



Universität Karlsruhe

Informatik 1 WS 00/01

Institut für Telematik, Forschungsgruppe C&M
Prof. Dr. S. Abeck, R. Scholderer

Abgabe bis: 13.02.2001 14:00 Uhr

Übungsblatt 14

Aufgabe

1. Gültigkeit und Lebensdauer (II)

Gegeben sei der folgende Abschnitt eines zuweisungsorientierten Programms:

```
┌ nat n = 2;

  proc erhoehe = (var nat x) :
    x := x+n;

  ┌ var nat n = 9;
    while n < 10 do erhoehe(n) od; n ┘
    * n ┘
```

Teilaufgabe 1:

Ordnen Sie den auftretenden Identifikatoren durch Pfeile Bindungen zu. (Setzen Sie wie üblich statische Bindung voraus.)

Teilaufgabe 2:

Geben Sie für die im Programmabschnitt auftretenden Identifikatoren ihre Lebensdauer und ihre Gültigkeit in einer Tabelle an. Führen Sie zunächst eine Umbenennung gleichnamiger Identifikatoren zur besseren Unterscheidung durch. (Auch hier ist natürlich statische Bindung vorauszusetzen.)

Teilaufgabe 3:

Welches Ergebnis liefert obiger Abschnitt

- bei statischer Bindung und
- bei dynamischer Bindung?

Aufgabe

2. Rekursive Prozeduren: Aufrufbaum

Gegeben sind folgende Deklarationen in einem zuweisungsorientierten Programm:

```
var seq nat s := conc(make(19), conc(make(13), conc(make(17), conc(make(11), make(15))));
```

```
proc sort = (nat u, nat o):  
  ⌈ if u ≥ o  
    then nop  
  else  
    nat m = (u+o) / 2;  
    sort(u,m); sort(m+1,o); merge(u,m,o)  
  fi  
⌋;
```

```
proc merge = (nat u, nat m, nat o):  
  ⌈ var seq nat t := empty;  
  var nat i, var nat j := u; m+1;  
  while i ≤ m ∧ j ≤ o do  
    if sel(i) ≤ sel(j)  
      then t := conc(t, make(sel(i))); i := i+1 od;  
    else t := conc(t, make(sel(j))); j := j+1 od;  
  fi  
od;  
while i ≤ m do t := conc(t, make(sel(i))); i := i+1 od;  
while j ≤ o do t := conc(t, make(sel(j))); j := j+1 od;  
i := u;  
while i ≤ o do  
  upd(i, first(t)); t := rest(t); i := i+1  
od⌋
```

Die weiteren im Code verwendeten Prozeduren und Funktionen haben die folgende Funktion:

fct conc = (nat seq p, nat seq q) nat seq verbindet die Sequenzen p und q zu einer Sequenz, indem s und t aneinander gehängt werden.

fct sel = (nat i) nat wählt das i-te Element aus s aus.

fct make = (nat x) seq macht aus der natürlichen Zahl x eine Sequenz mit dem Inhalt x.

proc upd = (nat i, nat j) kopiert die natürliche Zahl j an die i-te Stelle von s.

fct first = (seq l) nat gibt das erste Element von l zurück

fct rest = (seq l) seq gibt l ohne das erste Element von l zurück.

Teilaufgabe 1:

Zeichnen Sie den anfänglichen Wert von s auf.

Teilaufgabe 2:

Beschreiben Sie informell, welche Bedeutung die Parameter der Prozedur merge haben und was die Prozedur leistet.

Teilaufgabe 3:

Zeichnen Sie den Aufrufbaum zu sort(1,5). Geben Sie den Inhalt von s nach jedem Aufruf von merge an.

Aufgabe

3. Zusicherungsmethode

Gegeben sei folgendes zuweisungsorientiertes Programmstück S:

```

var seq m t := l; var m x := a; var nat n := 0;
while ¬isempty(t) do

    if first(t)  $\stackrel{?}{=} x$  then n := n + 1 else nop fi;
    t := rest(t);

od
    
```

Die im Code verwendeten Funktionen haben folgende Funktion:

fct conc = (nat seq p, nat seq q) nat seq verbindet die Sequenzen p und q zu einer Sequenz, indem s und t aneinander gehängt werden.

fct first = (seq l) nat gibt das erste Element von l zurück

fct rest = (seq l) seq gibt l ohne das erste Element von l zurück.

Teilaufgabe 1:

Geben Sie den Werteverlauf für die in S auftretenden Variablen t und n für l = [7, 3, 6, 4, 3, 1] und a = 3 auf. Was leistet das Programmstück S?

Teilaufgabe 2:

Erklären Sie informell, warum das Prädikat

$$P := \exists r: \text{conc}(r, t) = l$$

eine Invariante der Wiederholungsanweisung ist.

Teilaufgabe 3:

In S sollen möglichst starke Zusicherungen (in Form von Prädikaten) als Kommentare ergänzt werden.

Geben Sie die fehlenden Prädikate, in S durch (*) gekennzeichnet, an. Verwenden Sie dabei die Schreibweise #_u(s), um Anzahl der Vorkommnisse eines Wertes u in einer Sequenz s anzugeben.

(1)		{true}
(2)	var seq m t := l;	
(3)		{t = l}
(4)	var m x := a;	
(5)		{t = l ∧ x = a}

(6)	var nat n := 0:	
(7)		(*)
(8)		$\{\exists r: l = \text{conc}(r,t) \wedge n = \#_a(r) \wedge x = a\}$
(9)	while	
(10)	$\neg \text{isempty}(t)$	
(11)	do	
(12)		(*)
(13)	if	
(14)	$\text{first}(t) \stackrel{?}{=} x$	
(15)	then	
(16)		(*)
(17)		$\{\exists r: l = \text{conc}(r, \text{rest}(t)) \wedge n+1 = \#_a(r) \wedge x = a\}$
(18)	$n := n+1$	
(19)		(*)
(20)	else	
(21)		(*)
(22)		(*)
(23)	nop	
(24)		$\{\exists r: l = \text{conc}(r, \text{rest}(t)) \wedge n = \#_a(r) \wedge x = a\}$
(25)	fi;	
(26)		$\{\exists r: l = \text{conc}(r, \text{rest}(t)) \wedge n = \#_a(r) \wedge x = a\}$
(27)	$t := \text{rest}(t);$	
(28)		(*)
(29)	od	
(30)		$\{\text{isempty}(t) \wedge \exists r: l = \text{conc}(r, \text{rest}(t)) \wedge n = \#_a(r) \wedge x = a\}$
(31)		$\{n = \#_a(l)\}$
