194.026 Funktionale Programmierung WS 23

— Test 1 —

Donnerstag, 18.01.2024, 16:00–18:00 Uhr

Nachname	Vorname	Matrikel-nummer	Hörsaal	Platz-nummer	Unterschrift
			FH HS1		

Aufgabe	1	2	3	4	5	Summe Punkte
Punkte	19	15	18	21	27	100
Erreichte Punkte						

...schreiben Sie bitte auf jedes Blatt, das Sie abgeben, Ihren Namen und Ihre Matrikelnummer!

Aufgabe 1 (2∗0.5+12+6 = 19 Punkte)

Ein Element einer Liste heißt *Gipfel* gdw (genau dann, wenn) es ein linkes und ein rechtes Nachbarelement hat und echt größer ist als als jedes der beiden.

1. Gegeben sind die Listen `lst1` und `lst2` natürlicher Zahlen:

   (a) `lst1 = [5,8,7,6,0,9,5,4,7,9]`

   (b) `lst2 = [11,2,11,13,6,12,10]`

   Markieren Sie (durch Einkreisen) alle Gipfel in `lst1` und `lst2`.

2. Schreiben Sie eine Haskell-Rechenvorschrift `gipfel`: angewendet auf eine Liste `lst` natürlicher Zahlen liefert `gipfel` die aufsteigend geordnete Liste der Positionen, an denen in `lst` ein Gipfel steht. Dabei gilt: das erste Element einer Liste steht an Position 0, das zweite an Position 1 usw.

   ```
   type Nat0     = Int
   type Position = Nat0

   gipfel :: [Nat0] -> [Position]
   ```

3. Erklären Sie knapp, aber gut nachvollziehbar, wie ihre Implementierung von `gipfel` vorgeht.

Aufgabe 2 (5+5+5 = 15 Punkte)

Gegeben sind die Typen und Funktionen aus Aufgabe 1.

1. Was passiert, wenn Ihre Implementierung von `gipfel` auf die Liste `[4,7,-5,-1,-2,10]` angewendet wird?

   (a) Fehler zur Übersetzungszeit?
   (b) Fehler zur Laufzeit?
   (c) Weder ein Fehler zur Übersetzungszeit noch zur Laufzeit. Die Auswertung des Aufrufs terminiert regulär mit dem Ergebnis ...

   Begründen Sie Ihre Antwort. Im Fall von (c), geben Sie auch das Ergebnis der Auswertung an.

2. Lässt sich der Typ Ihrer Implementierung von `gipfel` verallgemeinern? Welches ist der allgemeinst mögliche Typ dafür? Begründen Sie Ihre Antwort.

3. Ist der allgemeinst mögliche Typ Ihrer Implementierung von `gipfel` monomorph oder polymorph? Begründen Sie Ihre Antwort und geben Sie im Fall von Polymorphie den Polymorphietyp so genau wie möglich an.

Aufgabe 3 (6+6+6 = 18 Punkte)

Richtig oder falsch? Begründen Sie Ihre Antwort jeweils.

1. Mit dem λ-Kalkül hat Alonzo Church die Grundlagen funktionaler Programmierung gelegt, ohne es zu wollen oder wissen.

2. Die Vorgabe "faul auszuwerten" legt die Folge der Auswertungsschritte eines Haskell-Ausdrucks eindeutig fest.

3. Haskell kennt ausschließlich faule Auswertung.

Aufgabe 4 (4+(3+3)+8+3 = 21 Punkte)

1. Variieren Sie das vordefinierte Funktional `filter` zu einem Funktional `filter2` mit Typ:

   ```
   filter2 :: (a -> Bool) -> (a -> Bool) -> [a] -> [a]
   ```

   Angewendet auf zwei Wahrheitswertfunktionen wf und wf' und eine Liste werden wf und wf' beginnend mit wf abwechselnd auf die Elemente der Liste angewendet. Liefert die jeweils angewandte Wahrheitswertfunktion den Wert `True`, für das Element, wird es in die Ergebnisliste übernommen, ansonsten verworfen.

2. Geben Sie die

   (a) curryfizierte
   (b) uncurryfizierte

   Lesart der Signatur von `filter2` an.

3. Verallgemeinern Sie die `filter2` zugrundeliegende Abbildungsidee zu einem Funktional `filter_n` mit Typ:

   ```
   filter_n :: [a -> Bool] -> [a] -> [a]
   ```

   Angewendet auf eine Liste von Wahrheitswertfunktionen und eine Liste wendet `filter_n` die Funktionen der Funktionsliste zyklisch (engl. *in round robin fashion*) auf die Elemente der Liste an, d.h. die erste Funktion wird auf das erste Listenelement angewendet, die zweite Funktion auf das zweite Listenelement usw. Ist die Werteliste kürzer als die Funktionenliste, kommen einige Funktionen nicht zum Zuge; ist umgekehrt die Funktionenliste kürzer als die Werteliste, wird die zyklische Anwendung der Funktionen schlagend.

4. Erklären Sie knapp, aber gut nachvollziehbar, wie ihre Implementierung von `filter_n` vorgeht. Erklären Sie dabei auch, ob es Randfälle gibt und wie Ihre Implementierung damit ggf. umgeht.

Aufgabe 5 (8+3+6+10 = 27 Punkte)

Gegeben sind die Typen `Rat` und `Rat'` zur Modellierung rationaler Zahlen:

```
type Nat0 = Int
type Nat1 = Int

type Zaehler    = Nat0
type Nenner     = Nat1
data Vorzeichen = P                 -- P für plus
                | M deriving Eq     -- M für minus

data Rat = R { vorzeichen :: Vorzeichen,
               zaehler    :: Zaehler,
               nenner     :: Nenner
             }

data ZN      = Z                    -- Z für Zähler
             | N deriving Eq        -- N für Nenner
newtype Rat' = R' (ZN -> Int)
```

1. Machen Sie den Typ `Rat` zu einer Instanz der Typklasse `Eq`. Der Gleichheitstest soll dabei als Wertegleichheit implementiert werden. Beachten Sie dabei, dass rationale Zahlen wertegleich sein können, ohne paarweise in Zähler und Nenner übereinzustimmen: z.B. sind die rationalen Zahlen $\frac{2}{3}, \frac{4}{6}, \frac{6}{9}, \frac{8}{12}$ wertegleich, ohne in Zähler und Nenner übereinzustimmen.

2. Erklären Sie knapp, aber gut nachvollziehbar, wie Ihre Instanzbildung und Implementierung des Gleichheitstests vorgeht. Gehen Sie dabei auch darauf ein, ob Sonderfälle zu beachten sind und wie sie ggf. behandelt werden.

3. Schreiben Sie eine Transformationsfunktion:

   ```
   t :: Rat -> Rat'
   ```

   die `Rat`-Darstellungen rationaler Zahlen in wertegleiche `Rat'`-Darstellungen überführt; dabei soll höchstens der Zähler der `Rat'`-Darstellung einen negativen Wert haben.

4. Schreiben Sie eine uncurryfizierte Funktion `add` zur Addition zweier rationaler Zahlen in `Rat'`-Darstellung. Geben Sie auch die Typsignatur von `add` an. Sehen Sie eine transparente Fehlerbehandlung vor, wenn die Argumente nicht wohldefiniert sind und beschreiben Sie in einem Kommentar, warum Ihre Fehlerbehandlung transparent ist. Das Ergebnis der Addition muss nicht gekürzt sein.