Technische Universität München
Fakultät für Informatik
D. Gmach, M. Rickert,
Dr. M. Schneider

WS 2005/2006
Lösungsvorschläge zu Blatt 9
23. Dezember 2005

Übungen zu Einführung in die Informatik I

Aufgabe 36 Nichtstrikte Funktionen (Lösungsvorschlag)

Sowohl der bedingte Ausdruck als auch die Konjunktion und Disjunktion werden in Ocaml gemäß der „Lazy"-Strategie ausgewertet. Im Falle der Konjunktion (Disjunktion) heißt das, dass das zweite Argument nur ausgewertet wird, wenn das erste `true` (`false`) ist. Bei dem bedingten Ausdruck wird je nachdem ob die Bedingung erfüllt oder nicht erfüllt ist nur der entsprechende Zweig ausgewertet.

Aufgabe 37 Ströme in Ocaml (Lösungsvorschlag)

a) Eine Datentstruktur für Ströme hat die Form:

```
type 'a stream = Nil | Cons of ('a * (unit -> 'a stream));;
```

Der einzige Unterschied zu normalen Sequenzen, liegt in der Deklaration des Listenrests; dieser ist nun eine Funktion

b) Die Funktionen head und tail lassen sich dann wie folgt implementieren:

```
let head ls = match ls with
        | Nil -> failwith "Empty"
        |(Cons(x,_)) -> x;;

let tail ls = match ls with
        | Nil -> failwith "Empty"
        |(Cons(_,s))  -> s ();;
```

c) Eine Funktion, die die ersten n-Elemente eines Stromes in eine Liste umwandelt, lautet:

```
let rec stream2list (n, str) = match (n, str) with
        | (0, _) -> []
        | (n, Nil) -> []
        | (n, Cons (x, f)) -> x :: stream2list (n-1, f ());;
```

d) Ein Datenstrom für ganze Zahlen, beginnend bei einer ganzen Zahl n, läßt sich durch

```
let rec from k = Cons(k, fun () -> from (k+1));;
```

realisieren. Die natürlichen Zahlen ergeben sich daraus durch

```
let natStream = from 0;;
```

Unter Verwendung der Funktion stream2list lassen sich beispielsweise die ersten 20 Zahlen durch

```
# stream2list (20, natStream);;
- : int list =
[0; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12; 13; 14; 15; 16; 17; 18;
    19]
```

ausgeben.

e) Die gesuchte Funktion lautet:

```
let rec addSeq arg = match arg with
        |(Nil, _) -> Nil
        |(_, Nil) -> Nil
        |(Cons(x, lx), Cons(y, ly)) -> Cons(x+y, fun () -> addSeq
           (lx (), ly ()));;
```

Aufgabe 38 Alkoholkonzentration im Blut (Lösungsvorschlag)

In der Lösung wird angenommen, dass ein 75 kg schwerer Mann einen Liter Bier mit 5% Alkoholgehalt trinkt.

a) Rekursive Version:

```
public class Alkoholkonzentration_rek {

    public static double verzehr =  10; // in 100ml
    public static double alkohol_gehalt = 5; // in Prozent
    public static double koerpergewicht = 75; // in KG
    public static double reduktionsfaktor = 0.7;
            // Verteilungsfaktor im Körper

    public static int reduktion(double promille) {

        System.out.println("Alkoholgehalt: " + promille);

        // * 0,8 entspricht einer Reduktion um 20% Prozent
        return promille > 0.3
            ? 1 + reduktion(promille * 0.8)
            : 0;
    }

    public static void main(String argv[]) {
        // 1 Liter Alkohol wiegt 0,8 kg
        double promille = 0.8 * verzehr * alkohol_gehalt /
            (koerpergewicht * reduktionsfaktor);

        System.out.println("Die Fahrtüchtigkeit wird nach "
                           + reduktion(promille)
                           + " Stunden wieder erreicht!");
    }
}
```

b) Iterative Version:

```
public class Alkoholkonzentration {
```

```
    public static double verzehr =  10; // in 100ml
    public static double alkohol_gehalt = 5; // in Prozent
    public static double koerpergewicht = 75; // in KG
    public static double reduktionsfaktor = 0.7;
            // Verteilungsfaktor im Körper

    public static int reduktion(double promille) {

        int stunden = 0;

        System.out.println("Alkoholgehalt␣nach␣" +  stunden
                           + "␣Stunden:␣" + promille);

        while(promille > 0.3) {
            stunden++;
            // * 0,8 entspricht einer Reduktion um 20% Prozent
            promille = promille * 0.8;
            System.out.println("Alkoholgehalt␣nach␣" +  stunden
                           + "␣Stunden:␣" + promille);
        }
        return stunden;
    }


    public static void main(String argv[]) {
        // 1 Liter Alkohol wiegt 0,8 kg
        double promille = 0.8 * verzehr * alkohol_gehalt /
            (koerpergewicht * reduktionsfaktor);

        System.out.println("Die␣Fahrtüchtigkeit␣wird␣nach␣"
                           + reduktion(promille)
                           + "␣Stunden␣wieder␣erreicht!");
    }
}
```

Ausgabe des Programms:

```
java Alkoholkonzentration_rek
Alkoholgehalt nach 0 Stunden: 0.7619047619047619
Alkoholgehalt nach 1 Stunden: 0.6095238095238096
Alkoholgehalt nach 2 Stunden: 0.4876190476190477
Alkoholgehalt nach 3 Stunden: 0.39009523809523816
Alkoholgehalt nach 4 Stunden: 0.31207619047619056
Alkoholgehalt nach 5 Stunden: 0.24966095238095246
Die Fahrtüchtigkeit wird nach 5 Stunden wieder erreicht!
```

Aufgabe 39 Primfaktorzerlegung (Lösungsvorschlag)

a) Kleinster Teiler:

```
    public static int kleinsterTeiler (int n){
            boolean b = false;
            int d = 2;
```

```
            while ((n % d) != 0)
                d++;
            }
            return d;
        }
```

b) PrimfaktorZerlegung:

```
        public static void primfaktorZerlegung (int n) {
            if (n>=2) {
                int d = kleinsterTeiler(n);
                    System.out.println(d);
                while (d < n) {
                    n = n / d;
                    d = kleinsterTeiler(n);
                    System.out.println(d);
                }
            }
        }
```

Aufgabe 40 Berechnung von π nach der Monte-Carlo-Methode (Lösungsvorschlag)

a) Rekursive Version:
In der rekursiven Version wurde die Anzahl der Stichproben auf 1000 heruntergesetzt, da bei höheren Zahlen ein Stackoverflow von Java auftreten kann.

```
import java.util.Random;

public class MonteCarlo_rek {

    public static int anzahl = 1000;
    public static Random generator = new Random();

    public static int versuch(int rest_anzahl, int treffer) {
            double x = generator.nextDouble();
            double y = generator.nextDouble();

            return rest_anzahl > 0
                ? ((x*x + y*y) < 1.0
                   ? versuch(rest_anzahl - 1, treffer + 1)
                   : versuch(rest_anzahl - 1, treffer))
                : treffer;
    }

    public static void main(String argv[]) {

        int treffer = versuch(anzahl, 0);
        double pi = 4 * ((double) treffer) / ((double) anzahl);

        System.out.println("Ermittelter Wert von PI: " + pi);
    }
}
```

b) Iterative Version:

```
import java.util.Random;

public class MonteCarlo {

    public static int anzahl = 1000;

    public static void main(String argv[]) {

        Random generator = new Random();

        int treffer = 0;

        for (int i = 0; i < anzahl; i++) {
            double x = generator.nextDouble();
            double y = generator.nextDouble();

            if ((x*x + y*y) < 1.0)
                treffer++;
        }

        double pi = 4 * ((double) treffer) / ((double) anzahl);

        System.out.println("Ermittelter Wert von PI: " + pi);
    }
}
```

Aufgabe 41 Potenz und allgemeine Wurzel (Lösungsvorschlag)

a) Potenz:

```
public static double powerInt (double x, int n){
        double result = 1.0;
        if (n<0) {
                x = 1/x;
                n = -n;
                }
        while (n > 0) {
                result = result*x;
                n = n-1;
        }
        return result;
}
```

b) Wurzel:

```
public static double abs (double x) {
        if (x<0) return -x;
        else return x;
}

public static double wurzel (double x, int n, double
   epsilon){
```

```
        double upper = x;
        double lower = 0;
        double mid = 0.5*(upper + lower);
        while (abs(upper-lower)> epsilon) {
            if (powerInt(mid, n) > x) upper = mid;
            else lower = mid;
            mid = 0.5*(upper + lower);
        }
        return mid;
}
```