

## Perlen der Informatik 2

### 7. Übung

Dies ist das letzte reguläre Übungsblatt in diesem Semester. In den folgenden Wochen wollen wir uns dann einem kleinen Verifikationsprojekt widmen, nämlich der Verifikation von Rot-Schwarz-Bäumen.

Rot-Schwarz-Bäume sind binäre Suchbäume mit einer zusätzlichen Invariante, die garantiert, dass die Bäume immer so balanciert bleiben, dass alle Operationen mit der Worst-Case-Zeitkomplexität  $O(\log n)$  laufen.

## 1 Binäre Suchbäume

Heute formalisieren wir zum Einstieg normale binäre Suchbäume. Wir definieren den Datentyp aber schon so, dass wir in Zukunft Rot-Schwarz-Bäume damit implementieren können. Also hat jeder Knoten zusätzlich eine Farbe  $R$  oder  $B$ .

```
datatype color = R | B
```

```
datatype rbt =  
  E  
| T color rbt nat rbt
```

Wie immer konzentrieren wir uns aufs Wesentliche: Unsere Bäume speichern nur die Schlüssel (vom Typ  $nat$ ), und keine weiteren Daten. Dementsprechend repräsentiert ein Baum eine Menge von natürlichen Zahlen.

▷ Definieren Sie eine Operation  $in\_tree :: nat \Rightarrow rbt \Rightarrow bool$ , die testet, ob ein Element in einem (beliebigen) Baum enthalten ist.

Suchbäume erfüllen im allgemeinen die Invariante, dass der in einem Knoten gespeicherte Wert größer als jedes Element des linken Teilbaums ist und kleiner als jedes Element des rechten Teilbaums.

▷ Definieren Sie eine Funktion  $st :: rbt \Rightarrow bool$ , die diese Invariante beschreibt.

Der Sinn dieser Invariante ist natürlich, dass man ein Element schneller finden kann.

▷ Schreiben Sie eine Funktion  $lookup :: nat \Rightarrow rbt \Rightarrow bool$ , die prüft, ob ein Element in einem Suchbaum enthalten ist.

▷ Zeigen Sie, dass sich  $lookup$  wie  $in\_tree$ , verhält, wenn der Baum ein Suchbaum ist. Dazu werden wie üblich ggfs. weitere Hilfsfunktionen und Lemmata benötigt.

```
lemma lookup_st:  
  "st t  $\implies$  lookup x t = in_tree x t"  
sorry
```

## 2 Rot-Schwarz-Bäume

Literaturhinweise zu Rot-Schwarz-Bäumen:

### Literatur

- [Wik] Wikipedia. Red-black trees. [http://en.wikipedia.org/wiki/Red\\_black\\_tree](http://en.wikipedia.org/wiki/Red_black_tree).  
Guter Einstiegspunkt
- [CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.  
Die kanonische Lektüre zu dem Thema. Die Implementierung ist aber imperativ und dementsprechend unübersichtlich
- [Oka98] Chris Okasaki. *Purely Functional Data Structures*. Cambridge University Press, Cambridge, UK, 1998.  
Aus Sicht eines funktionalen Programmierers. Leichter verständlich, aber es fehlt die Löschoption.
- [Oka99] Chris Okasaki. Red-black trees in a functional setting. *J. Funct. Program.*, 9(4):471–477, 1999. <http://www.eecs.usma.edu/webs/people/okasaki/jfp99.ps>.  
Ähnlich wie im Buch und auch ohne Löschoption. Angenehm zu lesen und online erhältlich.
- [Kah] Stefan Kahrs. Haskell-Implementierung. <http://www.cs.kent.ac.uk/people/staff/smk/redblack/Untyped.hs>.  
Implementiert auch die Löschoption.