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

WS 2012/13
06. 11. 2012
Abgabe: 13. 11. 2012, 15:30

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 Monoalphabetische Verschlüsselung

Bei einer monoalphabetischen Verschlüsselung wird jedem Zeichen des Klartextalphabets genau ein Zeichen des Schlüsselalphabets zugeordnet. Zum Verschlüsseln eines Textes werden die Klartextzeichen durch die dazugehörigen Schlüsselzeichen ausgetauscht. Das Entschlüsseln erfolgt umgekehrt.

Beispiel: Mit dem Schlüssel `[('a','x'),('H','e'),('l','P'),('o','M')]` wird der Text `"Hallo"` zu `"exPPM"`.

1. Schreiben Sie eine Funktion `crypt :: [(Char, Char)] -> [Char] -> [Char]`, die einen Schlüssel und einen Text nimmt und diesen monoalphabetisch verschlüsselt. Zeichen aus dem Text, die nicht im Schlüssel vorkommen, sollen durch `'_'` ersetzt werden.

   Beispiel: Mit dem Schlüssel `[('a','x'),('H','e'),('o','M')]` wird der Text `"Hallo"` zu `"ex__M"`.

   Es bietet sich an, eine Hilfsfunktion `cryptChar :: [(Char, Char)] -> Char -> Char` zu verwenden, die jedem Klartextzeichen sein Schlüsselzeichen (oder `'_'`) zuordnet.

2. Damit ein verschlüsselter Text auch wieder entschlüsselt werden kann, darf jedes Schlüsselzeichen höchstens einem Klartextzeichen zugeordnet sein.

   Schreiben Sie eine Funktion `isKeyReversible :: [(Char, Char)] -> Bool`, die dies überprüft. Sie dürfen dabei davon ausgehen, dass für jedes Klartextzeichen nur ein Paar im Schlüssel vorhanden ist.

3. Schreiben Sie QuickCheck-Tests für `isKeyReversible`. In diesen sollen allgemeine Eigenschaften der Funktion beschrieben werden, nicht nur das Verhalten für konkrete Eingaben.

Aufgabe G4.3 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 120). 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 G4.4 Mustervergleich

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 Strikt fallende Folgen (2 Punkte)

Eine Liste $[x_1, \ldots, x_n]$ ist *strikt fallend*, wenn $x_1 > \cdots > x_n$. Schreiben Sie eine Funktion `strictlyDescending :: [Integer] -> Bool`, die `True` zurückgibt, wenn die Eingabe eine strikt fallende Liste ist und ansonsten `False`.

Aufgabe H4.2 Listen zerschneiden (6 Punkte)

1. Gesucht ist eine Funktion `chunks :: Int -> [a] -> [[a]]`, die eine Liste in Teillisten einer gewissen Länge zerschneidet. Falls n ein Vielfaches von k ist, soll `chunks` $k\ [x_1, \ldots, x_n]$ eine Liste der Länge n/k zurückgeben, in der alle Teillisten die Länge k haben:

   $$\texttt{chunks}\ k\ [x_1, \ldots, x_n] = \big[[x_1, \ldots, x_k], [x_{k+1}, \ldots, x_{2k}], \ldots, [x_{n-k+1}, \ldots, x_n]\big]$$

   Ansonsten muss die Funktion eine Teilliste mit den restlichen n `` `mod` `` k Elementen anhängen, damit `concat (chunks` $k\ xs) = xs$ für alle $k > 0$ und xs gilt. Zur Vereinfachung können Sie annehmen, dass $k > 0$.

   Beispiele:

   ```
   chunks 1 [1 .. 5] == [[1], [2], [3], [4], [5]]
   chunks 3 [1 .. 5] == [[1, 2, 3], [4, 5]]
   chunks 3 [1 .. 6] == [[1, 2, 3], [4, 5, 6]]
   chunks 100 [1 .. 6] == [[1, 2, 3, 4, 5, 6]]
   chunks 100 [] == []
   ```

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.

3. Schreiben Sie jetzt eine Funktion `irregularChunks :: [Int] -> [a] -> [[a]]`, die die Funktion `chunks` verallgemeinert: Statt einer festen Länge k bekommt die Funktion eine Liste $[k_1, \ldots, k_m]$ von Längen als Argument. Die Funktion ist charakterisiert durch

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

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

   Beispiele:

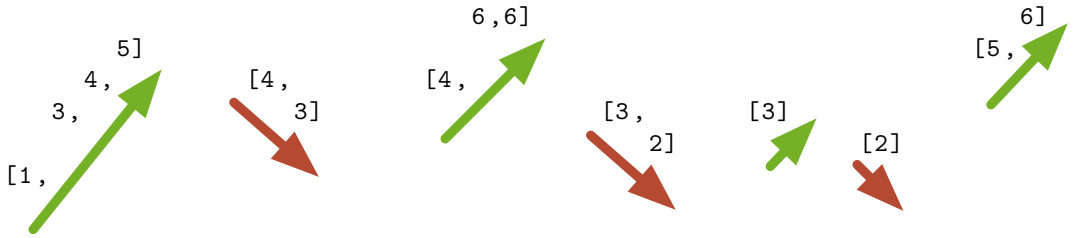
   ```
   irregularChunks [3, 1, 2] [1, 1, 1, 1, 1, 1] ==
       [[1, 1, 1], [1], [1, 1]]
   irregularChunks [1 .. 6] ['A' .. 'Z'] ==
       ["A", "BC", "DEF", "GHIJ", "KLMNO", "PQRSTU"]
   ```

Aufgabe H4.3 Berg- und Talfahrt (6 Punkte, Wettbewerbsaufgabe)

Für diese Woche hatte der Master of Competition vor, nach einer Funktion zu fragen, die die Liste der maximalen *steigenden* Teillisten einer Liste zurückgibt. Leider hat sich Professor Nipkow dieser Idee für seine Vorlesung bemächtigt (Folie 101, "accumulating parameter"). Der MC hat sich dann überlegt, nach der Liste der maximalen *fallenden* Teillisten zu fragen, aber das wäre zu einfach gewesen. Stattdessen hat er sich für die Liste der maximalen *abwechselnd steigenden/fallenden* Teillisten einer Liste entschieden. Zum Beispiel:

```
upsAndDowns [1, 3, 4, 5, 4, 3, 4, 6, 6, 3, 2, 3, 2, 5, 6] ==
  [[1, 3, 4, 5], [4, 3], [4, 6, 6], [3, 2], [3], [2], [5, 6]]
```

Graphisch:



Gesucht ist also eine Funktion `upsAndDowns :: Ord a => [a] -> [[a]]`, die zuerst eine maximale steigende Teilliste ausschneidet, dann eine maximale Fallende, dann wieder eine maximale Steigende, dann wieder eine maximale Fallende und so weiter.

Hinweis: Es kann hilfreich sein, zwei wechselseitig rekursive Hilfsfunktionen zu definieren, `upsAndDowns2` und `downsAndUps2` (beide mit der Typsignatur `Ord a => [a] -> [a] -> [[a]]`), die der Funktion `ups2` aus den Folien sehr ähnlich sind.

Für den Wettbewerb zählt wie üblich die Tokenanzahl (je kleiner, desto besser[1]). Um teilzunehmen müssen Sie die vollständige Lösung (außer `import`s) innerhalb der beiden Kommentare {-WETT-} und {-TTEW-} eingeben. Lösungen mit der gleichen Tokenanzahl werden bzgl. ihrer Lesbarkeit verglichen, wobei Formatierung und Namensgebung besonders gewichtet werden.

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.

Aufgabe H4.4 Länge einer Umdrehung (6 Punkte)

Beweisen Sie die Gleichung

```
length (reverse xs) = length xs
```

per Induktion mithilfe des "induction template" (Folie 120). Geben Sie zusätzlich die Gleichung, die Sie als Induktionshypothese (IH) benutzen, explizit an.

Die Funktionen `reverse :: [a] -> [a]` und `length :: [a] -> Int` sind wie folgt definiert:

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

[1] http://www21.in.tum.de/teaching/info2/WS1213/wettbewerb.html#token

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

Arithmetische Ausdrücke dürfen Sie direkt auswerten. Zusätzlich dürfen Sie die folgende, in der Vorlesung bereits bewiesene Gleichung benutzen:

```
length (xs ++ ys) = length xs + length ys  (length_append)
```

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