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

3. Angabe zu Funktionale Programmierung von Fr, 29.10.2021.

Erstabgabe: Fr, 05.11.2021, 12:00 Uhr

Zweitabgabe: Siehe „Hinweise zu Org. u. Ablauf der Übung" (TUWEL-Kurs)

(Teil A: beurteilt; Teil B: ohne Abgabe, ohne Beurteilung)

Themen: *Literate Haskell-Skripte, Typklassen, Instanzbildung, Rechnen mit Listen*
Stoffumfang: *Kapitel 1 bis Kapitel 9, besonders Kapitel 2 bis 6.*

- **Teil A, programmiertechnische Aufgaben:** Besprechung am ersten Übungsgruppentermin, der auf die *Zweitabgabe* der programmiertechnischen Aufgaben folgt.
- **Teil B, Papier- und Bleistiftaufgaben:** Besprechung am ersten Übungsgruppentermin, der auf die *Erstabgabe* der programmiertechnischen Aufgaben folgt.
- **Teil C, Organisatorische und Terminhinweise**

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

   `Angabe3.lhs`

   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 Angabe3.lhs`.

   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 Angabe3 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. 50 Punkte)

Erweitern Sie zur Lösung der programmiertechnischen Aufgaben die Rahmendatei `Angabe3.lhs`. Kommentieren Sie die Rechenvorschriften in Ihrem Programm zweckmäßig, aussagekräftig und problemangemessen. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten. 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:

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, $m, n \in IN_1$:

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

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

3. *Matrixaddition:* Sind M, N zwei Matrizen vom Typ (m, n), ist ihre Summe S die folgendermaßen definierte Matrix desselben Typs:

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

   mit

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

   und

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

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

4. *Matrixsubtraktion:* Sind M, N zwei Matrizen vom Typ (m, n), ist ihre Differenz D die folgendermaßen definierte Matrix desselben Typs:

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

   mit

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

   und

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

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

5. *Matrixmultiplikation:* Sind M, N zwei Matrizen vom Typ (m, n) und (n, p), ist ihr Produkt P die folgendermaßen definierte Matrix vom Typ (m, p):

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

mit

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

und

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

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

6. *Multiplikation mit einem Skalar:* Ist M eine Matrix vom Typ (m, n) und $s \in \mathbb{Z}$ eine ganze Zahl (ein sog. *Skalar*), so ist ihr Skalarprodukt SP die folgendermaßen definierte Matrix vom Typ (m, n):

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

mit

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

und

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

Wir modellieren Matrizen in der Folge zeilenweise als Listen von Listen ganzer Zahlen:

```
> type Nat1  = Int
> type Zeile = [Int]

Matrizen konzeptuell als Listen von Zeilen dargestellt:

> newtype Matrix = M [Zeile]
```

Der Wert `M [] :: Matrix` stellt keine keine gültige Matrix dar; wir verwenden diesen Wert deshalb, wo sinnvoll und nötig, als fehleranzeigenden Wert:

```
> fehlerwert = M [] :: Matrix
```

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

```
show (M [[1,2],[3,4]]) ->> "([1,2] [3,4])"
show (M [[1,2],[3,4],[5,6]]) ->> "([1,2] [3,4] [5,6])"
show (M [[1,2,3],[4,5],[6]]) ->> "([1,2,3] [4,5] [6])"
show (M [[1,2,3],[],[6]]) ->> "([1,2,3] [] [6])"
show (M [[],[],[]]) ->> "([] [] [])"
show (M []) ->> "()"
```

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

```
> data Matrixtyp = Mat (Nat1,Nat1) | KeineMatrix deriving (Eq,Show)

> matrixtyp :: Matrix -> Matrixtyp
```

Angewendet auf einen `Matrix`-Wert, liefert `matrixtyp` den Wert `(Mat (m,n))`, wenn das Argument eine Matrix vom Typ `(m,n)` repräsentiert; sonst den Wert `KeineMatrix`.

A.3 Machen Sie den Typ `Matrix` 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.4 Machen Sie den Typ `Matrix` 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 `(M [])` als fehleranzeigenden Wert liefern. Weiters soll gelten: Angewendet auf einen `Matrix`-Wert M bzw. eine ganze Zahl z, liefert

(a) `negate` das Skalarprodukt von -1 und M, wenn M eine Matrix darstellt, den Fehlerwert `(M [])` sonst.

(b) `abs` die Matrix M', die aus M entsteht, indem jedes Element von M durch seinen Absolutbetrag ersetzt wird, den Fehlerwert `(M [])` sonst.

(c) `signum` den Wert 1 (in Form der Matrix vom Typ $(1,1)$ mit einzigem Element 1), wenn alle Elemente von M echt positiv sind; den Wert 0 (in Form der Matrix vom Typ $(1,1)$ mit einzigem Element 0), wenn alle Elemente von M null sind; den Wert -1 (in Form der Matrix vom Typ $(1,1)$ mit einzigem Element -1), wenn alle Elemente von M echt negativ sind. Ansonsten terminiert `signum` mit dem Aufruf der Fehlerfunktion `error "Vorzeichenfunktion undefiniert"`.

(d) `fromInteger` die einelementige Matrix vom Typ $(1,1)$ mit z als Element.

A.5 **Ohne Abgabe, ohne Beurteilung:** Hätten Sie das Ziel einiger Instanzbildungen auch mithilfe von `deriving`-Klauseln erreichen können? Begründen Sie Ihre Antwort.

A.6 **Ohne Beurteilung:** Beschreiben Sie für jede Rechenvorschrift in einem Kommentar knapp, aber gut nachvollziehbar, wie die Rechenvorschrift vorgeht.

A.7 **Ohne Abgabe, ohne Beurteilung:** Testen Sie alle Funktionen umfassend mit aussagekräftigen eigenen Testdaten.

B Papier- und Bleistiftaufgaben (ohne Abgabe/Beurteilung)

B1. Warum ist in den Aufgaben aus Teil A der Typ `Matrix` als neuer Typ:

```
> newtype Matrix = M [Zeile]
```

nicht als Typalias:

```
> type Matrix = [Zeile]
```

eingeführt?

B.2 Sind Sie bei den Instanzbildungen für `Eq` und `Num` mit den jeweiligen Minimalvervollständigungen ausgekommen (s. Kapitel 4.3 zu Minimalvervollständigungen)? Konnten Sie also bei den Instanzbildungen die Protoimplementierungen ausnützen? Wenn nein, warum nicht?

B.3 Ändert sich die Bedeutung der Funktion `negate` gegenüber Aufgabe A.3, wenn wir sie in folgender Weise angeben?

- Angewendet auf einen `Matrix`-Wert M, liefert `negate` die Differenz aus der zu M typgleichen Nullmatrix (d.h. alle Matrixelemente haben Wert 0) und M, wenn M eine Matrix darstellt, den Fehlerwert `(M [])` sonst.

Implementieren Sie `negate` entsprechend und überprüfen Sie anschließend Ihre Vermutung mithilfe von `ghci` oder/und `hugs`!

B.4 Wiederholen Sie die Aufgaben aus Teil A&B mit spaltenweise definierten Matrizen:

```
> type Spalte     = [Int]
> newtype Matrix' = M' [Spalte]
```

Iucundi acti labores.
Getane Arbeiten sind angenehm.

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

C Organisatorische und Terminhinweise

1. **Kleinübungsgruppenzusammenlegung**: Aufgrund geringer Anmeldezahlen für einige freitags stattfinde Übungsgruppen ergeben sich folgende Änderungen gegenüber der ursprünglichen Planung:
   - KÜG 12 und KÜG 16 werden ersatzlos aufgelassen.
   - KÜG 17 geht in KÜG 15 auf (freitags, 15-16 Uhr, Hannes Siebenhandl)
2. **Beginn der Kleinübungsgruppen: Mittwoch, 03.11.2021, Freitag, 05.11.2021.**
   - ***Mo, 01.11.2021: Termine für KÜG 1-7 entfallen feiertagsbedingt!***

     Nehmen Sie bitte in dieser Woche nach freier Wahl an einem der anderen KÜG-Termine am Mittwoch, 03.11.2021, oder Freitag, 05.11.2021, teil!
   - KÜG 8: Mi, 03.11.2021, 12:00-13:00 Uhr, Zoom
   - KÜG 9: Mi, 03.11.2021, 12:00-13:00 Uhr, Zoom
   - KÜG 10: Mi, 03.11.2021, 12:00-13:00 Uhr, Zoom
   - KÜG 11: Fr, 05.11.2021, 12:00-13:00 Uhr, Zoom
   - KÜG 13: Fr, 05.11.2021, 12:00-13:00 Uhr, Zoom
   - KÜG 14: Fr, 05.11.2021, 15:00-16:00 Uhr, Zoom
   - KÜG 15: Fr, 05.11.2021, 15:00-16:00 Uhr, Zoom

   Mögliche zukünftige Änderungen werden via TUWEL- oder TISS-Aussendung oder/und unmittelbar in änderungsbetroffenen Kleinübungsgruppen bekanntgegeben.
3. **Nächste Vorlesung: Mittwoch, 03.11.2021, 10:15-11:45 Uhr, Zoom.**
   - 10:15-11:15 Uhr: Vorlesungsteil IV
   - 11:30-11:45 Uhr: Umgekehrtes Klassenzimmer zu Vorlesungsteil III
4. **Labor- und Vorlesungsfrage- und Antwortforen:**
   Regelmäßig mittwochs und donnerstags, jeweils 14:15-15:00 Uhr. Nächste Termine:
   - Laborfrage- und Antwortforum: Mi, 03.11.2021, 14:15-15:00 Uhr, Zoom
   - Vorlesungsfrage- und Antwortforum: Do, 04.11.2021, 14:15-15:00 Uhr, Zoom

Die Zoom-Teilnahme-URLs für KÜGs, Vorlesung, umgekehrtes Klassenzimmer und Labor- und Vorlesungsfrage- und Antwortforen stehen im TUWEL-Kurs der LVA bereit.