

## Praktikum zu Grundlagen der Programmierung

### Aufgabe 5 Türme von Hanoi (Lösungsvorschlag)

#### a) Position

```
# let string_of_turm turm =  
match turm with  
| 1 -> "eins"  
| 2 -> "zwei"  
| 3 -> "drei";;
```

Die Fehlermeldung "Warning: this pattern-matching is not exhaustive." bedenken und mögliche Alternative mit `| _ -> "Fehler"` anregen.

#### b) Zug

```
# let string_of_zug (start, ziel) =  
"von " ^ string_of_turm start ^  
" nach " ^ string_of_turm ziel ^ " ";;
```

#### c) Hanoi

```
# let rec hanoi anzahl start ziel =  
if anzahl <> 0  
then let hilfe = 6 - start - ziel in  
      let start_nach_hilfe = hanoi (anzahl - 1) start hilfe in  
      let hilfe_nach_ziel = hanoi (anzahl - 1) hilfe ziel in  
      start_nach_hilfe ^ string_of_zug (start, hilfe) ^ hilfe_nach_ziel  
else "";;
```

### Aufgabe 6 Babylonisches Wurzelziehen (Lösungsvorschlag)

Das Babylonische Wurzelziehen lässt sich mathematisch vom Newton-Verfahren herleiten. Das Newton-Verfahren berechnet für eine Funktion  $f$  eine Nullstelle mittels folgender Formel:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Zur Bestimmung der Nullstelle der Funktion  $f(x) = x^2 - a$  lautet die Formel folgendermaßen:

$$x_{n+1} = x_n - \frac{x_n^2 - a}{2x_n}$$

Weiter gilt:

$$\left(x + \frac{a}{x}\right) \frac{1}{2} = \frac{x^2 + a}{2x} = \frac{2x^2 - (x^2 - a)}{2x} = x - \frac{x^2 - a}{2x}$$

Die Ermittlung der Quadratwurzel kann geometrisch als die Bestimmung der Seitenlänge eines Quadrates mit Flächeninhalt  $a$  gedeutet werden. Hierbei startet man mit einem Rechteck der Seitenlänge  $x_0 = 1$  und  $y_0 = a/1 = a$ . In jedem Iterationsschritt wird das arithmetische Mittel der beiden Seitenlängen als neue Seitenlänge  $x_{n+1}$  verwendet.  $y_{n+1}$  ergibt sich dann aus  $a/x_{n+1}$ . Dadurch erreicht man in jedem Iterationsschritt durch die Bildung des arithmetischen Mittels eine bessere Näherung des Rechtecks an ein Quadrat mit dem Flächeninhalt  $a$ .

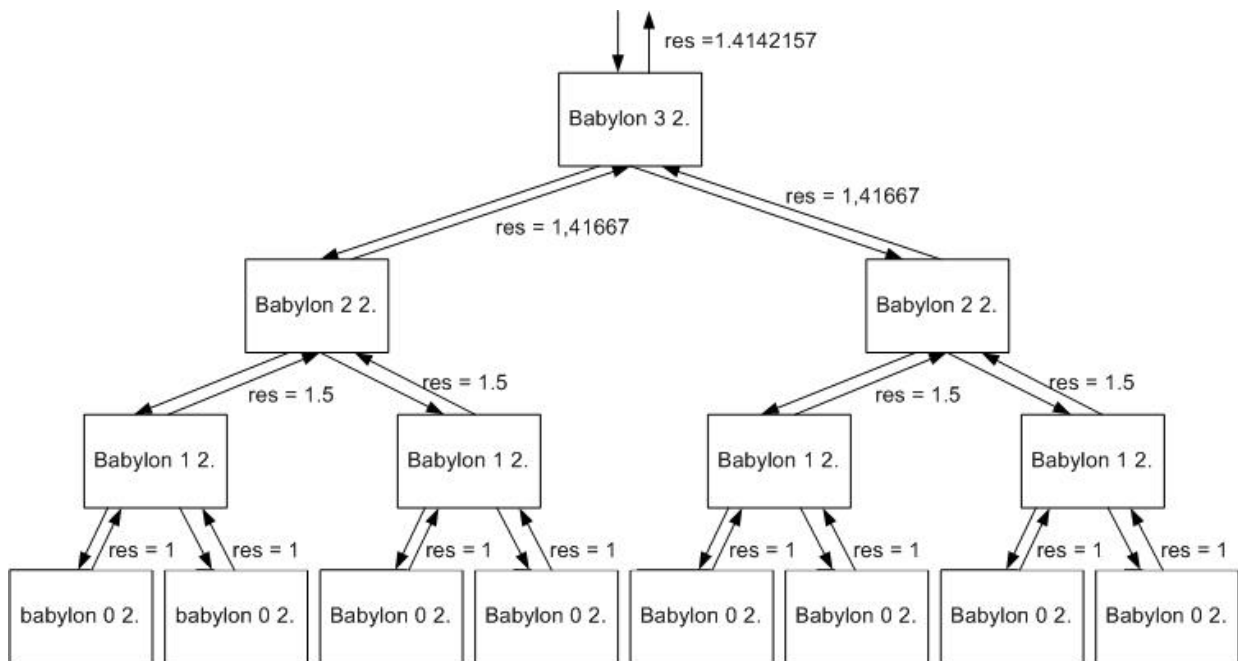
```
a) # let rec babylon k a =
      if k < 1
      then 1.
      else ((babylon (k-1) a) +. a /. (babylon (k-1) a)) /. 2.;;
val babylon : int -> float -> float = <fun>
```

```
b) # babylon 0 2.;;
- : float = 1.
# babylon 1 2.;;
- : float = 1.5
# babylon 2 2.;;
- : float = 1.4166666666666665
# babylon 3 2.;;
- : float = 1.4142156862745097
# babylon 4 2.;;
- : float = 1.4142135623746899
# babylon 5 2.;;
- : float = 1.4142135623730949
# babylon 6 2.;;
- : float = 1.4142135623730949
```

Das Verfahren konvergiert sehr schnell und liefert bereits nach wenigen Schritten eine gute Näherung.

c) Die obige Implementierung weist eine exponentielle Komplexität auf. Für die Berechnung sind  $2^{k+1} - 1$  Aufrufe erforderlich. Der Aufrufbaum sieht damit folgendermaßen aus:

```
d) # let rec intervall k a (l, r) =
      if k < 1
      then (l, r)
      else let m = (l +. r) /. 2. in
            if m *. m > a
            then intervall (k-1) a (l, m)
            else intervall (k-1) a (m, r);;
val intervall : int -> float -> float * float -> float * float = <fun>
# intervall 0 2. (0., 2.);;
- : float * float = (0., 2.)
# intervall 1 2. (0., 2.);;
- : float * float = (1., 2.)
# intervall 2 2. (0., 2.);;
- : float * float = (1., 1.5)
# intervall 3 2. (0., 2.);;
```



```
- : float * float = (1.25, 1.5)
# intervall 4 2. (0., 2.);
- : float * float = (1.375, 1.5)
# intervall 5 2. (0., 2.);
- : float * float = (1.375, 1.4375)
# intervall 6 2. (0., 2.);
- : float * float = (1.40625, 1.4375)
```

```
e) # let rec intervall_f f a (l, r) =
    if r -. l < f
    then (l, r)
    else let m = (l +. r) /. 2. in
        if m *. m > a
        then intervall_f f a (l, m)
        else intervall_f f a (m, r);;
val intervall_f : float -> float -> float * float -> float * float = <fun>
# intervall_f 0.0001 2. (0., 2.);
- : float * float = (1.4141845703125, 1.41424560546875)
```

### Aufgabe 7 Anzahl der Aufrufe von Fakultäts- und Fibonaccifunktion (Lösungsvorschlag)

a) Die Funktion, die die Aufrufe der Fakultätsfunktion zählt, lautet:

```
let rec anzahlFakultaet n = match n with
| 0 -> 1
| n -> 1 + anzahlFakultaet (n-1);;
```

b) Die Funktion, die die Aufrufe der Fibonaccifunktion zählt, lautet:

```
let rec anzahlFibonacci n = match n with
| 0 | 1 -> 1
| n -> 1 + anzahlFibonacci (n-1) + anzahlFibonacci (n-2)
;;
```

- c) Bereits nach wenigen Tests zeigt sich, dass die Anzahl der Aufrufe der Fakultätsfunktion proportional zum Funktionsparameter  $n$  ist. Die Anzahl der Aufrufe der Fibonaccifunktion hängt dagegen nichtlinear mit dem Funktionsparameter  $n$  zusammen. Man kann zeigen (später) dass die Anzahl der Aufrufe exponentiell wächst

### **Aufgabe 8 Passende Jahre für bestimmtes Osterdatum (Lösungsvorschlag)**

Eine Implementierung könnte folgendermaßen beschaffen sein:

```

type monat = Januar | Februar | März | April | Mai
            | Juni | Juli | August | September
            | Oktober | November | Dezember;;

type datum = Datum of (int * monat * int);;

let tabelle_m j = match j with
  | j when (j > 1582 && j < 1700) -> 22
  | j when (j > 1699 && j < 1900) -> 23
  | j when (j > 1899 && j < 2200) -> 24
  | j when (j > 2199 && j < 2300) -> 25
  | _ -> failwith "Falsche Eingabe";;

let tabelle_n j = match j with
  | j when (j > 1582 && j < 1700) -> 2
  | j when (j > 1699 && j < 1800) -> 3
  | j when (j > 1799 && j < 1900) -> 4
  | j when (j > 1899 && j < 2100) -> 5
  | j when (j > 2099 && j < 2200) -> 6
  | j when (j > 2199 && j < 2300) -> 0
  | _ -> failwith "Falsche Eingabe";;

let d j = (19 * (j mod 19) + tabelle_m j) mod 30;;

let e j = (2 * (j mod 4) + 4 * (j mod 7) + 6 * (d j) + tabelle_n j) mod 7;;

let ostersonntagOhneZusatzregel j = let d_plus_e = d j + e j
  in if ((22 + d_plus_e) < 32) then Datum(22 + d_plus_e, März, j)
  else Datum(d_plus_e - 9, April, j);;

let ostersonntag j =
  let tag = ostersonntagOhneZusatzregel j in
  match tag with
  | Datum(26, April, j) -> Datum(19, April, j)
  | Datum(25, April, j) when ((d j = 28) && (e j = 6) && (j mod 19 > 10))
    -> Datum(18, April, j)
  | _ -> tag;;

let rec jahrFuerOstertermin(vonJahr, bisJahr, amTag, imMonat) =
  if (vonJahr > bisJahr)
  then bisJahr
  else
    let ostern = ostersonntag vonJahr in
    match ostern with

```

```

    | Datum (t, m, vonJahr) when (t == amTag && m == imMonat) -> vonJahr
    | Datum (t, m, _) -> jahrFuerOstertermin(vonJahr + 1, bisJahr, amTag, imMonat);;

let rec jahreFuerOstertermin(vonJahr, bisJahr, amTag, imMonat, jahresliste) =
  if (vonJahr > bisJahr)
  then match jahresliste with
    | head :: tail -> tail
    | [] -> []
  else
    let j = jahrFuerOstertermin(vonJahr, bisJahr, amTag, imMonat) in
      jahreFuerOstertermin(j + 1, bisJahr, amTag, imMonat, [j] @ jahresliste);;

jahreFuerOstertermin(1600, 2200, 22, April, []);;

```