Technische Universität Dresden
Institut für Theoretische Informatik
Dr. Monika Sturm

Wintersemester 08/09

Funktionale Programmierung und Typtheorie

4. Übungsblatt

Hinweis

Am 20.11.2008 findet die Übung zur Lehrveranstaltung im Rechnerraum E046 statt. Die Besprechung der Aufgaben erfolgt zum Teil individuell während der Übung bzw. in einer abschliessenden Form in der Gruppe im Übungsraum.

Aufgabe 1

Kapitel 2 – polymorpher λ-Kalkül $\lambda 2$

Ein Typ *Nat* für natürliche Zahlen ist in $\lambda 2$ (explizite Typisierung) mit $Nat := \forall a.(a \to a) \to a \to a$ definiert. Mithilfe von *Nat* werden Funktionen über natürlichen Zahlen in $\lambda 2$ getypt, wie z.B. die Nachfolgerfunktion *suc* und die Ackermannfunktion *ack*.

a) Zeigen Sie mittels Herleitungsbaum, dass der Term

$$suc \equiv \lambda n : Nat. \Lambda a. \lambda f : a \to a. \lambda x : a. f\,(n\,a\,f\,x) : Nat \to Nat$$

wohlgetypt ist.

b) Konstruieren Sie einen Typ für die *Ackermannfunktion*. Benutzen Sie dazu den Term

$$ack \equiv (\lambda m.m\,(\lambda f n.n\ f\ (f\ \underline{1}))\ suc)$$

aus der Vorlesung.

Aufgabe 2

Kapitel 3 – rekursive Funktionsdefinitionen

a) Definieren Sie eine Funktion `power :: Int -> Int -> Int`.
`power k n` liefert den Wert k^n. Es gilt $n \geq 0$.

b) Geben Sie eine alternative Definition der Funktion `power` an, die folgende Gleichungen ausnutzt:

$$\begin{aligned} k^{2n} &= (k^n)^2 \\ k^{2n+1} &= k * (k^n)^2. \end{aligned}$$

Die Funktionen `div, mod :: Int -> Int -> Int` sind in Haskell vordefinierte Präfixfunktionen für die ganzzahlige Division und die Restbildung.

Aufgabe 3

Kapitel 3 – rekursive Funktionsdefinitionen
Implementieren Sie Sortierfunktionen für Listen von `Int`-Zahlen, die mit den folgenden Algorithmen arbeiten:

a) Insertion-Sort (Einfügen in sortierte Liste),

b) Bubble-Sort (Vertauschen benachbarter Elemente),

c) Merge-Sort (Verschmelzen schon sortierter Teillisten).

Aufgabe 4

Kapitel 3 – rekursive Funktionsdefinitionen
Führen Sie als neuen Datentyp den parametrischen binären Baum ein und definieren Sie folgende über diesem Datentyp operierende Funktionen:

a) Breite und Tiefe

b) Knotenanzahl

c) Test auf Balanciertheit

d) Generierung einer Liste mit allen Einträge an den Knoten

e) Generierung einer Liste mit allen Einträge an den Blättern

f) Löschen aller Knoteneinträge

g) Löschen aller Blatteinträge

h) `tree_map` (in Analogie zu `map` über Listen)

i) `tree_foldr` (in Analogie zu `foldr` über Listen).

Hinweis: Ein binärer Baum wird balanciert genannt, wenn sich für jeden Verzweigungsknoten die Tiefen des linken und rechten Teilbaums höchstens um 1 unterscheiden.

Aufgabe 5

Kapitel 4 – Polymorphie und Typinferenz
Geben Sie die Typen folgender Ausdrücke an:

a) `map map`

b) `map foldr`

c) `foldr map`

Aufgabe 6

Kapitel 4 – Polymorphie und Typinferenz
Geben Sie Funktionen mit folgenden Typen an:

a) `a -> Bool`

b) `a -> a -> b -> (b,a)`

c) `a -> b`

Aufgabe 7

Kapitel 4 – Polymorphie und Typinferenz
Im Schritt 3 der Typinferenz (in der Vorlesung) wird ein Unifikationsalgorithmus vorgestellt. Wenden Sie diesen Algorithmus zur Typinferenz der Funktion `span` an. Dabei ist `span` wie folgt definiert:

```
span p [] = ([],[])
span p (x:xs) = let (ys,zs) = span p xs
                in if p x then (x:ys, zs)
                    else ([], x:xs)
```

Welche Funktion realisiert `span`?

Aufgabe 8

Kapitel 5 – unendliche Listen

a) Definieren Sie eine Funktion `zeroone :: [String]`, die die Liste aller Zeichenketten der Form 0^n1^n für $n = 0, 1, 2, \ldots$ in aufsteigender Länge erzeugt.

b) Definieren Sie Funktionen zur Berechnung der i-ten Fibonacci-Zahl und zur Generierung einer Folge von Fibonacci-Zahlen bis zur i-ten. Geben Sie zusätzlich eine repetitiv rekursive Version zur ersten Funktion an.

Aufgabe 9

Kapitel 5 – Funktionen höherer Ordnung, Listenabstraktionen
Die Funktion `numChar :: Char -> String -> Int` liefert die Anzahl der Vorkommen eines Buchstabens in einer Zeichenkette. Definieren Sie diese Funktion

a) mit Pattern Matching und Rekursion,

b) mit `foldr` und

c) mithilfe einer Listenabstraktion.