Aufgabenblatt zur 2. Übung

Zeitraum: 20.04. bis 24.04.2009

1. Aufgabe: (AGS 11.23)

Gegeben sei der Typ

```
data Tree = Node Int Tree Tree | NIL
```

(a) Geben Sie eine Funktion

```
insert :: Tree -> [Int] -> Tree
```

an, die alle Werte einer Liste von Integer-Zahlen in einen bereits bestehenden Suchbaum des o. g. Typs so einfügt, dass die Suchbaumeigenschaft erhalten bleibt. Die Werte der Liste seien paarweise verschieden.

(b) Geben Sie eine HASKELL-Funktion einschließlich der Typ-Definiton an, die testet, ob zwei Binärbäume des o. g. Typs identisch sind.

2. Aufgabe: (AGS 11.21*)

(a) Geben Sie in HASKELL eine boolesche Funktion `compare` einschließlich der Typdefinition an, die zwei Listen mit Elementen des Typs `Int` auf Gleichheit prüft.

(b) Schreiben Sie in HASKELL eine Funktion `merge` einschließlich der Typdefinition, die aus zwei aufsteigend geordneten Listen mit Elementen des Typs `Int` durch Mischen eine aufsteigend geordnete Liste erzeugt.

(c) An den Knoten von Binärbäumen sollen entweder `Int`-Werte oder `Bool`-Werte gespeichert werden unter folgenden Bedingungen:

- Regel 1: Der Binärbaum besitzt mindestens einen Knoten; der Wurzelknoten soll einen `Int`-Wert speichern können.
- Regel 2: Wenn an einem Knoten ein `Int`-Wert gespeichert wird, sollen an beiden Nachfolgern `Bool`-Werte gespeichert werden und umgekehrt.
- Regel 3: Unabhängig von Regel 2 sollen an Blattknoten immer `Int`-Werte gespeichert werden können.

Geben Sie hierfür einen algebraischen Datentyp an.

3. Aufgabe: (AGS 11.11*)

Folgende Definition sei gegeben:

- Ein A–Baum ist entweder leer oder seine Wurzel ist mit A beschriftet und der erste Nachfolger von A ist ein Integerwert, der zweite Nachfolger von A ist ein A–Baum und der dritte Nachfolger von A ist ein B–Baum.
- Ein B–Baum ist entweder leer oder die Wurzel eines B–Baums ist mit B beschriftet und der erste Nachfolger von B ist ein Integerwert, der zweite Nachfolger von B ist ein B–Baum und der dritte Nachfolger von B ist ein A–Baum.

(a) Geben Sie algebraische Datentypen `TA` und `TB` für A–Bäume bzw. B–Bäume an.

(b) Geben Sie eine Haskell–Funktion `trans` an, die für jeden Knoten n eines A–Baums t, der mit einem Integerwert beschriftet ist, die aktuelle Beschriftung durch die Anzahl der auf dem Pfad von der Wurzel von t zu n vorkommenden B–Symbole ersetzt.
Geben Sie die Typen aller von Ihnen definierten Funktionen an.

(c) Geben Sie eine Haskell–Funktion `list` an, die aus einem A–Baum die Liste derjenigen gespeicherten Integerwerte generiert, die jeweils direkte Nachfolger eines mit A beschrifteten Knotens sind, und zwar in der Reihenfolge von links nach rechts gesehen.
Geben Sie die Typen aller von Ihnen definierten Funktionen an.

4. Aufgabe: (AGS 11.14)

Gegeben sei der folgende polymorphe algebraische Datentyp:

```
data Tree a = Branch (Tree a) (Tree a) | Leaf a
```

mit dessen Hilfe sich binäre Bäume konstruieren lassen, die an den Blättern Werte des Typs `a` speichern.

(a) Programmieren Sie eine Funktion `check :: Tree Bool -> Bool`, die genau dann `True` liefert, wenn der Eingabebaum mindestens ein Blatt mit dem gespeicherten Wert `False` besitzt.

(b) Programmieren Sie eine Funktion `toList :: Tree Int -> [Int]`, die aus einem Baum des Typs `Tree Int` eine Liste der gespeicherten Werte der Blätter generiert, und zwar in der Reihenfolge von **rechts nach links** gesehen.

(c) Programmieren Sie eine Funktion `toTree :: [Int] -> Tree Int`, die eine Liste von Zahlen als Argument nimmt und einen Baum erzeugt, welcher - zusätzlich zu einem Blatt mit dem gespeicherten Wert 42 - für jedes Element der Liste genau ein Blatt mit dessen Wert besitzt. (Hinweis: Die Form des Baumes spielt keine Rolle.)

(d) Programmieren Sie eine Funktion `transform :: [Bool] -> [Bool]`, die aus einer Liste von Wahrheitswerten alle Werte `False` herausstreicht und die Anzahl der Werte `True` verdoppelt.

Zusatzaufgabe: (AGS 11.12*)

Gegeben sei der folgende Datentyp `Tree` zur Realisierung von Binärbäumen mit Knotenbeschriftungen vom Typ `Int` in Haskell:

```
data Tree = Node Int Tree Tree | Nil
```

(a) Definieren Sie unter Verwendung der Haskell–Funktion `collapse`

```
collapse  :: Tree -> [Int]
collapse Nil               = []
collapse (Node x y z) = (collapse y)++[x]++(collapse z)
```

eine Haskell–Funktion `check :: Tree -> Bool`, die überprüft, ob es sich bei einem Baum vom Typ `Tree` um einen binären Suchbaum handelt, d. h. jeder Knoten ist mit einer ganzen Zahl beschriftet und es muss für jeden Knoten `x` gelten, dass seine Beschriftung größer oder gleich (bzw. kleiner oder gleich) allen Beschriftungen im linken (bzw. rechten) Teilbaum von `x` ist. Sie können die übliche Relation $\leq$ auf ganzen Zahlen benutzen.

(b) Definieren Sie eine Haskell–Funktion `insert :: Int -> Tree -> Tree`, die einen Integerwert in einen binären Suchbaum einfügt, so dass der resultierende Baum ebenfalls ein binärer Suchbaum ist.

(c) Definieren Sie unter Verwendung der Haskell–Funktion `insert` aus Teil (b) eine Haskell–Funktion `merge :: Tree -> Tree -> Tree` , die zwei binäre Suchbäume zu einem verschmilzt.