar` immer für alle beliebigen ganzen Zahlen terminiert!

c)
```
foobar :: Integer -> Integer -> Integer
foobar x y
 | x > 0     = 1 + (foobar (x-1) y)
 | y > 0     = foobar 7 (y-1)
 | otherwise = 0
```

d)
```
barfoo x
 | x < 111 = (barfoo (x+1) * barfoo (x+2) * x) `mod` 111
 | otherwise = 1
```

H3-1 ***Abstiegsfunktion II*** (2 Punkte; Abgabe: `H3-1.txt` oder `H3-1.pdf`)
Gegeben ist folgende Funktionsdefinition:

```
magic :: Integer -> Integer -> Integer -> String
magic a b c | even c, b > 0, a < 0 = 'x' : (magic (1+a)  b    (1+c))
            | odd  c, b > 0, a < 0 = 'y' : (magic    a  (b-1) (c+1))
            | otherwise            = show c
```

a) Warum ist die Funktion $f(a, b, c) := \max(a+c, 0)$ hier keine geeignete Abstiegsfunktion? Zeigen Sie dies durch Angabe eines Gegenbeispiels, also Werte für `a`,`b` und `c` so dass ein rekursiver Aufruf stattfindet, für welchen f verbotenerweise nicht kleiner wird.

b) Beweisen mithilfe einer geeigneten Abstiegsfunktion ausführlich, dass die folgend definierte Funktion für alle ganzen Zahlen terminiert.

Hinweise: Diese Funktion enthält für die Termination irrelevanten Code. Versuchen Sie die Abstiegsfunktion so einfach wie möglich zu wählen. Ihre Abstiegsfunktion muss nur *eine* obere Schranke an die rekursiven Aufrufe beschreiben; sie muss nicht exakt sein.

H3-2 ***Rekursion mit Listen*** (2 Punkte; Datei `H3-2.hs` als Lösung abgeben)
Implementieren zu Fuß[1] mit Pattern-Matching und Rekursion...

a) die Funktion `entferne :: Int -> [Int] -> [Int]` welche alle Vorkommen der gegebenen Zahl aus der Liste entfernt, aber ansonsten die Reihenfolge unverändert lässt:

```
> entferne 7 [4,5,6,7,7,8,9,8,7,6,7,4,7]
[4,5,6,8,9,8,6,4]
```

b) eine endrekursive Funktion `umdrehen :: [a] -> [a]` welche die Reihenfolge einer Liste umkehrt.

Abgabe: Lösungen zu den Hausaufgaben können bis Samstag, den 5.5.18, 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/`).

Es kann zu einem Punktabzug kommen, falls Ihre Abgabe nicht die geforderten Dateinamen genau einhalten, Funktionsnamen nicht richtig geschrieben wurden, andere Archive als `.zip` verwendet werden, oder wenn Syntaxfehler vorliegen.

Falls Sie an einer Stelle nicht weiter wissen, dann kommentieren Sie die entsprechenden Stellen mit einem Hinweisen aus und/oder vervollständigen Sie Ihre Abgabe mit einem Aufrufen der Funktion `error :: String -> a`. *Beispiel:*

```
foo _ [] = [] -- Fall ok!
foo _ [x] = error "H3-9b, foo: behandlung einelementiger Listen unklar" -- Hilfe!
-- foo x [h:t] = foo t ++ [h*x] -- Zeile kompiliert leider nicht. Hilfe!
```

Jeglicher Hinweistext sollte ordnungsgemäß als Kommentar in den Code geschrieben werden, d.h. hinter `--` oder in Kommentarklammern `{- mein hinweistext -}`

[1] Wie im Hinweis auf Blatt 2 angegeben ohne kompliziertere Funktionen aus der Standardbibliothek.