TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Sprachen und Beschreibungsstrukturen
Prof. Dr. Helmut Seidl

WS 2007/08
14. Februar 2007

Ausgegeben am:
Abgabe bis:

Lösungsvorschläge der Klausur zu Einführung in die Informatik II

Aufgabe 1 Verifikation funktionaler Programme (Lösungsvorschlag) (10 Punkte)

Zur Vereinfachung der Notation bezeichnen wir

$$\dfrac{\mathtt{f} = (\mathtt{fun\ l} \rightarrow \ldots) \quad (\mathtt{fun\ l} \rightarrow \ldots) \Rightarrow (\mathtt{fun\ l} \rightarrow \ldots)}{\mathtt{f} \Rightarrow (\mathtt{fun\ l} \rightarrow \ldots)}$$

mit π_f. Nach Voraussetzung gibt es einen Beweis π für $\mathtt{l} \Rightarrow [\mathtt{v_1}, \ldots, \mathtt{v_n}]$ für Werte $v_1 = (v'_1, v''_1), \ldots, v_n = (v'_n, v''_n)$. Wir zeigen die Behauptung per Induktion.

Induktionsanfang: $n = 0$:

$$\dfrac{\pi_\mathtt{f} \quad \pi \quad \dfrac{[\,] \Rightarrow [\,] \quad [\,] \Rightarrow [\,]}{(\mathtt{match}\ [\,]\ \mathtt{with}\ [\,]\ \rightarrow\ [\,] \mid (\mathtt{x},\mathtt{y}) :: \mathtt{r} -> \ \mathtt{x} :: \mathtt{y} :: (\mathtt{f\ r})) \Rightarrow [\,]}}{\mathtt{f\ l} \Rightarrow [\,]}$$

Induktionsschluss: $n > 0$:

Nach Induktionsvoraussetzung gibt es einen Wert $\mathtt{v}$ und einen Beweis π' für $f\ [v_2, \ldots, v_n] \Rightarrow v$. Damit gilt:

$$\dfrac{\pi_\mathtt{f} \quad \pi \quad \dfrac{[\mathtt{v_1}, \ldots, \mathtt{v_n}] \Rightarrow [\mathtt{v_1}, \ldots, \mathtt{v_n}] \quad \dfrac{\mathtt{v}'_1 \Rightarrow \mathtt{v}'_1 \quad \dfrac{\mathtt{v}''_1 \Rightarrow \mathtt{v}''_1 \quad \pi'}{\mathtt{v}''_1 :: (\mathtt{f}\ [\mathtt{v_2}, \ldots, \mathtt{v_n}]) \Rightarrow \mathtt{v}''_1 :: \mathtt{v}}}{\mathtt{v}'_1 :: \mathtt{v}''_1 :: (\mathtt{f}\ [\mathtt{v_2}, \ldots, \mathtt{v_n}]) \Rightarrow \mathtt{v}'_1 :: \mathtt{v}''_1 :: \mathtt{v}}}{(\mathtt{match}\ [\mathtt{v_1}, \ldots, \mathtt{v_n}]\ \mathtt{with}\ [\,]\ \rightarrow\ [\,] \mid (\mathtt{x},\mathtt{y}) :: \mathtt{r} -> \ \mathtt{x} :: \mathtt{y} :: (\mathtt{f\ r})) \Rightarrow \mathtt{v}'_1 :: \mathtt{v}''_1 :: \mathtt{v}}}{\mathtt{f\ l} \Rightarrow \mathtt{v}'_1 :: \mathtt{v}''_1 :: \mathtt{v}}$$

Punktevergabe:

a) Dass man Induktion nach Länge der List macht (sozusagen Rahmen): 2 Punkte

b) Induktionsanfang: 3 Punkte (Auf jede korrekte Regelanwendung einen Punkt)

c) Induktionsschluss: 5 Punkte (Auf jede korrekte Regelanwendung einen Punkt)

Aufgabe 2 **Verifikation imperativer Programme (Lösungsvorschlag)**

Wir benutzen im Folgenden die Konvention:

$$\sum_{j=k}^{l} \alpha = 0, \text{ wenn } k > l$$

- Wir wählen folgende Zusicherung:

$$A \equiv (\texttt{sum} = \sum_{j=0}^{i-1} 3^j) \wedge (\texttt{prod} = 3^{\texttt{i}}) \wedge (\texttt{n} \geq 0)$$

- Um die lokale Konsistenz am Zuweisung-Knoten `i=i+1` zu gewährleisten, wählt man für B die *weakest pre-condition* für die Zuweisung und A:

$$B \equiv \mathbf{WP}[\![\texttt{i = i+1}]\!](A) \equiv (\texttt{sum} = \sum_{j=0}^{i} 3^j) \wedge (\texttt{prod} = 3^{\texttt{i+1}}) \wedge (\texttt{n} \geq 0)$$

- Analog erhalten wir am Zuweisung-Knoten `prod = 3*prod`:

$$C \equiv \mathbf{WP}[\![\texttt{prod = 3*prod}]\!](B) \equiv (\texttt{sum} = \sum_{j=0}^{i} 3^j) \wedge (\texttt{prod} = 3^{\texttt{i}}) \wedge (\texttt{n} \geq 0)$$

- Analog erhalten wir am Zuweisung-Knoten `sum = sum+prod`:

$$D \equiv \mathbf{WP}[\![\texttt{sum=sum+prod}]\!](C) \equiv (\texttt{sum+prod} = \sum_{j=0}^{i} 3^j) \wedge (\texttt{prod} = 3^{\texttt{i}}) \wedge (\texttt{n} \geq 0)$$

- Um sicherzustellen, dass am Ende des Programms $\texttt{sum=3}^{\texttt{n}}$ für `n>=0` gilt, wählen wir für E:

$$E \equiv (\texttt{sum} = 3^{\texttt{n}})$$

- Die Konsistenz am `write(sum)`-Knoten ist trivialerweise gewährt durch:

$$F \equiv (\texttt{sum} = 3^{\texttt{n}})$$

- Um die lokale Konsistenz am Zuweisung-Knoten `sum=2*sum+1` zu gewährleisten, wählt man für G die *weakest pre-condition* für die Zuweisung und F:

$$G \equiv \mathbf{WP}[\![\texttt{sum=2*sum+1}]\!](F) \equiv (\texttt{2*sum+1} = 3^{\texttt{n}})$$

- Um die lokale Konsistenz am `if (i!=n)`-Knoten zu überprüfen, müssen wir jetzt zeigen, dass:

$$A \Rightarrow \mathbf{WP}[\![\texttt{i!=n}]\!](G, D)$$

  Wir zeigen dies, indem wir separat zeigen, dass:

(a) $A \wedge (i = n) \Rightarrow G$, d.h. $A \wedge (i = n) \Rightarrow (\texttt{2*sum+1} = 3^{\texttt{n}})$, UND

(b) $A \wedge (i! = n) \Rightarrow D$, d.h. $A \wedge (i! = n) \Rightarrow (\texttt{sum+prod} = \sum_{j=0}^{i} 3^j) \wedge (\texttt{prod} = 3^{\texttt{i}}) \wedge (\texttt{n} \geq 0)$

(a) Wir haben, dass:

$$A \wedge (i = n) \equiv (\texttt{sum} = \sum_{j=0}^{i-1} 3^j) \wedge (\texttt{prod} = 3^{\texttt{i}}) \wedge (\texttt{n} \geq 0) \wedge (i = n) \Rightarrow (\texttt{sum} = \sum_{j=0}^{n-1} 3^j)$$

Es bleibt zu zeigen, dass:

$$(\texttt{sum} = \sum_{j=0}^{n-1} 3^j) \wedge (\texttt{n} \geq 0) \Rightarrow (\texttt{2*sum+1} = 3^{\texttt{n}})$$

Wenn $n = 0$:

$$(\texttt{sum} = \sum_{j=0}^{n-1} 3^j = 0) \Rightarrow (\texttt{2*sum+1} = 1 = 3^{\texttt{0}} = 3^{\texttt{n}})$$

Wenn $n > 0$:

$$(\texttt{sum} = \sum_{j=0}^{n-1} 3^j = \frac{3^n - 1}{3 - 1}) \Rightarrow (\texttt{2*sum+1} = 2 * \frac{3^n - 1}{2} + 1 = 3^{\texttt{n}})$$

(b) Wir haben, dass:

$$A \wedge (i! = n) \equiv (\texttt{sum} = \sum_{j=0}^{i-1} 3^j) \wedge (\texttt{prod} = 3^{\texttt{i}}) \wedge (\texttt{n} \geq 0) \wedge (i! = n)$$

Darus folgt direkt, dass $(\texttt{prod} = 3^{\texttt{i}}) \wedge (\texttt{n} \geq 0)$.
Außerdem folgt es daraus, dass:

$$\texttt{sum+prod} = \sum_{j=0}^{i-1} 3^j + 3^{\texttt{i}} = \sum_{j=0}^{i} 3^j$$

- Um die lokale Konsistenz am Zuweisung-Knoten `i=0` zu gewährleisten, wählt man für H die *weakest pre-condition* für die Zuweisung und A:

$$H \equiv \mathbf{WP}[\![\texttt{i=0}]\!](F) \equiv (\texttt{sum} = \sum_{j=0}^{-1} 3^j) \wedge (\texttt{prod} = 3^{\texttt{0}}) \wedge (\texttt{n} \geq 0)$$

d.h.

$$H \equiv (\texttt{sum} = 0) \wedge (\texttt{prod} = 1) \wedge (\texttt{n} \geq 0)$$

- Um die lokale Konsistenz am Zuweisung-Knoten `prod=1` zu gewährleisten, wählt man für I die *weakest pre-condition* für die Zuweisung und H:

$$I \equiv (\texttt{sum} = 0) \wedge (\texttt{n} \geq 0)$$

- Um die lokale Konsistenz am Zuweisung-Knoten `sum=0` zu gewährleisten, wählt man für J die *weakest pre-condition* für die Zuweisung und I:

$$J \equiv (\mathtt{n} \geq 0)$$

- Um die lokale Konsistenz am Zuweisung-Knoten `n=-n` zu gewährleisten, wählt man für K die *weakest pre-condition* für die Zuweisung und J:

$$K \equiv (\mathtt{-n} \geq 0) \equiv (\mathtt{n} \leq 0)$$

- Die Konsistenz am `if(n>0)`-Knoten ist trivialerweise gewährt durch:

$$L \equiv \mathit{true}$$

- Schließlich ist die Konsistenz am `n=read()`-Knoten trivialerweise gewährt durch:

$$M \equiv \mathit{true}$$

Punktevergabe:

a) Schleifeninvariante: 3 Punkte

b) A -> B -> C -> D: 2 Punkte

c) E -> F -> G: 1 Punkt

d) G,D -> A: 2 Punkt

e) A -> H -> I -> J -> K: 1 Punkt

f) K,J -> L -> M: 1 Punkt

Aufgabe 3 **Verifikation rekursiver Programme (Lösungsvorschlag)** **(10 Punkte)**

Wir setzen $A :\equiv \mathtt{r} = l_\mathtt{r} \wedge i = l_\mathtt{i}$ und $B :\equiv \mathtt{r} = l_\mathtt{r} + l_\mathtt{i}^2 + 2l_\mathtt{i}$.

Aus der Gültigkeit von $A\ \mathtt{f}();B$ folgt die Gültigkeit von

$$\{(A[l_\mathtt{i} - 1/l_\mathtt{i}])[l_\mathtt{r} + 2l_\mathtt{i} + 1/l_\mathtt{r}]\}\ \mathtt{f}();\ \{(B[l_\mathtt{i} - 1/l_\mathtt{i}])[l_\mathtt{r} + 2l_\mathtt{i} + 1/l_\mathtt{r}]\}$$

Da

$$(B[l_\mathtt{i} - 1/l_\mathtt{i}])[l_\mathtt{r} + 2l_\mathtt{i} + 1/l_\mathtt{r}] \equiv \mathtt{r} = l_\mathtt{r} + 2l_\mathtt{i} + 1 + (l_\mathtt{i} - 1)^2 + 2(l_\mathtt{i} - 1) \equiv \mathtt{r} = l_\mathtt{r} + l_\mathtt{i}^2 + 2l_\mathtt{i}$$

ergibt sich explizit geschrieben die Gültigkeit von

$$\{r = l_\mathtt{r} + 2l_\mathtt{i} + 1 \wedge \mathtt{i} = l_\mathtt{i} - 1\}\ \mathtt{f}();\ \{\mathtt{r} = l_\mathtt{r} + l_\mathtt{i}^2 + 2l_\mathtt{i}\}$$

Damit ergibt sich:

$$C :\equiv r = l_\mathtt{r} + 2l_\mathtt{i} + 1 \wedge \mathtt{i} = l_\mathtt{i} - 1$$

und schließlich

$$D :\equiv \mathbf{WP}[\![\mathtt{i} = \mathtt{i} - 1]\!](C) \equiv r = l_\mathtt{r} + 2l_\mathtt{i} + 1 \wedge \mathtt{i} = l_\mathtt{i}$$

Damit haben wir

$$E :\equiv \mathbf{WP}[\![\mathtt{r} = \mathtt{r} + 2 * \mathtt{i} + 1]\!](D) \equiv r = l_\mathtt{r} \wedge \mathtt{i} = l_\mathtt{i}$$

Es bleibt die lokale Konsistenz an der bedingten Verzweigung zu zeigen. Wir müssen also zeigen, dass $A \equiv l_\mathtt{r} \wedge i = l_\mathtt{i}$

$$\mathbf{WP}[\![\mathtt{i!} = \mathtt{0}]\!](B,E) \equiv (\mathtt{i} = 0 \wedge \mathtt{r} = l_\mathtt{r} + l_\mathtt{i}^2 + 2l_\mathtt{i}) \vee (\mathtt{i} \neq 0 \wedge r = l_\mathtt{r} \wedge \mathtt{i} = l_\mathtt{i})$$

impliziert. Dazu nehmen wir als erstes an, dass A und $\mathtt{i} \neq 0$ gilt. Dann müssen wir zeigen, dass $r = l_\mathtt{r} \wedge \mathtt{i} = l_\mathtt{i}$ gilt, was offensichtlich ist. Als nächstes müssen wir annehmen, dass A und $i = 0$ gilt und müssen zeigen, dass dann $\mathtt{r} = l_\mathtt{r} + l_\mathtt{i}^2 + 2l_\mathtt{i}$ gilt. Aus $A \equiv \mathtt{r} = l_\mathtt{r} \wedge i = l_\mathtt{i}$ und $i = 0$ folgt, dass $\mathtt{r} = l_\mathtt{r}$ und $l_\mathtt{i} = 0$ woraus $\mathtt{r} = l_\mathtt{r} + l_\mathtt{i}^2 + 2l_\mathtt{i}$ direkt folgt.

Punktevergabe:

a) A und B korrekt: 1 Punkt

b) E, D und C korrekt: jeweils 1 Punkt

c) C -> D: 1 Punkt

d) D -> E: 1 Punkt

e) Adaption korrekt: 2 Punkte

f) B,E -> A: 2 Punkte

<u>Aufgabe 4</u> Schwach besetzte Vektoren (Lösungsvorschlag) (10 Punkte)

```
(*1*)

let sb_vektor v =
    let rec doit v p = match v with
        [] -> []
    |   x::xs -> if (x = 0) then doit xs (p+1) else (p,x)::doit xs (p+1)    in
    doit v 0

(*2*)

let rec add_sb_vektor v1 v2 =
    match (v1,v2) with
    ([],[]) -> [] |
    (_,[]) -> v1 |
    ([],_) -> v2 |
    ((i1,w1)::r1,(i2,w2)::r2) ->
        if i1 < i2 then (i1,w1)::add_sb_vektor r1 v2 else
        if i1 > i2 then (i2,w2)::add_sb_vektor v1 r2 else
        let sum = w1 + w2 in
        if sum = 0 then
            add_sb_vektor r1 r2 else
            (i1,sum)::add_sb_vektor r1 r2

(*3*)

let rec mult_sb_vektor v1 v2 =
    match (v1,v2) with
    ([],[]) -> 0 |
    (_,[]) -> 0 |
    ([],_) -> 0 |
    ((i1,w1)::r1,(i2,w2)::r2) ->
        if i1 < i2 then mult_sb_vektor r1 v2 else
        if i1 > i2 then mult_sb_vektor v1 r2 else
        let prod = w1*w2 in
            prod + mult_sb_vektor r1 r2
```

Punkteverteilung:

a) 3 Punkte fuer Hilfsfunktion + 1 Punkt fuer Initialisierung

b) 3 Punkte

c) 3 Punkte

<u>Aufgabe 5</u> Binäre Bäume (Lösungsvorschlag)

```
(* a *)
type 'a bintree = Leaf of 'a | Node of ('a bintree * 'a bintree)

(* b *)
let mapBintree f t =
  let rec doit t =
    match t with Leaf x -> Leaf (f x) | Node (t1,t2) -> Node (doit t1,doit t2)
  in doit t

(* c *)
let rec sumUp t = match t with Leaf x -> x | Node (t1,t2) -> (sumUp t1) + (sumUp t2)

(* d *)
let halveList l =
  let rec length l = match l with [] -> 0 | _::r -> 1 + length r in
  let rec take n l = match l with [] -> []
                  |  h::r ->  if n<=0 then []
                      else h::(take (n-1) r) in
  let rec drop n l = match l with [] -> []
                  |  h::r ->  if n<=0 then l
                      else drop (n-1) r in
  let half = (length l) / 2 in
  (take half l,drop half l)

(* e *)
exception EmptyList
let rec list2bintree l = match l with [] -> raise EmptyList
 | h::[] -> Leaf h
 |  _ -> let (l1,l2) = halveList l in Node (list2bintree l1,list2bintree l2)

(* f *)
let rec compress t =
  match t with
    Leaf x -> t
  | Node (t1,t2) ->
     match (compress t1,compress t2) with
       (Leaf x,Leaf y) -> if x=y then Leaf x
              else Node (Leaf x,Leaf y)
     | (t1',t2') -> Node (t1',t2')
```

Punkteverteilung:

a) 2 Punkte

b) 2 Punkte

c) 2 Punkte

d) 2 Punkte

e) 2 Punkte