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

SoS 2005
16. Juli 2005

Lösungsvorschläge der Zwischenklausur zu Einführung in die Informatik II

Aufgabe 1 Java-GUI

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

public class Main extends Applet implements ActionListener
{
  private static final Color[] colors =
      new Color[]{Color.red, Color.blue};
  private static int curColor = 0;
  private Button button = new Button("Change Colour");

  public void init()
  {
    setBackground(colors[curColor]);
    add(button);
    button.addActionListener(this);
  }

  public void actionPerformed(ActionEvent e)
  {
    setBackground(colors[curColor = (curColor + 1) % 2]);
  }
}
```

Aufgabe 2 Bäume

```
type ’a tree = Node of ’a * ’a tree list

let example_tree = Node("Das",
                     [Node("ist",[]);Node("ein",
                       [Node("kleiner",[]);Node("Test",[])])])

let rec size (Node(_, children)) =
  List.fold_left (fun s -> fun child -> s + size child) 1 children

let rec map f (Node(e, children)) =
  Node(f e, List.map (map f) children)

let fold f s (Node(e, children)) =
  let rec fold_i acc (Node(e, children)) =
    let acc = f acc e
    in List.fold_left (fun acc -> fun child -> fold_i acc child) acc children
  in
  fold_i s (Node(e, children))

let to_list =
  fold (fun list -> fun element -> element::list) []
```

Aufgabe 3 Graphen

```
type node = int

type graph = node list array

let example_graph = [|[1;4];[0;3];[1];[0;4];[]|]

let add_edge g source target =
  g.(source) <- target::g.(source)

let count g s =
  let visited = Array.make (Array.length g) false in
  let rec doit n =
    if visited.(n) then
      0
    else
    begin
      visited.(n) <- true;
      1 + List.fold_left (fun c -> fun n -> c + (doit n)) 0 g.(n)
    end
  in doit s

let make_bidirected g =
  let ret_val = Array.make (Array.length g) [] in
  for i = 0 to Array.length g - 1 do
    List.iter (fun node -> add_edge ret_val i node; add_edge ret_val node i) g.(i)
  done;
  ret_val
```

Aufgabe 4 Greedy-Algorithmen

```
let indepented_set g =
  let ws = Array.make (Array.length g) true in
  let rec doit node acc =
    if node = Array.length g then
      acc
    else
    begin
      if ws.(node) then
      begin
        List.iter (fun succ -> ws.(succ) <- false) g.(node);
        doit (node + 1) (node::acc)
      end
      else
        doit (node + 1) acc
    end
  in doit 0 []
```

Aufgabe 5 Verifikation funktionaler Programme (Lösungsvorschlag)

Der Beweis der Behauptung

$$\texttt{length (app ks ls) = length ls + length ks} \tag{1}$$

erfolgt durch Induktion über die Länge von `ls`.

Induktionsanfang: Sei `ls = []`. Dann folgt

$$\begin{aligned}
\texttt{length (app [] ks)} &\overset{Def.}{=} \texttt{length ks} \\
&= 0 + \texttt{length ks} \\
&= \texttt{length []} + \texttt{length ks} \\
&= \texttt{length ls} + \texttt{length ks}
\end{aligned}$$

Induktionsschluss: Sei `ls' = l :: ls`. Dann folgt:

$$\begin{aligned}
\texttt{length (app (l :: ls) ks)} &\overset{Def.}{=} \texttt{length l :: (app ls ks)} \\
&\overset{Def.}{=} 1 + \texttt{length (app ls ks)} \\
&\overset{IV}{=} 1 + \texttt{length ls} + \texttt{length ks} \\
&\overset{Def.}{=} \texttt{length (l :: ls)} + \texttt{length ks}
\end{aligned}$$

Aufgabe 6 Verifikation eines Min-Java-Programms (Lösungsvorschlag)

Das Prädikat `A` ist identisch mit der Vorbedingung der zu beweisenden globalen Hypothese:

$$A \equiv (m > 0 \wedge n \geq 0 \wedge m = l_2 \wedge n = l_1)$$

Mit der Regel für bedingte Anweisungen folgen daraus die Prädikate `B`

$$B \equiv ((0 \leq n < m) \wedge m = l_2 \wedge n = l_1)$$

und `C`

$$C \equiv (m > 0 \wedge n \geq m \wedge m = l_2 \wedge n = l_1).$$

Durch Äquivalenzumformung erhalten wir aus `C` das Prädikat `C'`:

$$C' \equiv (m > 0 \wedge (n-m) \geq 0 \wedge m = l_2 \wedge (n-m) = l_1 - l_2),$$

so dass sich mit dem Zuweisungsaxiom das Prädikat `E` ergibt:

$$E \equiv (m > 0 \wedge n \geq 0 \wedge m = l_2 \wedge n = l_1 - l_2).$$

Ebenso folgt mit dem Zuweisungsaxiom aus dem Prädikat `B` ein Prädikat `D'`:

$$D' \equiv (0 \leq ret < m \wedge m = l_2 \wedge ret = l_1),$$

woraus nach Äquivalenzumformung und Abschwächung das Prädikat `D`

$$D \equiv (0 \leq ret < m \wedge \exists z \in Z : (z \geq 0) \wedge (l_2 \cdot z + ret = l_1),$$

folgt (Der Existenzquantor ist erfüllt für z = 0). Schließlich folgt mit der Regel für Funktionsaufrufe aus dem Prädikat `E` das Prädikat `D''`:

$$D'' \equiv (0 \leq ret < m \wedge \exists z \in Z : (z \geq 0) \wedge (l_2 \cdot z + ret = (l_1 - l_2)),$$

woraus sich nach Äquivalenzumformung letztendlich das Prädikat `D`

$$D \equiv (0 \leq ret < m \wedge \exists z' \in Z : (z' \geq 0) \wedge (l_2 \cdot z' + ret = l_1),$$

ergibt.