Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
L. Noschinski, L. Hupel, Dr. J. Blanchette, M. Eberl

WS 2014/15
31.10.2014
Abgabe: 07.11.2014

Einführung in die Informatik 2
4. Übung

Aufgabe G4.1 Fibonacci-Liste

Eine Liste $[x_1, \ldots, x_n]$ hat die *Fibonacci-Eigenschaft*, wenn für alle $1 \leq i \leq n-2$ gilt $x_{i+2} = x_i + x_{i+1}$.

Beispiel: Die Listen $[0, 1, 1, 2, 3]$, $[3, -2]$, $[-3, 1, -2, -1, -3]$ und [] haben die Fibonacci-Eigenschaft; die Liste $[2, 1, 1]$ hat sie nicht.

Schreiben Sie eine rekursive Funktion `hasFibonacciProperty :: [Integer] -> Bool`, die genau dann `True` zurückgibt, wenn die Eingabeliste die Fibonacci-Eigenschaft hat.

Aufgabe G4.2 Umdrehung von `snoc`

Beweisen Sie die Gleichung

```
reverse (snoc xs x) = x : reverse xs
```

per Induktion. Halten Sie sich an der Induktionsvorlage auf der Übungsseite. In unserem Submission-System finden Sie ein Online-Tool um Ihre Beweise zu überprüfen.

Die Funktionen `(++) :: [a] -> [a] -> [a]`, `reverse :: [a] -> [a]` und `snoc :: [a] -> a -> [a]` sind wie folgt definiert:

```
[] ++ ys = ys
(x : xs) ++ ys = x : (xs ++ ys)

reverse [] = []
reverse (x : xs) = reverse xs ++ [x]

snoc [] y = [y]
snoc (x : xs) y = x : snoc xs y
```

Verwenden Sie in jedem Schritt *nur eine* dieser definierenden Gleichungen oder die Induktionshypothese und vergessen Sie nicht, den Namen der angewendeten Gleichung anzugeben.

Aufgabe G4.3 Permutationen

Gesucht ist eine Funktion `perms :: [Char] -> [[Char]]`, die alle Permutationen eines Wortes in umgedrehter alphabetischen (oder "ASCII-betischen") Reihenfolge zurückliefert. Die Liste der Permutationen soll keine Duplikate enthalten. Zum Beispiel:

```
perms "" == [""]
perms "a" == ["a"]
perms "ab" == ["ba","ab"]
perms "ba" == ["ba","ab"]
perms "aba" == ["baa","aba","aab"]
perms "abc" == ["cba","cab","bca","bac","acb","abc"]
perms "yhwh" ==
    ["ywhh","yhwh","yhhw","wyhh","whyh","whhy",
     "hywh","hyhw","hwyh","hwhy","hhyw","hhwy"]
perms "xxxxxxxxxxxxxxx" == ["xxxxxxxxxxxxxxx"]
```

Sie dürfen dafür die folgenden Funktionen aus der `List` Standardbibliothek benutzen, wo `a` einen beliebigen Typ darstellt (z.B. `Integer`, `Char` oder `[Char]`):

```
delete :: a -> [a] -> [a]  -- delete 3 [3,1,3] == [1,3]
nub :: [a] -> [a]          -- nub [1,3,1,2,2] == [1,3,2]
reverse :: [a] -> [a]      -- reverse [1,3,2] == [2,3,1]
sort :: [a] -> [a]         -- sort [1,3,2] == [1,2,3]
```

Basisfunktionen wie `++` sind erlaubt, `List.permutations` aber nicht!

Aufgabe G4.4 Mustervergleich (optional)

Implementieren Sie eine Funktion `match :: [Char] -> [Char] -> Bool`, die einen Zielstring mit einem Muster vergleicht. Das Muster ist selbst ein String, in dem das Sonderzeichen `?` ein beliebiges Zeichen und `*` eine beliebige Sequenz von (null oder mehr) Zeichen darstellt. Das Muster muss den vollständigen Zielstring erkennen. Im Aufruf `match ps ys` ist `ps` das Muster und `ys` der Zielstring.

Die folgenden Beispiele sollen alle `True` sein:

```
match "abc" "abc"              not (match "ab" "abc")
match "?bc" "abc"              not (match "a" "")
match "*" "abc"                not (match "a*b" "bba")
match "a*f" "abcdef"           not (match "a*f" "abcde")
match "a***b?" "abc"           not (match "a***b" "abc")
```

Wichtig: Wenn nicht explizit als "optional" markiert, ist die Wettbewerbsaufgabe immer auch als reguläre Hausaufgabe zu bearbeiten.

Aufgabe H4.1 Summe aller vorherigen Elemente

Schreiben Sie eine Funktion

```
allSumOfPrevious :: [Integer] -> Bool
```

die für eine gegebene Liste `xs` zurückgibt, ob jedes Element in dieser Liste (außer dem ersten) die Summe aller vorherigen Elemente ist.

Beispiel:

```
allSumOfPrevious [] == True
allSumOfPrevious [42] == True
allSumOfPrevious [3, 3, 6, 12] == True

allSumOfPrevious [1,0] == False
allSumOfPrevious [3, 3, 6, 10] == False
```

Aufgabe H4.2 Listen zerlegen

Wie man Listen verknüpft, ist in den bisherigen Aufgaben schon oft besprochen worden. In dieser Aufgabe geht es um den umgekehrten Weg: Eine Liste an bestimmten Punkten auftrennen. Gegeben sei eine Ursprungsliste und eine Liste von *Trennzeichen*. Gesucht ist eine Funktion `splitter :: Eq a => [a] -> [a] -> [[a]]`, die die Ursprungsliste anhand der Trennzeichen zerlegt. Die Trennzeichen sollen dabei der Reihe nach in der Ursprungsliste gesucht werden. Sofern eines oder mehrere Trennzeichen nicht gefunden werden, soll die verbleibende Liste nicht weiter aufgeteilt werden.

Beispiele:

```
splitter "aa$bb$cc:dd" ['$', '$', ':'] == ["aa", "bb", "cc", "dd"]
splitter "aa$bb$cc:dd" ['$', ':'] == ["aa", "bb$cc", "dd"]
splitter "aa$bb$cc:dd" ['#', '$', ':'] ==  ["aa$bb$cc:dd"]

splitter "Hello World!" [' ', 'o'] == ["Hello", "W", "rld!"]
splitter "Hello World!" ['x'] == ["Hello World!"]
splitter "Hello World!" [' ', 'x'] == ["Hello", "World!"]
splitter "Hello World!" ['H', 'e'] == ["", "", "llo World!"]

splitter "" ['x'] == []
```

Für den Fall, dass das letzte Element der Ursprungsliste zugleich ein gesuchtes Trennzeichen ist, sind folgende beiden Verhalten erlaubt:

```
splitter "x" "x" == [""]
splitter "abc" "c" == ["ab"]
splitter "abcabc" "cc" == ["ab","ab"]
```

bzw.

```
splitter "x" "x" == ["", ""]
splitter "abc" "c" == ["ab", ""]
splitter "abcabc" "cc" == ["ab","ab", ""]
```

1. Implementieren Sie diese Funktion.
2. Überlegen Sie sich Eigenschaften, die für diese Funktion gelten und implementieren Sie diese als QuickCheck-Tests.

Aufgabe H4.3 Consil

Beweisen Sie die Gleichung

```
snoc xs x = xs ++ [x]
```

per Induktion. Folgen Sie den Anweisungen aus G4.2 und laden Sie Ihren Beweis in einer Datei mit Namen `snoc.cprf` hoch.

Die Funktionen `(++) :: [a] -> [a] -> [a]` und `snoc :: [a] -> a -> [a]` sind wie folgt definiert:

```
[] ++ ys = ys
(x : xs) ++ ys = x : (xs ++ ys)

snoc [] y = [y]
snoc (x : xs) y = x : snoc xs y
```

Aufgabe H4.4 Neuer Auftrag aus Stuttgart (Wettbewerbsaufgabe)

> **Wichtig für Wettbewerbsteilnehmer:** In dieser Woche werden wir erstmals den Wettbewerb gemeinsam mit dem Karlsruher Institut für Technologie ausrichten. Für Studenten des KIT: Sie brauchen nur diese Aufgabe zu bearbeiten. Bitte schreiben Sie zusätzlich Ihren Namen innerhalb der {-WETT-}-Tags, sonst können wir Ihre Lösung nicht werten.

Vor zwei Wochen haben wir im Auftrag einer Stuttgarter Firma ausgearbeitet, wie man zwei Zahlen in eine kodiert und wieder dekodiert. Diese Woche ist unsere Aufgabe noch anspruchsvoller: Es geht darum, zwei Funktionen zu implementieren, die mit ganzen Listen arbeiten. Die Funktionen müssen die Typsignatur

```
encodeInts :: [Integer] -> Integer
decodeInts :: Integer -> [Integer]
```

haben und, für alle endlichen Listen `ns`, die folgende tolle Eigenschaft haben:

```
decodeInts (encodeInts ns) = ns
```

Für Ihre Implementierung dürfen Sie sich eine beliebige Kodierung ausdenken. Eine Möglichkeit wäre, die vor zwei Wochen implementierte `decodeIntPair` und eine passende `encodeIntPair` Funktion zu benutzen. Nach diesem Prinzip kann man die Liste `[81, 23, 43, 14]` als $(4, (81, (23, (43, 14))))$ ansehen, wobei das erste Element (4) die Länge angibt. Die Kodierung kann dann rekursiv erfolgen.

Für die Hausaufgabe dürfen Sie annehmen, dass alle Eingabezahlen nicht-negativ (d.h. ≥ 0) sind. Für den Wettbewerb gilt aber zusätzlich, dass Ihre Funktion auch korrekt mit negativen Zahlen arbeiten können muss.

Für den Wettbewerb zählt die Tokenanzahl (je kleiner, desto besser; siehe Aufgabenblatt 1). Lösungen, die gleichwertig bezüglich der Tokenanzahl sind, werden im Hinblick auf die Größe der generierten Zahlen verglichen (je kleiner, desto besser). Die vollständige Lösung (außer `import`-Direktiven) muss innerhalb von Kommentaren {`-WETT-`} und {`-TTEW-`} stehen. Zum Beispiel:

```
{-WETT-}
encodeInts :: [Integer] -> Integer
encodeInts zs = ...

decodeInts :: Integer -> [Integer]
decodeInts z = ...
{-TTEW-}
```

Wichtig: Wenn Sie diese Aufgabe als Wettbewerbsaufgabe abgeben, stimmen Sie zu, dass Ihr Name ggf. auf der Ergebnisliste auf unserer Internetseite veröffentlicht wird. Sie können diese Einwilligung jederzeit widerrufen, indem Sie eine Email an `fp@fp.in.tum.de` schicken. Wenn Sie nicht am Wettbewerb teilnehmen, sondern die Aufgabe allein im Rahmen der Hausaufgabe abgeben möchten, lassen Sie bitte die {`-WETT-`} . . . {`-TTEW-`} Kommentare weg. Bei der Bewertung Ihrer Hausaufgabe entsteht Ihnen hierdurch kein Nachteil.