Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Lars Noschinski, Lars Hupel, Dr. Jasmin Blanchette

WS 2013/14
05.11.2013
Abgabe: 12.11.2013

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. Benutzen Sie das aus der Vorlesung bekannte "induction template" (Folie 124). Geben Sie zusätzlich die Gleichung, die Sie als Induktionshypothese (IH) benutzen, explizit an.

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

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

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

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

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")
```

Aufgabe H4.1 Passende Wörter (4 Punkte)

Wir nennen eine Liste von Wörtern *passend*, wenn für zwei aufeinanderfolgende Wörter in dieser Liste gilt, dass der Endbuchstabe des vorderen mit dem Anfangsbuchstaben des hinteren Worts übereinstimmt.

Schreiben Sie eine Funktion `matchingWords :: [String] -> Bool`, die genau dann `True` liefert, wenn die Eingabeliste diese Bedingung erfüllt. Leere Wörter sollen vorher herausgefiltert werden.

Die folgenden Beispiele sollen alle `True` sein:

```
matchingWords []
matchingWords ["frobnicate"]
matchingWords ["wir", "reden", "", "nicht", "tatsächlich"]

not (matchingWords ["arthur", "dent"])
```

Aufgabe H4.2 Listen zerschneiden (6 Punkte)

1. Gesucht ist eine Funktion `chunks :: [Int] -> [a] -> [[a]]`, die zwei Listen als Eingabe nimmt und die zweite in Teillisten zerschneidet, wobei die erste Liste die Längen der Teillisten angibt. Die Funktion ist charakterisiert durch

   $$\texttt{chunks}\ [k_1, \ldots, k_m]\ [x_1, \ldots, x_n] = \\ \left[[x_1, \ldots, x_{k_1}], [x_{k_1+1}, \ldots, x_{k_1+k_2}], \ldots, [x_{k_1+\cdots+k_{m-1}+1}, \ldots, x_{k_1+\cdots+k_{m-1}+k_m}]\right]$$

   Sie dürfen annehmen, dass $k_i > 0$ für $1 \leq i \leq m$ und dass $n \geq k_1 + \cdots + k_m$ gilt.

   Beispiele:

   ```
   chunks [3, 1, 2] [1, 1, 1, 1, 1, 1] ==
       [[1, 1, 1], [1], [1, 1]]
   chunks [1 .. 6] ['A' .. 'Z'] ==
       ["A", "BC", "DEF", "GHIJ", "KLMNO", "PQRSTU"]
   ```

2. Schreiben Sie QuickCheck-Tests für die Funktion `chunks`. In diesen sollen allgemeine Eigenschaften der Funktion beschrieben werden, nicht nur das Verhalten für konkrete Eingaben. Sowohl der Inhalt als auch die Länge der Teillisten sollen getestet werden. Denken Sie auch an geeignete Vorbedingungen für die Tests!

   Hinweis: Auch hier kann es nötig sein, mit den QuickCheck-Parametern zu spielen. Nutzen Sie z.B. `quickCheckWith stdArgs{maxDiscardRatio=9001,maxSize=5} ...`, damit QuickCheck nicht so schnell aufgibt (`maxDiscardRatio` können Sie bedenkenlos erhöhen).

Aufgabe H4.3 ängeL (5 Punkte)

Beweisen Sie die Gleichung

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

per Induktion mithilfe des "induction template" (Folie 124). Geben Sie zusätzlich die Gleichung, die Sie als Induktionshypothese (IH) benutzen, explizit an.

Die Funktion `length :: [a] -> Int` ist wie folgt definiert:

```
length []       = 0                          (length_Nil)
length (x : xs) = length xs + 1              (length_Cons)
```

Arithmetische Ausdrücke dürfen Sie direkt auswerten. Für die Definition von `snoc` siehe Gruppenaufgabe.

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

Aufgabe H5.1 Verrückte Entfernungstabellen

(**zweiwöchige** Wettbewerbsaufgabe, 10 Punkte)

Diese Wettbewerbsaufgabe ist am 19.11.2013 als Teil von Blatt 5 abzugeben. Sie bringt bis zu 60 Wettbewerbspunkte. Die Hausaufgabenpunkte zählen für Blatt 5.

Entfernungstabellen geben die Entfernung zwischen einer Auswahl Städte an. In Haskell lassen sie sich als `[(String, Integer, String)]` darstellen. Zum Beispiel:

```
table =
  [("Cape Town",  1660, "Durban"),
   ("Cape Town",  1042, "East London"),
   ("Cape Town",  1402, "Johannesburg"),
   ("Durban",      667, "East London"),
   ("Durban",      598, "Johannesburg"),
   ("East London", 992, "Johannesburg")]
```

Laut dieser Tabelle beträgt die Distanz zwischen Kapstadt und Johannesburg 1402 km. Wir nehmen an, dass diese Entfernung für beide Fahrtrichtungen gilt und sparen uns somit die halbe Tabelle.

Für eine Dienstreise nach Südafrika sucht der MC diese Woche eine Haskell-Funktion

```
isDistanceTableSane :: [(String, Integer, String)] -> Bool
```

die überprüft, ob die gegebene Entfernungstabelle alle folgenden Eigenschaften erfüllt:

- **Nichtnegativität:** Keine Entfernungsangabe ist negativ.
- **Nichtreflexivität:** Die Tabelle enthält keine Einträge der Form $(x, \_, x)$.
- **Konsistenz:** Falls die Tabelle mehrere Tripel für die Städte x und y enthält, sind die Entfernungen gleich.[1]
- **Vollständigkeit:** Wenn x und y zwei ungleiche Städte sind, die in der Tabelle vorkommen, dann enthält die Tabelle ein Tripel der Form $(x, \_, y)$ oder $(y, \_, x)$.
- **Dreiecksregel:** Die Entfernung zwischen x und y plus die Entfernung zwischen y und z ist nicht kleiner als die Entfernung zwischen x und z.

Für den Wettbewerb zählt die Effizienz, vor allem die Skalierbarkeit. Ihre Implementierung sollte mit großen Tabellen, langen Wörtern und absurd vielen Duplikaten möglichst effizient vorgehen. Lösungen, die sich effizienztechnisch ähnlich verhalten, werden bezüglich ihrer Lesbarkeit verglichen, wobei Formatierung und Namensgebung besonders gewichtet werden.

[1] Dies schließt Symmetrie ein, d.h. der Abstand zwischen x und y muss gleich dem Abstand zwischen y und x sein.

Die vollständige Lösung (außer Standardfunktionen und `import`-Direktiven) muss innerhalb der Kommentare {-WETT-} und {-TTEW-} abgegeben werden, um als Wettbewerbsbeitrag zu zählen. Zum Beispiel:

```
import Test.QuickCheck
import Data.List

{-WETT-}
isDistanceTableSane :: [(String, Integer, String)] -> Bool
isDistanceTableSane table = ...
{-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.