6. Aufgabenblatt zu Funktionale Programmierung vom 26.11.2014.
Fällig: 03.12.2014 / 10.12.2014 (jeweils 15:00 Uhr)
Themen: *Typen, Typklassen, Polymorphie und Überladung*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe6.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.

Wir betrachten den Datentyp `Matrix` über positiven rationalen Zahlen mit

```
data Nat         = Z | S Nat
type PosRat      = (Nat,Nat)
type Skalar      = PosRat
type ProtoMatrix = [[Skalar]]
newtype Matrix   = M [[Skalar]]
type Fuellwert   = Integer
```

wobei für die Typen `Nat` und `PosRat` dieselben Bezeichnungen und Eigenschaften wie für Aufgabenblatt 4 gelten.

Ein Wert $M\ l$ vom Typ `Matrix` stellt eine $p \times q$-Matrix dar, p, q natürliche Zahlen, $p, q > 0$, wenn l eine Liste von p Listen der Länge q ist. Für $p = 1$ sprechen wir von $M\ l$ als Zeilenvektor, für $q = 1$ von einem Spaltenvektor. Für $p = q = 1$ sprechen wir von einer unechten Matrix.

1. Schreiben Sie eine Haskell-Rechenvorschrift

   ```
   mkMatrix :: ProtoMatrix -> Fuellwert -> Matrix
   ```

   Angewendet auf eine Protomatrix pm und einen Füllwert i liefert `mkMatrix` eine $p \times q$-Matrix m, wobei p durch die Länge von pm und q durch das Maximum max von 1 und der größten Länge g eines Elementes von pm gegeben ist. Elemente von pm, die kürzer sind als max, werden beim Übergang von pm zu m am Ende mit dem Füllwert $|i|$ dargestellt als `Nat`-Wertpaar aufgefüllt.

In der Folge gehen wir davon aus, dass Argumente des Matrixkonstruktors M stets rechteckig sind, d.h. Werte von $p \times q$-Matrizen repräsentieren.

2. Machen Sie den Datentyp `Matrix` zu einer Instanz der Typklasse `Show`. Werte des Datentyps `Matrix` sollen dabei in Form von Listen von Listen ausgegeben werden, deren Elemente in kanonischer (d.h. gekürzter) Form als Oktogonalzahlen ohne führende Nullen dargestellt sind, d.h. als 8-adische Zahlen ohne führende Nullen über den Ziffern Eins, Zwei, Drei, Vier, Fünf, Sechs, Sieben und Null.

   ```
   data OktoZiffern = E | Zw | D | V | F | Se | Si | N
   type OktoZahlen  = [OktoZiffern]
   ```

   Speziell gilt, dass die natürliche Zahl 0 die Oktozahldarstellung `[N]` hat.

3. Machen Sie den Datentyp `Matrix` zu einer Instanz der Typklasse `Eq`. Zwei Matrizen sind genau dann gleich, wenn sie dieselbe Dimension besitzen und jeweils an der gleichen Position stehende Matrixelemente wertgleich sind (beachte: wertgleich $\neq$ strukturgleich).

4. Wir führen die Typklasse

   ```
   class (Eq a) => OrdMat a where
     lsm, lem, grm, gem :: a -> a -> Bool
     cmpm :: a -> a -> OrderingMat
   ```

   ein mit

```
data OrderingMat = EQM | LTM | GTM | INC deriving (Eq,Show)
```

Machen Sie den Datentyp `Matrix` zu einer Instanz der Typklasse `OrdMat`. Dabei ist die Bedeutung der Relatoren `(lsm)`, `(lem)`, `(grm)` und `(gem)` durch die Relationen echt kleiner, kleiner oder gleich, echt größer und größer oder gleich auf Matrizen gegeben, stets im Wert-, nicht im Struktursinn. Zwei Matrizen m_1 und m_2 stehen dabei in der Relation echt kleiner genau dann, wenn sie dieselbe Dimension haben und jedes Matrixelement aus m_1 echt wertkleiner ist als das entsprechende Matrixelement aus m_2. Analog ist die Bedeutung der anderen drei Relatoren definiert. Die Abbildung `cmpm` vergleicht den Wert zweier Matrizen m_1 und m_2. Sind m_1 und m_2 im Sinn von Teilaufgabe 3 gleich, so liefert `cmpm` angewendet auf m_1 und m_2 den Wert `EQM`. Ist m_1 echt kleiner als m_2 oder ist m_1 echt größer als m_2 im Sinne von Teilaufgabe 4, so liefert `cmpm` angewendet auf m_1 und m_2 den Wert `LTM` bzw. den Wert `GTM`. Ansonsten liefert `cmpm` angewendet auf m_1 und m_2 den Wert `INC`, dessen Bezeichnung an "incomparable" erinnert.

5. Wir führen die Typklasse

   ```
   class (Eq a) => ArithMat a where
     addm, multm :: a -> a -> a
   ```

   ein. Dabei stehen die Operatoren `addm` und `multm` für die Addition und Multiplikation passend dimensionierter Matrizen. Die Summe und das Produkt zweier passend dimensionierter Matrizen ergibt sich dabei in der aus der Mathematik bekannten Weise. Die Summenmatrix zweier gleichdimensionierter Matrizen ist eine Matrix derselben Dimension, deren Elemente Summe der entsprechenden Elemente der Summanden sind. Die Produktmatrix einer $p \times q$-Matrix m_1 mit einer $q \times r$-Matrix m_2 ist eine $p \times r$-Matrix, deren Element an der Stelle (i, j) sich als Vektorprodukt des i-ten Zeilenvektors von m_1 und des j-ten Spaltenvektors von m_2 ergibt. Stets gilt, dass alle Elemente der Summen- bzw. Produktmatrix zweier Matrizen kanonisch, d.h. gekürzt, dargestellt sind. Sind die Matrizenargumente nicht passend dimensioniert für eine Addition oder Multiplikation, so wird als Resultat der Summe bzw. Multiplikation der (ungültige) Matrixwert `M [[(Z,Z)]]` geliefert.

Ergänzen Sie ggf. erforderliche weitere Instanzbildungen für Typen und Klassen; mit `deriving`-Klauseln, wo möglich, mit `instance`-Deklarationen, wo nötig.

Wichtig: Wenn Sie einzelne Rechenvorschriften aus früheren Lösungen für dieses oder spätere Aufgabenblätter wieder verwenden möchten, so kopieren Sie diese in die neue Abgabedatei ein. Ein `import` schlägt für die Auswertung durch das Abgabeskript fehl, weil Ihre frühere Lösung, aus der importiert wird, nicht mit abgesammelt wird. Deshalb: Kopieren statt importieren zur Wiederwendung!

Haskell Live

Der nächste *Haskell Live*-Termin ist am Freitag, den 28.11.2014.