Prof. Dr. Manfred Schmidt-Schauß
M.Sc. Nils Dallmeyer

Goethe-Universität Frankfurt am Main/ Institut für Informatik
Künstliche Intelligenz/Softwaretechnologie

Grundlagen der Programmierung 2

Sommersemester 2018

Aufgabenblatt Nr. 5

Abgabe: Mittwoch 16. Mai 2018 **vor!** der Vorlesung

Aufgabe 1 (20 Punkte)

Bäume mit (polymorphen) Markierungen aller Knoten können in Haskell definiert werden durch die folgenden rekursiven Datentypen, wobei `BBaum` ein Binärbaum und `NBaum` ein allgemeiner Baum ist:

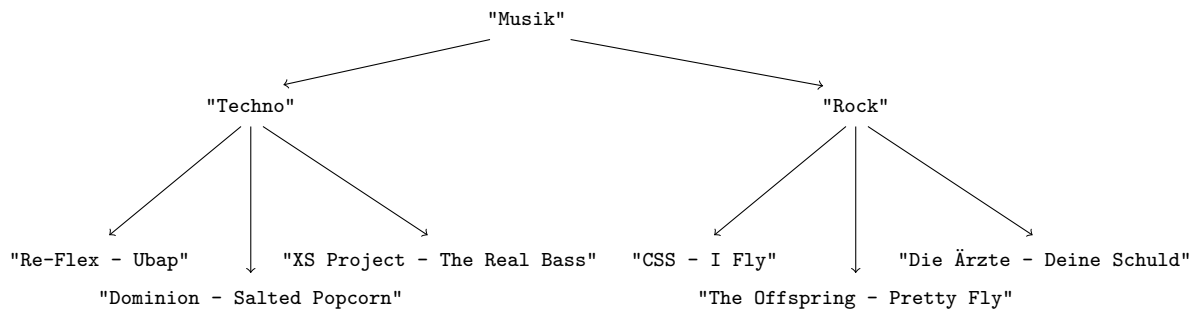
```
data BBaum a = BBlatt a | BKnoten a (BBaum a) (BBaum a)
  deriving(Eq,Show)

data NBaum a = NBlatt a | NKnoten a [NBaum a]
  deriving(Eq,Show)
```

a) Zeichnen Sie den folgenden Baum vom Typ `BBaum`: (4 Punkte)

```
BKnoten
  7
  (BKnoten
    99
    (BKnoten 55 (BBlatt 21) (BBlatt 13))
    (BKnoten 1001 (BBlatt 501) (BBlatt 301))
  )
  (BKnoten
    93
    (BKnoten 170 (BBlatt 9) (BBlatt 42))
    (BKnoten 2018 (BBlatt 130) (BBlatt 180))
  )
```

b) Geben Sie den folgenden Baum in der Haskell-Repräsentation unter Verwendung des Datentyps `NBaum` an und fügen Sie dabei noch eine Musikrichtung mit drei Stücken Ihrer Wahl hinzu:



(4 Punkte)

c) Implementieren Sie eine Haskell-Funktion `allaggr`, die einen Binärbaum entegegennimmt, welcher Zahlen als Knoten- und Blattmarkierungen besitzt und genau dann `True` zurückgibt, falls für jeden Knoten die Markierung mit dem Ergebnis der Addition der Markierungen seiner beiden Kinder übereinstimmt. (6 Punkte)

d) Implementieren Sie eine Haskell-Funktion

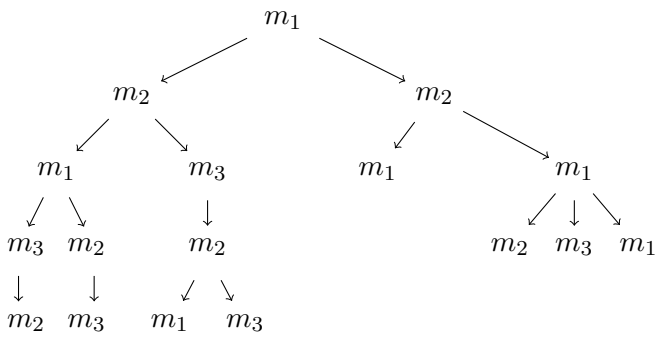
```
btreestontrees :: NBaum (BBaum a) -> NBaum (NBaum a)
```

die einen allgemeinen Baum entgegennimmt, der als Knoten- und Blattmarkierungen binäre Bäume vom Typ `BBaum a` hat und diese binären Bäume in allgemeine Bäume des Typs `NBaum a` umwandelt. (6 Punkte)

Aufgabe 2 (30 Punkte)

Der DJ und Komponist *D. Jay Layzie* lebt von einer beunruhigenden Entwicklung der heutigen Zeit. Er hat nämlich festgestellt, dass man heutzutage problemlos Melodien und Basslines im Techno-Bereich reproduzieren kann und man trotzdem nicht an Erfolg verliert. Aktuell liegt er im Urlaub am Strand und bereitet seine nächsten Stücke vor. Dabei arbeitet er seine Stücke nur sehr grob aus, lässt insbesondere das Schlagzeug weg:

- Jedem Stück ist eine über das ganze Stück gültige Tempoangabe in der Einheit beats-per-minute (bpm) zugeordnet, wobei alle Stücke im 4/4-Takt geschrieben sind.
- Damit aus wenigen Melodien und Basslines möglichst einfach ein neues Stück zusammengebaut werden kann, konstruiert er jeweils einen Baum für die Melodiefolgen und genauso einen für die Folgen von Basslines. Ein Beispielbaum für Melodien:



m_i sei dabei eine Melodie, die er nur kurz gepfiffen hat, also einstimmig ist und somit als Liste von Noten in Haskell dargestellt werden kann. Um ein Stück aus den beiden Bäumen zu generieren, wird aus beiden Bäumen jeweils ein vollständiger Pfad gleicher Länge gewählt. Zum Abschluss legt er dann noch ein paar Filter über das ganze Stück drüber, damit eventuelle Dissonanzen beseitigt werden und die Stücke alle schön einheitlich kommerziell klingen.

Wir stellen nun Noten in Haskell als 4-Tupel dar, wobei die Komponenten die folgende Bedeutung haben:

1. Ein Zahlenwert, der für einen Synthesizer die Oktave angibt, in der die Note liegt. Für Pausen kann hier ein beliebiger Zahlenwert gewählt werden.
2. Ein Zahlenwert, der die Tonhöhe auf der entsprechenden Oktave angibt, wobei 1 dem C, 2 dem Cis/Des, ..., 12 dem H entspricht. Dabei wird -1 zur Kodierung von Pausen verwendet.

3. Ein Zahlenwert, der die Länge der Note in Sechzehnteln angibt, das heißt für eine Viertelnote gibt man hier 4 an.

4. Ein Zahlenwert zwischen 0 und 120, der die Lautstärke des Tons angibt.

Also können Noten wie folgt in Haskell dargestellt werden:

```
type Noten = [(Int,Int,Int,Int)]
```

Die Bäume für die Melodien und Basslines werden mit dem folgenden Datentyp abgedeckt:

```
data Baum = Blatt Noten | Knoten Noten [Baum]
  deriving(Eq,Show)
```

Die gesamten Entwürfe werden durch den folgenden Datentyp dargestellt, bestehend von links nach rechts aus einem Baum für Melodien, einem Baum für Basslines und der Tempoangabe:

```
data Entwuerfe = Entwuerfe Baum Baum Int
  deriving(Eq,Show)
```

Ein komplettes Beispiel unter Verwendung dieser Datentypen findet sich in der Datei `blatt5.hs`. Ein aus dem obigen Melodiebaum abgeleitetes Musikstück im MP3-Format – bei dem die kommerzialisierenden Elemente wie die besagten Filter bewusst weggelassen wurden – findet sich ebenfalls beim Aufgabenblatt auf der Vorlesungsseite.

a) Implementieren Sie eine Haskell-Funktion `anzahlNotenMB :: Entwuerfe -> Int`, welche für Entwürfe in der oben definierten Darstellung ausrechnet, wie viele Noten im gesamten Melodiebaum vorhanden sind. Das heißt die Noten der selben Melodie werden mehrfach gezählt, falls die jeweilige Melodie mehr als einmal im Baum vorkommt. (10 Punkte)

b) Um sein Vorgehen besser zu vertuschen, verändert (d.h. transponiert) er hin und wieder die Tonhöhen der Melodien und Bässe. Implementieren Sie eine Haskell-Funktion `transponiere :: Entwuerfe -> Int -> Entwuerfe`, die für Entwürfe in der oben definierten Darstellung die Tonhöhe aller Noten entsprechend zum zweiten Argument ändert. Das zweite Argument bezieht sich direkt auf unsere Darstellung der Noten, also auf die ersten beiden Komponenten des 4-Tupels. Ist das zweite Argument zum Beispiel `11`, so wird aus einem C ein H in der selben Oktave, für `12` wird aus einem C das nächsthöhere C, für `-5` wird aus einem D ein A in der nächsttieferen Oktave usw.

   Tipps:

   - Implementieren Sie zunächst eine Hilfsfunktion, die eine Note und die Verschiebungszahl entgegennimmt und die entsprechende verschobene Note zurückliefert. Testen Sie diese Funktion systematisch.
   - Die Berechnung der Tonhöhe auf der entsprechenden Oktave kann unabhängig von der Berechnung der Oktave vorgenommen werden.
   - Beachten Sie, dass die Tonhöhe zwischen `1` und `12` liegt – bei der Verwendung von `div` und `mod` muss dies gegebenenfalls berücksichtigt werden.
   - Passend zur mathematischen Division ist der Rest einer Ganzzahldivision in Haskell stets positiv. Das heißt sie erhalten `-1` für `(-9) ‘div‘ 12` und nicht `0`.

   (20 Punkte)

Aufgabe 3 (50 Punkte)

Die Typen der Funktionen `map`, `iterate`, `flip` und `const` (mit jeweils frischen Typvariablen) sind:

```
map     :: (a -> b) -> [a] -> [b]
iterate :: (c -> c) -> c -> [c]
flip    :: (d -> e -> f) -> (e -> d -> f)
const   :: g -> h -> g
```

Für alle Teilaufgaben ist der Rechenweg erforderlich.

a) Berechnen Sie den Typ von `(map iterate)` mit der in der Vorlesung vorgestellten Typregel für die Anwendung, sowie dem Verfahren zur Unifikation von Typen. (15 Punkte)

b) Zeigen Sie, dass die Anwendung `(iterate map)` nicht typisierbar ist, indem Sie versuchen den Ausdruck mit der in der Vorlesung vorgestellten Typregel für Anwendungen zu typisieren. Rechnen Sie solange, bis ein Fehler auftritt. (15 Punkte)

c) Verwenden Sie die Anwendungsregel für mehrere Argumente und die Unifikation von Typen, um den Typ von `(iterate flip const)` zu berechnen. (20 Punkte)