WS 2017/2018

Funktionale Programmierung

3. Übungsblatt

Prof. Dr. Margarita Esponda

Ziel: Auseinandersetzung mit Listen und rekursiven Funktionsdefinitionen.

1. Aufgabe (4 Punkte)

Definieren Sie eine **arrow** Funktion, die bei Eingabe zweier natürlichen Zahlen **k** und **n**, die Zahl **k** **n**-mal potenziert. D.h. Die Anzahl der **k**-Elemente in dem Potenz-Turm ist gleich **n**. Vergessen sie dabei nicht, dass die Potenz-Funktion rechtsassoziativ ist.

$k \uparrow n = k^{k^{\cdot^{\cdot^{k}}}}$ (der Turm auf der rechten Seite der Gleichung hat **n** mal das Argument **k**)

2. Aufgabe (4 Punkte)

Eine Mersenne Zahl ist eine Zahl der Form 2^n-1. Schreiben Sie eine rekursive Haskell-Funktion, die bei Eingabe einer natürlichen Zahl **n** die ersten **n** Mersenne-Zahlen in einer Liste zurückgibt.

Anwendungsbeispiel:

mersenne 7 ⇒ [0, 1, 3, 7, 15, 31, 63]

3. Aufgabe (4 Punkte)

Schreiben Sie eine Funktion **chars2words**, die aus einem einfachen **String** die Liste aller Worte des Textes zurückgibt. Der Text besteht nur aus Buchstaben, Zahlen und Trennzeichen. Mit Trennzeichen sind Kommata, Punkte, Fragezeichen, Ausrufezeichen und Leerzeichen gemeint.

Die Funktion soll folgende Signatur haben:

chars2words :: [Char] -> [[Char]]

4. Aufgabe (6 Punkte)

Definieren Sie eine Haskell-Funktion, die aus einer beliebigen Binärzahl ***n*** (Zweierkomplement-Darstellung) die entsprechende negative Zahl (***-n***) berechnet. Es wird selbstverständlich angenommen, dass die Eingabe- und Ergebniszahl der Funktion immer die gleiche Bitlänge haben.

Anwendungsbeispiel: twoComplement [0,0,0,1,1,0,1,0] => [1,1,1,0,0,1,1,0]

5. Aufgabe (14 Punkte)

Wenn wir eine Menge ***M*** mit ***n*** verschiedenen Objekten haben, kann die Anzahl der verschiedenen ***k***-elementigen Teilmengen aus ***M*** mit Hilfe des bekannten Binomialkoeffizienten wie folgt berechnet werden.

Binomialkoeffizient (n, k) = $\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$ mit $0 \le k \le n$

a) Schreiben Sie eine Haskell-Funktion mit folgender Signatur

binom_naiv :: Integer -> Integer -> Integer

die mit Hilfe der vorherigen Definition und ohne Rekursion (außer innerhalb der Fakultätsfunktion) für beliebige natürliche Zahlen ***n*** und ***k*** den Binomialkoeffizienten berechnet.

Eine rekursive Definition der gleichen Funktion sieht wie folgt aus:

$$\binom{n}{k} = 0, \quad wenn \quad k > n$$

$$\binom{n}{0} = \binom{n}{n} = 1$$

$$\binom{n}{1} = \binom{n}{n-1} = n$$

$$\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1} \quad \textit{für alle} \quad n,k \in \mathrm{N} \quad \textit{mit} \quad 0 < k \leq n-1$$

b) Definieren Sie eine rekursive Haskell-Funktion dafür.

c) Aus der ersten Definition von a) kann folgende Gleichung abgeleitet werden:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \begin{cases} 1 & falls \quad k = 0 \\ \frac{n}{1} \cdot \frac{(n-1)}{2} \cdot \frac{(n-2)}{3} \cdots \frac{(n-k+2)}{(k-1)} \cdot \frac{(n-k+1)}{k} & falls \quad k > 0 \end{cases}$$

Definieren Sie eine möglichst effiziente rekursive Haskell-Funktion, die diese Definition des Binomialkoeffizienten verwendet.

d) Schreiben Sie eine Test-Funktion, die überprüft, dass alle drei Funktionen das gleiche Ergebnis liefern.

Wichtige Hinweise:

1) Verwenden Sie geeignete Namen für Ihre Variablen und Funktionsnamen, die den semantischen Inhalt der Variablen oder die Semantik der Funktionen wiedergeben.
2) Verwenden Sie vorgegebene Funktionsnamen, falls diese angegeben werden.
3) Kommentieren Sie Ihre Programme.
4) Verwenden Sie geeignete lokale Funktionen und Hilfsfunktionen in Ihren Funktionsdefinitionen.
5) Schreiben Sie in alle Funktionen die entsprechende Signatur.