Prof. Martin Hofmann, PhD
Dr. Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik
8. April 2013

1. Musterlösung zur Vorlesung

Programmierung und Modellierung

A1-1 ***Kartesisches Produkt*** Berechnen Sie mit Papier und Bleistift die Menge des jeweiligen kartesischen Produktes. Wenn Sie möchten, können Sie ihre Antworten mit GHCI überprüfen.

Hinweis: im Gegensatz zu Listen ist die Reihenfolge der Auflisting ist bei Mengen natürlich unerheblich.

a) $\{11, 7\} \times \{\clubsuit, \heartsuit\}$

LÖSUNGSVORSCHLAG: $\{(11, \clubsuit), (11, \heartsuit), (7, \clubsuit), (7, \heartsuit)\}$

b) $(\{1, 2\} \times \{3, 4\}) \times \{5, 6\}$

LÖSUNGSVORSCHLAG: $\{(1, 3, 5), (1, 3, 6), (1, 4, 5), (1, 4, 6), (2, 3, 5), (2, 3, 6), (2, 4, 5), (2, 4, 6)\}$

c) $\{1, 2\} \times (\{3, 4\} \times \{5, 6\})$

LÖSUNGSVORSCHLAG:

Abgesehen von der inneren Klammerung der Tripel ist das Ergebnis das Gleiche wie in der vorherigen Teilaufgabe. In der Mathematik ist es üblich, nicht zwischen diesen beiden Produkten zu unterscheiden und gleich Tripel zu verwenden. d.h. obwohl das kartesische Produkt eigentlich nicht assoziativ ist, wird es als assoziativ behandelt. Funktionale Sprachen unterscheiden jedoch üblicherweise zwischen solchen isomorphen Produkten, denn `(1,(2,3))` hat einen anderen Typ als `((1,2),3)` oder `(1,2,3)`.

d) $\{1, 2\} \times \{\}$

LÖSUNGSVORSCHLAG: Das Produkt ist hier die leere Menge.

e) Sei A eine Menge mit n Elementen. Wie viele Elemente gibt es in $A \times \{27, 69\}$?

LÖSUNGSVORSCHLAG: Doppelt so viele, da jedes Element aus A zwei Mal im Produkt mit $\{27, 69\}$ vorkommt; also insgesamt $2n$.

A1-2 ***Grundlegende Operatoren***

Die Funktionen für die logischen Operatoren UND, ODER und Negation wurden nicht in der Vorlesung erklärt. Finden Sie selbst heraus, wie diese Operatoren in Haskell heißen und testen Sie deren Funktionsweise kurz mit GHCI.

Hinweise: Die Vorlesungshomepage listet Links zur Lösung dieser Aufgabe. Zum Beispiel können Sie in der Dokumentation der Standardbibliotheken nachschlagen; grundlegende Funktionen findet man oft im Modul `Prelude` oder auch in den `Data`-Modulen, z.B. `Data.Bool`.

Die Haskell Suchmaschine Hoogle kann Bibliotheksfunktionen mithilfe des Typs schnell aufspüren; tippen Sie also alternativ den Typ der gesuchten Funktion in Hoogle ein.

Wenn Sie keinen Internetzugriff haben: Bei der Installation der Haskell Plattform sollte die Dokumentation der Standardbibliotheken auch lokal installiert worden sein. Eine weitere offline Möglichkeit zum Durchblättern der Dokumentation noch der GHCI Befehl `:browse`

LÖSUNGSVORSCHLAG:

```
Prelude> :browse Data.Bool
(&&) :: Bool -> Bool -> Bool
data Bool = False | True
not :: Bool -> Bool
otherwise :: Bool
(||) :: Bool -> Bool -> Bool

Prelude> True && False
False
Prelude> True || False
True
Prelude> not True
False
```

A1-3 ***Pattern-Matching*** Schreiben Sie eine Funktion `myAnd`,[1] welche den Typ `(Bool,Bool) -> Bool` hat und die logische Operation "und" implementiert. Natürlich dürfen Sie dabei *nicht* die Operatoren der Standardbibliothek aus der vorherigen Aufgabe verwenden!

a) Verwenden Sie bei der Definition ausschließlich Pattern-Matching.

b) Verwenden Sie bei der Definition ausschließlich Wächter (keine konstanten Patterns).

c) Verwenden Sie weder Pattern-Matching noch Wächter.

[1] Zur Vermeindung von Namenkonflikten mit der Standardbibliothek nicht "and" verwenden

LÖSUNGSVORSCHLAG:

a) Da der Typ vorgegeben war, können wir diese Aufgabe rein mechanisch lösen, indem wir für jeden möglichen Input eine Zeile mit dem Ergebnis hinschreiben:

```
myAnd1 :: (Bool,Bool) -> Bool
myAnd1 (False,False) = False
myAnd1 (False,True)  = False
myAnd1 (True ,False) = False
myAnd1 (True ,True)  = True
```

Wer sich etwas Schreibarbeit sparen will, kann Fälle mit dem gleichen Ergebnis zusammenfassen. Für Haskell macht das keinen Unterschied, aber für uns ist es wohl leichter lesbar. Der einzige Spezialfall `(True,True)` schreiben wir an erster Stelle, damit dieser zuerst geprüft wird. Da alle restlichen Fälle auf `False` abgebildet werden, können wir diese nun pauschal mit einem Wildcard Pattern abarbeiten, welche immer matched:

```
myAnd2 (True,True) = True
myAnd2 _           = False
```

Eine alternative Lösung mit einem Pattern Match weniger wäre:

```
myAnd3 (True,   y) = y
myAnd3 _           = False
```

Diese Variante nennt man auch nicht-strikt im zweiten Argument, da dieses nicht inspiziert wir: Der Ausdruck `myAnd2 (True,waldläufer)` terminiert, aber der Ausdruck `myAnd1 (True,waldläufer)` terminiert nicht (wobei `waldläufer x = waldläufer x`). Striktheit wird in der Vorlesung erst später behandelt.

b)
```
myAnd4 (x,y)
  | x, y      = True
  | otherwise = False
```

Alternative mit `case`-Ausdruck

```
myAnd5 xy = case xy of
  (x,y) | x && y    -> True
        | otherwise -> False
```

c)
```
myAnd6 (x,y) = if x then y else False
```

A1-4 *List Comprehension* 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

a) 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 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 nicht zu unterscheiden. Die Länge einer Liste berechnet man mit der Bibliotheksfunktionen `length :: [a] -> Int`

LÖSUNGSVORSCHLAG:

```
> length miaKombinationen
365
```

Mia stehen also 365 Kombinationen zur Verfügung; für den Code siehe Ende der Aufgabe.

b) Mia's Freundin, Ina die Informatikerin, hat jeweils 4 schwarze Klamotten von jedem Typ. Weiterhin hat sie 3 weiße und 3 fliederfarbene Tops, eine weiße Hose und ein paar fliederfarbene Schuhe. Pinke Klamotten hat sie keine. Wie viele Kombinationen stehen Ina zur Verfügung, wenn Sie jeden Tag mindestens ein schwarzes Kleidungsstück tragen will?

LÖSUNGSVORSCHLAG:

```
> length inaKombinationen
244
```

Ina stehen 244 Kombinationen zur Verfügung; für den Code siehe Ende der Aufgabe.

c) Wie groß ist die Wahrscheinlichkeit, dass an einem Tag Mia und Ina Kleidungsstücke in der gleichen Farbkombination tragen, wenn Sie sich jeden Tag zufällig gemäß der geschilderten Regeln anziehen?

Hinweis: Verwenden Sie zum Ausrechnen der Wahrscheinlichkeit folgende Funktion; in der Vorlesung wird die Konvertierung zwischen den Zahlentypen noch behandelt:

```
intDiv :: Int -> Int -> Double
intDiv z n = (fromIntegral z) / (fromIntegral n)
```

LÖSUNGSVORSCHLAG:

```
> chanceSame
5.227936222771166e-2
```

Also ungefähr $5,2\%$, d.h. durchschnittlich an mehr als einen Werktag pro Monat.

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]

-- Weitere Alternative:
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
-------------
miaKombinationen = [(t,h,s) | t<-miaTops, h<-miaHosen, s<-miaSchuhe, t/=h && (t==s || h==s)]
-------------
-- Teil b)
inaTops   = genCloth 4 3 0 3
inaHosen  = genCloth 4 1 0 0
inaSchuhe = genCloth 4 0 0 1

inaKombinationen = [(t,h,s)
  | t <- inaTops, h <- inaHosen, s <- inaSchuhe, t==schwarz || h==schwarz || s==schwarz
    -- t*h*s==0 -- Alternativer Hack, da schwarz == 0. Das Ausnutzen solcher Spezialfälle empfiehlt
    -- sich nicht, da eine Änderung der willkürlichen Konstanten Zeile 1-4 dann Korrektheit beeinflusst
  ]
-------------
-- Teil c)
chanceSame :: Double
chanceSame = anzahlGleich `intDiv` anzahlKombinationen
 where
    anzahlKombinationen = length [ True  | ina <- inaKombinationen, mia <- miaKombinationen]

    anzahlGleich = length [1 | ina <- inaKombinationen, mia <- miaKombinationen, ina == mia]
    -- Tripel kann man direkt vergleichen, aber zu Fuss wäre auch ok:
    -- anzahlGleich = length [1 | (lt,lh,ls) <- inaKombinationen,
    --                            (mt,mh,ms) <- miaKombinationen,
    --                            lt == mt, lh==mh, ls==ms ]

intDiv :: Int -> Int -> Double
intDiv z n = (fromIntegral z) / (fromIntegral n)
```

A1-5 ***Auswertung*** Berechnen Sie möglichst mit Papier und Bleistift den Wert, zu dem der gegebene Ausdruck jeweils auswertet. Überprüfen Sie Ihr Ergebnis mit GHCI.

a) `(4 == 5.0):[]`

LÖSUNGSVORSCHLAG:

`[False]`

b) `'c':'o':'o':'l':"!"`

LÖSUNGSVORSCHLAG:

`"cool!"`

c) `let mond="käse" in if mond=="käse" && 1==2 then False else 1+1==2`

LÖSUNGSVORSCHLAG:

`True`

d)
```
[ z |c <- "grotesk", c/='k', c/='r', c/='e' && c/='s',
                     let z = if c=='o' then 'u' else c]
```

LÖSUNGSVORSCHLAG:

`"gut"`

e) `[(a,b) | a<-[1..10], b<-[10..1], a/=b]`

LÖSUNGSVORSCHLAG:

`[]` *Zur Erinnerung:* Wenn man herunterzählen möchte, dann muss man die Schrittweite angeben. Die Liste `[10..1]` ist leer, die Liste `[10,9..1]` enthält dagegen 10 Zahlen.

f) `(\x->"nope!") [c | c <- "yes!"]`

LÖSUNGSVORSCHLAG:

`"nope!"` Der erste Teil definiert eine anonyme Funktion, welches Ihr Argument ignoriert und immer den String `"nope!"` zurückliefert. Der hintere Teil dieses Ausdrucks wertet wieder zu dem String `"yes!` aus.

H1-1 *List Comprehension II* (4 Punkte) (Geben Sie eine `.hs`-Datei als Lösung ab)
Robert der Rollenspieler möchte die Wahrscheinlichkeit berechnen, dass seine Spielfigur eine Fertigkeitsprüfung besteht, um die nächste Aufstufung seiner Spielfigur zu planen. In seinem Spiel wird eine Fertigkeitsprüfung wie folgt durchgeführt: Gegeben sind drei Attribute der Spielfigur und eine Schwierigkeit. Robert muss jeden der drei Attributswerte mit einem 20-seitigen Würfel (Zahlen von 1–20) unterwürfeln. Für jeden der drei Würfelwürfe wird die Differenz notiert, falls der Würfel über dem Attribut liegt und 0 sonst. Die Prüfung gelingt, wenn die Summe dieser abgeschnittenen Differenzen kleiner oder gleich dem Schwierigkeitswert ist. Zusätzlich gilt, dass die Prüfung immer gelingt, wenn mindestens zwei der drei Würfelwürfe eine 1 ergeben haben; aber die Prüfung misslingt auch schon, falls 2×20 fällt.

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 0.25 oder 2 Sekunden dauert, ist in seinem Anwendungsfall völlig egal.

a) Definieren Sie eine Funktion

```
erfolg :: (Int,Int,Int) -> Int -> (Int,Int,Int) -> Bool
```

welche als erstes Argument ein Tripel von Attributen, als zweites Argument einen Schwierigkeitswert und als drittes Argument das Ergebnis von drei Würfelwürfen bekommt, und entscheidet, ob eine Fertigkeitsprüfung gemäß der angegebenen Regeln erfolgreich war oder nicht. *Beispiel:*

```
> 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. Sie dürfen die Funktion `intDiv` aus Aufgabe A1-4 wiederverwenden. *Beispiel:*

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

Ein Fertigkeitsprüfung mit den Attributen 16, 13 und 8 und Schwierigkeit 5 wird also mit einer Chance von ungefähr 52% bestanden.

LÖSUNGSVORSCHLAG:

```
chance :: (Int,Int,Int) -> Int -> Double
chance abilities difficulty = anzahlErfolg `intDiv` anzahlMöglichkeiten
  where
    -- anzahlMöglichkeiten = length [1 | x<-[1..20], y<-[1..20], z<-[1..20] ]
    anzahlMöglichkeiten = 20 * 20 * 20
    anzahlErfolg = length [ 1 | x <- [1..20], y <- [1..20], z <- [1..20],
                                erfolg abilities difficulty (x,y,z)]

erfolg :: (Int,Int,Int) -> Int -> (Int,Int,Int) -> Bool
erfolg _          _          ( 1, 1, _) = True
erfolg _          _          ( 1, _, 1) = True
erfolg _          _          ( _, 1, 1) = True
erfolg _          _          (20,20, _) = False
erfolg _          _          (20, _,20) = False
erfolg _          _          ( _,20,20) = False
erfolg (a1,a2,a3) difficulty (w1,w2,w3) =
  difficulty >= (fehlpunkte a1 w1) + (fehlpunkte a2 w2) + (fehlpunkte a3 w3)

fehlpunkte :: Int -> Int -> Int   -- Anzahl benötigter Ausgleichpunkte
fehlpunkte skill würfel | skill >= würfel  = 0
                        | otherwise        = würfel - skill

intDiv :: Int -> Int -> Double
intDiv z n = (fromIntegral z) / (fromIntegral n)
```

H1-2 *Typen* (0 Punkte) (Keine Abgabe)
Welchen Typ haben folgenden Ausdrücke?

a) `['a','b','c']`

LÖSUNGSVORSCHLAG:

`String` oder `[Char]` (Typsynonym)

b) `('a','b','c')`

LÖSUNGSVORSCHLAG:

`(Char,Char,Char)`

c) `[(False,[1]),(True,[(2.0)])]`

LÖSUNGSVORSCHLAG:
`[(Bool, [Double])]`

d) `([True,True],('z','o','o'))`

LÖSUNGSVORSCHLAG:
`([Bool], (Char, Char, Char))`

e) `(\x -> ('a':x,False,([x])))`

LÖSUNGSVORSCHLAG:
`[Char] -> ([Char], Bool, [[Char]])`

f) `[(\x y-> (x*2,y-1)) m n | m <- [1..5], even m, n <- [6..10]]`

LÖSUNGSVORSCHLAG:
`[(Integer, Integer)]`

Hinweis: Kontrollieren Sie Ihre Antworten anschließend selbst mit GHCI! Auch wenn wir Typinferenz in der Vorlesung noch nicht behandelt haben, sollten Sie in der Lage sein, die meisten dieser einfachen Aufgaben bereits alleine mit Papier und Bleistift lösen zu können. Dies Aufgabe ist eigentlich auch einfacher als Aufgabe A1-5.

H1-3 *Rekursion* (2 Punkte) (Geben Sie eine `.hs`-Datei als Lösung ab)
In der Standardbibliothek gibt es eine Funktion `replicate :: Int -> a -> [a]`, welche einen gegebenen Wert wiederholt:

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

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

a) Implementieren Sie diese Funktion ohne Rekursion unter Verwendung von List-Comprehension!

LÖSUNGSVORSCHLAG:

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

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

LÖSUNGSVORSCHLAG:

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

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

In beiden Fällen dürfen Sie natürlich keine Funktionen der Standardbibliothek verwenden; abgesehen von den Grundoperationen `(==),(\=),(&&),(||),not,(+),(-),(*),(/),(:)`

Abgabe: Lösungen zu den Hausaufgaben können bis Dienstag, den 29.04.2013, 11:00 Uhr mit UniworX abgegeben werden.

Aufgrund des Klausurbonus müssen die Hausaufgaben von Ihnen alleine gelöst werden. Abschreiben bei den Hausaufgaben gilt als Betrug und kann zum Ausschluss von der Klausur zur Vorlesung führen.

Studenten, welche zugunsten eine Gruppenabgabe freiwillig auf den Klausurbonus aus einem Übungsblatt verzichten wollen, können die Einrichtung einer Gruppenabgabe per eMail beantragen.