Eine Sache lernt man, indem man sie macht.
Cesare Pavese (1908-1950)
italien. Schriftsteller

Für das Können gibt es nur einen Beweis, das Tun.
Marie von Ebner-Eschenbach (1830-1916)
österr. Schriftstellerin

6. Angabe zu Funktionale Programmierung von Fr, 19.11.2021.
Erstabgabe: Fr, 26.11.2021, 12:00 Uhr
Zweitabgabe: Siehe „Hinweise zu Org. u. Ablauf der Übung" (TUWEL-Kurs)
Themen: *Funktionen höherer Ordnung, Funktionen als Datenstrukturen, Rechnen mit Funktionen als Argument und Resultat, Typklassen, Überladung, Polymorphie, Feldsyntax*
Stoffumfang: *Kapitel 1 bis Kapitel 11, besonders Kapitel 4, 10 und 11.*

- **Teil A, programmiertechnische Aufgaben:** Besprechung am ersten Übungsgruppentermin, der auf die *Zweitabgabe* der programmiertechnischen Aufgaben folgt.
- Teil B, Papier- und Bleistiftaufgaben: Entfallen auf Angaben 5 bis 7.

Wichtig

1. Befolgen Sie die Anweisungen aus den 'Lies-mich'-Dateien (s. TUWEL-Kurs) zu den Angaben sorgfältig, um ein reibungsloses Zusammenspiel mit dem Testsystem sicherzustellen. Bei Fragen dazu, stellen Sie diese bitte im TUWEL-Forum zur LVA.
2. Erweitern Sie für die für diese Angabe zu schreibenden Rechenvorschriften die zur Verfügung gestellte Rahmendatei

   `Angabe6.hs`

   und legen Sie sie für die Abgabe auf oberstem Niveau in Ihrem *home*-Verzeichnis ab. Achten Sie darauf, dass "Gruppe" Leserechte für diese Datei hat. Wenn nicht, setzen Sie diese Leserechte mittels `chmod g+r Angabe6.hs`.

   Löschen Sie keinesfalls eine Deklaration aus der Rahmendatei! Auch dann nicht, wenn Sie einige dieser Deklarationen nicht oder nicht vollständig implementieren wollen. Löschen Sie auch nicht die für das Testsystem erforderliche Modul-Anweisung `module Angabe6 where` am Anfang der Rahmendatei.
3. Der Name der Abgabedatei ist für Erst- und Zweitabgabe ident!
4. Benutzen Sie keine selbstdefinierten Module! Wenn Sie (für spätere Angaben) einzelne Rechenvorschriften früherer Lösungen wiederverwenden möchten, kopieren Sie diese bitte in die neue Abgabedatei ein. Eine `import`-Anweisung für selbstdefinierte Module schlägt für die Auswertung durch das Abgabesystem fehl, weil Ihre Modul-Datei, aus der importiert werden soll, vom Testsystem nicht mit abgesammelt wird.
5. Ihre Programmierlösungen werden stets auf der Maschine `g0` mit der dort installierten Version von GHCi überprüft. Stellen Sie deshalb sicher, dass sich Ihre Programme (auch) auf der `g0` unter GHCi so verhalten, wie von Ihnen gewünscht.
6. Überzeugen Sie sich bei jeder Abgabe davon! Das gilt besonders, wenn Sie für die Entwicklung Ihrer Haskell-Programme mit einer anderen Maschine, einer anderen GHCi-Version oder/und einem anderen Werkzeug wie etwa Hugs arbeiten!

A Programmiertechnische Aufgaben (beurteilt, max. 100 Punkte)

Erweitern Sie zur Lösung der programmiertechnischen Aufgaben die Rahmendatei `Angabe6.hs`. Kommentieren Sie die Rechenvorschriften in Ihrem Programm zweckmäßig, aussagekräftig und problemangemessen. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Wertvereinbarungen für konstante Werte (z.B. `pi = 3.14 :: Float`). Versehen Sie alle Funktionen, die Sie zur Lösung der Aufgaben benötigen, mit ihren Typdeklarationen, d.h. geben Sie stets deren syntaktische Signatur (kurz: Signatur), explizit an.

Wir betrachten Matrizen über ganzen Zahlen, genau wie auf Angabe 3. Dieses Mal werden wir Matrizen allerdings nicht als Listen von Listen modellieren, sondern als Funktionen. Wir werden **Funktionen als Datenstruktur für Matrizen** verwenden! Rechnen mit Matrizen (addieren, subtrahieren, multiplizieren, etc.) wird damit in natürlicher Weise zum **Rechnen mit Funktionen!** Alles andere bleibt gleich!

1. *Matrix:* Eine ganzzahlige Matrix M vom Typ (m, n) ist ein nichtleeres zweidimensionales Schema ganzer Zahlen $a_{ij} \in \mathbb{Z}$ mit m Zeilen und n Spalten, $m, n \in I\!N_1$:

$$M = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \multicolumn{4}{c}{\dotfill} \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

   Matrizen vom Typ $(1, 1)$ identifizieren wir mit ihrem einzigen Eintrag, ein sog. *Skalar*.

2. *Matrixgleichheit:* Zwei Matrizen M, N sind *gleich* gdw. sie sind typgleich und ihre Einträge stimmen positionsweise überein, *ungleich* sonst.

3. *Matrixaddition:* Die Summe zweier typgleicher Matrizen M, N vom Typ (m, n) ist die typgleiche Matrix S:

$$M + N =_{df} \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \multicolumn{4}{c}{\dotfill} \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \multicolumn{4}{c}{\dotfill} \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{pmatrix} = S$$

   definiert durch:

$$S =_{df} \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \multicolumn{4}{c}{\dotfill} \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{pmatrix}$$

   mit

$$c_{ij} =_{df} a_{ij} + b_{ij},\ \ i = 1, \ldots, m,\ \ j = 1, \ldots, n$$

   Sind M, N typverschieden, sind Addition und Summe von M und N nicht definiert.

4. *Matrixsubtraktion:* Die Differenz zweier typgleicher Matrizen M, N vom Typ (m, n) ist die typgleiche Matrix D:

$$M - N =_{df} \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \multicolumn{4}{c}{\dotfill} \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} - \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \multicolumn{4}{c}{\dotfill} \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{pmatrix} = D$$

definiert durch:

$$D =_{df} \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \multicolumn{4}{c}{\dotfill} \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{pmatrix}$$

mit

$$c_{ij} =_{df} a_{ij} - b_{ij},\ i = 1, \ldots, m,\ j = 1, \ldots, n$$

Sind M, N typverschieden, sind Subtraktion und Differenz von M und N nicht definiert.

5. *Matrixmultiplikation:* Das Produkt zweier Matrizen M, N vom Typ (m, n) und (n, p) ist die Matrix P vom Typ (m, p):

$$M \cdot N =_{df} \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \multicolumn{4}{c}{\dotfill} \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \multicolumn{4}{c}{\dotfill} \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix} = P$$

definiert durch:

$$P =_{df} \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \multicolumn{4}{c}{\dotfill} \\ c_{m1} & c_{n2} & \cdots & c_{mp} \end{pmatrix}$$

mit

$$c_{qr} =_{df} \Sigma_{j=1}^{n} a_{qj} b_{jr},\ q = 1, \ldots, m,\ r = 1, \ldots, p$$

Sind M, N nicht von multiplikativ zueinander passenden Typen, sind Multiplikation und Produkt von M und N nicht definiert.

6. *Produkt mit einem Skalar:* Das Produkt einer Matrix M vom Typ (m, n) und einer ganzen Zahl $s \in \mathbb{Z}$ (ein sog. *Skalar*) ist die zu M typgleiche Matrix SP:

$$s \cdot M =_{df} s \cdot \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \multicolumn{4}{c}{\dotfill} \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} = SP = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \multicolumn{4}{c}{\dotfill} \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \cdot s \;{}_{df}\!= M \cdot s$$

definiert durch:

$$SP =_{df} \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \multicolumn{4}{c}{\dotfill} \\ c_{m1} & c_{n2} & \cdots & c_{mn} \end{pmatrix}$$

mit

$$c_{ij} =_{df} s \cdot a_{ij},\ i = 1, \ldots, m,\ j = 1, \ldots, n$$

Wir modellieren Matrizen in der Folge als Funktionen eines zweidimensionalen Definitionsbereichs entsprechend des Typs der modellierten Matrix in ganze Zahlen:

```
type Nat0        = Int
type Nat1        = Int
type Zeilenzahl  = Nat1
type Spaltenzahl = Nat1
type Zeile       = Nat1
type Spalte      = Nat1
type Skalar      = Int
type Matrixtyp   = (Zeilenzahl,Spaltenzahl)
type Matrixfkt   = Zeile -> Spalte -> Skalar  -- ausschliessl. total def. Abb.!

-- Matrizenwerte als Typ und funktionale Darstellung
data MatrixF = Mf { mtyp :: Matrixtyp, mf :: Matrixfkt }
```

`MatrixF`-Werte mit Typ `(0,0)` stellen keine gültige Matrix dar; einen dieser Werte verwenden wir deshalb, wo sinnvoll und nötig, als fehleranzeigenden Wert, und zwar:

```
fehler = Mf (0,0) (\_ _ -> 0) :: MatrixF
```

A.1 Machen Sie den Typ `MatrixF` zu einer Instanz der Typklasse `Show`. Implementieren Sie dazu die Funktion `show` entsprechend der in folgenden Beispielen gezeigten Weise:

```
-- m1 bis m4 sind gleichwertig und entsprechen M [[1,2],[3,4]] von Angabe 3
m1 = Mf (2,2) (\z s -> if z == 1 then s else z+s) :: MatrixF
m2 = Mf (2,2) (\z s -> s + ((z-1)*(snd (2,2)))) :: MatrixF
m3 = Mf (2,2) (\z s -> s + ((z-1)*(snd (mtyp m2)))) :: MatrixF
m4 = Mf (2,2) (\z s -> if z == 1 then (succ (fib (s-1)))
                                 else ((+) z (binom z (s-1)))) :: MatrixF

-- m5, m6 sind gleichwertig und entsprechen M [[1,2],[3,4],[5,6]] von Angabe 3
m5 = Mf (3,2) (\z s -> if z == 1 then s
                                 else if z == 2 then z+s
                                                else succ (z+s)) :: MatrixF
m6 = Mf (3,2) (\z s -> s + ((z-1)*(snd (mtyp m5)))) :: MatrixF

 -- m7, m8, alle anderen Matrixwerte mit Typ (0,0) entsprechen M [] v. Angabe 3
m7 = Mf (0,0) (\_ _ -> 0)   :: MatrixF
m8 = Mf (0,0) (\z s -> z+s) :: MatrixF

-- Für m4 verwendete Hilfsfunktionen
fib :: Nat0 -> Nat0
fib 0 = 0
fib 1 = 1
fib n = fib (n-2) + fib (n-1)

binom :: Nat0 -> Nat0 -> Nat1
binom n k
 | n==0 || n==k = 1
 | True         = binom (n-1) (k-1) + binom (n-1) k
```

```
-- Gewünschte Matrixdarstellungen als Zeichenreihen (ident zu Angabe 3)
show m1 ->> "([1,2] [3,4])"
show m2 ->> "([1,2] [3,4])"
show m3 ->> "([1,2] [3,4])"
show m4 ->> "([1,2] [3,4])"
show m5 ->> "([1,2] [3,4] [5,6])"
show m6 ->> "([1,2] [3,4] [5,6])"
show m7 ->> "()"
show m8 ->> "()"
```

A.2 Implementieren Sie die Haskell-Rechenvorschrift `matrixtyp`, die überprüft, ob das Argument eine Matrix ist und in diesem Fall ihren Typ bestimmt.

```
matrixtyp :: MatrixF -> Maybe Matrixtyp
```

Angewendet auf einen `MatrixF`-Wert, liefert `matrixtyp` den Wert `(Just (m,n))`, wenn das Argument eine Matrix vom Typ `(m,n)` repräsentiert; ansonsten den Wert `Nothing`.

A.3 **Ohne Abgabe, ohne Beurteilung:** Ihre Funktion `matrixtyp` liefert (hoffentlich) wesentlich seltener den Wert `Nothing` als die namensgleiche Funktion von Angabe 3 den entsprechenden Wert `KeineMatrix`. Warum ist das so? Warum ist die Matrixprüfung für `MatrixF`-Werte zudem noch viel einfacher als für `Matrix`-Werte?

Bei der Instanzbildung von `MatrixF` für `Show` haben wir ausgenutzt, dass die Funktionswerte von `MatrixF`-Werten nur auf einem endlichen Ausschnitt ihres konzeptuell nicht endlichen Definitionsbereichs (wenn wir von der größenmäßigen Implementierungsbeschränkung für `Int`-Wertr bzw. der Maschinenspeicherbeschränkung beim Übergang zu `Integer`-Werten absehen) betrachtet werden müssen, der durch den Matrixtyp bestimmt ist. Diese Beobachtung und sie auszunutzen, ist auch für die folgenden Instanzbildungen essentiell!

A.4 Machen Sie den Typ `MatrixF` zu einer Instanz der Typklasse `Eq`. Implementieren Sie die Bedeutung der Relatoren `(==)` und `(/=)` dabei so, dass sie ihre Argumente auf die oben eingeführte Matrixgleichheit bzw. -ungleichheit prüfen. Sind eine oder beide Argumente keine Matrizendarstellungen, so sollen die Gleichheits- und Ungleichheitsprüfung mit dem Aufruf der Fehlerfunktion `error "Gleichheit undefiniert"` bzw. `error "Ungleichheit undefiniert"` beendet werden.

A.5 Machen Sie den Typ `MatrixF` zu einer Instanz der Typklasse `Num`. Implementieren Sie dabei die Operatoren `(+)`, `(-)`, `(*)` als Matrixaddition, -subtraktion und -multiplikation. Für Argumente, für die die Operationen nicht definiert sind, sollen sie den fehleranzeigenden Wert `fehler` liefern. Weiters soll gelten: Angewendet auf einen `MatrixF`-Wert M bzw. eine ganze Zahl z, liefert

(a) `negate` das Produkt mit einem Skalar von -1 und M, wenn M eine Matrix darstellt; den Fehlerwert `fehler` sonst.

(b) `abs` die Matrix M', die aus M entsteht, indem jedes Element von M durch seinen Absolutbetrag ersetzt wird, wenn M eine Matrix darstellt; den Fehlerwert `fehler` sonst.

(c) `signum` den Wert 1 (in Form einer `MatrixF`-Wertdarstellung), wenn alle Elemente von M echt positiv sind; den Wert 0 (in Form einer `MatrixF`-Wertdarstellung), wenn alle Elemente von M null sind; den Wert -1 (in Form einer `MatrixF`-Wertdarstellung), wenn alle Elemente von M echt negativ sind und M jeweils eine Matrix darstellt. Ansonsten terminiert `signum` mit dem Aufruf der Fehlerfunktion `error "Vorzeichenfunktion undefiniert"`.

(d) `fromInteger` den Wert z (in Form einer `MatrixF`-Wertdarstellung).

Dabei gilt, dass die Matrixdarstellung einer Zahl $z \in \mathbb{Z}$ jeder `MatrixF`-Wert vom Typ $(1, 1)$ mit einem funktionalen Wert ist, der für Zeilen- und Spaltenwert 1 den Wert z liefert, für andere Zeilen- und Spaltenwerte einen beliebigen Wert aus $\mathbb{Z}$.

A.6 **Ohne Abgabe, ohne Beurteilung:**

(a) Hätten Sie das Ziel einiger Instanzbildungen auch mithilfe von `deriving`-Klauseln erreichen können? Begründen Sie Ihre Antwort.

(b) Wie könnten Sie statt zweidimensionaler Matrizen (rechteckige bzw. quadratische Zahlschemata) dreidimensionale kubische Matrizen (Quader-, würfelförmige Zahlschemata) mit Listendarstellungen wie auf Angabe 3 und funktionalen Darstellungen wie auf Angabe 6 modellieren? Erscheint Ihnen eine der beiden Modellierungen einfacher oder einfacher umzusetzen zu sein? Warum?

A.7 **Ohne Beurteilung:** Beschreiben Sie für jede Rechenvorschrift in einem Kommentar knapp, aber gut nachvollziehbar, wie die Rechenvorschrift vorgeht.

A.8 **Ohne Abgabe, ohne Beurteilung:** Testen Sie alle Funktionen umfassend mit aussagekräftigen eigenen Testdaten.

B Papier- und Bleistiftaufgaben

Entfallen auf Angaben 5 bis 7.

Iucundi acti labores.
Getane Arbeiten sind angenehm.

Cicero (106 - 43 v.Chr.)
röm. Staatsmann und Schriftsteller