Denken Sie daran, die unterfertigte eidesstattliche Erklärung mit abzugeben!

185.A03 Funktionale Programmierung WS 21

Schriftlicher Online-Test 1 auf Papier

Donnerstag, 13.01.2022, 16:00–18:00 Uhr

Aufgabe	**1**	**2**	**3**	**4**	**5**	**Summe Punkte**
Punkte	14	24	26	16	20	100
Erreichte Punkte						

...die Aufgaben beginnen auf Seite 2!

Denken Sie daran, die unterfertigte eidesstattliche Erklärung mit abzugeben!

Aufgabe 1 (8+4+2 = 14 Punkte)

Ganzzahlige Polynome

$$\sum_{i=0}^{n} c_i x^i$$

über der Variablen x lassen sich durch die Liste ihrer ganzzahligen Koeffizienten

$$[c_n, \ldots, c_2, c_1, c_0]$$

darstellen; die Koeffizienten $c_0, c_1, \ldots, c_n$ und die Variable x nehmen also ausschließlich ganzzahlige Werte an.

1. Schreiben Sie eine Rechenvorschrift `pw`, die den Wert solcher Polynome für beliebige Stellen x berechnet; `pw` soll rekursiv sein oder sich auf eine rekursive Funktion abstützen.

   Verwenden Sie aussagekräftige Typsynonyme und geben Sie die Signatur von `pw` an.

   Gibt es Fälle, die eine Ausnahmebehandlung erfordern? Wenn ja, behandeln Sie sie im Panikmodus.

2. Erklären Sie knapp, aber gut nachvollziehbar, wie ihre Implementierung vorgeht.

3. Von welchem Rekursionstyp ist Ihre Implementierung von `pw` bzw. die etwaiger Hilfsfunktionen, auf die sich Ihre Implementierung von `pw` abstützt? Begründen Sie Ihre Antwort.

Aufgabe 2 (2∗6+2∗3+2∗3 = 24 Punkte)

Für Aufgabe 2 kann die Funktion `pw` aus Aufgabe 1 als gegeben vorausgesetzt werden.

Eine Stelle w heißt *Wurzel* eines ganzzahligen Polynoms gdw (genau dann, wenn) gilt:

$$\sum_{i=0}^{n} c_i w^i = 0$$

1. Schreiben Sie zwei curryfizierte Rechenvorschriften `w1`, `w2`, die angewendet auf ein ganzzahliges Polynom p in der Darstellung aus Aufgabe 1, eine ganze Zahl k und eine ganze Zahl l in aufsteigender Anordnung die Liste der Wurzeln w von p mit $k \leq w \leq l$ liefern.

   Verwenden Sie aussagekräftige Typsynonyme und geben Sie die syntaktischen Signaturen von `w1`, `w2` vollständig, aber nicht überflüssig geklammert, an.

   Wenden Sie, wenn nötig, dieselbe Ausnahmebehandlung wie in Aufgabe 1 an und implementieren Sie

   (a) `w1` schlicht rekursiv.
   (b) `w2` als Realisierung des Generator/Filter-Prinzips.

2. Erklären Sie knapp, aber gut nachvollziehbar, wie ihre Implementierungen von

   (a) `w1`
   (b) `w2`

   vorgehen.

3. Geben Sie die

   (a) curryfizierte
   (b) uncuryfizierte

   Lesart der Signatur von entweder `w1` oder `w2` an.

Denken Sie daran, die unterfertigte eidesstattliche Erklärung mit abzugeben!

Aufgabe 3 (2∗1+2+2∗3+2∗2+3∗4 = 26 Punkte)

Gegeben sind die Funktionen `f` und `g` mit:

```
f :: Num a => a -> a -> a
f x y = (((x + y) * (x - y)) + (x * y))

g :: (a -> a) -> [a] -> [a]
g _ []     = []
g h (x:xs) = (((h . h) x) : (g h xs))
```

1. Geben Sie die Polymorphietypen von
   (a) `f`
   (b) `g`
   möglichst genau an.
2. Richtig oder falsch?
   Der Typ von `g` kann verallgemeinert werden zu:
   ```
   g :: (a -> b) -> [a] -> [b]
   ```
   Begründen Sie Ihre Antwort.
3. Richtig oder falsch?
   Bei Auswertung jedes gültigen Aufrufs von
   (a) `f`
   (b) `g`
   wird der exakt selbe Code ausgeführt.
   Begründen Sie Ihre Antwort jeweils.
4. Welche Klammern können im Rumpf von
   (a) `f`
   (b) `g`
   weggelassen werden, ohne die Bedeutung zu verändern oder syntaktische Korrektheit zu verlieren? Geben Sie die minimal geklammerten Rümpfe von `f` und `g` an und begründen Sie Ihre Antwort.
5. Geben Sie jeweils die ersten 6 Schritte der Auswertung des Ausdrucks:
   ```
   2 + 3 + f (3 + 5) (9 - 2 * 3) * 6 `div` 2
   ```
   bei
   (a) rechtsapplikativer
   (b) nicht fauler rechtsnormaler
   (c) fauler rechtsnormaler
   Auswertung an. Ein Schritt ist dabei ein Expansionsschritt oder eine Operatoranwendung eines Simplifikationsschritts. Ein Expansionsschritt konsumiert alle Argumente der expandierten Funktion. Simplifikation geht bei gleicher Operatorpriorität rechtsassoziativ vor, wenn nicht explizit angegebene Klammerung anderes verlangt.

Denken Sie daran, die unterfertigte eidesstattliche Erklärung mit abzugeben!

Aufgabe 4 (1+10+5 = 16 Punkte)

Für rationale Zahlen p und q gilt:

- p und q sind *gleich* gdw p und q sind wertgleich (z.B. $\frac{2}{3}$ ist wertgleich zu $\frac{4}{6}$ ist wertgleich zu $\frac{8}{12}$).
- p ist *kleiner/größer* q gdw p ist wertkleiner/wertgrößer als q.
- p ist *kleiner/größer oder gleich* q gdw p ist wertkleiner/wertgrößer als q oder wertgleich zu q.
- p ist in *Normalform* gdw Zähler und Nenner von p sind vollständig gekürzt und höchstens der Zähler ist kleiner als Null; Normalformdarstellung der Null ist $\frac{0}{1}$.

Rationale Zahlen lassen sich wie folgt modellieren:

```
data ZN   = Zaehler | Nenner deriving Eq
newtype Q = Q (ZN -> Int)
```

1. Gibt es `Q`-Werte, die keine rationale Zahl darstellen? Wenn ja, welche?
2. Schreiben Sie eine Haskell-Rechenvorschrift

   ```
   nf :: Q -> Q
   ```

   die ihr Argument in Normalform überführt. Sehen Sie eine Fehlerbehandlung vor, wenn das Argument keine rationale Zahl darstellt.
3. Erklären Sie knapp, aber gut nachvollziehbar, wie ihre Implementierung von `nf` vorgeht.

Aufgabe 5 (3∗4+4+4 = 20 Punkte)

Für Aufgabe 5 kann die Funktion `nf` aus Aufgabe 4 als gegeben vorausgesetzt werden.

1. Machen Sie den Typ der rationalen Zahlen `Q` aus Aufgabe 4 zu einer Instanz der Typklassen:
   (a) `Eq`, indem Sie die Funktion `(==)` implementieren.
   (b) `Ord`, indem Sie die Funktion `(<=)` implementieren.
   (c) `Show`, indem Sie die Funktion `show` implementieren. Rationale Zahlen sollen dabei in Normalform ausgegeben werden, wobei Zähler und Nenner durch einen Schrägstrich getrennt sind, wie nachstehend am Beispiel von `q1`, `q2` mit übereinstimmender Normalform gezeigt ist:

   ```
   q1 :: Q                                  q2 :: Q
   q1 = Q (\x -> case x of Zaehler -> 2     q2 = Q (\x -> case x of Zaehler -> 8
                           Nenner  -> 3)                            Nenner  -> 12)

   show q1 ->> "2/3"                         show q2 ->> "2/3"
   ```

   Sehen Sie eine verschleiernde Fehlerbehandlung vor, wenn Argumente von Funktionen keine rationale Zahlen darstellen.
2. Erklären Sie knapp, aber gut nachvollziehbar, wie ihre Implementierungen der Instanzfunktionen vorgehen und warum die Fehlerbehandlung verschleiernd ist.
3. Ist statt einer verschleierndern auch eine transparente Fehlerbehandlung mit Auffangwerten in Aufgabe 5.1 möglich? Begründen Sie Ihre Antwort für die verschiedenen Instanzfunktionen.