Informatik A, WS 2016/17 — 14. Übungsblatt

Abgabe bis Freitag, 3. Februar 2017, 12:00 Uhr in die Fächer der Tutor/inn/en

80. Teilsummen, 0 Punkte

(a) Die vordefinierte Funktion `scanl` ist ähnlich wie `foldl`, aber es wird eine Liste aller Zwischenergebnisse erzeugt:

```
scanl,scanl2 :: (a -> b -> a) -> a -> [b] -> [a]
scanl f q []     = [q]
scanl f q (x:xs) = q : scanl f (f q x) xs
```

(b) Was ist das Ergebnis von `scanl (+) 0 [3,5,-1,7]`?

(c) Weil auf der rechten Seite der Definition nur ein einziger rekursiver Aufruf mit einer um 1 kürzeren Liste steht, kann `scanl` mit einer Liste der Länge n in $O(n)$ Zeit berechnet werden, sofern die Funktion `f` in konstanter Zeit (in $O(1)$ Zeit) berechnet werden kann.
Wie lange dauert die Berechnung mit dem folgenden einfachen Programm?

```
scanl2 f q xs = [foldl f q (take i xs) | i <- [0..length xs]]
```

Geben Sie die Laufzeit mit O-Notation an.

(d) Ab welcher Listenlänge macht sich der Unterschied zwischen `scanl (+) 0` und `scanl2 (+) 0` in der Laufzeit bemerkbar, wenn man das Programm einmal in der Befehlszeile aufruft? Lassen Sie die Summe der Ergebnisliste ausrechnen, damit der Bildschirm nicht überschwemmt wird.

(e) Wie lange dauert die Berechnung von `length (scanl (+) 0 xs)` beziehungsweise `length (scanl2 (+) 0 xs)`, wenn `xs` eine Liste der Länge n ist?

81. Der Teilabschnitt mit der größten Summe, 10 Punkte

Die folgenden drei Funktionen `maxSum1, maxSum2, maxSum3 :: (Num a, Ord a) => [a] -> a` bestimmen für eine Liste $[a_1, \ldots, a_n]$ die größte Summe $a_i + a_{i+1} + \cdots + a_j$ einer in ihr enthaltenen *zusammenhängenden* Teilfolge.

```
maxSum1 list = maximum (0: [su i j | j <- [1..length(list)],
        i <- [1..j]])    -- direkt nach Definition
  where su a b = sum (drop (a-1) (take b list))
maxSum2 [] = 0 -- mit "scan"
maxSum2 (x:xs) = max (maxSumStart (x:xs)) (maxSum2 xs)
maxSumStart :: (Num a, Ord a) => [a] -> a
-- Teilfolgen, die am Anfang von xs beginnen:
maxSumStart xs = maximum (scanl (+) 0 xs)
maxSum3 = maxSumA 0  -- die größte Summe ist mindestens 0
maxSumA :: (Num a, Ord a) => a -> [a] -> a
maxSumA akk [] = akk -- ohne Erklärung. akk steht für "Akkumulator"
maxSumA akk (x:xs) | x <= -akk = max akk (maxSumA 0  xs)
                   | otherwise = max akk (maxSumA (akk+x) xs)
```

Bestimmen Sie eine obere Schranke für die Laufzeit für jede der drei Funktionen in Abhängigkeit von der Listenlänge n, in O-Notation. Begründen Sie Ihre Antworten.

82. Laufzeiten, 0 Punkte

Wir haben fünf Algorithmen, deren Laufzeit in Abhängigkeit von der Größe n der Eingabe auf einem bestimmten Rechner (1) $10n$, (2) n^3+5n, (3) $2^n/100$, (4) $3n^2+4n$, (5) $5n \log_2 n$ Mikrosekunden ist.

(a) Wie große Probleme kann man mit diesen Algorithmen (i) in einer Sekunde, (ii) in 1 Stunde, (iii) in 1 Woche lösen?

(b) Wie ändert sich die Antwort bei einem Rechner, der tausendmal so schnell ist?

83. Typklassen und Typinferenz, 0 Punkte

Bestimmen Sie die Typen der Standardfunktionen `curry f x y = f(x,y)`, `uncurry f (x,y) = f x y`, und `id x = x`. Stellen Sie fest, welche der folgenden zwölf Funktionen gültig sind, und bestimmen Sie gegebenfalls ihren Typ:

```
concat . map show                map . (++)
curry id                         uncurry curry
uncurry id                       curry uncurry
curry (curry id)                 uncurry uncurry
uncurry (uncurry id)             curry curry
uncurry (<=)                     uncurry . (==)
```

Verwenden Sie den Computer höchstens zum Überprüfen Ihrer Antworten. Füttern Sie jede gültige Funktion mit den erforderlichen Argumenten, sodass HASKELL ein Ergebnis ausgibt.

84. Typklassen und Typinferenz, 10 Punkte

Stellen Sie fest, welche der folgenden vier Funktionen gültig sind, und bestimmen Sie gegebenfalls ihren Typ, wobei `flip f x y = f y x` ist.

```
map reverse                      flip . curry
curry flip                       flip flip
```

Leiten Sie Schritt für Schritt her, wie sich der Typ aus den Restriktionen des Typsystems ergibt. Verwenden Sie den Computer höchstens zum Überprüfen der Antworten.

Gegeben Sie für jede gültige Funktion einen Beispielaufruf mit den nötigen zusätzlichen Argumenten an, für den HASKELL ein Ergebnis ausdruckt.

85. Ausgeglichene Klammern, Programmieraufgabe, 10 Punkte

Schreiben Sie eine Funktion `klammerTest`, die testet, ob die in einer Zeichenkette vorkommenden öffnenden und schließenden runden Klammern „`(`“ und „`)`“ einen korrekten Klammerausdruck bilden. Das bedeutet: in jedem Anfangsstück (Präfix) treten mindestens so viele öffnende wie schließende Klammern auf, und insgesamt gibt es ebenso viele öffnende wie schließende Klammern. Beispielsweise sind `()(())` und `((())()(()))` gültig, nicht aber `(()(` oder `)()()(`.

86. Induktion, 0 Punkte

Für welche Werte von k und l ist die Gleichung

```
(take k) . (take l) = take (min k l).
```

gültig? Beweisen Sie sie unter Verwendung folgender Definitionen:

```
take :: Int -> [a] -> [a]
take n _   | n <= 0 =  []                  -- take.0
take _ []           =  []                  -- take.1
take n (x:xs)       =  x : take (n-1) xs   -- take.2
min x y  | x <= y    =  x                  -- min.1
         | otherwise =  y                  -- min.2
```

Es bietet sich eine Fallunterscheidung an, je nachdem ob k oder l größer ist.