PHILIPPS-UNIVERSITÄT MARBURG
Fachbereich Mathematik und Informatik
Prof. Dr. R. Loogen

D-35032 Marburg
Hans Meerwein Straße
Lahnberge

Übungen zur „Praktischen Informatik III“, WS 2005/06

Nr. 2, Abgabe: 8. November 2005 vor der Vorlesung

A. Hausaufgaben

Die Lösungen sollten grundsätzlich schriftlich, Programme zusätzlich auf Diskette oder per E-Mail an Ihren Tutor oder Ihre Tutorin abgegeben werden. Die Abgabe ist in Gruppen bis zu zwei Personen erlaubt.

5. Konkrete vs. abstrakte Syntax **3 Punkte**

   Gegeben sei die folgende Grammatik G_T zur Erzeugung von einfachen Typausdrücken in Haskell.

   `T` $\longrightarrow$ `Int` | `Bool` | `(T,T)` | `(T -> T)`

   (a) Geben Sie den Syntaxbaum zu dem folgenden Typausdruck an. / 1
   `((Bool -> Int) -> (Bool -> (Int, Bool)))`

   (b) Bestimmen Sie eine abstrakte Syntaxbeschreibung zu G_T . / 1

   (c) Geben Sie den abstrakten Syntaxbaum für den Typausdruck aus (a) an. / 1

6. (a) Definieren Sie eine Funktion **5 Punkte**

   `power :: Int -> Int -> Int` / 1

   so daß `(power k n)` den Wert k^n liefert. Gehen Sie davon aus, dass $n \geq 0$ gilt.
   Der vordefinierte Operator `(^) :: Int->Int->Int` darf nicht verwendet werden.

   (b) Geben Sie eine ausführliche Auswertung des Ausdrucks `(power 3 3)` an. / 1

   (c) Geben Sie eine alternative Definition der Funktion `power` an, die folgende Gleichungen ausnutzt: / 3

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

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

7. Zeichenkettenverarbeitung **4 Punkte**

   Definieren Sie die folgenden Funktionen über Zeichenketten, die in Haskell als Listen von Zeichen realisiert sind: `type String = [Char]`.

   (a) `delete :: String -> Int -> String` / 2
   entfernt beim Aufruf `(delete text i)` das `i`-te Zeichen von `text`.

   (b) `insert :: String -> Int -> String -> String` / 2
   fügt beim Aufruf `(insert t1 i t2)` in `t1` ab der Position `i` `t2` ein.

B. Mündliche Aufgaben

8. Die Funktion `almostEqual :: (Int, Int) -> (Int, Int) -> Bool` vergleicht die Werte von zwei Wertepaaren des Typs `Int` miteinander. Sie liefert das Ergebnis `True`, falls beide Paare dieselben Werte enthalten, wobei deren Reihenfolge nicht von Belang ist. Beispielsweise gilt:

   ```
   almostEqual (3,4) (4,3)   liefert   True
   almostEqual (3,4) (3,5)   liefert   False
   ```

   Welche der folgenden Definitionen geben den korrekten Wert zurück? Welche Definitionen halten Sie für einen guten Programmierstil? Warum? Fügen Sie den Definitionen Kommentare hinzu, damit diese verständlicher und leichter lesbar werden.

   ```
   almostEqual1 (x1,y1) (x2,y2)
     | (x1 == x2) && (y1 == y2) = True
     | (x1 == y2) && (y1 == x2) = True
     | otherwise                = False

   almostEqual2 (x1,y1) (x2,y2)
     | (x1 == x2) = (y1 == y2)
     | (x1 == y2) = (y1 == x2)
     | otherwise  = False

   almostEqual3 pair1 pair2
     = (pair1 == pair2) ||
       (pair1 == swap pair2)
     where swap (x,y) = (y,x)

   almostEqual4 pair1 pair2
     = (pair1 == pair2) ||
       (swap pair1 == swap pair2)
     where swap (x,y) = (y,x)

   almostEqual5 (x1,y1) (x2,y2)
     = if (x1 == x2)
       then if (y1 == y2)
            then True
            else False
       else if (x1 == y2)
            then if (x2 == y1)
                 then True
                 else False
            else False
   ```

9. Listenkonstruktion mit `":"` und `"++"`

   Welche der folgenden Gleichungen auf Listen sind richtig? Begründen Sie kurz Ihre Antwort.

   (a) `x:y:z = [x,y,z]`
   (b) `xs:[] = [xs]`
   (c) `[]:xs = xs`
   (d) `(x:xs)++ys = x:(xs++ys)`
   (e) `[[]]++xs = [[],xs]`