Programmierung WS 2002 / 03:
Musterlösung zum 11. Übungsblatt

Prof. Dr. Gert Smolka, Dipl.-Inform. Thorsten Brunklaus

Aufgabe 11.1: Geschlossene Ausdrücke und freie Bezeichner $(16 = 8 + 8)$

(a)
```
fun closed' (Con _) ids           = true
  | closed' (Id v)  ids           = List.exists (fn x => x=v) ids
  | closed' (Op(e,_,e')) ids      = closed' e ids andalso closed' e' ids
  | closed' (If(e,e',e'')) ids    =
    closed' e ids andalso closed' e' ids andalso closed' e'' ids
  | closed' (Abs(v,_,e)) ids      = closed' e (v::ids)
  | closed' (App(e,e')) ids       = closed' e ids andalso closed' e' ids
  | closed' (Rec(v,v',_,_,e)) ids = closed' e (v::v'::ids)

fun closed exp = closed' exp nil
```

(b)
```
fun freeIds' ids (Con _)           = nil
  | freeIds' ids (Id v) =
  if List.exists (fn x => x=v) ids then nil else [v]
  | freeIds' ids (Op(e,_,e'))      = (freeIds' ids e) @ (freeIds' ids e')
  | freeIds' ids (If(e,e',e''))    =
  (freeIds' ids e) @ (freeIds' ids e') @ (freeIds' ids e'')
  | freeIds' ids (Abs(v,_,e))      = freeIds' (v::ids) e
  | freeIds' ids (App(e,e'))       = (freeIds' ids e) @ (freeIds' ids e')
  | freeIds' ids (Rec(v,v',_,_,e)) = freeIds' (v::v'::ids) e

fun freeIds exp = freeIds' nil exp
```

Aufgabe 11.2: Umgebungen (5)

```
signature Env =
  sig
    type 'a env
    exception Unbound of id
    val empty  : 'a env
    val insert : id * 'a * 'a env -> 'a env
    val lookup : id * 'a env -> 'a  (* Unbound *)
end


structure Env :> Env =
  struct
    type 'a env = id -> 'a
    exception Unbound of id
    val empty = (fn id => raise (Unbound id))
    fun insert (id, v, env) = (fn x => if id=x then v else env x)
    fun lookup (id, env) = env id
  end
```

Aufgabe 11.3: Rekursive Prozeduren ($15 = 5 + 5 + 5$)

```
datatype exp = ...
             | Rec of id * id * ty * ty * exp

fun elab T ...
  | elab T (Rec(f,x,t',t,e)) =
      if (elab (insert(x,t',(insert(f,Arrow(t',t),T)))) e) = t then Arrow(t',t)
      else raise Error("ill-typed recursive procedure")

datatype value = ...
               | RProc of id * id * exp * value env

fun eval V ...
  | eval V (Rec(s,s',_,_,e)) = RProc(s,s',e,V)
  | eval V (App(e1,e2))    = (case (eval V e1, eval V e2) of
                                (Proc(s,e,V'), v) =>
                                  eval (insert(s,v,V')) e
                              | (p as RProc(s,s',e,V'), v') =>
                                eval (insert(s', v', insert(s, p, V'))) e
                              | _ => raise Error "type error")
```

Aufgabe 11.4: Erweiterung von F um Paare ($17 = 3 + 3 + (1 + 3 + 3 + 1 + 3)$)

(a)

$$\frac{T \vdash e_1 \Rightarrow t_1 \quad T \vdash e_2 \Rightarrow t_2}{T \vdash (e_1, e_2) \Rightarrow t_1 * t_2} \qquad \frac{T \vdash e \Rightarrow t_1 * t_2}{T \vdash \texttt{fst}\ e \Rightarrow t_1} \qquad \frac{T \vdash e \Rightarrow t_1 * t_2}{T \vdash \texttt{snd}\ e \Rightarrow t_2}$$

(b)

$$\frac{V \vdash e_1 \Rightarrow v_1 \quad V \vdash e_2 \Rightarrow v_2}{V \vdash (e_1, e_2) \Rightarrow (v_1, v_2)} \qquad \frac{V \vdash e \Rightarrow (v_1, v_2)}{V \vdash \texttt{fst}\ e \Rightarrow v_1} \qquad \frac{V \vdash e \Rightarrow (v_1, v_2)}{V \vdash \texttt{snd}\ e \Rightarrow v_2}$$

(c)

```
datatype ty  = ...
             | Pair of ty * ty

datatype exp = ...
             | Tup of exp * exp | Fst of exp | Snd of exp

fun elab T ...
  | elab T (Tup(e1,e2)) = let val t1 = elab T e1
                              val t2 = elab T e2
                          in Pair(t1, t2)
                          end
  | elab T (Fst e) = let val t = elab T e
                     in case t of
                          Pair(t,t') => t
                        | _ => raise Error "ill-typed fst"
                     end
  | elab T (Snd e) = let val t = elab T e
                     in case t of
                          Pair(t,t') => t'
                        | _ => raise Error "ill-typed snd"
                     end
```

```
datatype value = ...
               | VPair of value * value

fun eval V ...
  | eval V (Tup(e1,e2)) = VPair(eval V e1, eval V e2)
  | eval V (Fst e) = (case eval V e of
                        VPair(v, _) => v
                      | _           => raise Error "type error")
  | eval V (Snd e) = (case eval V e of
                        VPair(_, v) => v
                      | _           => raise Error "type error")
```

Aufgabe 11.5: Erweiterung von F um Listen $(27 = 5 + 5 + (1 + 5 + 5 + 1 + 5))$

(a)

$$\frac{}{T \vdash \texttt{nil}\colon t \Rightarrow t\ \texttt{list}} \qquad \frac{T \vdash e_1 \Rightarrow t \quad T \vdash e_2 \Rightarrow t\ \texttt{list}}{T \vdash e_1 :: e_2 \Rightarrow t\ \texttt{list}}$$

$$\frac{T \vdash e \Rightarrow t\ \texttt{list}}{T \vdash \texttt{null}\, e \Rightarrow \texttt{bool}} \qquad \frac{T \vdash e \Rightarrow t\ \texttt{list}}{T \vdash \texttt{hd}\, e \Rightarrow t} \qquad \frac{T \vdash e \Rightarrow t\ \texttt{list}}{T \vdash \texttt{tl}\, e \Rightarrow t\ \texttt{list}}$$

(b)

$$\frac{}{V \vdash \texttt{nil}\colon t \Rightarrow 0} \qquad \frac{V \vdash e_1 \Rightarrow v_1 \quad V \vdash e_2 \Rightarrow v_2}{V \vdash e_1 :: e_2 \Rightarrow v_1 :: v_2}$$

$$\frac{V \vdash e \Rightarrow 0}{V \vdash \texttt{null}\, e \Rightarrow 1} \qquad \frac{V \vdash e \Rightarrow v \quad v \neq 0}{V \vdash \texttt{null}\, e \Rightarrow 0}$$

$$\frac{V \vdash e \Rightarrow v_1 :: v_2}{V \vdash \texttt{hd}\, e \Rightarrow v_1} \qquad \frac{V \vdash e \Rightarrow v_1 :: v_2}{V \vdash \texttt{tl}\, e \Rightarrow v2}$$

(c)

```
datatype ty  = ...
             | List of ty

datatype exp = ...
             | Nil of ty | Cons of exp * exp
             | Null of exp | Hd of exp | Tl of exp
```

```
fun elab T ...
  | elab T (Nil t) = List t
  | elab T (Cons(e, e')) = let val t = elab T e
                               val t' = elab T e'
                           in if (List t) = t' then List t
                              else raise Error "ill-typed cons"
                           end
  | elab T (Null e) = (case elab T e of
                         List _ => Bool
                       | _      => raise Error "ill-typed null")
  | elab T (Hd e) = (case elab T e of
                       List t => t
                     | _      => raise Error "ill-typed hd")
  | elab T (Tl e) = (case elab T e of
                       (t as List t') => t
                     | _              => raise Error "ill-typed tl")


datatype value = ...
               | VCons of value * value

fun eval V ...
  | eval V (Nil _) = (IV 0)
  | eval V (Cons(e1,e2)) = VCons(eval V e1, eval V e2)
  | eval V (Null e) = (case eval V e of
                         IV 0 => IV 1
                       | _    => IV 0)
  | eval V (Hd e) = (case eval V e of
                       VCons(v,_) => v
                     | _          => raise Error "type error")
  | eval V (Tl e) = (case eval V e of
                       VCons(_,v) => v
                     | _          => raise Error "type error")
```

Aufgabe 11.6: Programme $(20 = 10 + 10)$

(a)
```
fun elabPro env nil             = env
  | elabPro env ((id, exp)::es) = let val t = elab env exp
                                  in elabPro (insert(id, t, env)) es
                                  end
```

(b)
```
fun evalPro env  nil            = env
  | evalPro env ((id, exp)::es) = let val v = eval env exp
                                  in evalPro (insert(id, v, env)) es
                                  end
```