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

**WS 2012/13**
**11.11.2012**
**Abgabe: 18.12.2012, 15:30**

# Einführung in die Informatik 2
## 9. Übung

### Aufgabe G9.1 Gespiegelte Binärbäume

Beweisen Sie die Gleichung

```
mirror (mirror t) = t
```

per Induktion über Bäume. Benutzen Sie das aus der Vorlesung bekannte "induction template" (Folie 234). Geben Sie zusätzlich die Gleichungen, die Sie als Induktionshypothesen (IH1 und IH2) benutzen, explizit an.

Die Funktion `mirror :: Tree a -> Tree a` ist wie folgt definiert:

```
mirror Empty = Empty                              --mirror_Empty
mirror (Node x l r) =                             --mirror_Node
  Node x (mirror r) (mirror l)
```

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

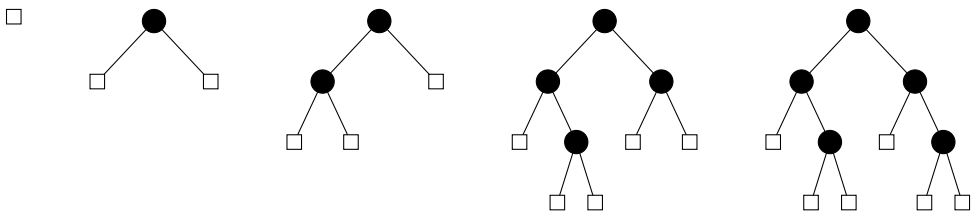
### Aufgabe G9.2 Balancierte Binärbäume

In einem Binärbaum ist der Abstand eines Knotens von der Wurzel die Anzahl der Kanten über die man gehen muss, um von diesem Knoten zur Wurzel zu kommen. Ein Binärbaum ist *balanciert*, wenn es ein $n$ gibt, so dass alle Blätter entweder $n$ oder $n+1$ Knoten von der Wurzel entfernt sind.

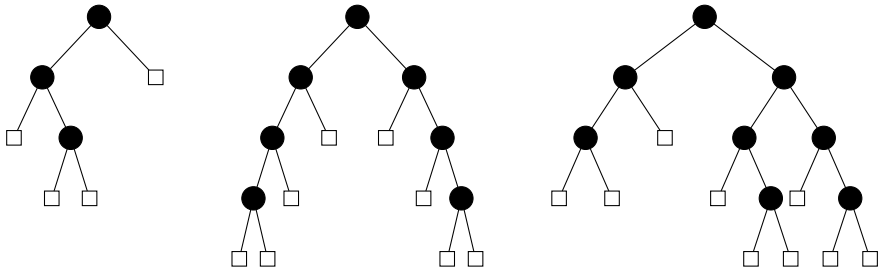
Im Datentyp `Tree` werden Blätter durch `Empty` repräsentiert und innere Knoten durch `Node`. In den Beispielen unten nehmen wir dafür leere Rechtecke und ausgefüllte Kreise. Außerdem lassen wir dort den Wert in den `Node`s weg, da er für die Balanciertheit nicht relevant ist.

Schreiben Sie eine Funktion `balanced :: Tree -> Bool`, die testet, ob ein Baum balanciert ist. Dabei ist ein `Node` ein innerer Knoten und `Empty` ein Blatt.

**Beispiele für balancierte Bäume:**



**Beispiele für unbalancierte Bäume:**



**Aufgabe G9.3** Das Bild vor lauter Bäumen nicht sehen (optional)

Der Waldrand ist häufig ein guter Platz für die Landschaftsmalerei und daher ist unser Thema jetzt die Darstellung von geometrischen Formen.

1. Definieren Sie einen Datentyp `Shape` zur Darstellung von Strecken und Kreisen. Eine Strecke soll beschrieben werden durch ihre zwei Endpunkte, ein Kreis durch Position und Radius. Koordinaten seien durch den Typ `type Coords = (Int, Int)` beschrieben.
2. Definieren Sie eine Funktion `move :: Coords -> Shape -> Shape`, die ein Shape um den durch die Koordinaten beschriebenen Vektor verschiebt.
3. Schreiben Sie eine Funktion `polygon :: [Shape] -> Bool`, die testet, ob eine Liste von `Shape`s ein Polygon beschreibt. Ein Polygon sei hier eine nicht-leere Liste von Strecken, so dass der Endpunkt einer Strecke der Startpunkt der nächsten (und der Endpunkt der letzten der Startpunkt der ersten Strecke) ist.

**Aufgabe H9.1** Flundern im Spiegel (5 Punkte)

Beweisen Sie die Gleichung

```
flat (mirror t) = reverse (flat t)
```

per Induktion über Bäume. Benutzen Sie das aus der Vorlesung bekannte "induction template" (Folie 234). Geben Sie zusätzlich die Gleichungen, die Sie als Induktionshypothesen (IH1 und IH2) benutzen, explizit an.

Die Funktionen `mirror :: Tree a -> Tree a`, `flat :: Tree a -> [a]`, `reverse :: [a] -> [a]` und `++ :: [a] -> [a] -> [a]` sind wie folgt definiert:

```
mirror Empty = Empty                              --mirror_Empty
mirror (Node x l r) =                             --mirror_Node
  Node x (mirror r) (mirror l)

flat Empty = []                                   --flat_Empty
flat (Node x l r) = flat l ++ [x] ++ flat r       --flat_Node
```

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

[] ++ ys = ys                                      --append_Nil
(x : xs) ++ ys = x : (xs ++ ys)                    --append_Cons
```

Zusätzlich dürfen Sie die folgenden, aus der Vorlesung bereits bekannten Gleichungen benutzen:

```
(xs ++ ys) ++ zs = xs ++ (ys ++ zs)                --append_assoc
reverse (xs ++ ys) = reverse ys ++ reverse xs      --reverse_append
```

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

**Aufgabe H9.2** Baum-Origami (5 Punkte)

Es gibt verschiedene Arten, einen Baum in eine Liste umzuwandeln, insbesondere In-Order-, Pre-Order- und Post-Order-Durchläufe.

In-Order bedeutet, dass die Knoten in der Ausgabe in der Reihenfolge

> Knoten des linken Teilbaums—Wurzel—Knoten des rechten Teilbaums

vorzufinden sind (Knoten der Teilbäume sind natürlich entsprechend rekursiv geordnet). Pre- bzw. Post-Order steht für die Reihenfolge

> Wurzel—Knoten des linken Teilbaums—Knoten des rechten Teilbaums bzw.
> Knoten des linken Teilbaums—Knoten des rechten Teilbaums—Wurzel

Beispiel für

```
t = Node 4
      (Node 2 (Node 1 Empty Empty) (Node 3 Empty Empty))
      (Node 5 Empty Empty)

inorder t == [1, 2, 3, 4, 5]
preorder t == [4, 2, 1, 3, 5]
postorder t == [1, 3, 2, 5, 4]
```

1. Definieren Sie die Funktionen `inorder :: Tree a -> [a]`, `preorder :: Tree a -> [a]` und `postorder :: Tree a -> [a]`.
2. Für volle Punktzahl definieren Sie die obigen Funktionen *nicht-rekursiv* mit Hilfe einer allgemeineren Funktion `foldTree`. Diese rekursive Funktion soll unter anderem einen Parameter vom Typ `a -> b -> b -> b` haben und diesen auch verwenden.

   `foldTree` soll nicht speziell für das das Umwandeln von Bäumen in Listen geschrieben sein, sondern sich durch geeignete Wahl der Parameter auch eignen, um z.B. die Summe aller Elemente im Baum zu berechnen. Lassen Sie sich von den `fold`-Funktionen auf Listen inspirieren.

**Wichtig:** Benutzen Sie für Ihre Definitionen *keine* der von uns vorgegebenen Funktionen auf Bäumen, nur Pattern-Matching und die Verwendung der Konstruktoren `Node` und `Empty` sind erlaubt.

**Aufgabe H9.3** Die Nacht der Lebenden Bäume (5 Punkte)

In Aufgabe G9.2 haben wir uns bereits mit balancierten Bäumen beschäftigt.

1. Definieren Sie einen Datentyp `SkeleTree`, der (ähnlich wie `Tree`) Binärbäume darstellt, bei denen aber die Knoten nicht mit einem Wert beschriftet sind. Ein `SkeleTree` stellt also nur das Skelett, die Struktur eines Baumes dar.
2. Schreiben Sie eine Funktion `makeBalanced :: Int -> SkeleTree`, die für eine nichtnegative Zahl $n$ einen balancierten Binärbaum mit $n$ inneren Knoten zurückgibt (zur Definition siehe wieder Aufgabe G9.2).
3. Schreiben Sie QuickCheck-Tests, die sicherstellen, dass `makeBalanced` nur Bäume mit der beschriebenen Eigenschaft zurückliefert.

   Hinweis: Um einen QuickCheck-Test `prop` zu testen, können Sie statt `quickCheck prop` auch

   ```
   quickCheckWith stdArgs{maxSize=10} prop
   ```

   verwenden, um nur kleine Beispiele zu testen.
4. Schreiben Sie eine Funktion `makeAllBalanced :: Int -> [SkeleTree]`, die für eine nichtnegative Zahl $n$ alle balancierte Binärbäume (ohne Duplikate) mit $n$ inneren Knoten zurückgibt.

*Hinweis:* Da Sie bei dieser Aufgabe den Datentyp selbst definieren müssen, können wir Ihnen für diese Aufgabe keine Tests bereitstellen.

**Aufgabe H9.4** 2-3 Bäume (5 Punkte)

Wir definieren Bäume deren innere Knoten 2 oder 3 Kinder haben als:

```
data Tree23 a = Leaf a
              | Branch2 (Tree23 a) a (Tree23 a)
              | Branch3 (Tree23 a) a (Tree23 a) a (Tree23 a)
```

Die im Baum enthaltene Werte sollen lediglich in den Blättern zu finden sein. Werte in den inneren Knoten dienen wie folgt als Suchstruktur:

- für `Branch2 t1 x t2` gilt:
  - alle Werte in `t1` sind echt kleiner `x`
  - alle Werte in `t2` sind größer gleich `x`
- für `Branch3 t1 x t2 y t3` gilt:
  - alle Werte in `t1` sind echt kleiner `x`

– alle Werte in `t2` sind größer gleich `x` und echt kleiner `y`

– alle Werte in `t3` sind größer gleich `y`

Definieren Sie eine Funktion `find23 :: Ord a => a -> Tree23 a -> Bool`, die unter Verwendung der obigen Bedingungen testet ob ein gegebener Wert im gegebenen Baum enthalten ist.

Zum Beispiel ergeben folgende Aufrufe `True` für `t23 = Branch3 (Branch2 (Leaf 1) 2 (Leaf 2)) 3 (Leaf 4) 5 (Leaf 42)`:

```
find23 1 t23                    not (find23 5 t23)
find23 2 t23                    not (find23 23 t23)
not (find23 3 t23)              find23 42 t23
find23 4 t23
```

**Aufgabe W9.1** Ein boolescher Löser (zweiwöchige Wettbewerbsaufgabe, **0 Punkte**)

Diese Wettbewerbsaufgabe ist im Aufgabenblatt 8 beschrieben, ist jedoch als Teil von Blatt 9 abzugeben. Sie bringt dem Löser bis zu 40 Wettbewerbspunkte, dafür aber *keine* Hausaufgabenpunkte. Um teilzunehmen müssen Sie Ihre Lösung innerhalb der Kommentare {`-WETT-`} und {`-TTEW-`} eingeben.

**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.