Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
L. Noschinski, L. Hupel, M. Eberl

WS 2014/15
23.01.2015
Abgabe: 30.01.2015 (außer Wettbewerb)

Einführung in die Informatik 2
14. Übung

Aufgabe G14.1 Endrekursive Funktionen I

Entscheiden Sie für die folgenden Funktionen, ob diese endrekursiv sind:

1. ```
   prod :: Num a => a -> [a] -> a
   prod n [] = n
   prod n (m:ms) = prod (n*m) ms
   ```

2. ```
   prod :: Num a => [a] -> a
   prod [] = 1
   prod (m:ms) = if m = 0 then 0 else m * prod ms
   ```

3. ```
   prod :: Num a => a -> [a] -> a
   prod n [] = n
   prod n (m:ms) = if m == 0 then 0 else prod (n*m) ms
   ```

Aufgabe G14.2 Endrekursive Funktionen II

Wir betrachten die Funktion `concat :: [[a]] -> [a]`, die eine Liste von Listen nimmt und diese konkateniert:

```
concat [[1,2],[],[5,6],[7]] = [1,2,5,6,7]
```

Geben Sie eine endrekursive Implementierung von `concat` an.

Aufgabe G14.3 Reduktion

Werten Sie die folgenden Ausdrücke mit Haskells Auswertungsstrategie vollständig aus:

```
map (*2) (1 : threes) !! 1
(\f -> \x -> x + f 2) (\y -> y * 2) (3 + 1)
head (filter (/=3) threes)
```

Welche Auswertungen terminieren nicht? Dabei seien die verwendeten Funktionen wie folgt definiert:

```
map _ [] = []
map f (x:xs) = f x : map f xs

filter _ [] = []
```
```

```
filter f (x:xs) | f x = x : filter f xs
                | otherwise = filter f xs

(x:xs) !! n | n == 0 = x
            | otherwise = xs !! (n - 1)

threes = 3 : threes
```

Aufgabe G14.4 QuickCheck (optional)

Gegeben sei eine Ihnen unbekannte Implementierung der Funktion

```
uniqueElems :: Eq a => [a] -> [a]
```

Angeblich liefert `uniqueElems xs` eine duplikatfreie Liste der Elementen aus der Liste `xs`, in einer beliebigen Reihenfolge. Überprüfen Sie diese Behauptung, indem Sie eine vollständige Sammlung von QuickCheck-Tests für `uniqueElems` definieren. Eine Sammlung von Tests ist *vollständig*, wenn im Falle eines Fehlers in der Implementierung es Eingaben für die Tests gibt, die den Fehler aufdecken, d.h. der Test schlägt fehl. (Ob QuickCheck tatsächlich innerhalb von n Iterationen diese Eingabe generiert, ist eine andere Frage.) Vergleichen Sie mehrere Test-Sammlungen in der Gruppe, prüfen Sie die Vollständigkeit und diskutieren Sie Vor- und Nachteile dieser Vorschläge.

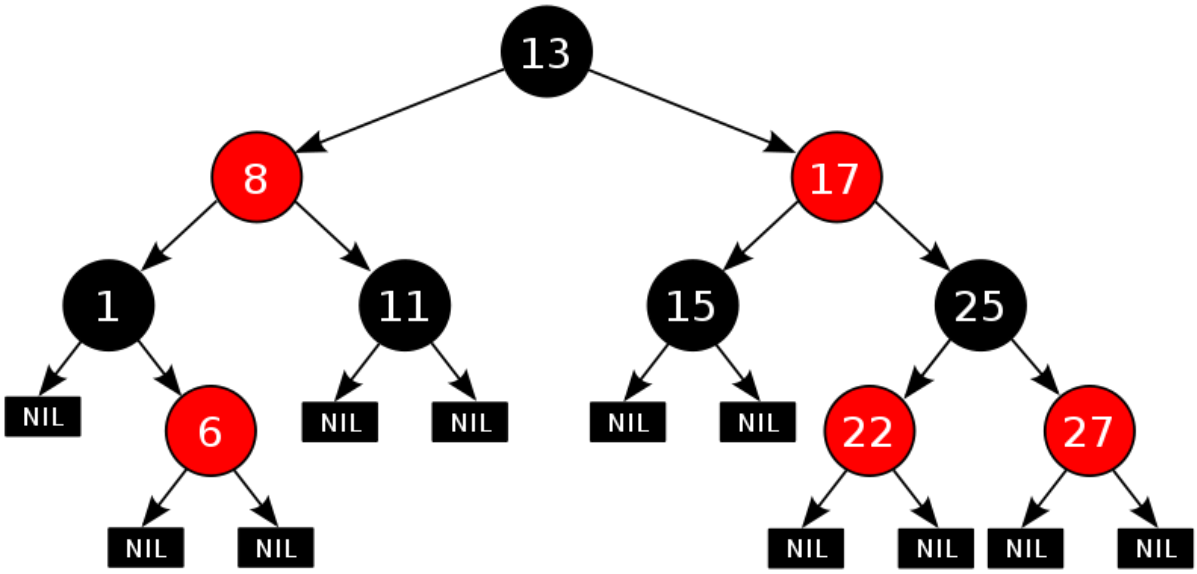

Abbildung 1: Ein gültiger Rot-Schwarz-Baum. Der Knoten 8 hat die Schwarztiefe $k = 2$, der Knoten 13 die Schwarztiefe $k = 3$. Die mit `NIL` beschrifteten Blätter entsprechen in der obigen Definition `Empty`.

Aufgabe H14.1 Am Alleroptimalsten

Implementieren Sie eine optimierte Version der Funktion `f`, die keine unnötige Arbeit verrichtet. Dabei können die Techniken zur Optimierung funktionaler Programme hilfreich sein, die Sie in der Vorlesung und der letzten Übung kennengelernt haben.

```
f :: ([a] -> Integer) -> [a] -> (Integer -> Integer) -> Integer -> Integer
f h xs g 0 = 1
f h xs g 1 = 1 + h xs
f h xs g n
    | g n > - g n = f h xs g (n - 1) + 2 * (h xs - f h xs g (n - 2))
    | otherwise   = f h xs g (n - 1) - h xs + g n
```

Aufgabe H14.2 Schwarzrotmalerei

Gegeben sei eine etwas erweiterte Definition von binären Bäumen und eine Ihnen unbekannte Implementierung einer Funktion, die eine Liste in einen Baum umwandelt:

```
data Color = Red | Black
data Tree a = Empty | Node Color a (Tree a) (Tree a)
fromList :: Ord a => [a] -> Tree a
```

Das Ergebnis der Funktion `fromList` soll dabei ein sogenannter *Rot-Schwarz-Baum* mit folgenden Eigenschaften sein:

Suchbaumeigenschaft Für jeden Knoten n gilt, dass alle Werte links unterhalb des Knotens kleiner als der Wert des Knotens n und alle Werte rechts unterhalb des Knotens größer als der Wert des Knotens n sind.

Farbeigenschaft Jeder Knoten ist entweder *rot* oder *schwarz*. Die Wurzel des Baums ist explizit als schwarz markiert. Blätter (`Empty`) zählen implizit als schwarz.

Rote Kinder Beide Kindknoten eines roten Knotens müssen schwarz sein.

Schwarztiefe Für jeden Knoten n gibt es eine Ganzzahl k, so dass gilt: Jeder Pfad von n zu einem darunterliegenden Blatt enthält k schwarze Knoten.

Elemente Die im Baum enthaltenen Elemente müssen den Elementen der Liste entsprechen. In der Liste mehrfach vorkommende Elemente kommen genau einmal im Baum vor.

Benutzen Sie bitte die Übungsschablone, um den QuickCheck-Test einzutragen. Dort finden Sie nähere Anweisungen zur Benutzung von QuickCheck für diese Aufgabe.

Aufgabe W14.1 Für die van Goghs dieser Welt (optional, Wettbewerbsaufgabe)

> **Wichtig:** Die Abgabefrist für die Wettbewerbsaufgabe ist bereits am 29.01.2015 um 12:00 Uhr. Die Auswertung erfolgt bis zur Vorlesung am 30.01., wo dann auch eine offizielle Preisverleihung stattfinden wird. Laden Sie bitte Ihre Abgabe im Übungssystem unter *Wettbewerb 14* hoch. Fehlermeldungen des Abgabesystems können Sie getrost ignorieren.

In dieser allerletzten Wettbewerbsaufgabe geht es darum, ein Pixel- oder Vektor-Bild in einem Standardformat (z.B. PNG, SVG, PDF) zu generieren. Alle Grafikbibliotheken von Drittanbietern sind ausdrücklich erlaubt. Siehe zum Beispiel:

```
http://www.cs.yale.edu/homes/hudak/SOE/
http://www.haskell.org/haskellwiki/Applications_and_libraries/Graphics
https://hackage.haskell.org/package/JuicyPixels
https://hackage.haskell.org/package/Rasterific
```

Die Bewertungskriterien sind Ästhetik (*hübsches Bild!*) und Technik (*toller Code!*). Die Jury besteht aus dem Meta-Master und den Masters of Competition.

Diese Wettbewerbsaufgabe ist getrennt in einer Datei namens `Gogh.hs` einzureichen. Zusammen mit `Gogh.hs` sollten sie auch das generierte Bild einreichen unter den Namen `gogh.`*`xxx`*, wobei *`xxx`* z.B. `png` ist.

> **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.