Dr. Steffen Jost
Leah Neukirchen

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

Lösungsvorschlag zur 02. Übung zur Vorlesung

Programmierung und Modellierung

A2-1 ***Substitutionsmodell I*** Werten Sie folgenden Ausdruck gemäß dem in der Vorlesung behandelten Substitutionsmodell aus: $\Big((\backslash x \texttt{ -> } (\backslash(y : \_) \texttt{ -> } x - y))\ (3+4)\Big)\ (\texttt{tail}\ [1,2])$

LÖSUNGSVORSCHLAG:

Wir verzichten hier gleich auf alle überschüssige Klammern gemäß der Klammerkonvention.

Weiterhin rufen wir uns vorher noch die Definition von `tail` aus Kapitel 2 (oder auch aus der Standardbibliothek) in Erinnerung: $(tail)\ (\_ : t) = t$. Damit man besser verstehen kann, wie das Pattern-Matching bei der Funktionsanwendung von `tail` und der anonymen Funktion abläuft, schreiben wir anstatt $[1,2]$ gleich $(1 : (2 : []))$, beide Schreibweisen sind ja äquivalent.

$$
\begin{aligned}
& (\backslash x \texttt{ -> } (\backslash(y : \_) \texttt{ -> } x - y))\ \underline{(3+4)}\ (\texttt{tail}\ (1 : (2 : []))) \\
\rightsquigarrow\ & (\backslash x \texttt{ -> } (\backslash(y : \_) \texttt{ -> } x - y))\ 7\ \underline{(\texttt{tail}\ (1 : (2 : [])))} \\
\rightsquigarrow\ & (\backslash x \texttt{ -> } (\backslash(y : \_) \texttt{ -> } x - y))\ 7\ (2 : []) \\
\rightsquigarrow\ & (\backslash(y : \_) \texttt{ -> } 7 - y))\ (2 : []) \\
\rightsquigarrow\ & 7 - 2 \rightsquigarrow 5
\end{aligned}
$$

A2-2 ***Replicate*** Die Funktion `replicate :: Int -> a -> [a]` aus der Standardbibliothek wiederholt einen gegebenen Wert n-mal:

```
> replicate 3 "Ha!"
["Ha!","Ha!","Ha!"]

> replicate 7 5
[5,5,5,5,5,5,5]
```

a) Implementieren Sie die Funktion unter Verwendung einer List-Comprehension, **ohne** Rekursion! *Hinweis:* geben Sie der Funktion einen anderen Namen, z.B. `meinRep`

LÖSUNGSVORSCHLAG:

```
meinRep n e = [e | _x <-[1..n] ]
```

Bemerkung: Nicht jede rekursive Funktion läßt sich mithilfe eine List-Comprehension ausdrücken, umgekehrt gilt dies schon, da Rekursion allgemeiner ist.

b) Implementieren Sie diese Funktion erneut, dieses Mal mit Hilfe von Rekursion, aber ohne Verwendung von List-Comprehension!

LÖSUNGSVORSCHLAG:

```
meinRep n e | n > 0     = e : meinRep (n-1) e
            | otherwise = []
```

c) Überprüfen Sie, welche Art der Rekursion Ihre Lösung zur vorherigen Teilaufgabe verwendet! Formulieren Sie ggf. jetzt noch eine **endrekursive** Version.

LÖSUNGSVORSCHLAG:

Unsere Lösung war linear rekursiv, aber nicht endrekursiv. Hier ist nun eine endrekursive Variante, welche eine Hilfsfunktion verwendet. Die Hilfsfunktion bekommt ein zusätzliches Argument für das vorläufige Endergebnis der Berechnung, welches auch als Akkumulator bezeichnet wird.

```
meinRep n e = meinRepAux n []
  where
    meinRepAux 0 acc = acc
    meinRepAux n acc = meinRepAux (n-1) (e:acc)
```

A2-3 ***List Comprehension I*** Mia die modische Mathematikerin hat folgende Klamotten:

	Schwarz	Weiß	Pink	Flieder
Tops	3	3	3	4
Hosen	4	3	2	1
Schuhe	3	2	1	1

Mia möchte wissen, wie viele verschiedene Kombinationen Ihr damit zum Anziehen zur Verfügung stehen. Allerdings zieht Mia keine einfarbigen Outfits an; und die Farbe Ihrer

Schuhe müssen zur Farbe Ihres Tops oder zu Ihrer Hose passen (gleiche Farbe). Als Mathematikerin kann Mia die Menge aller akzeptablen Kombinationen fix hinschreiben:

$$\Big\{(t,h,s) \;\Big|\; t \in M_\mathrm{t}, h \in M_\mathrm{h}, s \in M_\mathrm{s},\ \text{mit } \mathtt{f}(t) \neq \mathtt{f}(h) \text{ und } \big(\mathtt{f}(t) = \mathtt{f}(s) \text{ oder } \mathtt{f}(h) = \mathtt{f}(s)\big)\Big\}$$

wobei $M_\mathrm{t}, M_\mathrm{h}$ und M_s jeweils die Menge ihrer Tops, Hosen und Schuhe bezeichnen, und die Funktion `f` ein Kleidungsstück auf seine Farbe abbildet.

Typisch für Mathematiker ist diese Antwort zweifellos richtig, aber irgendwie nutzlos. Rechnen Sie die Anzahl der Kombinationen mit einer List-Comprehension fix aus!

Hinweis: Zur Vereinfachung modellieren wir Klamottenmengen durch Listen von ganzen Zahlen, wobei jeder Zahlwert für eine der möglichen Farben steht. Zum Beispiel kodieren wir M_s durch die Liste `[0,0,0,1,1,2,3]`. Listen können Elemente mehrfach enthalten; Schuhe gleicher Farbe brauchen wir ja hier nicht weiter zu unterscheiden. Die Länge einer Liste berechnet man mit der Bibliotheksfunktion `length :: [a] -> Int`

LÖSUNGSVORSCHLAG:

```
-- Konstanten für Farben
schwarz = 0
weiss = 1
pink = 2
flieder = 3

-- Möglichkeiten, den Kleiderschrank zu modellieren:
-- miaTops   = [schwarz, schwarz, schwarz, weiss, weiss, weiss, pink, pink, pink,
--                                                    flieder, flieder, flieder, flieder]
-- miaHosen  = (replicate 4 schwarz) ++ (replicate 3 weiss) ++ (replicate 2 pink) ++ (replicate 1 flieder)
-- miaSchuhe = [schwarz, schwarz, schwarz, weiss, weiss, pink, flieder]

-- Generische Alternative mit List Comprehension:
genCloth :: Int -> Int -> Int -> Int -> [Int]
genCloth b w p f =
  [schwarz | _ <- [1..b]] ++ [weiss | _ <- [1..w]] ++ [pink | _ <- [1..p]] ++ [flieder | _ <- [1..f]]

miaTops   = genCloth 3 3 3 4
miaHosen  = genCloth 4 3 2 1
miaSchuhe = genCloth 3 2 1 1

-- Übersetzung der gegebenen mathematischen Formel als List-Comprehension:
miaKombinationen = [(t,h,s) | t<-miaTops, h<-miaHosen, s<-miaSchuhe, t/=h && (t==s || h==s)]
```

Die Liste `miaKombinationen` enthält alle für Mia akzeptablen Klamottenkombinationen. Jetzt müssen wir nur noch mit GHCi deren Größe ermitteln:

```
> length miaKombinationen
365
```

Mia stehen also 365 Kombinationen zur Verfügung; für jeden Tag im Jahr ein Outfit!

Ende der Lösungsvorschläge für die Präsenzaufgaben.
Lösungsvorschläge für die Hausaufgaben folgen nach Ende der Abgabezeit.

Hinweis Hausaufgaben: Für die Bearbeitung der Aufgaben und Hausaufgaben sind bis auf weiteres die Verwendung von Funktionen der Standardbibliothek tabu, abgesehen von den Grundoperationen wie `(:)`, `(++)`, `(>)`, `(>=)`, `(>)`, `(<=)`, `div`, `mod`, `not`, `(&&)`, `max`, `min`, etc. Von Ihnen implementierte Funktionen aus vorangegangenen Aufgaben sind zulässig.

H2-1 ***Substitutionsmodell II*** (2 Punkte; Abgabe: `H2-1.txt` oder `H2-1.pdf`)
Gegeben sind folgende zwei Funktionsdefinitionen

```
const x y =  x
negate x  = -x
```

und ein Ausdruck `const const (negate 1) (negate 2) 3`

a) In dem gegebenen Ausdruck müssen wir die implizite Klammerkonvention berücksichtigen, siehe Folie 1.50. Geben Sie den Ausdruck noch einmal mit vollständiger, expliziter Klammerung an!

b) Werten Sie nun den Ausdruck schrittweise gemäß dem in der Vorlesung behandelten Substitutionsmodell vollständig aus; unterstreichen Sie jeweils den bearbeiteten Teilausdruck.

H2-2 ***Rekursion*** (2 Punkte; Datei `H2-2.hs` als Lösung abgeben)

a) Schreiben Sie die rekursive Funktion `quersumme :: Int -> Int`, welche die Quersumme einer nicht-negativen Zahl berechnet.

```
> quersumme 1234
10
> 1 + 2 + 3 + 4
10
```

Hinweis: Aus der Standardbibliothek könnten die Funktionen `mod :: Int -> Int -> Int` und `div :: Int -> Int -> Int` hilfreich sein. `x `mod` 10` berechnet die letzte Dezimalziffer von `x` und `x `div` 10` entfernt die letzte Dezimalziffer.

b) In der Standardbibliothek gibt es eine Funktion `concat :: [[a]] -> [a]` welche eine Liste von Listen aneinanderhängt:

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

Implementieren Sie diese Funktion unter Verwendung von Rekursion selbst, natürlich ohne vorher den Code in der Standardbibliothek anzuschauen!

Die Infix-Funktion `(++) :: [a] -> [a] -> [a]` aus der Standardbibliothek dürfen Sie dabei verwenden, wenn Sie das möchten.

Zusatzfrage: Können Sie eine endrekursive Version entwickeln?

H2-3 ***List Comprehension II*** (2 Punkte; Datei `H2-3.hs` als Lösung abgeben)
Robert der Rollenspieler möchte die Wahrscheinlichkeit berechnen, dass seine Spielfigur eine Fertigkeitsprüfung besteht. In seinem Spiel wird diese Fertigkeitsprüfung wie folgt durchgeführt: Drei mal muss Robert mit einem 20-seitigen Würfel (Zahlen von 1–20) Attribute seiner Spielfigur (ebenfalls zwischen 1 und 20) unterwürfeln. Für jeden der drei Würfe wird die Differenz zwischen Würfelwert und Attributwert notiert, falls diese positiv ist und 0 sonst. Die Prüfung gelingt, wenn die Summe dieser abgeschnittenen Differenzen kleiner oder gleich dem Fertigkeitswert ist. Zusätzlich gilt: die Prüfung gelingt immer, wenn 2 oder 3 Würfelwürfe eine 1 ergeben haben; die Prüfung misslingt immer, falls 2–3 mal die 20 fällt.

Beispiel: Eine Fertigkeitsprüfung verlangt die Attribute Stärke, Geschicklichkeit und Charisma, für die Roberts Spielfigur die Werte 15, 12 und 6 hat. Robert würfelt der Reihe nach 16, 18 und 1. Für die Stärke-Prüfung fehlt 1 Punkt, für die Geschicklichkeit 6 und die Charisma-Prüfung wurde locker bestanden. Für Fertigkeit 7 oder mehr gelingt diese Prüfung also.

Robert möchte nun allgemein die Wahrscheinlichkeiten ausrechnen, bei welchen Attribut-Werten (a, b, c) und Fertigkeitswert s solch eine Prüfung gelingt. Robert hat leider fast alles vergessen, was er in der Schule über Wahrscheinlichkeitsrechnung gelernt hat. Seine mathematisch begabte Freundin Mia ist über das Wochenende leider zum Shopping verreist. Er weiß allerdings noch, dass er für die Berechnung der Wahrscheinlichkeit lediglich die Anzahl der günstigen Möglichkeiten durch die Anzahl aller Möglichkeiten für den Ausgang der 3 Würfelwürfe teilen muss. Da sich Robert mit Haskell's List Comprehensions auskennt, kann er sein Problem schnell lösen! "Schnell" bedeutet für Robert hier primär, dass er schnell ein korrektes Programm hingeschrieben hat. Ob die Berechnung dann 0,2 oder 2 Sekunden dauert, ist in seinem Anwendungsfall völlig egal.

a) Definieren Sie eine Fkt. `erfolg :: (Int,Int,Int) -> Int -> (Int,Int,Int) -> Bool` welche als erstes Argument ein Tripel von Attributen, als zweites Argument einen Fertigkeitswert und als drittes Argument das Ergebnis von drei Würfelwürfen bekommt, und entscheidet, ob eine Fertigkeitsprüfung gemäß der angegebenen Regeln erfolgreich war oder nicht. *Beispiele:*

```
> erfolg (16,13,8) 5 (1,17,10)
False
```

```
> erfolg (16,13,8) 5 (1,17,9)
True
```

b) Definieren Sie eine Funktion `chance :: (Int,Int,Int) -> Int -> Double` welche die Wahrscheinlichkeit berechnet, dass eine Fertigkeitsprüfung bestanden wird. *Beispiel:*

```
> chance (16,13,8) 5
0.518625
```

Ein Fertigkeitsprüfung mit den Attributen 16, 13 und 8 und Fertigkeit 5 wird also mit einer Chance von ungefähr 52 % bestanden. *Hinweis:* Das Ergebnis einer Gleitkomma-Division zweier ganzer Zahlen erhalten Sie mit folgender Definition:

```
myDiv :: Int -> Int -> Double  -- Typsignatur verhindert Fehlermeldung!
myDiv z n = (fromIntegral z) / (fromIntegral n)
```

Abgabe: Lösungen zu den Hausaufgaben können bis Samstag, den 28.4.18, 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/`).