8. Aufgabenblatt zu Funktionale Programmierung vom 08.12.2009. Fällig: 16.12.2009 (wg. Feiertag) / 12.01.2010 (jeweils 15:00 Uhr)

Themen: *Knobeleien*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe8.hs` ablegen. Sie sollen für die Lösung dieses Aufgabenblatts also ein "gewöhnliches" Haskell-Skript schreiben. Versehen Sie wieder wie auf den bisherigen Aufgabenblättern alle Funktionen, die Sie zur Lösung brauchen, mit ihren Typdeklarationen und kommentieren Sie Ihre Programme aussagekräftig. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

Im einzelnen sollen Sie die im folgenden beschriebenen Problemstellungen bearbeiten.

1. Ein Schachbrett kann durch eine $8 * 8$-Liste des Datentyps `Chessboard` modelliert werden,

   ```
   type Chessboard = [[Char]]
   ```

   wobei jedes Element der Listenliste eine Zeile des Schachbretts modelliert. Unbesetzte Schachbrettfelder sind mit dem Leerzeichen ' ' besetzt, mit einer Dame besetzte Felder sind mit dem Zeichen 'B' besetzt. Die Zeilen und Spalten des Schachbretts werden dabei von 1 bis 8 durchnummeriert.

   Auf einem Schachbrett können bis zu 8 Damen so positioniert werden, dass keine Dame eine andere Dame bedroht.

   (a) Ein Wert b vom Typ `Chessboard` ist *gültiges Schachbrett*, wenn b von der Größe $8 * 8$ ist und alle Felder entweder unbesetzt oder mit einer Dame besetzt sind.

   Schreiben Sie eine Wahrheitswertfunktion `isValidChessboard` mit der Signatur `isValidChessboard :: Chessboard -> Bool`, die überprüft, ob ein Wert vom Typ `Chessboard` ein gültiges Schachbrett ist.

   (b) Schreiben Sie eine Haskell-Rechenvorschrift `threatens` mit der Signatur `threatens :: Chessboard -> (Int,Int) -> [(Int,Int)]`, die angewendet auf ein gültiges Schachbrett die Koordinaten aller auf dem Schachbrett positionierten Damen ausgibt, die von einer Dame auf dem Feld (p, q) bedroht werden, wobei (p, q) durch den Wert des zweiten Arguments gegeben ist. Die Ergebnisliste soll dann lexikographisch aufsteigend sortiert sein. Wird `threatens` auf ein nicht-gültiges Schachbrett angewendet oder ist die Position (p, q) nicht innerhalb des Schachbretts, so ist das Ergebnis von `threatens` die leere Liste.

   (c) Schreiben Sie eine Wahrheitswertfunktion `noThreats` mit der Signatur `noThreats :: Chessboard -> Bool`, die angewendet auf ein gültiges Schachbrett feststellt, ob keine der Damen eine andere Dame auf dem Schachbrett bedroht. In diesem Fall liefert die Funktionsanwendung den Wert `True` zurück, in allen anderen Fällen, insbesondere bei Anwendung auf ein nicht-gültiges Schachbrett den Wert `False`.

   (d) Schreiben Sie eine Haskell-Rechenvorschrift `fullBoard` mit der Signatur `fullBoard :: [(Int,Int)] -> Chessboard`, die ein gültiges mit 8 sich wechselweise nicht bedrohenden Damen besetztes Schachbrett zurückliefert, wenn dies möglich ist, das leere Schachbrett sonst. Dabei soll auf dem Ergebnisschachbrett auf den ersten bis zu 8 gültigen Positionen aus der Argumentliste der Funktion eine Dame stehen (nicht-gültige Positionen in der Argumentliste werden überlesen; ebenso über 8 hinausgehende Positionen).

2. In vielen Tageszeitungen findet sich heute neben einem Kreuzworträtsel auch ein $3 * 3$-Sudoku als Rätselaufgabe ($3*3$ bezieht sich hier auf die Grösse der Unterquadrate des Sudokus. Ein $3*3$-Sudoku besteht also aus $3 * 3 = 9$ Unterquadraten mit je $3 * 3 = 9$-Feldern, insgesamt also aus 3*3*3*3=81 Feldern.) Analog zu $3 * 3$-Sudokus lassen sich generell $n * n$-Sudokus definieren. In dieser Aufgabe betrachten wir $n * n$-Sudokus mit $1 \leq n \leq 3$, die durch geeignet dimensionierte Werte des Typs

```
type Sudoku = [[Int]]
```

modelliert werden können.

   (a) Ein $n * n$-Sudoku, $1 \leq n \leq 3$, heißt $n * n$-*Anfangs-Sudoku*, wenn in jeder Zeile und in jeder Spalte und in jedem Unterquadrat die Zahlen von 1 bis $n*n$ höchstens einmal vorkommen und alle anderen Felder mit 0 besetzt sind. Es heißt $n * n$-*Sudoku*, wenn es $n * n$-*Anfangs-Sudoku* ohne Nullen ist.

   Schreiben Sie eine Wahrheitswertfunktion `isValidStart` mit der Signatur `isValidStart :: Int -> Sudoku -> Bool`, die überprüft, ob ein Wert vom Typ Sudoku ein gültiges $n * n$-Anfangs-Sudoku, $1 \leq n \leq 3$, ist, wobei der Wert für n der Funktion `isValidStart` als erstes Argument übergeben wird.

   (b) Schreiben Sie eine Wahrheitswertfunktion `isValid` mit der Signatur `isValid :: Int -> Sudoku -> Bool`, die überprüft, ob ein Wert vom Typ Sudoku ein gültiges $n * n$-Sudoku, $1 \leq n \leq 3$, ist, wobei der Wert für n der Funktion `isValidStart` als erstes Argument übergeben wird.

   (c) Schreiben Sie eine Rechenvorschrift `solve :: Int -> Sudoku -> Sudoko`, die angewendet auf ein $n * n$-Anfangs-Sudoku, $1 \leq n \leq 3$, wobei der Wert für n der Funktion `solve` als erstes Argument übergeben wird, ein Sudoku zurückgibt, wenn möglich (das Sudoku muss durch das Anfangs-Sudoku nicht eindeutig bestimmt sein); ansonsten das ausschließlich mit Nullen besetzte $n * n$-Anfangs-Sudoku.