Übungsblatt zu Vorlesung 04— Laziness

1 Alte Bekannte

Auch auf `LazyList` gibt es etliche Funktionen, die Sie bereits kennen.

Implementieren Sie `map`, `filter`, `append` und `flatMap` jeweils via `foldRight` im Enum. Teil dieser Aufgabe ist es, die Signaturen selbst zu schreiben. Die Implementierung dieser Methoden funktioniert sehr ähnlich zu dem, was Sie bereits kennen. Die Signatur für `append` ist gegeben:

```
def append[B >: A](b: => LazyList[B]): LazyList[B] = ???
```

Hinweis: Benutzen Sie die *smart constructors* aus dem companion object.

2 takeWhile

In dieser Aufgabe implementieren Sie die Funktion `takeWhile` auf drei verschiedene Arten. `takeWhile` macht sozusagen das Gegenteil von `dropWhile`, indem alle Elemente vom Beginn einer `LazyList` zurückgegeben werden, für die ein übergebenes Prädikat `true` ist. Sobald die Funktion das erste Element gefunden hat, bei dem das Prädikat `false` ist, bricht sie ab.

Gegeben ist hier die Signatur der Funktion wenn sie im `LazyList` enum implementiert wird.

```
def takeWhile(p: A => Boolean): LazyList[A] = ???
```

a) Implementieren Sie `takeWhile` auf LazyList via explizitem Pattern Matching!

b) Implementieren Sie `takeWhile` via `foldRight`!

c) Implementieren Sie `takeWhile` via `unfold`!

3 tails

Implementieren Sie die Funktion `tails` via `unfold` (und ggf. `append`)!

Für eine gegebene `LazyList` gibt `tails` in einer `LazyList` alle Suffixe, d.h. alle Teillisten, die man durch entfernen von Elementen vom Anfang erhalten kann, zurück, angefangen mit der ursprünglichen `LazyList`. Falls beispielsweise der `LazyList(1,2,3)` gegeben ist, würde `tails` die Ausgabe `LazyList(LazyList(1,2,3), LazyList(2,3), LazyList(3), LazyList())` zurückgeben.

Im `LazyList` Enum implementiert sieht die Signatur von `tails` wie folgt aus:

```
def tails: LazyList[LazyList[A]] = ???
```