Vergessen Sie nicht, die unterfertigte eidesstattliche Erklärung mit abzugeben!

185.A03 Funktionale Programmierung WS 2020/2021

Schriftlicher Online-Test 2 auf Papier

Freitag, 05.03.2021, 15:00–17:00 Uhr

Aufgabe	**1**	**2**	**3**	**4**	**5**	**6**	**Summe Punkte**
Punkte	19	20	10	18	10	23	100
Erreichte Punkte							

Aufgabe 1 (10+(4+4)+1 = 19 Punkte)

Die *Quersumme* einer natürlichen Zahl $n \geq 0$ ist die Summe ihrer Ziffern.

1. Schreiben Sie eine direkt rekursive Funktion `qs`, die angewendet auf nichtnegative ganze Zahlen deren Quersumme liefert und angewendet auf andere Zahlen eine verschleiernde Fehlerbehandlung vornimmt. Geben Sie auch die Signatur von `qs` an.
2. Erklären Sie knapp, aber gut nachvollziehbar,
   (a) wie Ihre Funktion `qs` vorgeht.
   (b) warum und in welcher Weise die Fehlerbehandlung verschleiernd ist.
3. Welchen Rekursionstyp hat Ihre Funktion `qs`? Woran erkennt man das?

Aufgabe 2 ((8+4)+(4+2)+2 = 20 Punkte)

1. Implementieren Sie `qs` aus Aufgabe 1 nichtrekursiv mithilfe einer Listenkomprehension, ebenfalls mit verschleiernder Fehlerbehandlung:

   ```
   qs n = sum ziffern_von_n where ziffern_von_n = [ . | ... ]
   ```

   Ersetzen Sie dazu die Punkte in `[ . | ... ]` geeignet. Sie dürfen dabei annehmen, dass Ihnen eine Funktion `zz` angewendet auf eine ganze Zahl die Anzahl der Ziffern dieser Zahl liefert. Für die volle Punktzahl müssen Sie auch `zz` einschließlich seiner Typsignatur implementieren.
2. Erklären Sie knapp, aber gut nachvollziehbar, wie Ihre Implementierungen für
   (a) `qs`
   (b) `zz`

   vorgehen, warum die Fehlerbehandlung von `qs` verschleiernd ist und ob sie das in der gleichen oder in einer anderen Weise als in Aufgabe 1 ist.
3. Wie sieht die von Ihrer Implementierung aus Aufgabe 2.1 aufgebaute Liste `ziffern_von_n` aus für den Aufruf:

   ```
   qs 156040
   ```

   Begründen Sie Ihre Antwort.

Aufgabe 3 (2+(3+3)+(1+1) = 10 Punkte)

Eine natürliche Zahl ist entweder Null oder der Nachfolger einer natürlichen Zahl.

1. Definieren Sie einen rekursiven Datentyp, dessen Werte die natürlichen Zahlen in dieser Weise modellieren.
2. Machen Sie Ihren Datentyp mithilfe von `instance`-Deklarationen zu einer Instanz der Typklassen
   (a) `Eq`
   (b) `Ord`

   Implementieren Sie dazu `(==)` oder `(/=)` für `Eq` und `(<=)` oder `compare` für `Ord`.
3. Lässt sich Ihre `instance`-Deklaration für
   (a) `Eq`
   (b) `Ord`

   bedeutungsgleich durch eine `deriving`-Klausel ersetzen? Begründen Sie Ihre Antwort.

Aufgabe 4 (4+4+9+1 = 18 Punkte)

1. Klammern Sie vollständig, aber nicht überflüssig:

   ```
   (\f p x y -> if p x y then f x + succ y else f (pred y) - x) (\z -> z+1) (\q r -> q < r) (2+3*5) 5-3
   ```
2. Geben Sie den allgemeinsten Typ an von:

   ```
   \f p x y -> if p x y then f x + succ y else f (pred y) - x
   ```

   Begründen Sie Ihre Antwort.
3. Geben Sie die ersten 9 Schritte der schrittweisen linksnormalen Auswertung an von:

   ```
   (\f p x y -> if p x y then f x + succ y else f (pred y) - x) (\z -> z+1) (\q r -> q < r) (2+3*5) 5-3
   ```

   Ein Schritt ist dabei ein Expansionsschritt oder eine Operatoranwendung eines Simplifikationsschritts. Eine Funktion konsumiert bei einem Expansionsschritt genau ein Argument.
4. Welchen Wert hat der vollständig ausgewertete Ausdruck aus Aufgabe 4.3?

Aufgabe 5 ((0.5+0.5)+5+4 = 10 Punkte)

Schere-Stein-Papier heißt ein altes Kinderspiel. Zwei Kinder, die gegeneinander spielen, machen auf ein Zeichen hin gleichzeitig eine Handgeste, die für eine Schere, einen Stein oder ein Blatt Papier stehen: ein v-förmig gespreizter Zeige- und Mittelfinger für eine *Schere*, eine Faust für einen *Stein* und eine flach ausgestreckte Hand für ein Blatt *Papier*. Zeigen beide Kinder dieselbe Geste, hat das Spiel keinen Gewinner. Ansonsten gewinnt das Kind, das die 'stärkere' Geste zeigt; diese ist wie folgt bestimmt:

- *Schere* schlägt *Papier* und verliert gegen *Stein*.
- *Stein* schlägt *Schere* und verliert gegen *Papier*.
- *Papier* schlägt *Stein* und verliert gegen *Schere*.

Alice und Bob spielen mehrfach hintereinander gegeneinander. Wer von ihnen gewinnt öfter? Oder gewinnen beide gleich oft? Dabei gilt: Hat Alice weniger Spielzüge (= Gesten) vorbereitet als Bob, werden Bobs überzählig vorbereitete Spielzüge nicht beachtet und umgekehrt. Haben Alice oder Bob gar keinen Spielzug vorbereitet, geht das Gesamtspiel unentschieden aus.

```
type/newtype/data Schere_Stein_Papier = Schere | Stein | Papier
type/newtype/data Kind                = Alice | Bob
type/newtype/data Spielzuege_Alice    = [Schere_Stein_Papier]
type/newtype/data Spielzuege_Bob      = [Schere_Stein_Papier]
type/newtype/data Sieger              = Sieger_ist Kind | Unentschieden

sieger :: Spielzuege_Alice -> Spielzuege_Bob -> Sieger
```

1. Geben Sie das Ergebnis folgender Aufrufe an:
   (a) `sieger [Schere,Stein,Schere,Papier] [Papier,Papier,Papier,Papier]`
   (b) `sieger [Stein,Schere,Papier] [Stein,Papier,Schere]`
2. Mit welchem Schlüsselwort können die verschiedenen Datentypen jeweils eingeführt werden? Streichen Sie jeweils zwei.
3. Ist die Wahl der nichtgestrichenen Schlüsselwörter in der vorigen Teilaufgabe für jeden Typ eindeutig bestimmt oder gibt es für einige Typen mehrere Möglichkeiten, wenn alles andere unverändert bleibt? Begründen Sie Ihre Antwort.

Aufgabe 6 ((10+4)+6+3 = 23 Punkte)

1. (a) Vervollständigen Sie die Implementierung der Funktion

   ```
   sieger :: Spielzuege_Alice -> Spielzuege_Bob -> Sieger
   ```

   entsprechend der Beschreibung in Aufgabe 5.
   (b) Müssen einige Typen Instanzen von Typklassen sein, damit Ihr Programm erfolgreich interpretiert und das Ergebnis von `sieger` auf dem Bildschirm ausgegeben werden kann? Wenn ja, nehmen Sie mit `deriving`-Klauseln oder/und `instance`-Deklarationen die nötigen Instantiierungen vor; aber keine über die unbedingt erforderlichen hinaus.
2. Erklären Sie knapp, aber gut nachvollziehbar, wie ihre Funktion `sieger` vorgeht.
3. Geben Sie den Aufrufgraphen der Rechenvorschriften an, die Sie für Aufgabe 6 implementiert haben.

Vergessen Sie nicht, die unterfertigte eidesstattliche Erklärung mit abzugeben!