Prof. Martin Hofmann, PhD
Dr. Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik
5. Mai 2015

3. Übung zur Vorlesung

Programmierung und Modellierung

A3-1 ***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 gleichen Typ sind, essen Sie sie auf und füllen eine 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 Frage: Terminiert dieser Vorgang? Falls ja, wie viele Schmankerl verbleiben am Ende im Fass? Falls nein, wie entwickelt sich das Verhältnis von Weißwürsten zu Brezn? Begründen Sie Ihre Antwort!

A3-2 ***Abstiegsfunktion*** 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))
```

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

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

A3-3 ***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.

H3-1 ***Abstiegsfunktion II*** (3 Punkte; Abgabeformat: Text oder PDF)
Gegeben ist folgende Funktionsdefinition:

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

a) Warum ist die Funktion $f(x, y, z) := \max(x + y, 0)$ hier keine geeignete Abstiegsfunktion? Zeigen Sie dies durch Angabe eines Gegenbeispiels, also Werte für `x`,`y` und `z` 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 ***Benutzerdefinierte Listen*** (2 Punkte; Datei `H3-2.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 Assisstenten oder Tutor, falls Sie mit dieser polymorphen, rekursiven Infix-Definition noch Schwierigkeiten haben!

Viele Teilnehmer haben jedoch Probleme mit benutzerdefinierten Datentypen, weshalb auf Vorlesungsfolien 04-14 und 04-18 gezeigt wurde, wie man gewöhnliche Listen äquivalent "zu Fuss" deklarieren müsste, falls diese in Haskell nicht eingebaut wären:

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

Ihre Aufgabe: Schreiben Sie eine Funktion `verketten :: List a -> List a -> List a`, welche zwei Listen mit einander verkettet. Sie müssen also lediglich den oben angegeben Code zur Benutzung des Datentyps auf Folie 04-18 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))
```

H3-3 ***Benutzerdefinierte Datentypen*** (4 Punkte; Datei `H3-3.hs` als Lösung abgeben)
Gegeben sind folgende Datentypdeklarationen

```
data Vergleich = Schlechter | Gleich | Besser          deriving Show
data Spiel     = Brettspiel String Int | Bausatz Int   deriving Show
```

Ein Spiel ist entweder ein Brettspiel, beschrieben durch einen Namen und eine Anzahl an erlaubten Spielern; oder einen Bausatz mit einer gewissen Anzahl an Bauteilen.

Schreiben Sie eine Funktion `vergleiche :: Spiel -> Spiel -> Vergleich`, welche zwei gegebene Spiele vergleicht.

Die Spiele vergleichen wir nach der Anzahl der möglichen Mitspieler: Je mehr, desto besser. Die Anzahl der Mitspieler eines Bausatzes ergibt sich aus der Anzahl der Teile dividiert durch 50 (abgerundet). Zum Beispiel gelten zwei Bausätze mit 212 und 243 Teilen gelten also ebenfalls als `Gleich`, da beide 4 Spieler erlauben.

Hinweis: Verwenden Sie die Funktion `div` aus der Standardbibliothek zur abgeschnittenen Division zweier ganzer Zahlen. Wie immer dürfen Sie gerne Hilfsfunktionen definieren, falls es Ihnen hilft.

Abgabe: Lösungen zu den Hausaufgaben können bis Dienstag, den 12.05.2015, 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.

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 -}`