Algorithmieren und Programmieren

Sommersemester 2023

Lehrstuhl Programmiersprachen und Compilerbau
Prof. Dr. rer. nat. habil. Petra Hofstedt
Ilja Becker
Andreas Eberle, Jan Robert Meyer, Oliver Pflug

Brandenburgische
Technische Universität
Cottbus - Senftenberg

Übungsblatt 9

Abgabedatum: 02.07.2023

Hinweise

- Beachten Sie, dass die Tutoren unkommentierte Programme nicht als Lösung akzeptieren, selbst wenn die Programme richtig funktionieren. Zu einer richtigen Lösung gehören immer sinnvolle Kommentare, deren Umfang der Komplexität des Programms angemessen ist.
- **Halten Sie sich an die in den Übungsblättern vorgegebenen Namen von Funktionen und Dateien, Funktionstypen (Typsignaturen), Reihenfolge der Parameter und verwenden Sie — sofern vorhanden — die Vorgaben!**
- Auf den Übungsblättern finden Sie einige Haskell-Quelltextfragmente. Diese sind, der besseren Lesbarkeit wegen, unter Nutzung einiger mathematischer Sonderzeichen wiedergegeben.
- Für diese Veranstaltung wird die Verwendung des ***Glasgow Haskell Compiler (GHC)*** (`https://www.haskell.org/`) empfohlen. Weitere Informationen finden Sie im Software-Reiter auf Moodle.
- Als **Tutoriumsaufgabe** markierte (Teil-)aufgaben werden in den Übungen ausführlicher besprochen. Die schriftlichen bzw. elektronischen Lösungen müssen jedoch trotzdem mit abgegeben werden. Bitte schauen Sie Sich diese Aufgaben im Vorfeld der Übung an und bereiten Sie Sich darauf vor.
- Die Abgabe Ihrer Lösungen erfolgt vor Ablauf der Abgabefrist digital über die Moodle-Plattform an Ihren Tutor. Erstellen Sie dazu ein PDF-Dokument, das die Lösungen Ihrer schriftlichen Aufgaben enthält. Laden Sie dieses PDF-Dokument und die erzeugten Hakell-Dateien, mit den in den Aufgaben vorgegebenen Namen, bei Moodle hoch.

Informationsquellen

- Sie finden unter `http://haskell.org/` sehr viele Informationen über die Programmiersprache Haskell. Von besonderem Interesse sind dabei sicherlich die Übersicht über zahlreiche online verfügbare Haskell-Tutorials (`https://wiki.haskell.org/Tutorials`) sowie die Suchmaschine Hoogle (`http://hoogle.haskell.org/`) für die Haskell-API, die Ihnen mit zunehmender Haskell-Erfahrung wertvolle Dienste leisten wird.

Sie können maximal **(7 Punkte)** mit diesem Übungsblatt erreichen.

Aufgabe 1 (Induktion über den natürlichen Zahlen) 2 Punkte

Beweisen Sie mittels Induktion über den natürlichen Zahlen, dass die im Folgenden definierten Prädikate P_i für alle natürlichen Zahlen gültig sind. Beachten Sie dabei, dass für beliebige Funktionen $f : \mathbb{N} \to \mathbb{N}$ und natürliche Zahlen m und n mit $m > n$ die Gleichung $\sum_{i=m}^{n} f(i) = 0$ gilt.

1. $P_1(n) \Leftrightarrow \sum_{i=1}^{n} (2i - 1) = n^2$
2. **(Tutoriumsaufgabe)** $P_2(n) \Leftrightarrow \sum_{i=1}^{n} i^3 = (\frac{n \cdot (n+1)}{2})^2$
3. $P_3(n) \Leftrightarrow \prod_{i=0}^{n-1} (1 + z^{2^i}) = \frac{1 - z^{2^n}}{1 - z}$ wobei $z \neq 1$

Aufgabe 2 (Strukturelle Induktion über einen rekursiven Datentyp) 1 Punkt

Die natürlichen Zahlen können durch einen rekursiven Datentyp `Nat` beschrieben werden, indem ausgehend vom Element `Zero` für die Zahl 0 fortlaufend alle Nachfolger durch Aneinanderreihung von Successor-Elementen `Succ` entstehen. Auf diese Weise kann man die natürlichen Zahlen nacheinander abzählen, ohne jedoch an ein Ende zu kommen, denn die Menge der natürlichen Zahlen ist abzählbar unendlich groß. Der rekursive Datentyp

```
data Nat =  Zero|
            Succ Nat
```

beschreibt die natürlichen Zahlen wie folgt:

`Zero`	0
`Succ Zero`	1
`Succ Succ Zero`	2
⋮	⋮

1. **(Tutoriumsaufgabe)** Geben Sie eine rekursive Funktion `add :: Nat →Nat →Nat` an, die zwei gegebene natürliche Zahlen addiert. Sie können dabei das Rekursionsschema der Addition ausnutzen, wonach $x + 0 = x$ und $x + (y + 1) = (x + y) + 1$. In diesem Rekursionsschema wird die Addition der Zahlen x und $(y + 1)$ auf das kleinere Problem der Addition von x und y zurückgeführt, bis y zu 0 abgebaut ist. Der Term „+1“ symbolisiert jeweils die Nachfolgerbildung.
2. **(Tutoriumsaufgabe)** Das Assoziativgesetz der Addition besagt, dass $(x + y) + z = x + (y + z)$ für beliebige natürliche Zahlen x, y, z. Beweisen Sie mittels struktureller Induktion, dass für die Addition natürlicher Zahlen vom Typ `Nat` ebenso das Assoziativgesetz gilt, also:

   ```
   add (add x y) z = add x (add y z)
   ```

Aufgabe 3 (Induktion über Listen) 2 Punkte

Betrachten Sie die folgenden Funktionsdefinitionen:

```
(++) :: [el] → [el] → [el]
[] ++ liste = liste
(el : els) ++ liste = el : (els ++ liste)

length :: [el] → Int
length [] = 0
length (_ : els) = 1 + length els

reverse :: [el] → [el]
reverse [] = []
reverse (el : els) = reverse els ++ [el]
```

Beweisen Sie mittels struktureller Induktion folgende Aussagen:

1. **(Tutoriumsaufgabe)** `liste ++ [] = liste`
2. **(Tutoriumsaufgabe)** `length (liste1 ++ liste2) = length liste1 + length liste2`
3. `reverse liste1 ++ reverse liste2 = reverse (liste2 ++ liste1)`
4. `reverse (reverse liste) = liste`

Aufgabe 4 (Induktion über Bäumen) 2 Punkte

Wir betrachten wieder die Definition des algebraischen Datentyps `Baum`:

```
data Baum el =  Blatt el |
                Verzweigung (Baum el) (Baum el)
```

Die Funktionen `blaetter` und `hoehe` seien wie folgt definiert:

```
blaetter :: Baum el → [el]
blaetter (Blatt el) = [el]
blaetter (Verzweigung links rechts) = blaetter links ++ blaetter rechts

hoehe :: Baum el → Int
hoehe (Blatt _) = 0
hoehe (Verzweigung links rechts) = 1 + max (hoehe links) (hoehe rechts)
```

Beweisen Sie mittels struktureller Induktion die Gültigkeit der Aussage

$$length(blaetter\ baum) \leq 2^{(hoehe\ baum)}$$

Verwenden Sie dazu neben den obigen Definitionen die Definitionen von (++) und `length` aus Aufgabe 3.