

## Übungen zu Einführung in die Informatik I

### Aufgabe 32      Funktionen höherer Ordnung: fold\_left

In Ocaml gibt es neben der in der Vorlesung vorgestellten Funktion fold\_right noch die Funktion fold\_left:

$$\text{fold\_left } f \ x \ [x_1; \dots; x_n] = f \ ( \dots (f \ (f \ x \ x_1) \ x_2) \ \dots ) \ x_n$$

- Implementieren Sie die Funktion fold\_left in OCaml.
- Implementieren Sie basierend auf fold\_left eine Funktion list\_sum, welche die Werte einer Integer-Liste addiert.
- Machen Sie sich den Unterschied zwischen fold\_left und fold\_right deutlich. Führen sie hierfür die Auswertung der Funktion list\_sum [2;4;6;8] 0 Schritt für Schritt durch. Wie sieht im Gegensatz dazu Auswertung für fold\_right sum [2;4;6;8] 0 aus?
- Implementieren Sie die den Existenz- und Allquantor aus Aufgabe 26 basierend auf fold\_left.

### Aufgabe 33      Von OCaml zu Java

Auf den bisherigen Übungsblättern wurden rekursive Funktionen in OCaml implementiert, so z.B. die Berechnung der Zahlensumme:

$$\text{sum}(n) = \sum_{i=1}^{i=n} i$$

Ziel dieser Aufgabe ist es, bereits erlernte Verfahren in eine weitere Programmiersprache, in diesem Fall Java, zu übertragen. Ein geeignete rekursive OCaml-Funktion zur Lösung eben genannter Aufgabe könnte wie folgt aussehen:

```
(* Rekursive Berechnung der Summe *)
let rec sum n = match n with
  1 -> 1
  | _ -> n + sum (n - 1);;

(* Aufruf für den Wert 4 und Ausgabe mit Zeilenumbruch *)
print_endline (string_of_int (sum 4));;
```

Eine Umsetzung dieser Funktion in ein entsprechendes Java-Programm lautet:

```
public class Sum {
    /* Rekursive Berechnung der Summe */
    public static int sum(int n) {
        if (n == 1) {
```

```
        return 1;
    } else {
        return n + sum(n - 1);
    }
}

public static void main(String[] args) {
    // Aufruf für den Wert 4 und Ausgabe mit Zeilenumbruch
    System.out.println(sum(4));
}
}
```

- a) Vergleichen Sie beide Implementierungen miteinander und arbeiten Sie die für Java wichtigen Konzepte heraus.
- b) Üben Sie Funktionen in OCaml und Java anhand eines Beispiels `fullName`, das aus den Parametern `firstName` und `lastName` den vollständigen Namen zurückliefert. Für die Darstellung von Zeichenketten kann der Java-Typ `String` verwendet werden.
- c) Wiederholen Sie die Implementierung einer geeigneten Funktion zur Berechnung der Fibonacci-Zahlen in OCaml.

$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2) \qquad \text{fib}(0) = 0, \quad \text{fib}(1) = 1$$

- d) Implementieren Sie nun Ihre OCaml-Funktion geeignet in Java.
- e) Testen Sie beide Lösungen mit geeigneten Eingaben.

### Aufgabe 34      Aufrufstrategien bei OCaml

- a) Gegeben sei die OCaml-Funktion `let zero x = 0`. D.h. es handelt sich um eine Funktion, die unabhängig vom Parameter die Null zurückgibt. Was passiert, wenn diese Funktion mit `zero (fib (200))` aufgerufen wird ? (Testen Sie dies direkt am Rechner!)
- b) Die Strategie nach der OCaml Funktionsaufrufe auswertet, nennt man „Eager Evaluation“. Was könnte sich dahinter verbergen und inwiefern ist diese Strategie problematisch?
- c) Betrachten Sie die Funktion höherer Ordnung `map (f, ls)`! Weshalb kann bei Argumenten, die Funktionen sind, die Aufrufstrategie „Eager Evaluation“ nicht angewandt werden?
- d) In OCaml ist das leere Tupel `()` ein spezieller Datentyp, der nur aus einem einzigen Element namens `unit` besteht. Der Datentyp des leeren Tupels läßt sich dazu verwenden, um die Auswertung eines Ausdrucks zu verzögern und damit eine „Lazy Evaluation“ Strategie umzusetzen. Testen Sie folgende Funktionsdeklarationen:

```
let f x = match x with
  | () -> fib (20);;
```

Der Ausdruck `fib (20)` wird vorerst nicht ausgewertet, sondern nur die Signatur angegeben: `val f : unit -> int = <fun>`. Was ist zu tun, um die Auswertung des Ausdrucks `fib (20)` zu erzwingen?

- e) Bei einer Funktion höherer Ordnung läßt sich die Signatur einer Funktion, die als Argument übergeben wird, detailliert angeben. Beispielsweise könnte man die Funktion `map` auch durch

```
let rec map (f : (int -> int)) ls = match ls with  
  | [] -> []  
  | x :: rest -> (f x) :: (map f rest);;
```

festlegen. (Testen Sie diese Funktion!) Inwiefern bedeutet dies eine Einschränkung gegenüber der bekannten map-Funktion? Führen Sie diese Einschränkung auch bei der Funktion `filter` durch!

- f) Definieren Sie die Funktion `zero` als Funktion höherer Ordnung, die die Argumente nur bei Bedarf auswertet, also letztendlich eine „Lazy“-Strategie umsetzt und rufen sie die Funktion mit `fib (200)` gemäß der „Lazy“-Strategie auf.

### **Aufgabe 35 (H) Türme von Hanoi in Java**

**(2+2+6+0 Punkte)**

In Aufgabe 6 des 2. Übungsblattes wurde bereits eine Lösung für das Knobelspiel *Türme von Hanoi* in OCaml programmiert. Benutzen Sie die Lösung dieser Aufgabe um eine ähnliche Implementierung in Java zu entwickeln.

- a) Definieren Sie eine statische Hilfsfunktion `stringOfTower`, welche die Position des Turmes darstellt.
- b) Definieren Sie eine statische Hilfsfunktion `stringOfMove`, die einen Zug darstellt.
- c) Definieren Sie die statische Funktion `hanoi`, die den Turm von Hanoi verschiebt.
- d) Testen Sie Ihre Implementierungen geeignet.