Listen mit Haskell[1]

Grundlagen

- Aufbewahren von Daten gleichen Typs
- Einzelne Elemente können komplex sein
- Im Unterschied zu mathematischen Mengen sind doppelte Elemente erlaubt
- Schreibweise: [2,4,3,4]
- Punktschreibweise ist möglich: [3..12] oder [1,3..99]
- Auch unendliche Mengen sind definiert: [1..] (Menge der natürlichen Zahlen)
  Macht aber nur Sinn unter Verwendung anderer Funktionen (Endlosschleife!)

Grundstruktur

- Eine Liste ist entweder leer
  [] (leere Menge)
  oder enthält ein erstes Element und eine Restliste
  `Kopf:Rest` (mind. 1 Element)
  Kopf - erstes Element der Liste
  Rest - Restliste
  : - trennt den Kopf vom Rest
  Achtung: Kopf und Rest haben unterschiedlichen Typ:
  Kopf ist ein **Element** (z.B. Int)
  Rest ist eine **Liste** (z.B. [Int])
- Zwei Listen werden mit ++ verkettet
- Mit : wird ein Element vor eine Liste gesetzt (Listenkonstruktion)
- es hat sich die Schreibweise `x:xs` durchgesetzt (Notation)

Basisfunktionen

Haskell kennt eine Reihe von vordefinierten Funktionen auf Listen:

- tail xs -- Rest einer Liste
- head xs -- Kopf einer Liste
- init xs -- Anfang einer Liste
- last xs -- letztes Element
- reverse xs -- Liste umkehren
- length xs -- Länge einer Liste
- xs!!n -- n-tes Element

Listcomprehension (Listgenerator)

Besonders bequem ist das Erzeugen von Listen in einer beschreibenden Form („Listcomprehension”):

[x | x <- [1..100] , div x 5 > 4 , mod x 7 == 0] – was erzeugt dieser ListGenerator?

Lies:

- x <- [1..] - -Generator
- div x 5 > 4 -- 1. Filter
- mod x 7 == 0 -- 2. Filter
- alle Elemente, die **beide** Filter überwinden, landen in der **Menge**
- die Filter werden durch ein **Komma** getrennt

Ergebnis: [56,63,70,77,84,91,98]

[1] mit Material von W.Gussmann

High-Order-Funktionen

Die map-Funktion

Häufig muss mit jedem Element einer Liste etwas **gemacht** werden.

Beispiel:

Bestimme alle Wurzeln von 1 *bis* 10

```
map1 sqrt [1..10]
```

Lösung 1: „Listcomprehension" - Listengenerator

```
map1 f xs = [f x| x<-xs]
```

Die Funktion f muss passend gewählt werden:

f :: a -> b , und

map1 :: (a -> b) -> [a] -> [b]

map2 rekursiv

Weiteres Beispiel:

Bestimme alle Wurzeln von 1 *bis* 10

```
map2 sqrt [1..10]
```

Lösung 2: „Rekursion"

```
map2 f []  = []
map2 f (x:xs) = (f x) : map2 f xs
```

Die map-Funktion ist in Haskell vordefiniert.

Die filter-Funktion

Eine **Teilliste** erreicht man mit der filter-Funktion:

```
filter ::  (a -> Bool) -> [a] -> [a]
```

Die Filterfunktion wendet die boolsche-Funktion auf alle Elemente an. In die **Ergebnismenge** kommen nur Elemente, die True liefern.

Beispiel: *Extrahiere alle Primzahlen einer Liste*

```
filter prim [1,2,4,5,7,6,8,9,3]
prim ::Int -> Bool
prim n = [x|x<-[2..n],mod x n == 0]==[n]
```

oder

```
prim n = filter (teilbar n) [2..n] == [n]
teilbar b :: Int -> Bool
teilbar b c = mod b c == 0
```