

## Übungen zu „Prinzipien von Programmiersprachen“ Blatt 4

**Aufgabe 14.** Harry Hacker hat sich folgenden Variantentyp ausgedacht, um natürliche Zahlen zu repräsentieren.

```
data Nat = Zero | One Nat | Two Nat
```

Zahlen werden wie im Binärsystem repräsentiert, nur dass anstelle der Ziffern 0 und 1 die Ziffern 1 und 2 verwendet werden. Harry preist als entscheidenden Vorteil an, dass die Darstellung nicht redundant ist (da es keine Nullen und somit keine führenden Nullen gibt).

1. Aber, hat jede natürliche Zahl überhaupt eine Darstellung in diesem System?
2. Programmier die Nachfolgerfunktion und die Addition von Zahlen in diesem System.

**Aufgabe 15.** Implementiere Heapsort. Verwende den folgenden Datentyp, um Heaps darzustellen.

```
data Heap  $\alpha$  = Empty | Node (min :  $\alpha$ , left : Heap  $\alpha$ , right : Heap  $\alpha$ )
```

Schreibe eine Funktion, die eine ungeordnete Liste in einen Heap überführt, und eine Funktion, die einen Heap in eine geordnete Liste überführt.

```
toheap :  $\langle \alpha \rangle \rightarrow ((\alpha, \alpha) \rightarrow Bool) \rightarrow (List \langle \alpha \rangle \rightarrow Heap \langle \alpha \rangle)$   
unheap :  $\langle \alpha \rangle \rightarrow ((\alpha, \alpha) \rightarrow Bool) \rightarrow (Heap \langle \alpha \rangle \rightarrow List \langle \alpha \rangle)$ 
```

Welche Laufzeit hat Deine Implementierung von Heapsort?

**Aufgabe 16\*.** Parametrisierte Datentypen können erstaunliche viele Invarianten ausdrücken. Die folgende Deklaration definiert zum Beispiel sogenannte perfekte Bäume, das sind Bäume, deren Blätter sich alle auf einer Ebene befinden.

```
data Perfect  $\alpha$  = Zero  $\alpha$  | Succ (Perfect  $\langle (fst : \alpha, snd : \alpha) \rangle$ )
```

Was ist an dieser Deklaration ungewöhnlich? Wie sehen konkrete Elemente dieses Variantentyps aus? Schreibe Funktionen, die die Tiefe und die Größe perfekter Bäume bestimmen. Programmier einen Iterator für diesen Datentyp:

```
perfect :  $\langle \alpha, \beta \rangle \rightarrow (\alpha \rightarrow \beta) \rightarrow (Perfect \langle \alpha \rangle \rightarrow Perfect \langle \beta \rangle)$ 
```

Verwende den Iterator, um eine Funktion zu definieren, die einen perfekten Baum halbiert.

```
split :  $\langle \alpha \rangle \rightarrow Perfect \langle (fst : \alpha, snd : \alpha) \rangle \rightarrow (fst : Perfect \langle \alpha \rangle, snd : Perfect \langle \alpha \rangle)$ 
```

Gibt es noch andere Möglichkeiten, diese Funktion zu definieren?