Technische Universität München
Fakultät für Informatik
Prof. Dr. H. Seidl

SoS 2005
12. Oktober 2005

Lösungsvorschläge der Wiederholungsklausur zu Einführung in die Informatik II

Aufgabe 1 Java-GUI

```
import java.awt.*;
import java.awt.event.*;

public class Main extends Frame implements ActionListener
{
  private Button button = new Button("Quadriere");
  private TextField value = new TextField();

  public Main()
  {
    add(value, BorderLayout.NORTH);
    add(button);
    button.addActionListener(this);
  }

  public void actionPerformed(ActionEvent e)
  {
    try
    {
      int x = Integer.parseInt(value.getText());
      value.setText((x * x) + "");
    }
    catch(Exception ex) {}
  }

  public static void main(String[] args)
  {
    Frame frm = new Main();
    frm.setVisible(true);
  }
}
```

Aufgabe 2 **Datalog**

a) `sohn(X,Y) :- maennlich(X),kind(X,Y).`

b) `mutter(X,Y) :- weiblich(X),kind(Y,X).`

c)
```
nachfahre(X,Y) :- kind(X,Y).
nachfahre(X,Z) :- kind(X,Y),nachfahre(Y,Z).
```

d) `gemeinsames_kind(X,Y) :- kind(K,X),kind(K,Y).`

e)

$$\frac{\texttt{kind}(\texttt{hieronymus},\texttt{rosalinde}) \quad \dfrac{\texttt{kind}(\texttt{rosalinde},\texttt{konstanze})}{\texttt{nachfahre}(\texttt{rosalinde},\texttt{konstanze})}}{\texttt{nachfahre}(\texttt{hieronymus},\texttt{konstanze})}$$

Aufgabe 3 **Ocaml**

```
let rec map2 l1 l2 f =
  match (l1,l2) with
  (h1::r1,h2::r2) -> (f h1 h2)::(map2 r1 r2 f)
  | _ -> []

let zip l1 l2 = map2 l1 l2 (fun a b -> (a,b))

let v = zip [1;2;3] [4;5;6]

type node = int
type graph = node list array

let inverse g =
  let gi = Array.make (Array.length g) []
  in Array.iteri
       (fun i neighbours ->
          List.iter
            (fun neighbour -> gi.(neighbour) <- i::gi.(neighbour))
            neighbours)
       g; gi

let g = Array.of_list [[4];[0];[0;3];[1];[1;2;3]]
let g1 = inverse g
```

Aufgabe 4 Bipartiter graph

```
type node = int
type graph = node list array

exception NotBipartite

let rec isIn i l = match l with [] -> false | h::r -> if i=h then true else isIn i r

let makeBipartition g =
  let visited = Array.make (Array.length g) false
  in
    let rec doit i forPart1 (part1,part2) =
      (* forPart is true if the current partition is partition no 1 *)
      let visitNeighbours =
        (* auxiliary function to start the traversal for the neighbour nodes *)
            List.fold_left (fun (part1,part2) j -> doit j (not forPart1) (part1,part2))
      in
        if i>= Array.length g then (part1,part2)
        else
          if forPart1 then
    (* the current node is supposed to be in the first partition *)
            if (isIn i part2) then raise (NotBipartite)
    else
      if visited.(i) then (* the node was already visited *) (part1,part2)
         else (visited.(i) <- true; visitNeighbours (i::part1,part2) g.(i))
     else
  (* the current node is supposed to be in the second partition *)
          if (isIn i part1) then raise (NotBipartite)
  else if visited.(i) then (part1,part2)
       else (visited.(i) <- true; visitNeighbours (part1,i::part2) g.(i))
  in
    let rec visitAll i (part1,part2) =
      (* auxiliary function necessary if the graph is not connected *)
      if i>= Array.length g then (part1,part2)
      else if visited.(i) then visitAll (i+1) (part1,part2)
           else visitAll (i+1) (doit i true (part1,part2))
    in visitAll 0 ([],[])

let g = Array.of_list [[1;4];[2];[5];[];[3];[3];[7];[6]]
let g1 = makeBipartition g

(* Alternative *)

let bipart g =
  let nodeCount = Array.length g in
  let visited = Array.make nodeCount (-1) in
  let partitions = Array.make 2 [] in
  let rec dfs p n =
    if visited.(n) = -1 then
    begin
      visited.(n) <- p;
      partitions.(p) <- n::partitions.(p);
      List.iter (dfs ((p+1) mod 2)) g.(n)
```

```
  end
  else if visited.(n) != p then
    raise NotBipartite
in
  for n = 0 to nodeCount - 1 do
    if visited.(n) = -1 then
      dfs 0 n
  done;
  (partitions.(0),partitions.(1))
```

Aufgabe 5 Verifikation eines Min-Java-Programms (Lösungsvorschlag)

Für die Schleifen-Invariante raten wir:

$$F \equiv y = (1+x)^i \wedge xz = y-1$$

Dann ergibt sich:

$$D \equiv WP[\![i == n]\!](F,E) \equiv (i \neq n \wedge F) \vee (i = n \wedge E) \equiv (i \neq n \wedge F) \vee (i = n \wedge F) \equiv F$$
$$H \equiv WP[\![i = i+1]\!](D) \equiv y = (1+x)^{i+1} \wedge xz = y-1$$
$$G \equiv WP[\![y = y*(x+1)]\!](H) \equiv y(x+1) = (1+x)^{i+1} \wedge xz = y(x+1)-1$$
$$C \equiv WP[\![i = 0]\!](D) \equiv y = 1 \wedge xz = y-1$$
$$B \equiv WP[\![y = 1]\!](C) \equiv 1 = 1 \wedge xz = 1-1 = 0 \equiv xz = 0$$
$$A \equiv WP[\![z = 0]\!](B) \equiv 0 = 0 \equiv true$$

Schlussendlich bleibt noch festzustellen, dass gilt:

$$\begin{array}{rcl} WP[\![z = z+y]\!](G) & \equiv & y(x+1) = (1+x)^{i+1} \wedge x(z+y) = y(x+1)-1 \\ & \equiv & y(x+1) = (1+x)^{i+1} \wedge xz = y-1 \\ & \Leftarrow & y = (1+x)^i \wedge xz = y-1 \equiv F \end{array}$$

Aufgabe 6 Verifikation funktionaler Programme

Zu zeigen ist das Prädikat

flip (flip list) = list

Der Beweis erfolgt durch Induktion über die Länge der Liste `list`. Die Länge der Liste `list` bezeichnen wir im Folgenden mit n.

Induktionsanfang ($n = 0$):

```
     =      flip []
  =(Def.)   (fun list -> match list with
                             [] -> []
                           | (x,y)::tail -> (y,x)::(flip tail)) []
     =      match [] with
                [] -> []
              | (x,y)::tail -> (y,x)::(flip tail)
     =      []
```

Da aus $n = 0$ folgt, dass `list = []` ergibt sich:

`flip(flip list) = flip(flip []) = flip [] = [] = list`

Induktionsschluß: ($n \rightarrow n+1$) : Die Liste `list` hat die Länge $n+1$. Daraus folgt, dass eine Liste `list'` der Länge n und ein Paar `(x',y')` existiert mit `list = (x',y')::list'`.

Weiterhin ergibt sich:

```
     =      flip (a,b)::l
  =(Def.)   (fun list -> match list with
                             [] -> []
                           | (x,y)::tail -> (y,x)::(flip tail)) (a,b)::l
     =      match (a,b)::l with
                [] -> []
              | (x,y)::tail -> (y,x)::(flip tail)
     =      (b,a)::(flip l)
```

Daraus folgt:

```
            flip(flip list)
     =      flip(flip ((x',y')::list'))
     =      flip ((y',x')::(flip list'))
     =      (x',y')::(flip (flip list'))
  =(I.V.)   (x',y')::list'
     =      list
```