3. Übungsblatt

Ausgabe: 01.12.05
Abgabe: 15.12.05

5 *Ordnung muss sein* *20 Punkte*

Jetzt naht wieder die Weihnachtszeit, und da stehen überall ungeordnete Weihnachtsbäume herum. Das muss nicht so sein! Deshalb werden wir in diesem Übungsblatt einen Datentyp für *geordnete* Weihnachts- und sonstige Bäume implementieren, und natürlich die Korrektheit der Implementation beweisen.

Der Datentyp soll folgende Schnittstelle besitzen:

```
type 'a tree

empty    :: 'a::linorder tree
is_empty :: 'a::linorder tree => bool
insert   :: 'a::linorder => 'a tree => 'a tree
elem     :: 'a::linorder => 'a tree => bool
remove   :: 'a::linorder => 'a tree => 'a tree
```

Spezifikation, Implementation und Beweis erfolgen in drei einfachen Schritten:

1. Zuerst definieren wir einen Datentyp `'a tree` von (binären) Bäumen. Ein Baum kann entweder leer sein, oder ein Knoten mit einem Element (dem Knotenwert) und zwei Kindern sein.

   Geordnete Bäume sind solche, die folgende *Invariante* erfüllen: für einen nichtleeren Baum mit dem Knotenwert x sind alle Knotenwerte des linken Unterbaums kleiner als x und alle Knotenwerte des rechten Unterbaums größer oder gleich x, und beide Unterbäume erfüllen wieder die Invariante. Diese Invariante formulieren wir als Prädikat

   ```
   ordered :: 'a::linorder tree => bool
   ```

   (*5 Punkte*)

2. `insert` und `elem` sollen einen Wert in den Baum ordnungserhaltend einfügen, bzw. testen, ob der Wert in dem Baum enthalten ist.

   Spezifizieren Sie Korrektheit der Operationen (bei beiden können wir annehmen, dass die Invariante für die Argumente gilt; `insert` muss die Invariante erhalten). Geben Sie eine Implementation (d.h. eine ausführbare, rekursiv definierte Funktion), und beweisen Sie deren Korrektheit bezüglich der Spezifikation. (*10 Punkte*)

3. Zu guter Letzt spezifizieren Sie die Korrektheit von `remove`, geben Sie eine Implementation an, und beweisen Sie die Korrektheit bezüglich der Spezifikation.

   Dazu implementieren Sie eine Hilfsfunktion

   ```
   merge :: 'a tree => 'a tree => 'a tree
   ```

   die zwei geordnete Bäume ordnungserhaltend zusammenfügt. Hierbei sind alle Knotenwerte des ersten Argumentes kleiner als alle Knotenwerte des zweite Arguments. Der neue Baum hat den rechten unteren Knoten des ersten (oder wahlweise den linken unteren Knoten des zweiten) Baums als neue Wurzel. (*5 Punkte*)