

## Übungen zu Einführung in die Informatik I

### Aufgabe 23      **Sortierte Bäume in OCaml**

In dieser Aufgabe wenden wir uns sortierten binären Bäumen zu. Dazu verwenden wir eine Sortierfunktion  $f: 'a \rightarrow 'a \rightarrow \text{int}$ , die zwei Elemente  $a$  und  $b$  von beliebigen Typ vergleicht und  $-1$  für  $a < b$ ,  $0$  für  $a = b$  und  $1$  für  $a > b$  zurückgibt. Ein Baum ist bezüglich  $f$  sortiert, wenn für jeden Knoten gilt: *alle Elemente im linken Teilbaum  $\leq$  Element des Knotens  $<$  alle Elemente im rechten Teilbaum*

- Verwenden Sie die Datenstruktur aus Aufgabe 18. In dem Baum sollen zweidimensionale Koordinaten  $(x,y)$  gespeichert werden. Schreiben Sie eine Sortierfunktion `comp_xy` die die Koordinaten lexikographisch vergleicht.
- Schreiben Sie eine Funktion `in_order`, die eine Liste der Elemente des Baumes in *InOrder*-Reihenfolge (Linker Teilbaum - Knoten - Rechter Teilbaum) zurückgibt.
- Schreiben Sie die Funktion `sort_tree f btree`. Diese Funktion sortiert den Baum `btree` bezüglich der Funktion  $f$ . Implementieren Sie dazu einen Algorithmus der mittels sortierten Einfügen rekursiv einen sortierten Baum erzeugt (Insertsort). (Hinweis: In dem sortierten Baum soll jedes Element nur einmal vorhanden sein, d.h. es werden Elemente nur eingefügt, solange sie noch nicht im Baum enthalten sind). Verwenden Sie für die Implementierung nur funktionale Konzepte und testen Sie Ihre Funktionen anhand eines geeigneten Beispielbaumes. Verwenden Sie zum Testen Koordinatenwerte  $< (1000, 1000)$ .
- Optional: Schreiben Sie eine Funktion `pre_order` und `post_order`, die eine Liste der Elemente des Baumes in *PreOrder* bzw. *PostOrder*-Reihenfolge zurückgibt.
- Schreiben Sie eine Funktion `map_tree`, die eine Funktion  $f$  auf jedes Element des Baumes anwendet. Die Struktur des Baumes bleibt dabei erhalten. Linearisieren Sie die Elemente des Baumes durch Anwendung von `map_tree`. (Hinweis: Speichern Sie hierfür in den Knoten den Wert  $x \cdot 1000 + y$ ).

### Aufgabe 24      **Verifikation funktionaler Programme**

Gegeben sei folgende OCaml-Funktion zur Umkehrung einer Liste:

```
let rec rev list = match list with
  | [] -> []
  | el :: rest -> app (rev rest) [el]
```

sowie die Funktion `app`:

```
let rec app x y =
  match x with
  | [] -> y
  | x :: xs -> x :: app xs y
```

Ziel dieser Aufgabe ist es zu zeigen, dass die Funktion `rev` involutorisch (oder selbst-invers ist), d.h. es gilt:

$$\text{rev}(\text{rev list}) = \text{list} \quad (1)$$

a) Um die Gültigkeit des Prädikates (1) zu zeigen, muss zunächst Folgendes gezeigt werden:

$$\text{rev}(\text{app xs ys}) = \text{app}(\text{rev ys})(\text{rev xs}) \quad (2)$$

b) Zeigen Sie nun die Gültigkeit des Prädikates (1)!

### **Aufgabe 25 (H) Verifikation funktionaler Programme**

**(10 Punkte)**

Die Funktion `length` ist folgendermaßen gegeben:

```
let rec length ls = match ls with
| [] -> 0
| x :: xs -> 1 + length xs
```

Die Funktion `app` sei wie folgt definiert:

```
let rec app x y =
  match x with
  | [] -> y
  | x::xs -> x :: app xs y
```

Zeigen Sie, dass für alle Listen *ls* und *ks* gilt:

$$\text{length}(\text{app ls ks}) = \text{length ls} + \text{length ks}$$

### **Aufgabe 26 (H) Existenz- und Allquantor**

**(5+5 = 10 Punkte)**

In der Prädikatenlogik treten Ausdrücke mit dem sogenannten Existenzquantor  $\exists$  und dem so genannten Allquantor  $\forall$  auf, wie etwa  $\forall x \in \mathbb{N} : f(x)$  oder  $\exists x, y \in \mathbb{N} : g(x, y)$ ; hierbei handelt es sich bei *f* und *g* um boolesche Funktionen, d.h ihr Ergebnis ist ein Wahrheitswert.

Auf Sequenzen seien nun in analoger Weise die Funktionen `existiert` und `fuer_alle` in folgender Weise definiert: Sei *f* eine einstellige, boolesche Funktion über  $\mathbb{Z}$  und *l* eine Liste über  $\mathbb{Z}$ . Dann gilt:

$$\text{existiert}(f, l) = \begin{cases} \text{true} & \text{wenn die Liste ein Element } x \text{ enthält mit } f(x) = \text{true} \\ \text{false} & \text{sonst} \end{cases}$$

und

$$\text{fuer_alle}(f, l) = \begin{cases} \text{true} & \text{wenn für alle Elemente der Liste gilt: } f(x) = \text{true} \\ \text{false} & \text{sonst} \end{cases}$$

Modellieren Sie die Funktionen `existiert` und `fuer_alle` durch eine geeignete rekursive Funktion in OCaml. Geben Sie dabei auch die Signatur der Funktionen an!