PHILIPPS-UNIVERSITÄT MARBURG
Fachbereich Mathematik und Informatik
Prof. Dr. R. Loogen

D-35032 Marburg
Hans Meerwein Straße
Lahnberge

19. November 2003

Übungen zur „Praktischen Informatik III“, WS 2003/04

Nr. 5, Besprechung bzw. Abgabe: 26. und 27. November in den Übungen

---

A. Mündliche Aufgaben

22. Listeninduktion

    Beweisen Sie die folgenden Aussagen für beliebige endliche Listen `xs, xs1, xs2` über einem Elementtyp `a` mittels Listeninduktion:

    (a) `xs ++ [] = xs`

    (b) `reverse (xs1 ++ xs2) = (reverse xs2) ++ (reverse xs1)`

23. Warteschlangen

    Zeigen Sie, dass die in der Vorlesung vorgestellte Implementierung eines abstrakten Datentyps `Queue a` als Paar von Listen

```
type Queue a = ([a],[a]) -- (ws,rs) Schreib-Ende >> Lese-Ende
```

(siehe Datei `queue.hs` auf der Vorlesungsseite) der folgenden Gleichung aus der Gleichungsspezifikation der Warteschlange genügt:

```
(isEmptyQueue q = False) -> (dequeue (enqueue q x) = enqueue (dequeue q) x)
```

Die Gleichheit von Warteschlangen sei dabei wie folgt festgelegt:

`(ws1,rs1) = (ws2,rs2)` gilt genau dann, wenn
`rs1 ++ (reverse ws1) = rs2 ++ (reverse ws2).`

B. Hausaufgaben

24. Programmtransformation **5 Punkte**

    Gegeben seien die folgenden Funktionen

```
my_unlines     :: [String] -> String
my_unlines css =  concat (addnewline ccs)

addnewline          :: [String] -> [String]
addnewline []       =  []
addnewline (cs:css) =  (cs ++ "\n") : addnewline ccs

concat          :: [[a]] -> [a]
concat []       =  []
concat (xs:xss) =  xs ++ concat xss
```

(a) Analysieren Sie den Zeitaufwand der Funktion `my_unlines`. / 1

(b) Entwickeln Sie eine zu `my_unlines` äquivalente Funktion, die effizienter arbeitet, d.h. mit weniger Schritten dasselbe Ergebnis bestimmt. / 2

(c) Zeigen Sie die Äquivalenz Ihrer Funktion zu `my_unlines`. / 2

25. Countdown Problem **7 Punkte**

Das *Countdown Problem* lautet wie folgt: Gegeben seien eine Folge positiver ganzer Zahlen, die Quellzahlen, und eine positive ganze Zahl, die Zielzahl. Gesucht wird ein arithmetischer Ausdruck, in dem jede der Quellzahlen höchstens einmal auftritt und der als Ergebnis die Zielzahl liefert. Als Operatoren können in dem Ausdruck $+$, $-$, $*$ und $/$ verwendet werden, wobei jedes Zwischenergebnis ebenfalls eine positive ganze Zahl sein muss. Zum Beispiel löst für die Quellzahlen $[1, 3, 7, 10, 25, 50]$ und die Zielzahl 765 der Ausdruck $(1 + 50) * (25 - 10)$ das Problem.

In dieser Aufgabe wird schrittweise ein Programm zur Lösung dieses Problems entwickelt. Eine Datei `countdown.hs` mit den im folgenden vorgegegenen Definitionen steht auf der Vorlesungsseite bereit.

```
data Op   = Add | Sub | Mul | Div      deriving (Show)
data Expr = Val Int | App Op Expr Expr deriving (Show)

apply         :: Op -> Int -> Int -> Int
apply Add x y =  x+y
apply Sub x y =  x-y
apply Mul x y =  x*y
apply Div x y =  x `div` y
```

(a) Schreiben Sie eine Funktion `valid :: Op -> Int -> Int -> Bool` zum Test, ob die Anwendung einer Operation auf zwei positive ganze Zahlen wieder eine solche ergibt. / 1

(b) Schreiben Sie eine Funktion `values :: Expr -> [Int]`, die die Liste aller in einem Ausdruck vorkommenden ganzen Zahlen bestimmt. / 1

Gegeben sei weiterhin die folgende Hilfsfunktion, die zu einem arithmetischen Ausdruck seinen Wert in einer einelementigen Liste bestimmt. Falls der Ausdruck keine positive ganze Zahl ergibt oder ein Zwischenergebnis keine positive ganze Zahl ist, liefert die Funktion die leere Liste als Resultat.

```
eval               :: Expr -> [Int]
eval (Val n)       =  [n | n>0]
eval (App op l r) =  [apply op x y | x <- eval l, y <- eval r, valid op x y]
```

(c) Schreiben Sie unter Verwendung von Funktionen, die in den Aufgaben 13 und 16 (siehe Blatt 3) definiert wurden, eine Funktion `subbags :: [a] -> [[a]]`, die die Liste aller Permutationen aller Teillisten einer gegebenen Liste bestimmt. / 2

(d) Definieren Sie eine Funktion `exprs :: [Int] -> [Expr]`, die zu einer Zahlenliste `ns` die Liste aller Ausdrücke `e` bestimmt, für die gilt `values e == ns`. / 3

Verwenden Sie hierzu einen Divide-and-Conquer-Ansatz, bei dem alle Möglichkeiten betrachtet werden, eine Zahlenliste mit mindestens zwei Elementen in zwei nicht-leere Teillisten zu zerlegen.

Die Lösungsfunktion für das Countdown-Problem kann mit diesen Hilfsfunktionen wie folgt angegeben werden:

```
solutions      :: [Int] -> Int -> [Expr]
solutions ns n =  [ e | ns' <- subbags ns, e <- exprs ns', eval e == [n]]
```