1. Übungsaufgabe zu
Fortgeschrittene funktionale Programmierung
Thema: Ströme, Generatoren und Selektoren
ausgegeben: Di, 14.03.2017, fällig: Di, 28.03.2017

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der im folgenden angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `AufgabeFFP1.hs` auf **oberstem Niveau in Ihrem Gruppenverzeichnis** ablegen. Kommentieren Sie Ihre Programme aussagekräftig und benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

- Implementieren Sie den Generator
  - `repeat`

  und die Selektoren
  - `within`
  - `relative`

  aus Kapitel 1 der Vorlesung so typallgemein wie möglich in Haskell und testen Sie unterschiedliche Kombinationen dieser Generatoren und Selektoren anhand der Beispiele zur Approximation von Quadratwurzeln positiver reeller Zahlen, der näherungsweisen numerischen Integration stetiger einstelliger reeller Funktionen sowie der näherungsweisen Ableitung solcher Funktionen an einer Stelle, wobei der im Prelude definierte Name `repeat` mittels `hiding`-Klausel verborgen werden soll.

  Implementieren Sie dazu die entsprechenden Rechenvorschriften
  - `next`
  - `easyintegrate`
  - `integrateNaive` (erste Variante von `integrate` aus Kap. 1)
  - `integrateEfficient` (zweite Variante von `integrate` aus Kap. 1)
  - `easydiff`
  - `differentiate`

  zusammen mit ihren Hilfsfunktionen sowie die entsprechenden Generator/Selektor-Kombinationen
  - `sqrt :: InitialApprox -> Epsilon -> SquareArg -> Approx`
  - `relativesqrt :: InitialApprox -> Epsilon -> SquareArg -> Approx`
  - `intgrt :: Map -> Low -> High -> Epsilon -> Area`
    (Analogon zur Generator/Selektor-Kombination `sqrt`)
  - `relativeintgrt :: Map -> Low -> High -> Epsilon -> Area`
    (Analogon zu `relativesqrt`)

- `intgrteff :: Map -> Low -> High -> Epsilon -> Area`
  (Effiziente Variante zu `intgrt`)
- `relativeintgrteff :: Map -> Low -> High -> Epsilon -> Area`
  (Effiziente Variante zu `relativeintgrt`)
- `diff :: Map -> XCoordinate -> InitialH -> Epsilon -> Slope`
  (Analogon zu `sqrt`)
- `relativediff :: Map -> XCoordinate -> InitialH -> Epsilon -> Slope`
  (Analogon zu `relativesqrt`)

aus Kapitel 1 der Vorlesung, wobei

```
type InitialApprox = Double   -- Auschliesslich Werte > 0
type Epsilon       = Double   -- Auschliesslich Werte > 0
type SquareArg     = Double   -- Auschliesslich Werte > 0
type Approx        = Double   -- Auschliesslich Werte > 0
type Map           = Double -> Double
type Low           = Double   -- Untere Intervallgrenze
type High          = Double   -- Obere Intervallgrenze
type Area          = Double
type XCoordinate   = Double
type InitialH      = Double   -- Auschliesslich Werte > 0
type Slope         = Double
```

Benutzen Sie für Listen bzw. Ströme den Standardlistenkonstruktor `[]` und für reelle Zahlen den Typ `Double` in Haskell. Alle Funktionen geben jeweils die zuletzt berechnete Näherung zurück, also die genaueste Approximation zum Abbruchzeitpunkt.

Ohne Abgabe: `integrateNaive`, `integrateEfficient` und `differentiate` sind selbst Generatoren. Anders als `differentiate` stützen sich die Generatoren `integrateNaive` und `integrateEfficient` jedoch nicht auf den Generator `repeat` ab.

Wie könnte ein Generator `repeat2` aussehen, mit dessen Hilfe sich `integrateNaive` und `integrateEfficient` analog zur Abstützung des Generators `differentiate` auf `repeat` implementieren ließen und sich ähnlich wie `repeat` für andere Aufgabenstellungen wieder benutzen ließe?

- Betrachten Sie die Folgen $(x_i)_{i \in \mathbb{N}_0}$ reeller Zahlen, deren Elemente nach der Regel

$$x_{n+1} = a x_n (1 - x_n)$$

gebildet werden, wobei a eine reellwertige Konstante ist mit $0 \leq a \leq 4$ und die Anfangswertswerte x_0 ebenfalls reellwertig mit $0 \leq x_0 \leq 1$ gewählt werden.

Schreiben Sie eine Haskell-Rechenvorschrift `next2`

```
type Avalue        = Double -- 0 <= a <= 4
type InitialValue  = Double -- 0 <= x0 <= 1
type SequenceValue = Double
next2 :: Avalue -> InitialValue -> SequenceValue
```

und mithilfe der Generatoren und Selektoren `repeat`, `within` und `relative` aus Aufgabenteil 1 Generator/Selektor-Kombinationen

- sequence
- relativesequence

analog zu den Generator/Selektor-Kombinationen `sqrt` und `relativesqrt`.

Untersuchen Sie das Konvergenzverhalten der Generator/Selektor-Kombinationen `seqence` und `relativesequence` in Abhängigkeit des Wertes von a. Wählen Sie dazu unterschiedliche Werte von a mit

- $0 \leq a < 1$
- $1 \leq a < 3$
- $3 \leq a \leq 3.449$
- $3.449 < a \leq 4$

und gewinnen Sie so eine Vermutung über das Verhalten der Folgenglieder und ob die Selektoren `within` und `relative` für alle Werte von a sinnvoll sind, ggf. auch dadurch, dass Sie den Generator(ausdruck) mit Selektoren wie `take n` kombinieren für wachsende Werte von `n`$\in \mathbb{N}$.

- Sei $f : IR \to IR$ eine reelle stetige Funktion. Die Funktion f hat einen Vorzeichenweichsel im Intervall $I = [a, b] \subseteq IR$, wenn ein Teilintervall $I_0 = [a_0, b_0] \subseteq I$ existiert mit der Eigenschaft

$$f(a_0)f(b_0) < 0$$

Nach dem Zwischenwertsatz für reelle stetige Funktionen gibt es dann im Intervall $I_0 = [a_0, b_0]$ mindestens eine Nullstelle von f, d.h., ein $x \in IR$ mit $a_0 \leq x \leq b_b$ mit $f(x) = 0$.

Mithilfe eines Intervallschachtelungsverfahrens lässt sich eine solche Nullstelle auf folgende Weise annähern:

Sei $I_t = [a_t, b_t]$ ein Intervall mit $f(a_t)f(b_t) < 0$ und sei $x_t = \frac{1}{2}(a_t + b_t)$ der Mittelpunkt des Intervalls I_t.

- Gilt $f(x_t) = 0$, so ist eine Nullstelle x_t von f gefunden und das Intervallschachtelungsverfahren ist beendet.
- Gilt $f(x_t) \neq 0$, und $f(a_t)f(x_t) < 0$, so wird ein neues Intervall $I_{t+1} = [a_{t+1}, b_{t+1}]$ bestimmt nach der Vorschrift

$$a_{t+1} = a_t \quad \text{und} \quad b_{t+1} = x_t.$$

– Gilt $f(x_t) \neq 0$, $f(a_t)f(x_t) > 0$ und $f(x_t)f(b_t) < 0$, so wird ein neues Intervall $I_{t+1} = [a_{t+1}, b_{t+1}]$ bestimmt nach der Vorschrift

$$a_{t+1} = x_t \quad \text{und} \quad b_{t+1} = b_t.$$

Schreiben Sie eine Haskell-Rechenvorschrift

```
type Interval        = (Double,Double)
type InitialInterval = Interval
type Map             = Double -> Double
type Epsilon         = Double -- Auschliesslich Werte > 0
nextinterval :: Map -> Interval -> Interval
```

und darauf aufbauend einen Generator

```
intervalnesting :: Map -> InitialInterval -> [Interval]
```

die für eine Abbildung f und ein Anfangsintervall I den Strom der Intervalle nach obigem Verfahren liefert.

Kombinieren Sie den Generator `intervalnesting` mit den Selektoren `within` und `relative` aus Aufgabenteil 1 zu zwei Generator/Selektor-Kombinationen

– `null :: Map -> InitialInterval -> Epsilon -> Interval`

– `relativenull :: Map -> InitialInterval -> Epsilon -> Interval`

die das Intervallschachtelungsverfahren abbrechen, wenn der Absolutbetrag der Differenz bzw. des Verhältnisses zweier aufeinanderfolgender Intervalle einen vorgegebenen Wert $\epsilon > 0$ annimmt oder unterschreitet. Zurückgegeben wird dabei das zuletzt berechnete Intervall, also die genaueste Approximation zum Abbruchzeitpunkt.

Wichtige Hinweise:

- **Zugangsdaten:** Bis spätestens zum 15. März 2017 sollten Sie per email an Ihre generische Adresse `e<matrikelnummer>@student.tuwien.ac.at` Ihre Zugangsinformationen für die Maschine `g0.complang.tuwien.ac.at` erhalten. Loggen Sie sich damit bitte baldmöglich auf der `g0` ein und ändern Sie Ihr initiales Losungswort auf ein von Ihnen gewähltes neues Losungswort.
- **Aufgabenabgabe:** Die Aufgaben werden für die automatische Überprüfung unter Hugs auf der Maschine `g0` ausgeführt. Wenn Sie für die Programmentwicklung eine andere Maschine oder einen anderen Interpretierer benutzen, überprüfen Sie bitte auf jeden Fall rechtzeitig vor dem Abgabezeitpunkt, dass Ihre Programme auch unter Hugs auf der `g0` wie von Ihnen beabsichtigt laufen.