Aufgabenblatt zur 2. Übung

Zeitraum: 26.04. bis 30.04.2010

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 (Klausuraufgabe 02.2010):

(a) Schreiben Sie in Haskell eine Funktion `incEntry :: [Int] -> Int -> [Int]`, so dass für jede Liste `l :: [Int]` und jede ganze Zahl `i :: Int` das Ergebnis der Funktionsanwendung `incEntry l i` die Liste ist, die aus `l` entsteht, indem der `i`-te Eintrag um 1 erhöht wird. Wenn die Liste `l` weniger als `i` Einträge hat, soll diese zuerst durch wiederholtes Anhängen von Einträgen mit dem Wert 0 auf die Länge `i` verlängert werden.
Beispiel: `incEntry [2, 3]  4 = [2, 3, 0, 1]`.

(b) Schreiben Sie in Haskell eine Funktion `rsum :: [Int] -> [Int]`, die zu einer Liste `l` eine Ausgabeliste gleicher Länge erzeugt, deren erster Eintrag der letzte Eintrag von `l`, deren zweiter (bzw. dritter) Eintrag die Summe der letzten beiden (bzw. drei) Einträge von `l`, u.s.w. ist.
Beispiel: `rsum [4, 2, 3, 8] = [8, 3+8, 2+3+8, 4+2+3+8] = [8, 11, 13, 17]`.
Sie dürfen hierzu die Funktion `reverse :: [t] -> [t]` zum Invertieren von Listen verwenden, die wie folgt definiert ist:

```
reverse [] = []
reverse (x:xs) = reverse xs ++ [x]
```

Hinweis: Sollten Sie in den Aufgabenteilen **(a)** oder **(b)** weitere Hilfsfunktionen nutzen, so sind diese ebenfalls vollständig zu definieren.

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.