Prof. Martin Hofmann, PhD
Dr. Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik
6. Mai 2013

3. Übung zur Vorlesung

Programmierung und Modellierung

A3-1 ***Wiederholung: Listen, Rekursion*** *Schnell und jeder für sich, egal ob am Computer oder mit Papier und Bleistift:* Implementieren Sie die Funktion `concat :: [[a]] -> [a]` aus der Standardbibliothek, welche die Elemente einer Liste von Listen miteinander verkettet. Verwenden Sie dazu Pattern-Matching auf Listen und Rekursion; und zur Vermeidung von Namenskonflikten einen frischen Funktionsnamen, z.B. `myConcat`.

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

> concat ["Hi ", "there", "!"]
"Hi there!"

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

A3-2 ***Abstiegsfunktion*** Zeigen Sie jeweils mithilfe einer geeigneten Abstiegsfunktion, dass jede der folgenden Funktionsdefinitionen für alle ganzen Zahlen terminiert!

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

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

A3-3 ***Induktion*** Betrachten Sie die folgende rekursive Funktion `foo :: (Integer,Integer) -> Integer`

```
foo (n,m)
  | n == 0 = m+1
  | n >  0 = foo (n-1, foo (n-1,m))
  | otherwise = error "First Argument for foo is not allowed to be negative."
```

Raten Sie zuerst durch Ausprobieren dieser Funktion eine (nicht-rekursive) mathematische Formel, welche den Wert von `foo (n,m)` für beliebige natürliche Zahlen $n, m \in \mathbb{N}$ berechnet.

Beweisen Sie anschließend mit Induktion, dass Sie zuvor richtig geraten haben!

Generelle Hinweise zur Hausaufgabenabgabe:

Wird zu einer Hausaufgabe die Abgabe einer .hs-Datei gefordert, so muss diese als einzelne Datei pro Aufgabe abgegeben werden (Teilaufgaben in einer gemeinsamen Datei). Die Abgabe kann mit 0 Punkten bewertet werden, falls diese Datei von GHC/GHCi mit einer Fehlermeldung abgelehnt wird! Korrigieren Sie also alle Syntaxfehler vor der Abgabe! Falls Sie dies nicht hinbringen, dann kommentieren Sie die entsprechenden Stellen mit Hinweisen aus und vervollständigen Sie Ihre Abgabe mit Aufrufen der Funktion `error :: String -> a`.
Beispiel:

```
foo _ [] = [] -- Fall ok!
foo _ [x] = error "A0-3b, 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 -}`

Wenn nichts anderes angegeben ist, dürfen Sie Code von Vorlesungsfolien und vorangegangenen Aufgaben wiederverwenden. Die Verwendung von Funktionen der Standardbibliothek ist tabu, abgesehen von den Grundoperationen wie `(:)`, `(++)`, `(>)`, `(<=)`, `div`, `mod`, etc.

H3-1 ***Wdh.: Listen, Rekursion*** **(0 Punkte)** (`.hs`-Datei als Lösung abgeben)
Implementieren Sie flott, lediglich mit Pattern-Matching und Rekursion:

a) die Infix-Funktion `(++) :: [a] -> [a] -> [a]` aus der Standardbibliothek selbst. Diese Funktion verkettet zwei einzelne Listen miteinander, also `[1,2,3] ++ [4,5] == [1,2,3,4,5]`. Zur Vermeidung von Namenskonflikten geben Sie Ihrer Version den Namen `myAppend :: [a] -> [a] -> [a]` in Präfix-Notation, also:

```
> myAppend [1,2,3] [4,5]
[1,2,3,4,5]
```

b) eine endrekursive Version von `reverse :: [a] -> [a]` welche die Reihenfolge einer Liste umkehrt.

H3-2 ***Abstiegsfunktion II*** **(4 Punkte)** (Abgabeformat: Text oder PDF)
Beweisen mithilfe einer geeigneten Abstiegsfunktion ausführlich, dass die folgend definierte Funktion für alle ganzen Zahlen terminiert:

```
barfoo :: Integer -> Integer
barfoo x
  | x >= 123       = x - 666
  | x <= 1, x >= 0 = 42
  | otherwise      = barfoo (x+1) * barfoo (x*x) * 3
```

H3-3 ***Benutzerdefinierte Datentypen*** **(4 Punkte)** (`.hs`-Datei als Lösung abgeben)

a) Gegeben sind folgende Definitionen:

```
data Brotzeit = Weisswurscht Int | Breze Int Brotzeit
              | Leberkas Double  | Radi Double Brotzeit
  deriving (Show)

bz_steffen = Breze 3 ( Radi 0.5 (Weisswurscht 2))
bz_martin  = Breze 1 ( Radi 0.02 ( Radi 0.01 ( Breze 1 ( Leberkas 1.6 ))))
```

Eine `Brotzeit` ist also entweder eine Anzahl Weisswürschte, oder eine Anzahl Brezen als Beilage einer anderen Brotzeit, oder ein Leberkas mit einem Gewicht in Kilogramm, oder ein Radi mit Gewicht in Kilogramm als Beilage einer anderen Brotzeit. `bz_steffen` beinhaltet zum Beispiel 3 Brezen, ein Radi mit 500g und zwei Weisswürschte.

Schreiben Sie eine Funktion `anzahl :: Brotzeit -> Int` welche die Summe der Objekte einer `Brotzeit` ausrechnet. Ein Radi und ein Leberkas zählen unabhängig vom Gewicht als ein Objekt. *Beispiele:*

```
> anzahl bz_steffen
6
> anzahl bz_martin
5
```

b)
```
data Korb = Korb { weisswürschte, brezen :: Int,
                   leberkas :: Double, radi :: (Int,Double) }
    deriving (Show)

leererKorb = Korb { weisswürschte=0, brezen=0, leberkas=0, radi=(0,0) }
tcs_korb = leererKorb { weisswürschte=2, brezen=3, leberkas=1.6, radi=(3,0.53) }
```

Ein `Korb` enthält also eine Anzahl Weisswürschte und eine Anzahl an Brezen, und einen Leberkäse mit einem angegebenen Gesamtgewicht in Kilogramm und eine Anzahl Radi mit einem Gesamtgewicht in Kilogramm.

Schreiben Sie eine Funktion `brotzeit2korb :: Brotzeit -> Korb` welche einen Wert des Typs `Brotzeit` in einen Wert des Typs `Korb` umrechnet, so dass äquivalente Lebensmittel enthalten sind. *Beispiele:*

```
> brotzeit2korb bz_steffen
Korb {weisswürschte = 2, brezen = 3, leberkas = 0.0, radi = (1,0.5)}
> brotzeit2korb bz_martin
Korb {weisswürschte = 0, brezen = 2, leberkas = 1.6, radi = (2,3.0e-2)}
```

Abgabe: Lösungen zu den Hausaufgaben können bis Dienstag, den 13.05.2013, 11:00 Uhr mit UniworX abgegeben werden. Die Hausaufgaben müssen von Ihnen alleine gelöst werden; Abschreiben wird als Betrug gewertet und ans Prüfungsamt gemeldet.