

## Übungen zu Informatik I

**Aufgabe 14-1**                      **Wiederholung: Typisierung**                      (keine Abgabe)

Die SML-Funktionen haben folgenden Typ

- a) `fun add1 n = n + 1`  
Typ: `int → int`
- b) `fun add2 x = x + 2.0`  
Typ: `real → real`
- c) `fun rev nil = nil`  
| `rev (x::xs) = (rev xs) @ [x]`  
Typ: `'a list → 'a list`
- d) `fun addl(n,nil) = nil`  
| `addl(n,m::ms) = (n+m)::(addl(n,ms))`  
Typ: `int * int list → int list`
- e) `fun mappend f [] = []`  
| `mappend f (x::xs) = (f x) @ (mappend f xs);;`  
Typ: `('a → 'b list) → 'a list → 'b list`

**Aufgabe 14-2**                      **Wiederholung: Additive Terme**                      (keine Abgabe)

`datatype Term = Z of int | Sum of Term * Term | Dif of Term * Term`

```
fun eval (Z n) = n
  | eval (Sum(t1,t2)) = eval(t1) + eval(t2)
  | eval (Dif(t1,t2)) = eval(t1) - eval(t2);
```

```
eval(Sum(Z(5),Dif(Z(10),Z(2))));
```

**Aufgabe 14-3**                      **Addition**                      (keine Abgabe)

- a) Wir schreiben  $f$  statt  $add$  und ersetzen von  $f$  durch  $F(f)$  auf der linken Seite; intuitiv:

$F(f)(x, y) = \text{if } x = 0 \text{ then } y \text{ else } f(x-1, y) + 1$

Formal:

$$F(f)(x, y) = \begin{cases} y & x = 0 \\ f(x-1, y) + 1 & \text{sonst} \end{cases}$$

b) Durch Induktion:

$$F^i(\Omega)(x, y) = \begin{cases} x + y & x < i \\ \omega & \text{sonst} \end{cases}$$

c) Berechnung von  $\text{fix}(F)$  nach Kleene:

$$\begin{aligned} \text{fix}(F)(x, y) &= \sup\{F^i(\Omega)(x, y) \mid i \in \mathbb{N}_0\} \\ &= \sup\{F^i(\Omega)(x, y) \mid x < i\} \cup \{F^i(\Omega)(x, y) \mid x \geq i\} \\ &= \sup\{x + y, \omega\} \\ &= x + y \end{aligned}$$

#### Aufgabe 14-4

#### Imperative Programmierung

(keine Abgabe)

a) Die Lösung in Pseudocode-Notation:

```
function potit(x: real, n: int) real:  
  var erg: real;  
  var j: int;  
  erg := 1; j := 1;  
  while j ≤ n do erg := x * erg; j := j + 1 end;  
  erg
```

b) Die Funktion *istquadratit* lässt sich wie folgt definieren:

```
function istquadratit(n: int) bool:  
  var erg: bool;  
  var j: int;  
  erg := false;  
  j := 0;  
  while j ≤ n ∧ ¬erg do  
    if n = j2 then erg := true  
    else j := j + 1 end;  
  end;  
  erg
```

c) Eine mögliche Lösung ist die folgende:

```
function teilersummeit(n: int) int:  
  var erg: int;  
  var i: int;  
  erg := 0;  
  i := 1;  
  while i ≤ n do  
    if n mod i = 0 then erg := erg + i; i := i + 1  
    else i := i + 1 end;  
  end;  
  erg
```

d) Die Funktion *mapit* in Pseudocode-Notation:

```
function mapit( $f : \alpha \rightarrow \beta$ ,  $l : \alpha$  list)  $\beta$  list :  
  var erg :  $\beta$  list ;  
  erg := nil ;  
  while  $\neg$ null(l) do erg := erg@[f(hd(l))]; l := tl(l)  
  end ;  
  erg
```

e) Die Funktion *foldlit* kann man wie folgt definieren:

```
function foldlit ( $f : \alpha \times \beta \rightarrow \beta$ ,  $l : \alpha$  list ,  $z : \beta$ )  $\beta$  :  
  var erg :  $\beta$  ;  
  erg := z ;  
  while  $\neg$ null(l) do erg := f(hd(l), erg); l := tl(l) end ;  
  erg
```