Aufgabe 1: Effiziente funktionale Schlangen (12)

```
structure FQueue :>
  sig
    type 'a queue
    val empty  : 'a queue
    val insert : 'a * 'a queue -> 'a queue
    val head   : 'a queue -> 'a        (* Empty *)
    val tail   : 'a queue -> 'a queue  (* Empty *)
  end
=
struct
  type 'a queue = 'a list * 'a list

  val empty = ([], [])

  fun insert (a, ([], _)) = ([a], [])
    | insert (a, (q , r)) = (q, a::r)

  fun head (q, r) = hd q

  fun tail ([x], r) = (rev r, [])
    | tail (q  , r) = (tl q, r)
end
```

Aufgabe 2: Rot-Schwarz-Bäume (2+8+2+2)

(a)
```
datatype color = R | B
datatype ctree = E | N of color * ctree * int * ctree

fun inpro  E            = []
  | inpro (N(_,t,x,t')) = inpro t @ [x] @ inpro t'

fun check (x::x'::xr) = x<x' andalso check(x'::xr)
  | check     _       = true

fun searchTree t = check(inpro t)
```

(b) Ein C-Baum heißt *rot-balanciert* genau dann, wenn keiner seiner Pfade zwei aufeinanderfolgende rote Knoten enthält.

(c) Ein C-Baum heißt *schwarz-balanciert* genau dann, wenn seine maximalen Pfade alle die gleiche Anzahl schwarzer Knoten enthalten.

Aufgabe 3: Zahldarstellung (6+6)

```
fun from10 b x = if x<b then [x]
                 else from10 b (x div b) @ [x mod b]

fun to10 b = foldl (fn (x,s) => s*b+x) 0
```

Aufgabe 4: Freie Bezeichner (12)

```
fun freeIds' xs (Con c)           = []
  | freeIds' xs (Id  s)           = if List.exists (fn x => x=s) xs
                                    then [] else [s]
  | freeIds' xs (Op(e1,ops,e2))   = freeIds' xs e1 @ freeIds' xs e2
  | freeIds' xs (If(e1,e2,e3))    = freeIds' xs e1 @ freeIds' xs e2
                                                   @ freeIds' xs e3
  | freeIds' xs (Abs(s,t,e))      = freeIds' (s::xs) e
  | freeIds' xs (App(e1,e2))      = freeIds' xs e1 @ freeIds' xs e2

fun freeIds e = freeIds' [] e
```

Aufgabe 5: Statische Semantik von F (9)

$$\frac{T \vdash e_1 \Rightarrow \texttt{bool} \quad T \vdash e_2 \Rightarrow t \quad T \vdash e_3 \Rightarrow t}{T \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \Rightarrow t}$$

$$\frac{T + \{x \mapsto t\} \vdash e \Rightarrow t'}{T \vdash \texttt{fn}\ x : t\ \texttt{=>}\ e \Rightarrow t \to t'}$$

$$\frac{T \vdash e_1 \Rightarrow t' \to t \quad T \vdash e_2 \Rightarrow t'}{T \vdash e_1\ e_2 \Rightarrow t}$$

Aufgabe 6: Dynamische Semantik von F (9)

$$\frac{V \vdash e_1 \Rightarrow 0 \qquad V \vdash e_3 \Rightarrow v}{V \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \Rightarrow v}$$

$$\frac{}{V \vdash \texttt{fn}\ x : t\ \texttt{=>}\ e \Rightarrow (x, e, V)}$$

$$\frac{V \vdash e_1 \Rightarrow (x, e, V') \quad V \vdash e_2 \Rightarrow v_2 \quad V' + \{x \mapsto v_2\} \vdash e \Rightarrow v}{V \vdash e_1\ e_2 \Rightarrow v}$$

Aufgabe 7: Kontextfreie Syntax von arithmetischen Ausdrücken (4+8+8)

(a)

$$\begin{aligned} \mathit{exp} &= [\ \mathit{exp}\ \texttt{"+"}\ ]\ \mathit{mulexp} \\ \mathit{mulexp} &= [\ \mathit{mulexp}\ \texttt{"*"}\ ]\ \mathit{atexp} \\ \mathit{atexp} &= \mathit{identifier} \mid \texttt{"("}\ \mathit{exp}\ \texttt{")"} \end{aligned}$$

(b)

```
fun exp (Add(e1, e2))    = (exp e1) ^ "+" ^ (mulexp e2)
 |  exp  e               = mulexp e
and mulexp (Mul(e1, e2)) = (mulexp e1) ^ "*" ^ (atexp e2)
 |  mulexp  e            = atexp e
and atexp (Id s)         = s
 |  atexp  e             = "(" ^ (exp e) ^ ")"
```

(c)

$$\begin{aligned} \mathit{exp} &= \mathit{mulexp}\ \ \mathit{exp}' \\ \mathit{exp}' &= [\ \texttt{"+"}\ \ \mathit{mulexp}\ \ \mathit{exp}'\ ] \\ \mathit{mulexp} &= \mathit{atexp}\ \ \mathit{mulexp}' \\ \mathit{mulexp}' &= [\ \texttt{"*"}\ \ \mathit{atexp}\ \ \mathit{mulexp}'\ ] \\ \mathit{atexp} &= \mathit{identifier} \mid \texttt{"("}\ \mathit{exp}\ \texttt{")"} \end{aligned}$$

Aufgabe 8: Generatoren (8)

```
fun newGenerator f =
  let
    val r = ref 0
  in
    fn () => f(!r) before r:= !r+1
  end
```

Aufgabe 9: Imperative Listen (12)

```
fun refs’ r rs =
  case !r of
    Nil        => rs
  | Cons(_,r’) => if List.exists (fn r’’ => r’’=r’) rs
                  then rs
                  else refs’ r’ (r’::rs)

fun refs Nil          = nil
  | refs (Cons(_, r)) = refs’ r [r]
```

Aufgabe 10: Maschinennahe Programmierung (10+6+8)

```
[proc(1, 4),
 getF~1, con 0, callR 4,
 proc(2,18),
 getF~1, getF~1, mul, con 1, getF~2, add, leq, cbranch 5,
 con 1, getF~1, sub, return,
 getF~2, con 1, getF~1, add, callR 4]
```

```
fun sqrt x =
  let
    val n = ref 0
  in
    while !n * !n <= x do n:= !n+1 ;
    !n-1
  end
```

```
[con 0,
 getS 0, getS 1, getS 1, mul, leq, cbranch 6,
 con 1, getS 1, add, putS 1, branch~10,
 con 1, getS 1, sub, halt]
```

Aufgabe 11: Übersetzung (18)

```
fun compile env e =
  case e of
    Con i        => [con i]
  | Var i        => [getS(env i)]
  | Add(e1,e2)   => compile env e2 @ compile env e1 @ [add]
  | Leq(e1,e2)   => compile env e2 @ compile env e1 @ [leq]
  | If(e1,e2,e3) =>
      let
        val c1 = compile env e1
        val c2 = compile env e2
        val c3 = compile env e3
      in
        c1 @ [cbranch (length c2 + 2)] @
        c2 @ [branch (length c3 + 1)] @
        c3
      end
```