

Übungen zur „Praktischen Informatik III“, WS 2006/07

Nr. 6, Abgabe: 28. November 2006 vor der Vorlesung

Achtung: Ab dem 22.11.2006 findet die Vorlesung mittwochs von 11.30 Uhr bis 13.15 Uhr im Hörsaal IV auf den Lahnbergen statt.

15. Countdown Problem

12 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 soll ein Programm zur Lösung dieses Problems entwickelt werden. Das Programm soll nach entsprechenden Eingabeaufforderungen die Zahlenliste und die Zielzahl einlesen und anschließend eine erste Lösung des Countdown-Problems berechnen und ausgeben. Weitere Lösungen sollen nur nach Eingabe von $\backslash n$ (Returntaste) ausgegeben werden. Bei Betätigung einer anderen Eingabetaste oder, falls keine weiteren Lösungen existieren, soll das Programm stoppen.

Eine Datei `countdown.hs` mit den im folgenden vorgegebenen 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
```

Eine Lösung für das Countdown-Problem kann wie folgt angegeben werden:

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

wobei gilt:

- `subbags :: [a] -> [[a]]` bestimmt die Liste aller Permutationen aller Teillisten einer gegebenen Liste.
- `exprs :: [Int] -> [Expr]` liefert zu einer Zahlenliste `ns` die Liste aller Ausdrücke `e`, für die gilt `values e == ns`, wobei `values :: Expr -> [Int]` die Liste aller in einem Ausdruck vorkommenden ganzen Zahlen bestimmt.
- `eval :: Expr -> [Int]` bestimmt zu einem arithmetischen Ausdruck seinen Wert in einer einelementigen Liste. Falls der Ausdruck keine positive ganze Zahl ergibt oder ein Zwischenergebnis keine positive ganze Zahl ist, liefert die Funktion die leere Liste als Resultat.