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

D-35032 Marburg
Hans Meerwein Straße
Lahnberge

22. Oktober 2002

Übungen zu „Parallelität in funktionalen Sprachen“

Nr. 1, Abgabe: 29. Oktober in der Vorlesung

Hinweise: Die Lösungen sollten grundsätzlich schriftlich, Programme zusätzlich auf Diskette oder per E-Mail an `eden@mathematik.uni-marburg.de` abgegeben werden.
Die Abgabe ist in Gruppen bis zu zwei Personen erlaubt.

---

1. Parallelisierbarkeit eines Beispielprogramms — 20 Punkte

   Gegeben sei das folgende (sequentielle) Haskell-Programm.

   (a) Untersuchen Sie das Haskell-Programm und geben sie an, wozu das Programm dient. Vervollständigen Sie die Kommentare im Quelltext und die fehlenden Typdeklarationen.

   (b) Finden Sie voneinander unabhängige Programmteile und überlegen Sie sich eine Parallelisierung des Programms, bei der möglichst viele Teile gleichzeitig ausgeführt werden. Gehen Sie zunächst davon aus, dass Ihnen zur Ausführung ein Netz mit beliebig vielen Rechnern zur Verfügung steht.

   Können Sie allgemeine Regeln zum Auffinden von Parallelität in funktionalen Programmen aufstellen?

```
main :: IO ()
main =  putStr (concat (map find hidden))

-- find:
--------------------------------------------------------------------------
find :: [Char] -> String
find word = word ++ " " ++ (concat dirs) ++ "\n"
  where dirs = map snd (forw ++ back)

        forw = filter (any (contains word) . fst)
               [(r,"right "), (d,"down "), (dl,"downleft "), (ul,"upleft ")]
        back = filter (any (contains drow) . fst)
               [(r,"left "), (d,"up "), (dl,"upright "), (ul,"downright ")]
        drow = reverse word

        r  = grid
        d  = transpose grid
        dl = diagonals grid
        ul = diagonals (reverse grid)
```

```
-- transpose:
--------------------------------------------------------------------------
transpose :: [[a]] -> [[a]]
transpose []             = []
transpose ([] : xss)     = transpose xss
transpose ((x:xs) : xss) = (x : [h | (h:t) <- xss]) :
                           transpose (xs : [t | (h:t) <- xss])

-- diagonals:
-- zipinit:
--------------------------------------------------------------------------
diagonals [r]    = map (:[]) r
diagonals (r:rs) = zipinit r ([]:diagonals rs)

zipinit []     ys     = ys
zipinit (x:xs) (y:ys) = (x : y) : zipinit xs ys

-- contains:
-- prefix:
-- suffixes:
--------------------------------------------------------------------------
contains xs ys = any (prefix xs) (suffixes ys)

suffixes [] = []
suffixes xs = xs : suffixes (tail xs)

prefix []     ys     = True
prefix xs     []     = False
prefix (x:xs) (y:ys) = x == y && prefix xs ys

--------------------------------------------------------------------------
grid = [['Y', 'I', 'O', 'M', 'R', 'E', 'S', 'K', 'S', 'T'],
        ['A', 'E', 'H', 'Y', 'G', 'E', 'H', 'E', 'D', 'W'],
        ['Z', 'F', 'I', 'A', 'C', 'N', 'I', 'T', 'I', 'A'],
        ['N', 'T', 'O', 'C', 'O', 'M', 'V', 'O', 'O', 'R'],
        ['E', 'R', 'D', 'L', 'O', 'C', 'E', 'N', 'S', 'M'],
        ['Z', 'O', 'U', 'R', 'P', 'S', 'R', 'N', 'D', 'A'],
        ['O', 'Y', 'A', 'S', 'M', 'O', 'Y', 'E', 'D', 'L'],
        ['R', 'N', 'D', 'E', 'N', 'L', 'O', 'A', 'I', 'T'],
        ['F', 'I', 'W', 'I', 'N', 'T', 'E', 'R', 'R', 'C'],
        ['F', 'E', 'Z', 'E', 'E', 'R', 'F', 'T', 'F', 'I'],
        ['I', 'I', 'D', 'T', 'P', 'H', 'U', 'B', 'R', 'L'],
        ['C', 'N', 'O', 'H', 'S', 'G', 'E', 'I', 'O', 'N'],
        ['E', 'G', 'M', 'O', 'P', 'S', 'T', 'A', 'S', 'O'],
        ['T', 'G', 'F', 'F', 'C', 'I', 'S', 'H', 'T', 'H'],
        ['O', 'T', 'B', 'C', 'S', 'S', 'N', 'O', 'W', 'I']]

hidden = ["COSY", "SOFT", "WINTER", "SHIVER", "FROZEN", "SNOW",
          "WARM", "HEAT", "COLD", "FREEZE", "FROST", "ICE" ]
```