Einführung in die Informatik 2

Prof. Dr. Seidl, J. Kranz, N. Hartmann J. Brunner — WS 2016/17

Übungsblatt 6 — Abgabefrist: 12.12.2016

Aufgabe 6.1 (P) Binäre Buchen

Gegeben sei folgende Typ-Definition eines binären Baumes:

```
type bin_tree =
    Node of bin_tree_node
  | Leaf
and bin_tree_node = {
  key : int;
  left : bin_tree;
  right : bin_tree;
}
[@@deriving show]
```

Bearbeiten Sie darauf aufbauend folgende Teilaufgaben.

1. Implementieren Sie die Ocaml-Funktion

   `val insert : int -> bin_tree -> bin_tree`,

   die einen Wert in den Baum einfügt und dabei einen neuen Baum zurückliefert. Beim Einfügen in den Suchbaum sollen jeweils kleinere Elemente weiter links im Baum zu finden sein.

2. Implementieren Sie die Ocaml-Funktion

   `val sum : bin_tree -> int`,

   die alle Schlüssel des Baumes aufsummiert.

3. Implementieren Sie die Ocaml-Funktion

   `val height : bin_tree -> int`,

   die die Höhe des Baumes berechnet. Ein Blatt habe dabei die Höhe 0.

4. Implementieren Sie die Ocaml-Funktionen

   `val max : bin_tree -> int option`,

   und

   `val min : bin_tree -> int option`,

   die das kleinste bzw. größte Element eines Baumes zurückliefern, sofern ein solches existiert.

5. Implementieren Sie die Ocaml-Funktion

   `val remove : int -> bin_tree -> bin_tree`,

   die ein Element aus dem Baum entfernt und dazu einen neuen Baum zurückliefert.

Hinweis: Nutzen Sie die Funktion `val [%derive.show: bin_tree] : bin_tree -> string`, um einen Baum in einen String zu konvertieren. Geben Sie Ihren Baum so auf der Konsole aus und testen Sie Ihre Implementierung!

Hinweis: Sie müssen sich in dieser Aufgabe nicht um Fehlerbehandlung kümmern.

Lösungsvorschlag 6.1

Die Lösung befindet sich in der Datei `ocaml/p6_sol.ml`.

Allgemeine Hinweise zur Hausaufgabenabgabe

Die Hausaufgabenabgabe bezüglich dieses Blattes erfolgt über das Ocaml-Abgabesystem. Sie erreichen es unter `https://vmnipkow3.in.tum.de`. Sie können hier Ihre Abgaben einstellen und erhalten Feedback, welche Tests zu welchen Aufgaben erfolgreich durchlaufen. Sie können Ihre Abgabe beliebig oft testen lassen. Bitte beachten Sie, dass Sie Ihre Abgabe stets als **eine einzige UTF8-kodierte Textdatei mit dem Namen *ha6.ml* hochladen müssen**. Die hochgeladene Datei muss ferner den Signaturen in *ha6.mli* entsprechen.

Aufgabe 6.2 (H) Avolla [20 Punkte]

In dieser Aufgabe geht es darum, einen AVL-Baum[1] für Integer-Zahlen zu implementieren. Im Baum werden nur die Schlüssel abgelegt, auf assoziierte Werte wird verzichtet. Eine Vorlage finden Sie in der Datei `ocaml/ha6.ml`; fügen Sie Ihre Implementierung an den markierten Stellen hinzu. Die Datei `ocaml/ha6_angabe.ml` enthält die in der Aufgabe verwendeten Typen. Sie müssen diese Typen in Ihrer Lösung verwenden und dürfen sie nicht verändern.

Gehen Sie wie folgt vor:

1. `[3 Punkte]` Implementieren Sie die Ocaml-Funktion

   ```
   val valid_avl : avl_tree -> bool,
   ```

   die einen gegebenen Baum auf Validität prüft. Es sollen dabei folgenden Eigenschaften überprüft werden:

   - Alle Teilbäume sind valide.
   - Alle Schlüssel im linken Teilbaum sind höchstens so groß wie der Schlüssel des Wurzelknotens.
   - Alle Schlüssel im rechten Teilbaum sind mindestens so groß wie der Schlüssel des Wurzelknotens.
   - Die Balancierung des Baumes wird korrekt im Feld `balance` gespeichert.
   - Die Balancierung ist ein Wert zwischen -1 und 1.

2. `[3 Punkte]` Implementieren Sie die Ocaml-Funktion

   ```
   val dot : avl_tree -> string,
   ```

   die einen AVL-Baum in das Graphviz-Dot-Format[2] überführt. Überlegen Sie sich dazu zunächst eine sinnvolle Darstellung von AVL-Bäumen im Dot-Format. Der resultierende Graph sollte mindestens die Schlüssel enthalten und die Beziehungen der Knoten zueinander ausdrücken. Es ist sehr empfehlenswert, sich die eigenen Graphen der folgenden Teilaufgaben mit z.B. Xdot[3] anzusehen, um Fehler besser zu verstehen.

[1] `http://www2.in.tum.de/hp/file?fid=1347`
[2] `http://www.graphviz.org/`
[3] `https://github.com/jrfonseca/xdot.py`

3. [4 Punkte] Implementieren Sie die Ocaml-Funktion

   ```
   val rotate_single : direction -> avl_tree -> avl_tree,
   ```

   die eine einfache Rotation in eine gegebene Richtung auf einem AVL-Baum ausführt und einen neuen AVL-Baum zurückliefert.

4. [4 Punkte] Implementieren Sie die Ocaml-Funktion

   ```
   val rotate_double : direction -> avl_tree -> avl_tree,
   ```

   die eine doppelte Rotation in eine gegebene Richtung auf einem AVL-Baum ausführt und einen neuen AVL-Baum zurückliefert.

5. [3 Punkte] Implementieren Sie die Ocaml-Funktion

   ```
   val rebalance : avl_tree -> avl_tree,
   ```

   die die Balancierung eines AVL-Baumes prüft (durch das Auslesen des `balance`-Feldes) und den Baum ggf. durch maximal eine einzige Rotation korrigiert, wenn dieser an der Wurzel in eine Richtung Überhang aufweist (also eine Balancierung von -2 oder 2 hat).

6. [3 Punkte] Implementieren Sie schließlich die Ocaml-Funktion

   ```
   val insert : int -> avl_tree -> avl_tree,
   ```

   die einen Schlüssel in einen AVL-Baum einfügt, indem sie einen neuen AVL-Baum zurückliefert, der zusätzlich den Schlüssel enthält.

Hinweis: Sie können bei der Implementierung Ihrer Funktionen davon ausgehen, dass diese valide Eingaben erhalten; Fehler müssen also nicht erkannt oder behandelt werden. Um Warnungen des Ocaml-Compiler zu vermeiden, sollten Sie dennoch alle Fälle von `match`-Ausdrücken ausprogrammieren. Sie können die Funktion `val failwith : string -> 'a` nutzen, um bei einem Fehler das Programm mit einer Nachricht abzubrechen.

Lösungsvorschlag 6.2

Die Lösung befindet sich in der Datei `ocaml/ha6_sol.ml`.