5. Aufgabenblatt zu Funktionale Programmierung vom Mi, 14.11.2018.
Fällig: Mi, 21.11.2018 (15:00 Uhr)

Themen: *Funktionen über algebraischen Datentypen, Typklassen und Instanzbildungen*

Zur Frist der Zweitabgabe: Siehe „Hinweise zu Organisation und Ablauf der Übung“ auf der Homepage der LVA.

Aufgabe

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe5.hs` ablegen. Sie sollen für die Lösung dieses Aufgabenblatts also wieder ein 'gewöhnliches' Haskell-Skript schreiben.

Versehen Sie wieder wie auf den bisherigen Aufgabenblättern alle Funktionen, die Sie zur Lösung benötigen, mit ihren Typdeklarationen und kommentieren Sie Ihre Programme aussagekräftig. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und (Wertvereinbarungen für) Konstanten.

1. Wir führen die Typklasse `FFGB` ein, um durch Überladung ein unterschiedliches Verhalten der bekannten *Fakultäts-*, *Fibonacci-*, *Größter-gemeinsamer-Teiler-* und *Binomial*-Funktionen für unterschiedliche Typen mittels typspezifischer Instanzbildungen zu erreichen:

```
newtype IN_0 = IN_0 Integer deriving (Eq,Ord,Show)

instance Num IN_0 where
 ... -- von Aufgabenblatt 4 uebernehmen!

class (Eq a, Ord a, Show a, Num a) => FFGB a where
 fac      :: a -> a           -- Fakultaetsfunktion
 fib      :: a -> a           -- Fibonacci-Funktion
 ggt      :: a -> a -> a      -- Groesster-gemeinsamer-Teiler-Funktion
 binom    :: a -> a -> a      -- Binomialkoeffizientenfunktion
 div_ffgb :: a -> a -> a      -- Ganzzahlige Divisionsfunktion (ohne Protoimpl.)

 fac n
  | n == 0 = 1
  | sonst  = n * fac (n-1) where sonst = True
 fib n
  | n == 0 = 0
  | n == 1 = 1
  | True   = fib (n-1) + fib (n-2)
 ggt m n
  | m == n = n
  | m > n  = ggt (m-n) n
  | m < n  = ggt m (n-m)
 binom n k = div_ffgb (fac n) (fac k * fac (n-k))
```

Machen Sie die Typen `Int`, `Integer` und `IN_0` zu Instanzen der Typklasse `FFGB` und nutzen Sie dafür die Protoimplementierungen der Klasse `FFGB` bestmöglich aus. Für die Instanzbildungen soll (nach jeweiliger typspezifischer Vervollständigung von `div_ffgb`) gelten:

- `Int`: Die Funktionen `fac`, `fib`, `ggt` und `binom` verhalten sich für alle Argumentwerte wie durch ihre Protoimplementierungen angegeben.
- `Integer`:
  - Werden die Funktionen `fac` und `fib` mit nichtnegativen Argumenten, die Funktion `ggt` mit echt positiven Argumenten aufgerufen, verhalten sie sich wie ihre mathematischen Gegenstücke, ansonsten liefern sie das Resultat -1.
  - Wird die Funktion `binom` mit mindestens einem echt negativen Argumentwert aufgerufen, so liefert die Funktion den Wert -1, ansonsten verhält sich die Funktion wie die Protoimplementierung.

- `IN_0`:
  - Werden die Funktionen `fac` und `fib` mit gültigen `IN_0`-Werten aufgerufen (s. Aufgabenblatt 4), die Funktion `ggt` mit echt positiven gültigen `IN_0`-Werten, verhalten sie sich wie ihre mathematischen Gegenstücke, ansonsten liefern `fac` und `fib` den Argumentwert als Ergebnis, `ggt` den Wert des ersten Arguments, wenn dieser nicht gültig ist, sonst den des zweiten Arguments.
  - Wird die Funktion `binom` mit gültigen Argumentwerten n, k, $k \leq n$ aufgerufen, so verhält sich die Funktion wie ihr mathematisches Gegenstück, ansonsten liefert sie den Wert `IN_0 (-99)` als Resultat, um den Fehlerfall anzuzeigen.

Testen Sie die verschiedenen Instanzen durch Aufrufe mit geeigneten Argumentwerten (ohne Abgabe!). Bei Aufrufen mit `Int`- und `Integer`-Werten muss der gemeinte Argumenttyp beim Aufruf spezifiziert werden, um die Überladung auflösen zu können:

```
fac 5               ->> Fehler: Nicht aufgeloeste Ueberladung
fac (5 :: Int)      ->> 120
fac (-5 :: Int)     ->> ... ->> ... terminiert nicht regulaer.
fac (5 :: Integer)  ->> 120
fac (-5 :: Integer) ->> -1
```

Die Divisionsfunktion `div_ffgb` muss für alle drei Typen `Int`, `Integer` und `IN_0` selbst implementiert werden. Ihre Bedeutung soll dabei (für alle `Int`- und `Integer`-Werte sowie für gültige `IN_0`-Werte) mit der bekannten Divisionsfunktion `div` für die ganzzahlige Division übereinstimmen, die Operator der von uns noch nicht behandelten Typklasse `Integral` ist. Die Typen `Int` und `Integer` sind vordefinierte Instanzen der Klasse `Integral`, so dass Sie die Implementierung von `div_ffgb` bei der Instanzbildung für `Int` und `Integer` auf `div` abstützen können, für `IN_0` ist diese (unmittelbare) Abstützung nicht in gleicher Weise möglich. Für nichtgültige `IN_0`-Werte (s. Aufgabenblatt 4) soll `div_ffgb` den nichtgültigen `IN_0`-Wert `IN_0 (-1)` liefern, um die Fehlersituation anzuzeigen; ebenso, wenn das Divisorargument den `IN_0`-Wert 0 hat.

Gottfried Wilhelm Leibniz (1.7.1646-14.11.1716), Philosoph und Universalgelehrter, Begründer von Infinitesimal- und Binärrechnung, verdanken wir auch die Determinantenrechnung. Wir nehmen die 302-te Wiederkehr seines Todestags am heutigen Ausgabetag von Aufgabenblatt 5 zum Anlass, uns mit Determinanten von Matrizen zu beschäftigen. Dafür führen wir folgende Begriffe ein, wobei $M =_{df} \{1, 2, \ldots, n\}$ die Menge natürlicher Zahlen von 1 bis n bezeichne.

- **Permutation** von M: Eine *Permutation* von M ist eine (vollständige und duplikatfreie) Anordnung der Elemente von M in einer bestimmten Reihenfolge, z.B. $\langle 1, 2, 3, \ldots, n\rangle$, $\langle n, n-1, \ldots, 2, 1\rangle$, $\langle 1, 3, 2, 4, 5, 7, 6, 8, \ldots\rangle$, etc. Die Menge aller Permutationen von M bezeichnen wir mit $\mathcal{P}(M)$.
- **Inversion** zweier Permutationselemente: Sei $P = \langle p_1, p_2, \ldots, p_n\rangle$ eine Permutation von M. Die Elemente p_i und p_j, $i < j$, von P bilden eine *Inversion* gdw. $p_i > p_j$.

  Beispiel: Für $P = \langle 5, 2, 1, 4, 3\rangle$ Permutation von $M = \{1, 2, 3, 4, 5\}$ gilt:

  - 5 bildet eine Inversion mit 2, 1, 4 und 3.
  - 2 bildet eine Inversion mit 1.
  - 1 bildet keine Inversion.
  - 4 bildet eine Inversion mit 3.
  - 3 bildet keine Inversion.

  Die Anzahl der Inversionen von P bezeichnen wir mit $\mathcal{I}(P)$. Im Beispiel gilt: $\mathcal{I}(P) = 6$.
- **Charakter** einer Permutation: Die Zahl $\chi(P) =_{df} (-1)^{\mathcal{I}(P)}$ heißt *Charakter* von P.
- **Matrix** vom Typ (m, n): Eine *Matrix* ist eine zweidimensionale Anordnung A von $m * n$ Zahlen, $m, n \in IN_1$, in m Zeilen zu je n Spalten:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} = (a_{jk})_{(m,n)}$$

A heißt dann Matrix vom *Typ* (m, n). Für $m \neq n$ heißt A *rechteckig*, für $m = n$ heißt A *quadratisch.*

- **Determinante** einer quadratischen Matrix: Die *Determinante*

$$d_n = |A| = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \multicolumn{4}{c}{\dotfill} \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}$$

einer quadratischen Matrix A vom Typ (n, n)

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \multicolumn{4}{c}{\dotfill} \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

ist definiert durch:

$$d_n = |A| = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \multicolumn{4}{c}{\dotfill} \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix} =_{df} \sum_{\langle p_1,\ldots,p_n\rangle \in \mathcal{P}(\{1,\ldots,n\})} \chi(\langle p_1, \ldots, p_n\rangle)\; a_{1p_1} a_{2p_2} \ldots a_{np_n}$$

Beispiele:

$$\begin{aligned} d_1 &= |a_{11}| = a_{11} \\ d_2 &= \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21} \\ d_3 &= \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} - a_{13}a_{22}a_{31} \end{aligned}$$

Anmerkung: Das Koeffizientenschema $(a_{jk})_{(n,n)}$ eines linearen Gleichungssystems mit mehreren Unbekannten $x_1, x_2, \ldots, x_n$

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= c_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= c_2 \\ &\cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= c_n \end{aligned}$$

bildet die sog. *Koeffizientenmatrix* des Gleichungssystems. Die Determinante der Koeffizientenmatrix erlaubt es, die Lösbarkeit des zugehörigen Gleichungssystems zu bestimmen und das Gleichungssystem, so Lösbarkeit gegeben ist, aufzulösen. Der Determinantenbegriff geht wesentlich auf Leibniz zurück, der ihn zur Lösung linearer Gleichungssysteme eingeführt und verwendet hat. Die obige Formel zur Berechnung von Determinanten ist als *Leibniz-Formel* bekannt. Für nichtquadratische Matrizen ist der Determinantenbegriff nicht definiert.

2. Wir modellieren (quadratische) Matrizen wie folgt:

```
type Matrioid     = [[Integer]]
type Typ          = (IN_0,IN_0)
data Matrix       = Matrix Typ Matrioid deriving (Eq,Show)
data Determinante = Determinante_ist Integer
                    | Determinante_ist_undefiniert deriving (Eq,Show)

class Quadratisch a where
 ist_quadratisch :: a -> Bool
```

   Machen Sie den Typ `Matrix` zu einer Instanz der Typklasse `Quadratisch`. Dabei gilt: Ein `Matrix`-Wert ist *quadratisch* gdw. das Matrioidelement vom Typ (n, n), $n \geq 1$, also ein Matriod mit n Listen der Länge n ist, und das Typelement diesen Typ als Paar von `IN_0`-Werten korrekt ausweist.

3. Schreiben Sie eine Haskell-Rechenvorschrift `determinante` mit der Signatur:

```
determinante :: Matrix -> Determinante
```

   Angewendet auf eine quadratische Matrix m liefert `determinante` die Determinante von m als `Determinante_ist`-Wert; anderenfalls den Wert `Determinante_ist_undefiniert`.

4. Machen Sie abschließend auch die Typen

   4.1 `IN_0`

   4.2 `newtype Zeichenreihe = Z String deriving (Eq,Show)`

   4.3 `newtype Tripel = T (IN_0,Zeichenreihe,Matrix) deriving (Eq,Show)`

   4.4 `newtype Liste = L [Tripel] deriving (Eq,Show)`

   je zu einer Instanz der Typklasse `Quadratisch`. Dafür legen wir fest:

   - Ein `IN_0`-Wert heißt *quadratisch* gdw. der Wert das Produkt eines (gültigen) `IN_0`-Werts mit sich selbst ist.
   - Ein `Zeichenreihe`-Wert `(Z s)` heißt *quadratisch* gdw. `s` lässt sich als k-fache Konkatenation einer Zeichenreihe `t` der Länge k darstellen.

     Beispiel: Für `s="abcabcabc"` ist `(Z s)` quadratisch, da für `t="abc"` gilt:
     `s == "abcabcabc" == "abc"++"abc"++"abc"==t++t++t`.
   - Ein `Tripel`-Wert heißt *quadratisch* gdw. jeder seiner Komponentenwerte ist quadratisch.
   - Ein `Liste`-Wert heißt *quadratisch* gdw. alle seine Elemente sind quadratisch.

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 alte Lösung, aus der importiert wird, nicht mit abgesammelt wird. Deshalb: Kopieren statt importieren zur Wiederwendung!

Haskell Live

An einem der kommenden *Haskell Live*-Termine, der nächste ist am Freitag, den 16.11.2018, werden wir uns u.a. mit der Aufgabe *World of Perfect Towers* beschäftigen.

World of Perfect Towers

In diesem Spiel konstruieren wir *Welten perfekter Türme*. Dazu haben wir n Stäbe, die senkrecht auf einer Bodenplatte befestigt sind und auf die mit einer entsprechenden Bohrung versehene Kugeln gesteckt werden können. Diese Kugeln sind ebenso wie die Stäbe beginnend mit 1 fortlaufend nummeriert.

Die auf einen Stab gesteckten Kugeln bilden einen Turm. Dabei liegt die zuerst aufgesteckte Kugel ganz unten im Turm, die zu zweit aufgesteckte Kugel auf der zuerst aufgesteckten, usw., die zuletzt aufgesteckte Kugel ganz oben im Turm. Ein solcher Turm heißt *perfekt*, wenn die Summe der Nummern zweier unmittelbar übereinanderliegender Kugeln eine Zweierpotenz ist. Eine Menge von n perfekten Türmen heißt *n-perfekte Welt*.

In diesem Spiel geht es nun darum, n-perfekte Welten mit maximaler Kugelzahl zu konstruieren. Dazu werden die Kugeln in aufsteigender Nummerierung, wobei mit der mit 1 nummerierten Kugel begonnen wird, so auf die n Stäbe gesteckt, dass die Kugeln auf jedem Stab einen perfekten Turm bilden und die Summe der Kugeln aller Türme maximal ist.

Schreiben Sie in Haskell oder einer anderen Programmiersprache ihrer Wahl eine Funktion, die zu einer vorgegebenen Zahl n von Stäben die Maximalzahl von Kugeln einer n-perfekten Welt bestimmt und die Türme dieser n-perfekten Welt in Form einer Liste von Listen ausgibt, wobei jede Liste von links nach rechts die Kugeln des zugehörigen Turms in aufsteigender Reihenfolge angibt.

Haskell Private

Die Anmeldung zu *Haskell Private* ist offen. Nutzen Sie die Möglichkeit! Auch die Möglichkeit, eigene Terminvorschläge zu machen. Nähere Hinweise und die URL zur Anmeldungsseite finden Sie auf der Homepage der Lehrveranstaltung.