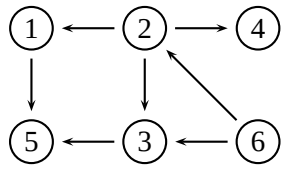
Programmierung WS 2002 / 03:
Musterlösung zum 15. Übungsblatt

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

Aufgabe 15.1: Graphen $(16 = 4 * 4)$

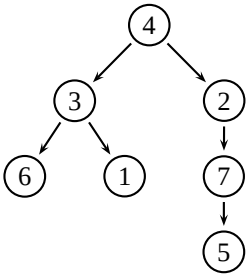
(a)



Größe: 6, Tiefe: 3, Quellen: {6}, Senken: {4, 5}

G ist azyklisch. G ist zusammenhängend, aber nicht stark zusammenhängend, denn 2 ist von 1 aus nicht erreichbar. G ist kein Wald, denn 3 hat zwei Vorgänger.

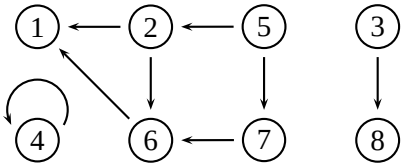
(b)



Größe: 7, Tiefe: 3, Quellen: {4}, Senken: {1, 5, 6}

G ist azyklisch. G ist zusammenhängend, aber nicht stark zusammenhängend, denn 2 ist von 1 aus nicht erreichbar. G ist ein Baum.

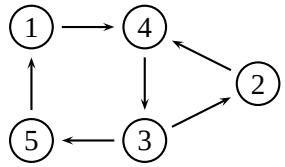
(c)



Größe: 8, Tiefe: 3, Quellen: {3, 5}, Senken: {1, 8}

G ist zyklisch mit Zyklus $\langle 4, 4 \rangle$. G ist nicht zusammenhängend. G ist kein Wald, denn 6 hat zwei Vorgänger.

(d)



Größe: 5, Tiefe: 4, Quellen: $\emptyset$, Senken: $\emptyset$

G ist zyklisch mit Zyklus $\langle 3, 2, 4, 3 \rangle$. G ist stark zusammenhängend. G ist kein Wald, denn G ist zyklisch.

Aufgabe 15.2: Graphen mit Standard ML $(24 = 3 * 8)$

(a)
```
fun reset nil                                = ()
  | reset ((g as ref(G(true, x, gs)))::xr) =
    (g := G(false, x, gs); reset (gs @ xr))
  | reset ((g as ref(G(false, x, gs)))::xr) = reset xr
```

(b)
```
fun tree' nil                                 = true
  | tree' ((ref(G(true, _, _)))::xr)          = false
  | tree' ((g as ref(G(false, x, gs)))::xr) =
    (g := G(true, x, gs); tree' (gs @ xr))

fun tree (g as ref(G(_, x, gs))) =
    (g := G(true, x, gs); (tree' gs before reset [g]))
```

(c)
```
fun count' n nil          = n
  | count' n ((g as ref(G(false, x, gs)))::xr) =
    (g := G(true, x, gs); count' (n + 1) (gs @ xr))
  | count' n (g as ref(G(true, x, gs))::xr) = count' n xr

fun count gs = count' 0 gs before reset gs
```

Aufgabe 15.3: Listen mit VH $(24 = 3 * 8)$

(a)
```
val cons = [proc(2, 5), arg 2, arg 1, new 2, return]

val null = [proc(1, 5), con 0, arg 1, eq, return]

val hd   = [proc(1, 4), arg 1, getH 1, return]

val tl   = [proc(1, 4), arg 1, getH 2, return]
```

(b)
```
val gen = [proc(1, 4), arg 1, con 0, callR 4,
           proc(2, 14), con ~1, arg 2, leq, cbranch 3,
           arg 1, return,
           con 1, arg 2, sub, arg 1, arg 2, new 2, callR 4]
```

(c)
```
val sum = [proc(1, 4), arg 1, con 0, callR 4,
           proc(2, 14), con 0, arg 2, eq, cbranch 3,
           arg 1, return,
           arg 2, getH 2, arg 2, getH 1, arg 1, add, callR 4]
```

Aufgabe 15.4: Speicher $(36 = 16 + 20)$

```
structure Store :> STORE =
  struct
    exception Error

   type value = int

    val heapSize = 100
    val heap = Array.array(heapSize, 0)
    val fha = valOf(Int.maxInt)-heapSize+1  (* first heap address *)
    val bp = ref fha     (* first cell in current heap section *)
    val hp = ref(fha-1)  (* topmost heap cell allocated *)

    fun isI v = v<fha

    fun getH ha = Array.sub(heap, ha-fha)

    fun putH(ha,v) = Array.update(heap, ha-fha, v)

    fun alloc v = (hp:= !hp+1 ; putH(!hp,v))

    fun cons(x, xs) = if (!hp - fha + 4) > heapSize
                      then raise Error
                      else (alloc(~1);
                            alloc(x);
                            alloc(xs);
                            !hp-2)

    val wc  = ref ~1  (* counter for while loop  *)

    (* Interface Section *)

    fun fromInt i = if isI i then i else raise Error
    fun toInt v = if isI v then v else raise Error
    fun isInt v = isI v
    fun pair(x, xs) = cons(x, xs)
    fun first v = getH (v + 1)
    fun second v = getH (v + 2)
    fun updateF(v, x) = putH(v + 1, x)
    fun updateS(v, x) = putH(v + 2, x)

    val iwc = ref ~1  (* counter for inner while loop  *)

    fun clone' ha = (iwc:= ha ;
                     while !iwc < ha+3 do
                       (alloc(getH(!iwc)) ; iwc:= !iwc+1))

    fun copyBlock oldba =
        if not(isI(getH oldba)) then getH oldba
        else let
            val newba = !hp+1
          in
            clone' oldba;
            putH(oldba,newba);
            newba
          end
```

```
  fun clone v =
      if isI v then v
      else
        let
          val v' = copyBlock v
        in
          (wc := v';
           while !wc <= !hp do
             (if isI(getH(!wc)) then ()
              else putH(!wc, copyBlock(getH(!wc))) ;
              wc:= !wc+1);
           wc := fha;
           while !wc < v' do
             (putH(!wc, ~1);
              wc := !wc + 3);
           v')
        end
end
```