Lösungen zu den Übungsaufgaben (Version 1.2)

Lösungsskizzen zu den Aufgaben aus dem Buch „**Formale Modelle der Softwareentwicklung**“ von Stephan Kleuker aus dem Friedr. Vieweg & Sohn Verlag / GWV Fachverlage GmbH, 2010 (alle Rechte vorbehalten)

Die folgenden Lösungen stellen meist jeweils eine der möglichen Antworten auf die jeweilige Aufgabenstellung dar. Eine wichtige Erkenntnis des Software-Engineerings ist, dass, wenn zwei Leute das Gleiche sagen, sie nicht unbedingt das Gleiche meinen. Dies gilt insbesondere, wenn ein Auftragnehmer die Wünsche des Kunden verstehen will. Ähnliches gilt für Aufgabenstellungen und deren Lösungen. Da Aufgabenstellungen teilweise Interpretationen erlauben, erweitert sich die Anzahl sinnvoller Lösungen noch mehr.

Bilder und Programmcode der Lösungen sind von der Web-Seite des Buches herunterladbar.

Lösungen zu Kapitel 1

zu 1) Ansatz mit Kategorien: Verteidigung, Weltraum, Medizin, Verkehr, Meteorologie, Auto, öffentliche Dienstleistungen

zu 2) Folgende Fehlerarten finden meist besondere Beachtung:

- öffentlich wirksame Fehler (Fehler, die viele Personen potenziell betreffen können, z. B. Fehler in Behörden)
- eventuell Gefahr für Leib und Leben
- hohe Kosten

Häufig treten Fehler bei Einmalentwicklungen ohne oder nur mit geringer Korrekturmöglichkeit auf, z. B. in der Raumfahrt.

Bei Projekten in den genannten Bereichen hat die Qualitätssicherung eine besondere Bedeutung.

zu 3)

- Raketensteuerung, Triebwerke automatisch an/abschalten
- Identifizierung Freund/Feind beim Militär

Typisch bei schwierig zu automatisierenden Systemen, bei denen überhaupt über Sinn einer Automatisierung nachgedacht werden muss und jede automatisierte Regel auf alle Eventualfälle geprüft werden muss.

zu 4)

Wie in 2) erwähnt, Gefahr für Leib und Leben (Ausfall von Maschinen, falsche auomatisierte Prozesse z. B. bei der Flugzeuglandung oder dem automatischen Abschluss von Feinden), hohe Kosten (Verlust einer Rakete, Entwicklungskosten, die enorm ansteigen), Imageverlust (für Firmen, die Projekte nicht fertigstellen, für die Politik, wenn Verwaltungssysteme unsauber umgesetzt werden).

Lösungen zu Kapitel 2

zu 1)

a) Es sollten von Hand Wege zu x=2, x=3 und x=4 gefunden werden.

b) Z. B. durch interaktive Simulation der folgenden Spezifikation

```
byte x=254;

active proctype Grenzen(){
  do
  :: x=x+1
  :: x=x-1
  od
}
```

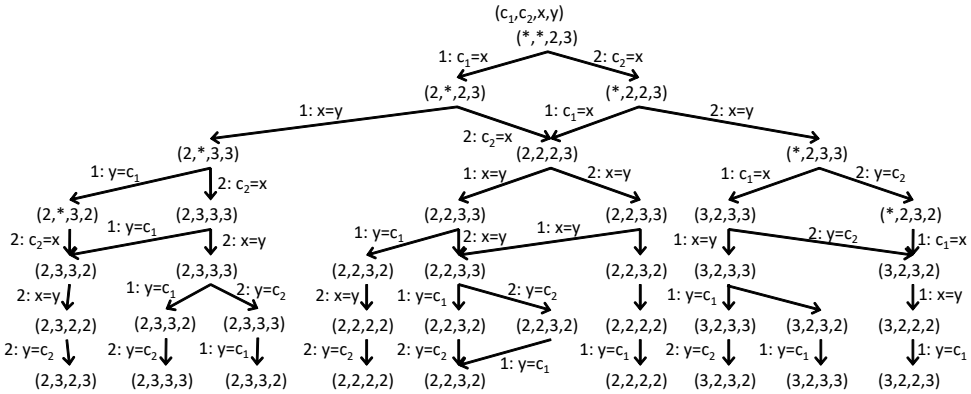
Es wird „im Kreis“ gerechnet, nach 255 folgt 0, vor 0 ist der Wert 255. Im Ausgabefenster wird man bei der Simulation darauf hingewiesen.

```
selected: 1
selected: 1
spin: line   5 "pan_in", Error: value (256->0 (8)) truncated in assignment
selected: 2
spin: line   6 "pan_in", Error: value (-1->255 (8)) truncated in assignment
```

c)

```
byte x=0;

active proctype Zwischen0und100(){
  do
  :: x<99 -> x=x+1
  :: x%10 !=0 -> break
  od
}
```

zu 2)



zu 3)

Variante mit atomic:

```
#define PROZESSE 5
bool drucken[PROZESSE];
bool frei=true;  /* kann Drucker genutzt werden */
```

```
proctype P(byte id){
  do
  ::atomic{
       frei;
       frei=false;
       drucken[id]=true;
       drucken[id]=false;
       frei=true;
    }
  od;
}

init{
  atomic{
    byte i=0;
    do
    :: i==PROZESSE-> break;
    :: i<PROZESSE ->
       run P(i);
       i=i+1;
    od;
   }
}
```

Variante ohne atomic mit Prüfprozess:

```
#define PROZESSE 5
bool drucken[PROZESSE];
bool frei=true;  /* kann Drucker genutzt werden */
bool bearbeitet =false; /* ist letzte Anmeldung bearbeitet worden */
bool fertig=false; /* Drucken abgeschlossen? */
int anmelden=-1; /* wer möchte drucken */
int dran=-1;  /* wer darf drucken */

proctype P(byte id){
  do
  :: frei ->
       anmelden=id;
       if
       :: bearbeitet && dran==id->
          drucken[id]=true;
          drucken[id]=false;
          fertig=true;
       :: bearbeitet && dran!=id  -> skip;
       fi;
  od;
}

active proctype Drucker(){
  do
  :: anmelden!=-1 ->
     frei=false;
     fertig=false;
     dran=anmelden; /* letzter Anmelder gewinnt */
     bearbeitet=true;
     fertig==true;  /* Überprüfung ! */
     bearbeitet=false;
     anmelden=-1;
     frei=true;
  od;
}

init{
```

```
  atomic{
    byte i=0;
    do
    :: i==PROZESSE-> break;
    :: i<PROZESSE ->
         run P(i);
         i=i+1;
    od;
   }
}

byte aktiv;

active proctype Pruefung(){
  atomic{
      byte i=0;
      do
       :: i<PROZESSE ->
          aktiv=aktiv+drucken[i];
          i=i+1;
       :: else -> break;
      od;
      assert(aktiv<=1);
  }
}
```

zu 4) (nur Variante für N)

```
#define N 10
byte wert[N];
byte dran=N;

proctype P(byte id){
  do
  :: dran==id && wert[id]<10 ->
     wert[id]=wert[id]+1;
     dran=(dran+1)%N;
  :: dran==id && wert[id]==10 -> break;
  od;
}

init{
  byte i=0;
  do
  :: i<N ->
     run P(i);
     i=i+1;
  :: else -> break;
  od;
  dran=0;
}
```

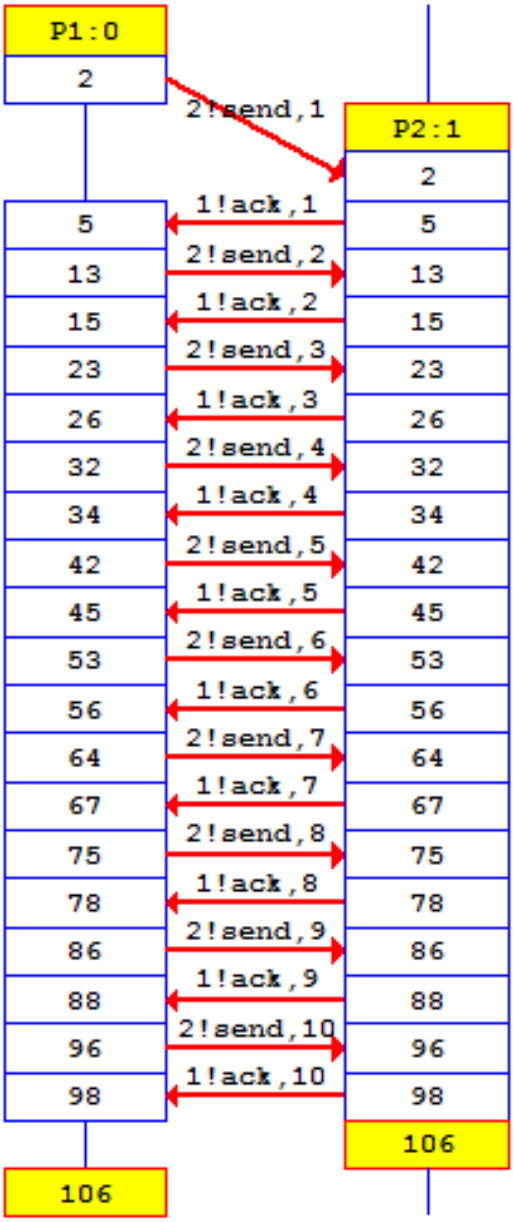
zu 5)

active proctype Q(){

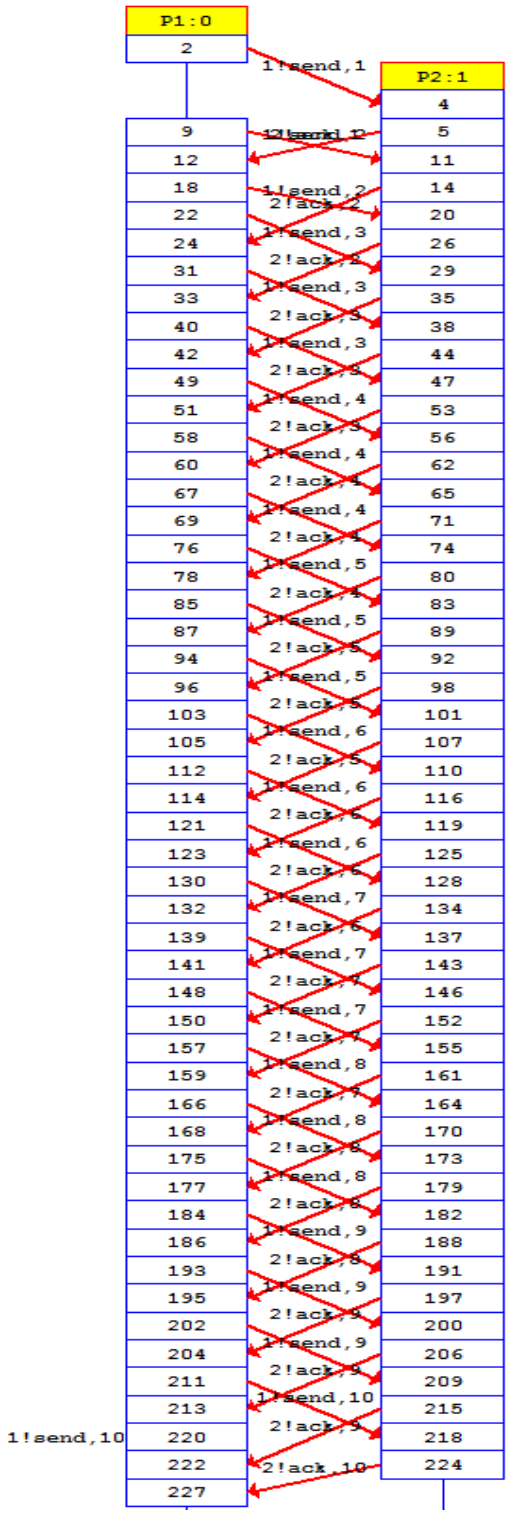
```
 do
  :: x==2 -> break;
  :: x==1 -> x=x+2;
 od;
}
```

zu 6)

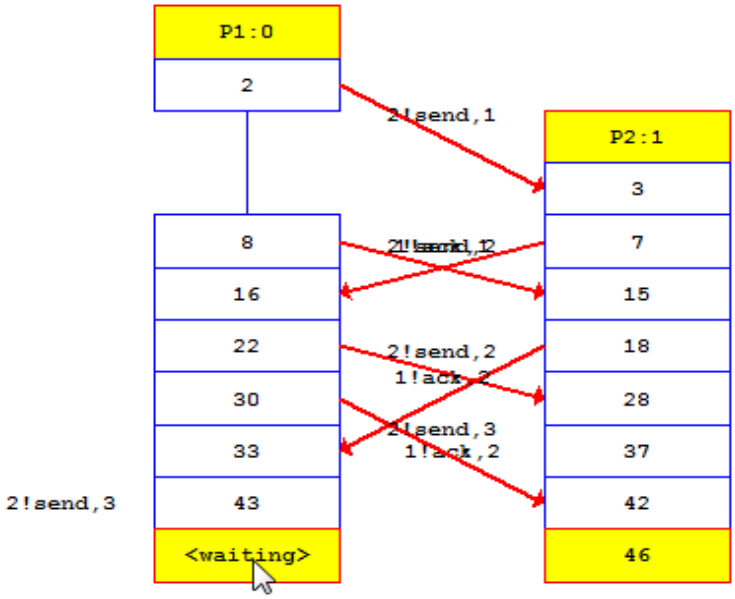
a)


P1:0
2
2!send,1
P2:1
2
5
1!ack,1
5
13
2!send,2
13
15
1!ack,2
15
23
2!send,3
23
26
1!ack,3
26
32
2!send,4
32
34
1!ack,4
34
42
2!send,5
42
45
1!ack,5
45
53
2!send,6
53
56
1!ack,6
56
64
2!send,7
64
67
1!ack,7
67
75
2!send,8
75
78
1!ack,8
78
86
2!send,9
86
88
1!ack,9
88
96
2!send,10
96
98
1!ack,10
98
106
106

b)


P1:0
2
9
12
18
22
24
31
33
40
42
49
51
58
60
67
69
76
78
85
87
94
96
103
105
112
114
121
123
130
132
139
141
148
150
157
159
166
168
175
177
184
186
193
195
202
204
211
213
1!send,10
220
222
227
1!send,1
P2:1
4
5
11
14
20
26
29
35
38
44
47
53
56
62
65
71
74
80
83
89
92
98
101
107
110
116
119
125
128
134
137
143
146
152
155
161
164
170
173
179
182
188
191
197
200
206
209
215
218
224
2!ack,10

c)



zu 7)

```
#define PUFFER 0
#define MAX 10

chan tom1 = [PUFFER] of {byte};
chan tom2 = [PUFFER] of {byte};
chan fromm1 = [PUFFER] of {byte};
chan fromm2 = [PUFFER] of {byte};

active proctype Sender(){
  byte wert=1;
  do
  :: wert<=MAX ->
     if
     :: tom1!wert;
     :: tom2!wert;
     fi;
     wert=wert+1;
  :: wert>MAX ->
     break;
  od;
}

active proctype Empfaenger(){
  byte wert;
  do
  :: fromm1?wert->
     if
     :: wert>MAX ->
        break;
     :: else ->
        skip;
     fi;
```

```
  :: fromm2?wert->
     if
     :: wert>MAX ->
        break;
     :: else ->
        skip;
     fi;
  od;
}

active proctype Medium1(){
  byte wert;
  byte summe=0;
  do
  :: tom1?wert ->
     summe=summe+wert;
     fromm1!wert;
     if
     :: wert>MAX ->
         break;
     :: else ->
          skip;
     fi;
  od;
}

active proctype Medium2(){
  byte wert;
  byte summe=0;
  do
  :: tom2?wert ->
     summe=summe+wert;
     fromm2!wert;
     if
     :: wert>MAX ->
        break;
     :: else ->
        skip;
     fi;
  od;
}
```

zu 8) Lösung zu f), funktioniert mit beliebiger Puffergröße

```
#define MIN 5
#define MAX 10
#define PUFFER 0
mtype={send,term,ack,termack,repeat};
byte summe1=0;
byte summe2=0;
byte summe3=0;
chan c12=[PUFFER] of {mtype,byte};
chan c23=[PUFFER] of {mtype,byte};
chan c21=[PUFFER] of {mtype,byte};
chan c32=[PUFFER] of {mtype,byte};

chan c1f=[PUFFER] of {mtype,byte};
chan c2f=[PUFFER] of {mtype,byte};
chan c3f=[PUFFER] of {mtype,byte};
```

```
active proctype P1(){
  byte wert=1;
  do
  :: wert>MIN ->
     c12!term,0;
     do
     :: c21?termack,_;->
        break;
     :: c21?repeat,_ ->
        c12!term,0;
     od;
     break;
  :: wert<=MAX ->
     summe1=summe1+wert;
     c12!send,wert;
     do
     :: c21?ack,eval(wert) ->
        wert=wert+1;
        break;
     :: c21?repeat,_ ->
        c12!send,wert;
     od;
  od;
  c1f!send,summe1;
}

active proctype P2(){
  byte wert;
  do
  :: c12?send,wert ->
     c21!repeat,0;
  :: c12?send,wert ->
     summe2=summe2+wert;
     c21!ack,wert;
     c23!send,wert;
     do
     :: c32?ack,eval(wert) ->
        break;
     :: c32?repeat,_ ->
        c23!send,wert;
     od;
  :: c12?term,_ ->
     c21!repeat,0;
  :: c12?term,_ ->
     c21!termack,0;
     c23!term,0;
     do
     :: c32?termack,_ ->
        break;
     :: c32?repeat,_ ->
        c23!term,0;
     od;
     break;
  od;
  c2f!send,summe2;
}

active proctype P3(){
  byte wert;
  do
  :: c23?send,wert ->
     c32!repeat,0;
  :: c23?send,wert ->
     summe3=summe3+wert;
```

```
      c32!ack,wert;
  :: c23?term,_ ->
      c32!repeat,0;
  :: c23?term,_ ->
      c32!termack,0;
      break;
  od;
  c3f!send,summe3;
}

active proctype Invariante(){
  assert(summe1>=summe2 && summe2>=summe3);
}

active proctype Final(){
  byte s1;
  byte s2;
  byte s3;
  byte erhalten=0;
  do
  :: erhalten==3 ->
      break;
  :: c1f?send,s1->
      erhalten=erhalten+1;
  :: c2f?send,s2->
      erhalten=erhalten+1;
  :: c3f?send,s3->
      erhalten=erhalten+1;
  od;
  assert(s1==s2 && s2==s3);
}
```

zu 9)

```
chan send=[0] of {byte};
chan ack=[0] of {bit};
chan repeat=[0] of {bit};

active proctype P1(){
  byte wert=0;
  byte wdh=0;
  do
  :: send!wert;
     if
     :: ack?_ ->
        progress1:  wert=wert+1;
     :: repeat?_ ->
        /* skip;       Lösung für b) */
        if
        :: wdh<3 ->
           wdh=wdh+1;
        :: else ->
           wdh=0;
           progress2: wert=wert+1;
        fi;
     fi;
  od;
}

active proctype P2(){
  do
  :: send?_ ->
     if
     :: ack!0;
```

```
     :: repeat!0;
     fi;
  od;
}

/* d)
trace{
  do
   :: send?_ ->
      if
      :: ack?_;
      :: repeat?_;
      fi;
  od;
}
*/

/* e)
trace{
  do
  :: ack?_;
  :: repeat?_;
     if
     ::ack?_;
     ::repeat?_;
       if
       ::ack?_;
       ::repeat?_;
         ack?_;
       fi;
     fi;
  od;
}
*/
```

zu 10)

Da in der Aufgabenstellung die Prozesse ab Eins nummeriert sind, wird in der folgenden Spezifikation ein Array mit N+1-Elementen betrachtet, wobei das nullte Feld ignoriert wird. Eine Anpassung mit einem nullten Prozess mit PROZESSE=FELDGROESSE ist leicht möglich.

```
#define sortiert (i>PROZESSE && wert[1]<=wert[2] && wert[2]<=wert[3]
                  && wert[3]<=wert[4] && wert[4]<=wert[5])
/*  <> sortiert  */

/* Folgende drei Werte hängen zusammen, jeweils +/- 1 */
#define PROZESSE 5
#define FELDGROESSE 5
#define KANALZAHL  4
#define MAX 3
byte wert[FELDGROESSE];
byte anfang[FELDGROESSE];
chan cij[KANALZAHL]= [0] of {byte};
chan cji[KANALZAHL]=[0] of {byte};
byte i=1;

proctype P(byte id){
  byte tmp;
  byte zaehl=1;
  do
  :: zaehl<=PROZESSE->
     if
     :: id%2==0 ->
```

```
            cji[id-2]!wert[id];
            cij[id-2]?wert[id];
         :: id%2==1 && id!=PROZESSE ->
            cji[id-1]?tmp;
            if
            :: tmp<wert[id] ->
               cij[id-1]!wert[id];
               wert[id]=tmp;
            :: else ->
               cij[id-1]!tmp;
            fi;
         :: else -> skip
         fi;
         if
         :: id%2==0  && id!=PROZESSE->
            cij[id-1]!wert[id];
            cji[id-1]?wert[id];
         :: id%2==1 && id!=1->
            cij[id-2]?tmp;
            if
            :: tmp>wert[id] ->
               cji[id-2]!wert[id];
               wert[id]=tmp;
            :: else ->
               cji[id-2]!tmp;
            fi;
         :: else -> skip;
         fi;
         zaehl=zaehl+1;
      :: else-> break;
      od;
   }

   init{
      atomic{
      byte zufall=1;
      do
      :: i<=PROZESSE ->
         do
         :: true -> break;
         :: zufall<MAX ->
            zufall=zufall+1;
         od;
         wert[i]=zufall;
         anfang[i]=zufall;
         run P(i);
         zufall=1;
         i=i+1;
      :: else ->break;
      od;
      }
   }
```

zu 11)

```
#define xungerade (x%2==1)
#define xist10 (x==10)
#define ykleinergleichx (y<=x)
#define yungleichx (y!=x)
#define ygleichx (y==x)
```

a) [] xungerade (f)

b) <> xist10 (f)

c) <> !(xist10) (w)

d) <> [] !xungerade (f)

e) [] <> xungerade (wahr nur für schwache fairness)

f) <>[] xist10 (f)

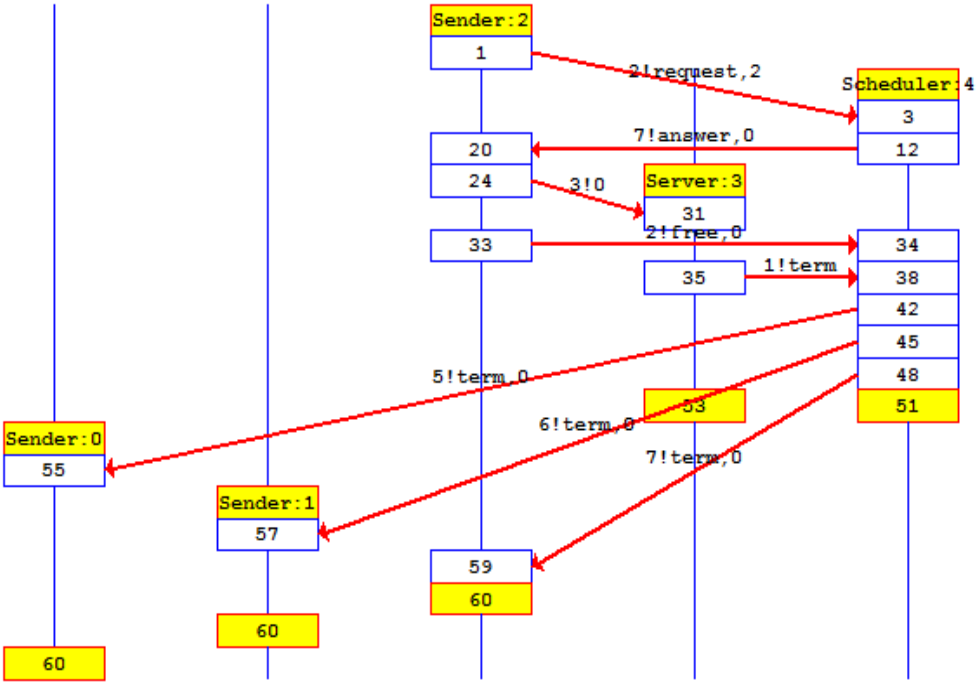
g) [] ykleinergleichx (f)

h) [] (yungleichx -> <> ygleichx) (f)

i) [] (ygleichx -> <> yungleichx) (w)

zu 12)

a) und b) zusammen



c) Es gibt verschiedene Ansätze, am einfachsten wird der Scheduler so realisiert, dass er folgende Fälle getrennt abarbeitet:

do

:: Anfrage liegt vor → Anfrage puffern

:: Freigabe liegt vor → freigegebenen Kanal merken (puffern)

:: Anfrage im Puffer und freier Kanal im Puffer → Kanalnummer an Anfrager senden

:: Terminierung vom Server → Terminierungsprozess einleiten

od

Man kann für Anfragen und Kanäle Puffer einrichten. Trickreich ist der Ansatz, für die Puffer lokale Kanäle zu nutzen, so dass man die Puffer nicht selbst spezifizieren muss. Die folgende Spezifikation nutzt die Idee nicht, freie Kanäle und Anfragen werden in Arrays verwaltet.

```
#define N 3
#define M 2
#define PUFFER 1
mtype={send,term,request,answer,free};
chan toServer[M] = [PUFFER] of {bool};
chan stop =[PUFFER] of {mtype};
chan toScheduler=[PUFFER] of {mtype,byte};
chan toSender[N]=[PUFFER] of {mtype,byte};
bool requested[N]=false; /* für Verifikation eingeführt */
bool answered[N]=false; /* für Verifikation eingeführt */
bool sent[N]=false; /* für Verifikation eingeführt */
bool terminated=false;

active [N] proctype Sender(){
  unsigned kanal:3;
  do
  ::  toScheduler!request,_pid ->
      requested[_pid]=true;
      if
      :: toSender[_pid]?answer,kanal; ->
         d_step{
           answered[_pid]=true;
           requested[_pid]=false;
         }
      :: toSender[_pid]?term,_ -> break;
      fi;
      answered[_pid]=false;;
      if
      :: toServer[kanal]!0->
         sent[_pid]=true;
      :: toSender[_pid]?term,_ -> break;
      fi;
      progress: if
      :: toScheduler!free,kanal ->
         sent[_pid]=false;
      :: toSender[_pid]?term,_ -> break;
      fi;
  ::  toSender[_pid]?term,_ ->
      break;
  od;
}

active proctype Server(){
  xs stop;
  unsigned zaehler:3=0;
  progress: do
  :: toServer[zaehler]?[_] ->
     toServer[zaehler]?_;
     zaehler=(zaehler+1)%M;
  :: !(toServer[zaehler]?[send] ) ->  zaehler=(zaehler+1)%M;
  :: stop!term; ->
     terminated=true;
     break
  od;
}
```

```
active proctype Scheduler(){
  xr toScheduler;
  xr stop;
  unsigned belegtAnzahl:3=0;
  bool belegt[M]=false; /*Ist Kanal belegt?*/
  byte anfragen[N];
  unsigned  frei:3=0;
  unsigned  anfrager:3; /*wer will was bzw. welcher Kanal frei*/
  unsigned  tmp:3; /*macht in Schleifen seine Dienste */
  do
    :: stop?term -> break;
    :: toScheduler?request,anfrager ->
       d_step{
         anfragen[frei]=anfrager;
         frei++;
       }
    :: toScheduler?free,anfrager ->
       d_step{
         belegt[anfrager]=false;
         belegtAnzahl=belegtAnzahl-1;
       }
    :: frei>0 && belegtAnzahl<M ->
        /* suche freien Kanal */
       d_step{
         tmp=0;
         do
         :: belegt[tmp] -> tmp=tmp+1;
         :: ! belegt[tmp] -> break;
         od;
         belegt[tmp]=true;
         belegtAnzahl++;
       }
       toSender[anfragen[0]]!answer,tmp;
       /*reorganisieren der Queue*/
       d_step{
         tmp=0;
         do
         :: tmp<frei  && tmp<N-1 ->
            anfragen[tmp]=anfragen[tmp+1];
            tmp++;
         :: else -> break;
         od;
         frei--;
       }
  od;
  tmp=0;
  do
  :: tmp<N ->
     toSender[tmp]!term,0;
     tmp++;
  :: else-> break;
  od;
}
```

d) Zur Verifikation werden einige Boolesche Hilfsvariablen eingeführt. Die Anforderung iii wird mit progress-Marken gezeigt, iv mit der Prüfung auf invalid endstates und v kann man auf iv zurückführen; könnte aber auch mit einer LTL-Formel gezeigt werden.

Wichtig ist der sorgfältige Einsatz der Booleschen Flags, da es sonst schnell möglich wird, eine Eigenschaft zu prüfen, die das System nicht erfüllt.

Anforderung i wird wie folgt übersetzt, dabei wird nur ein Prozess betrachtet:

```
#define frage requested[0]
#define antwort answered[0]
#define ende terminated
[] ((frage) -> (<> (antwort || ende)))
```

Anforderung ii kann ähnlich wie i behandelt werden, wenn man die Deadlock- und Livelock-Freiheit berücksichtigt.

```
#define gesendet (sent[0] || sent[1] || sent[2])
[] ((!ende) -> (<> (gesendet || ende)))
```

e) Die Spezifikation läuft synchron und asynchron, am Ende können bei asynchroner Kommunikation noch Nachrichten in den Puffern vorliegen.

Lösungen zu Kapitel 3

zu 1)

a) Der Zustand s2 muss nach maximal 25 Zeiteinheiten (ZE) verlassen werden. Bis 10 ZE können die obere und untere Kante genutzt werden, insofern die Bedingung bzgl. i erfüllt ist. Im Intervall (10ZE,15ZE] kann nur die untere Kante, im Intervall (15ZE,17ZE) keine Kante genutzt werden. Ab 17 ZE steht die mittlere Kante zur Verfügung, insofern i>=3 gilt.

b) Es gibt folgende mögliche Pfade nach s3:

i) s1, obere Kante nach s2, untere Kante nach s3; dann ist x in [0,15) und eine Schleife ist in diesem Zeitintervall mit i), ii), iv) möglich.

ii) s1, mittlere Kante nach s2, untere Kante nach s3; dann ist x in [10,15) und eine Schleife ist in diesem Zeitintervall mit i), ii), iv) möglich.

iii) s1, mittlere Kante nach s2, mittlere Kante nach s3; dann ist x in [17,25) und eine Schleife ist in diesem Zeitintervall mit iii), v), vi) möglich.

iv) s1, untere Kante nach s2, untere Kante nach s3; dann ist x in [5,15) und eine Schleife ist in diesem Zeitintervall mit i) , ii), iv) möglich.

v) s1, untere Kante nach s2, mittlere Kante nach s3; dann ist x in [17,25) und eine Schleife ist in diesem Zeitintervall mit iii) , v), vi) möglich.

vi) s1, untere Kante nach s2, obere Kante nach s3; dann ist x in [5,10) und eine Schleife ist in diesem Zeitintervall mit iii) , v), vi) möglich.

c) Da man beliebig lang in s3 bleiben kann, muss die ausgehende Kante nicht genutzt werden. Im Intervall [20,40) ist zwar die Kantenbedingung erfüllt, aber nicht die Zustandsinvariante von s1.

zu 2)

a) Der Zustand s1 muss nach weniger als 20 ZE verlassen werden, ab 2 ZE ist die mittlere Transition, ab 6 ZE sind beide Transitionen nutzbar.

b) Da der Zustand s2 nicht verlassen werden muss, besteht dazu ab 8 ZE auch keine Möglichkeit mehr.

c) In s3 befindet sich x im Intervall (2,20).

d) A[] P.x<30

e) E[] !P.s2

zu 3)

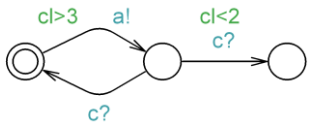
a) Globale Variablen

```
chan a;
chan b;
chan c;
clock cl; /* Loeschen um Deadlock zu Erzeugen */
```

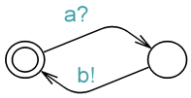
Gesamtspezifikation:

```
system A,B,C;
```

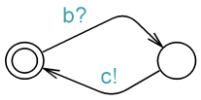
A:



B:



C:



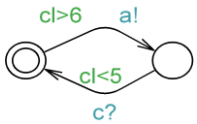
b) Globale Variablen:

```
chan a;
chan b;
chan c;
clock cl; /* Loeschen zum Deadlock entfernen */
```

Gesamtspezifikation:

```
system A,B,C;
```

A:



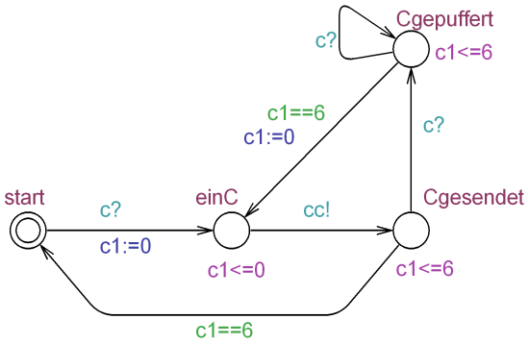
B und C wie in a).

zu 4)

```
system Empfaenger, Erzeuger, Filter;
```

Filter mit lokaler Variable:

```
clock c1;
```



zu 5)

Die Initialisierung und die Prüfung können aus dem Buch übernommen werden.

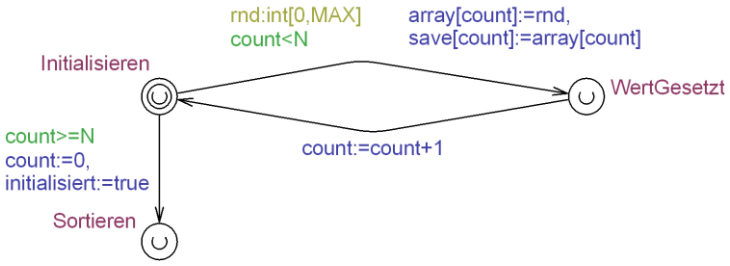
Globale Variablen:

```
const int N=5;
const int MAX=4;

int[0,MAX] array[N];
int[0,MAX] save[N];

int[0,N] count:=0;
int[0,MAX+1] count2:=0;
int[0,N] j:=0;
int[0,MAX] tmp;
int[0,N] anzahl1;
int[0,N] anzahl2;

chan up[N];
chan down[N];

bool initialisiert=false;
int sortiert=0;
```

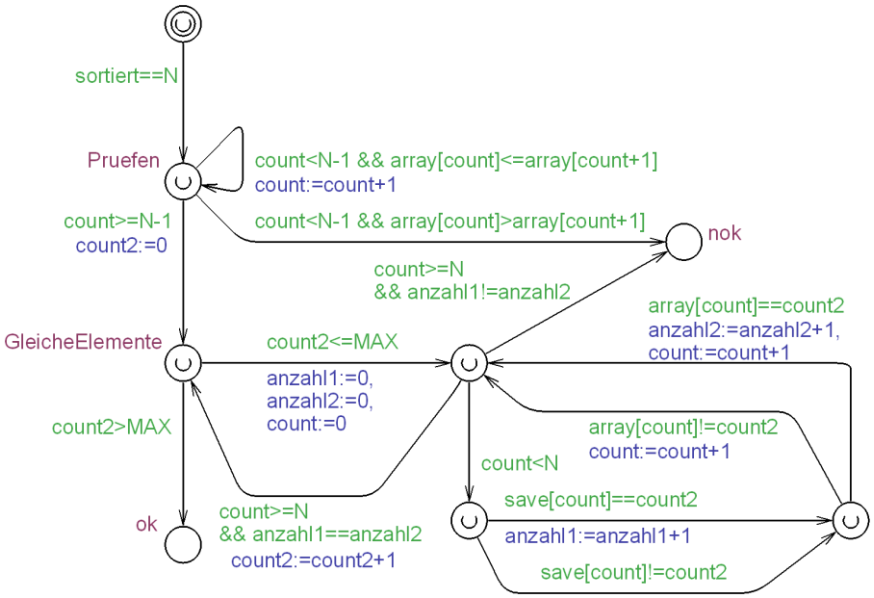
Gesamtspezifikation:

```
S0=Sort(0);
S1=Sort(1);
S2=Sort(2);
S3=Sort(3);
S4=Sort(4);  /* für N=5 */
// List one or more processes to be composed into a system.
system Initialisierer,S0,S1,S2,S3,S4,Pruefer;
```

Initialisierer:



Pruefer:

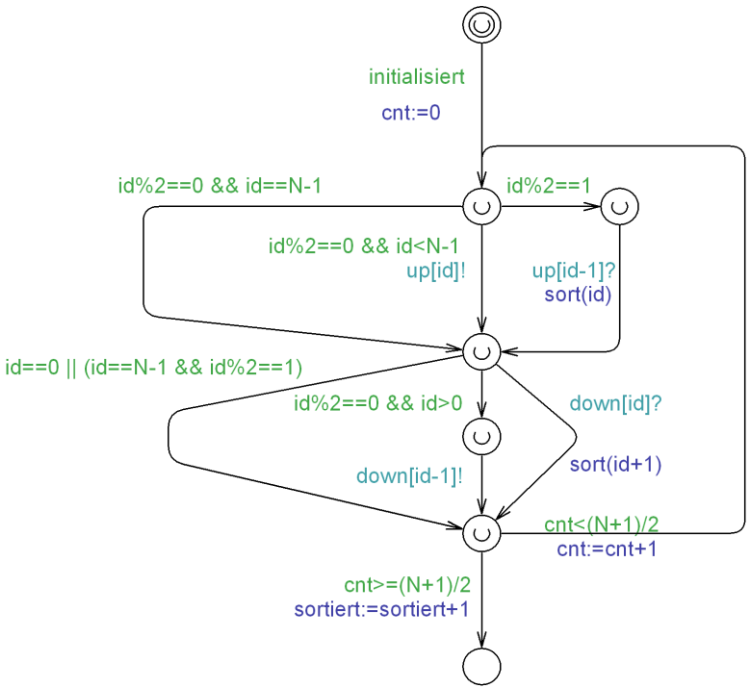


Sort: mit Parameter

```
int id
```

mit lokaler Variablen und lokaler Funktion:

```
int cnt; //lokale Zählvariable
void sort(int id){
  if(array[id]<array[id-1]){
     int tmp=array[id-1];
     array[id-1]=array[id];
     array[id]=tmp;
  }
}
```



zu 6)

Dieser Fahrstuhl speichert Anfragen in einer Queue, er kann damit keine kürzesten Wege und Zwischenhalte berücksichtigen. Dies kann in die Steuerung eingebaut werden.

Globale Variablen:

```
const int STOCKWERKE 4;
chan knopf[STOCKWERKE];
chan tuer[STOCKWERKE];
chan rauf;
chan runter;
chan bereit;
// Nachbildung einer Queue (Schlange)
int[0,STOCKWERKE] anforderung[STOCKWERKE];
int[-1,STOCKWERKE] naechster=-1;
// Fahrstuhl startet im 0.ten Stock, -1 für fahrenden Fahrstuhl
int[-1,STOCKWERKE] zuletzt=0;
```

Gesamtspezifikation

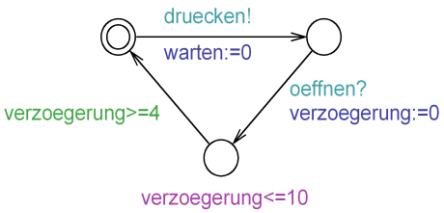
```
Stock0 := Stock(knopf[0],tuer[0]);
Stock1 := Stock(knopf[1],tuer[1]);
Stock2 := Stock(knopf[2],tuer[2]);
Stock3 := Stock(knopf[3],tuer[3]);

system
  Stock0, Stock1, Stock2, Stock3, Fahrstuhl, Knopfverwaltung, Steuerung;
```

Stock mit Parameter:

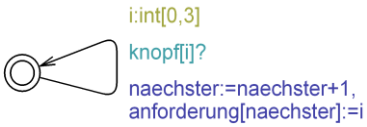
```
 chan &druecken, chan &oeffnen
```

mit lokalen Variablen:

```
clock warten;
clock verzoegerung;
```

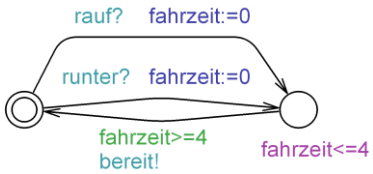


Knopfverwaltung



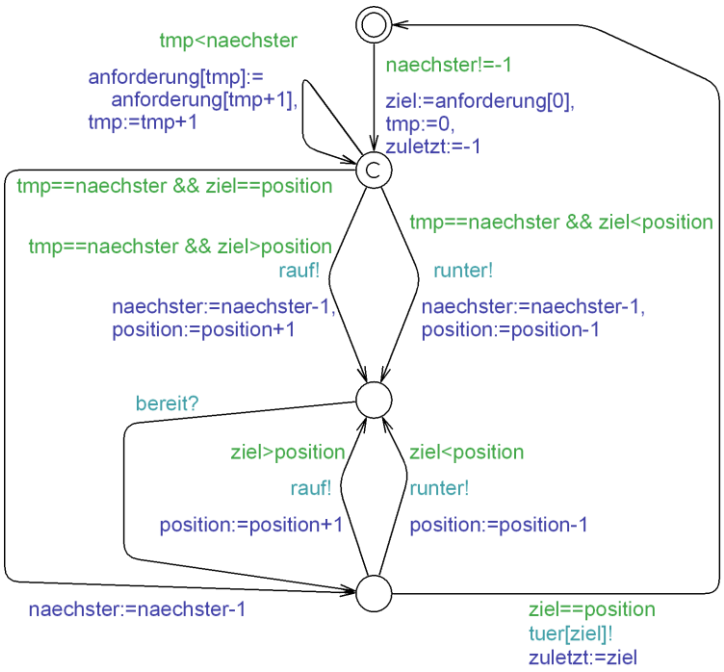
Fahrstuhl mit lokaler Variable:

```
clock fahrzeit;
```

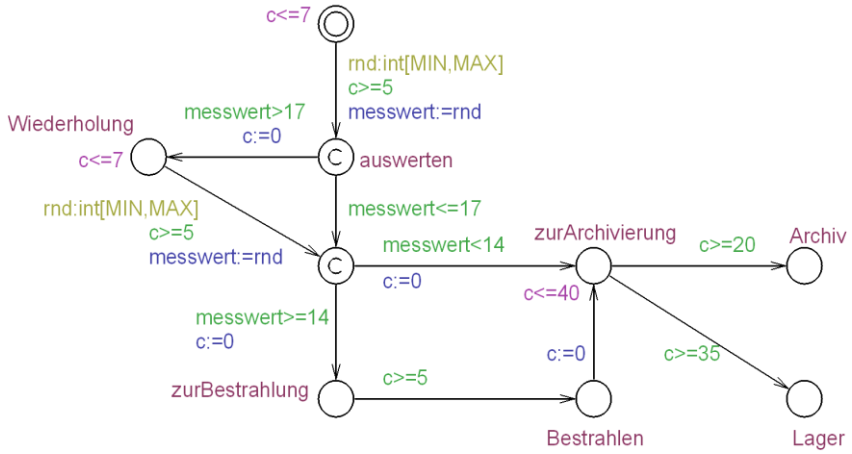


Steuerung mit lokalen Variablen:

```
int[0,STOCKWERKE] position:=0;
int[0,STOCKWERKE] ziel;
// zur Queuesteuerung
int[0,STOCKWERKE] tmp;
```



zu 7)



zu 8)

a) A[] P.x<20, nicht erfüllt, da beliebig lang in s6

b) A[] (!P.s1 imply P.i%2!=0), erfüllt

c) A[] (!(P.s1 || P.s2 || P.s3) imply P.j%2!=0) , nicht erfüllt über s1, s2, s4

d) A[] P.j<7, nicht erfüllt

e) E[] P.j<7, erfüllt

f) A<> P.j<7, erfüllt

g) A<> P.i<P.j, nicht erfüllt

h) E[] P.j<=P.i, erfüllt

i) E<> P.j>P.i, erfüllt

j) P.s5 --> P.j==P.i, erfüllt

k) A[] !deadlock, nicht erfüllt, da Deadlock in s6 möglich

zu 9)

Die Spezifikation zeigt eine Möglichkeit, recht flexibel eine Queue zu spezifizieren, die für Anfragen und freie Kanäle jeweils genutzt wird.

Globale Variablen:

```
const int N:=3;
const int M:=2;

// zur Speicherung der Anfragen und der freien Kanale werden Queues genutzt
bool kanalvorhanden:=false;
bool anfragevorhanden:=false;
chan pushanfrage;
chan pushkanal;
chan getanfrage;
chan getkanal;
int inprozessnummer; // diesen Wert in zugehörige Queue
int inkanalnummer:=-1; // diesen Wert in zugehörige Queue
int outprozessnummer; // diesen Wert in zugehörige Queue
int outkanalnummer; // diesen Wert in zugehörige Queue

chan request[N];
urgent chan free[N];
chan answer[N];
chan term[N];
chan terminate; // vom Server zum Scheduler
chan toS[M];
int[0,M-1] channel[N]; // welcher Kanal wurde i-tem Sender zugeordnet
```

Gesamtspezifikation

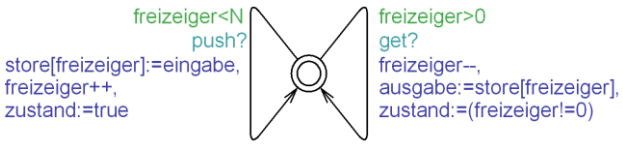
```
Q1:=Queue(inprozessnummer,outprozessnummer,anfragevorhanden,pushanfrage,get
anfrage);
Q2:=Queue(inkanalnummer,outkanalnummer,kanalvorhanden,pushkanal,getkanal);
Send0 = Sender(0);
Send1 = Sender(1);
Send2 = Sender(N-1);
system Send0,Send1,Send2,Scheduler,Q1,Q2,Server;
```

Queue mit Parametern:

```
int &eingabe, int &ausgabe, bool &zustand, chan &push, chan &get
```

mit lokale Variablen:

```
int store[N];
int [0,N] freizeiger;
```

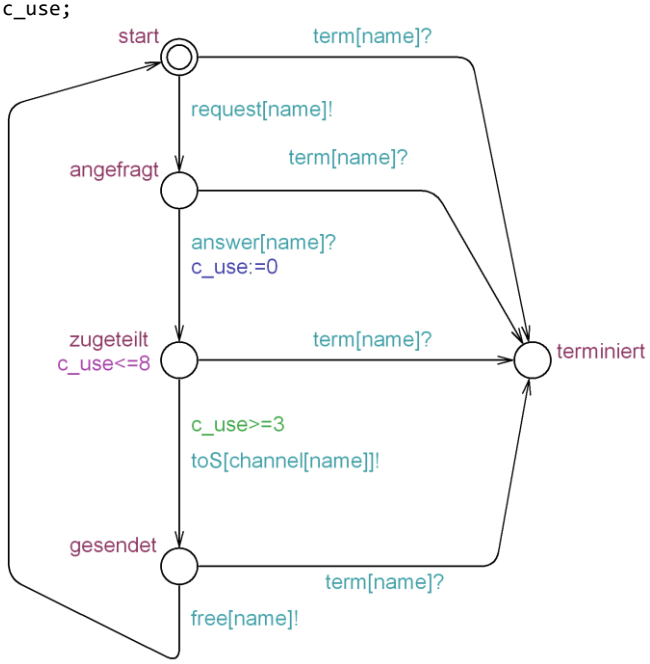


Sender mit Parameter:

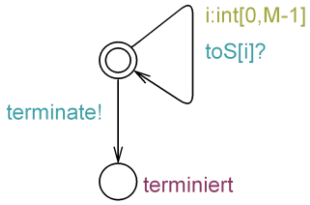
```
int[0,N-1] name
```

mit lokaler Variable:

```
clock c_use;
```

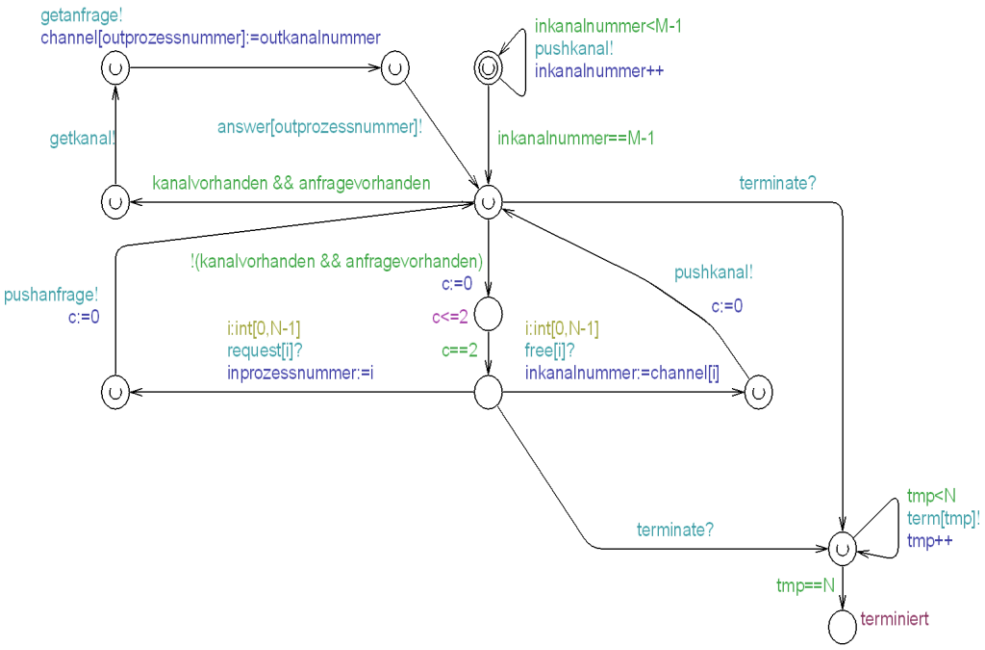


Server:



Scheduler mit lokalen Variablen:

```
clock c;
int[0,N] tmp;
```



Bei der Verifikation ist die Deadlockfreiheit leicht zu zeigen.

A[] !deadlock || (Server.terminiert && Scheduler.terminiert
&& Send0.terminiert && Send1.terminiert && Send2.terminiert)

Bei den restlichen Anforderungen ist das Problem, dass das System nicht voran schreiten muss. Werden Zustände als „Urgent“ gekennzeichnet, entstehen zeitabhängige Deadlocks. Aus diesem Grund wurde der interne Schritt der Prüfung, ob eine Kanalzuteilung möglich ist, von den restlichen Möglichkeiten getrennt, so dass das System voranschreiten muss.

Anforderung i kann wie folgt umgesetzt werden:

Send0.angefragt --> Send0.zugeteilt || Server.terminiert

Die Anfrage ist wie die Folgenden für jeden Sender zu formulieren. Bei genauerer Betrachtung erlaubt die Anforderung noch, dass auf mehrere Anfragen nur eine Zuteilung erfolgt. Dies könnte durch eine Zählvariable geprüft werden, die feststellt, dass es maximal eine Zuteilung weniger als Anfragen gibt.

Anforderung ii kann wie folgt umgesetzt werden, dabei wird kein Sender zu request gezwungen, die Kanäle free allerdings als `urgent` gekennzeichnet, damit die zugehörige Aktion auch ausgeführt wird.

```
urgent chan free[N];
```

(!Server.terminiert && !Send0.start) -->
(Server.terminiert || Send0.gesendet || Send1.gesendet || Send2.gesendet)

Anforderung iii kann geprüft werden, indem sichergestellt ist, dass jeder Sender voranschreitet. Dazu wird für alle Nicht-Startzustände und Nicht-Endzustände überprüft, dass sie verlassen werden. Der Zustand `angefragt` wurde bereits mit i überprüft.

Send0.zugeteilt --> !Send0.zugeteilt

Beim Zustand `gesendet` ist zu beachten, dass andere Prozesse überholen können, so dass nur der Gesamtfortschritt verifizierbar ist. Es ist z. B. nicht verifizierbar, dass Folgendes gilt:

(Send0.gesendet || Send1.gesendet || Send2.gesendet) -->
!(Send0.gesendet || Send1.gesendet || Send2.gesendet)

Konkret müsste für jede Zustandskombination über alle Automaten mit Send0.gesendet gezeigt werden, dass diese Kombination verlassen wird, z. B. :

(Send0.gesendet && Send1.gesendet) --> !(Send0.gesendet && Send1.gesendet)

Anforderung v kann wie folgt umgesetzt werden:

Server.terminiert --> Send0.terminiert && Send1.terminiert
&& Send2.terminiert && Scheduler.terminiert

Lösungen zu Kapitel 4

Hinweis: Einige Ergebnisse sind mit Hilfe von Netlab entstanden.

zu 1)

a)

```
M001:   0 (  1  1  0  0) ---t1---> M002:   1 (  0  0  1  1)
M002:   1 (  0  0  1  1) ---t2---> M003:   2 (  1  0  0  1)
                          ---t3---> M004:   2 (  0  1  1  0)
M003:   2 (  1  0  0  1) ---t3---> M001:   0 (  1  1  0  0)
M004:   2 (  0  1  1  0) ---t2---> M001:   0 (  1  1  0  0)
```

b)

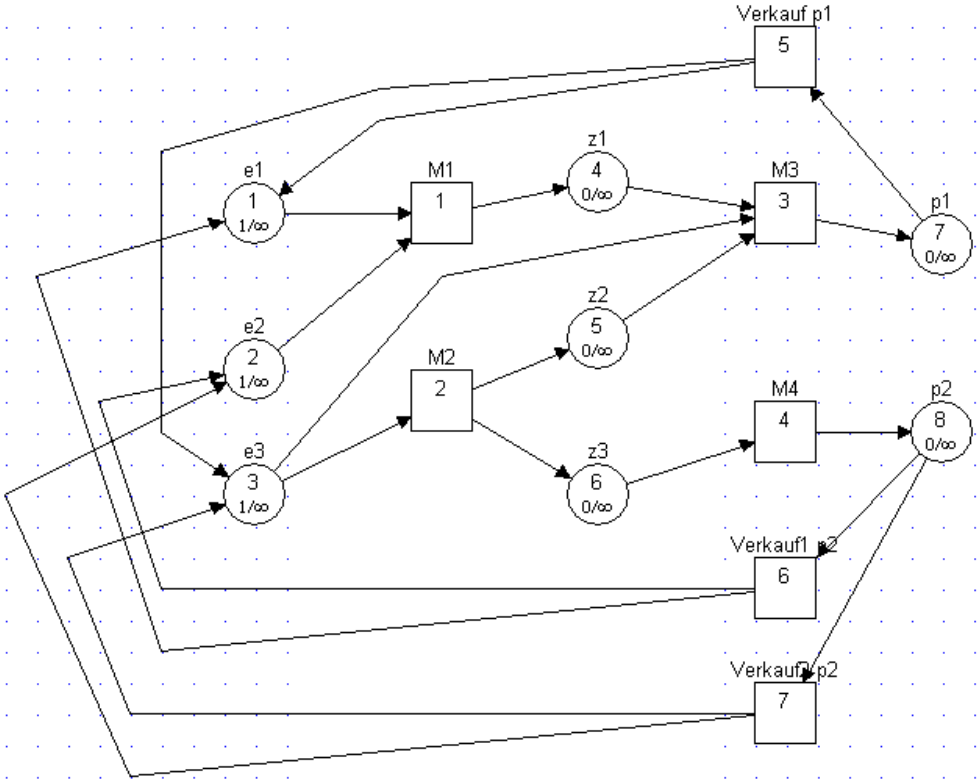
```
M001:   0 (  1  0  0) ---t1---> M002:   1 (  0  1  1)
M002:   1 (  0  1  1) ---t2---> M003:   2 (  1  0  *)
                      ---t3---> M004:   2 (  1  *  0)
M003:   2 (  1  0  *) ---t1---> M005:   3 (  0  1  *)
                      ---t3---> M006:   3 (  *  0  *)
M004:   2 (  1  *  0) ---t2---> M007:   3 (  *  *  0)
                      ---t1---> M008:   3 (  0  *  1)
M005:   3 (  0  1  *) ---t2---> M003:   2 (  1  0  *)
                      ---t3---> M009:   4 (  *  1  *)
M006:   3 (  *  0  *) ---t1---> M010:   4 (  *  *  *)
                      ---t3---> M006:   3 (  *  0  *)
M007:   3 (  *  *  0) ---t2---> M007:   3 (  *  *  0)
                      ---t1---> M010:   4 (  *  *  *)
M008:   3 (  0  *  1) ---t2---> M011:   4 (  *  *  1)
                      ---t3---> M004:   2 (  1  *  0)
M009:   4 (  *  1  *) ---t2---> M006:   3 (  *  0  *)
                      ---t1---> M010:   4 (  *  *  *)
                      ---t3---> M009:   4 (  *  1  *)
M010:   4 (  *  *  *) ---t2---> M010:   4 (  *  *  *)
                      ---t1---> M010:   4 (  *  *  *)
                      ---t3---> M010:   4 (  *  *  *)
M011:   4 (  *  *  1) ---t2---> M011:   4 (  *  *  1)
                      ---t1---> M010:   4 (  *  *  *)
                      ---t3---> M007:   3 (  *  *  0)
```

c)

```
M001:   0 (  1  1  0  0  0  0  0) ---t1---> M002:   1 (  0  1  1  1  0  0  0)
                                  ---t2---> M003:   1 (  1  0  0  0  1  0  0)
M002:   1 (  0  1  1  1  0  0  0) ---t2---> M004:   2 (  0  0  1  1  1  0  0)
M003:   1 (  1  0  0  0  1  0  0) ---t1---> M004:   2 (  0  0  1  1  1  0  0)
M004:   2 (  0  0  1  1  1  0  0) ---t4---> M005:   3 (  0  1  0  1  0  1  1)
M005:   3 (  0  1  0  1  0  1  1) ---t2---> M006:   4 (  0  0  0  1  1  1  1)
                                  ---t3---> M007:   4 (  1  1  0  0  0  0  *)
M006:   4 (  0  0  0  1  1  1  1) ---t3---> M008:   5 (  1  0  0  0  1  0  *)
M007:   4 (  1  1  0  0  0  0  *) ---t1---> M009:   5 (  0  1  1  1  0  0  *)
                                  ---t2---> M008:   5 (  1  0  0  0  1  0  *)
M008:   5 (  1  0  0  0  1  0  *) ---t1---> M010:   6 (  0  0  1  1  1  0  *)
M009:   5 (  0  1  1  1  0  0  *) ---t2---> M010:   6 (  0  0  1  1  1  0  *)
M010:   6 (  0  0  1  1  1  0  *) ---t4---> M011:   7 (  0  1  0  1  0  1  *)
M011:   7 (  0  1  0  1  0  1  *) ---t2---> M012:   8 (  0  0  0  1  1  1  *)
                                  ---t3---> M007:   4 (  1  1  0  0  0  0  *)
M012:   8 (  0  0  0  1  1  1  *) ---t3---> M008:   5 (  1  0  0  0  1  0  *)
```

zu 2)

Das folgende Netz zeigt die Lösung zu b); für a) muss nur die Kante von e3 nach m3 gelöscht werden. In a) ist das Netz deadlockfrei, die Tokenanzahl kann beliebig steigen. In b) ist ein Deadlock möglich.



zu 3)

a) Netz mit S-Invarianten überdeckt, also endlich, T-Invariante realisierbar.

```
S1:  1    0    1    1
S2:  1    0    0    0
S3:  0    1    0    0
S4:  0    1    0    0
S5:  0    0    1    1
S6:  0    0    1    0
S7:  0    0    0    1
T1:  1
T2:  1
T3:  1
T4:  1
```

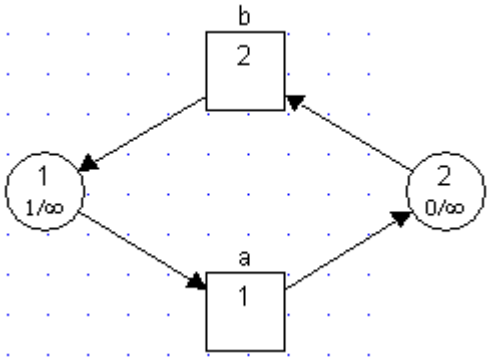
b) S-Invariante umfasst nur zwei Stellen, T-Invariante realisierbar, Realisierung zeigt, dass Stellen, die nicht in der S-Invariante sind keine Beschränkung der Tokenzahl haben.

```
S1:  1
S2:  0
S3:  0
S4:  1

T1:  3
```

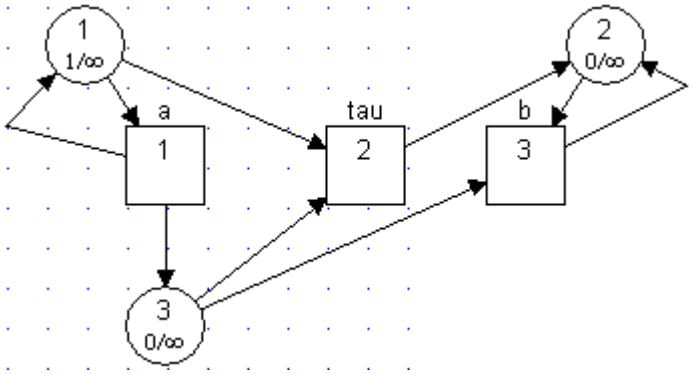
```
T2:  1
T3:  1
T4:  1
```

zu 4)

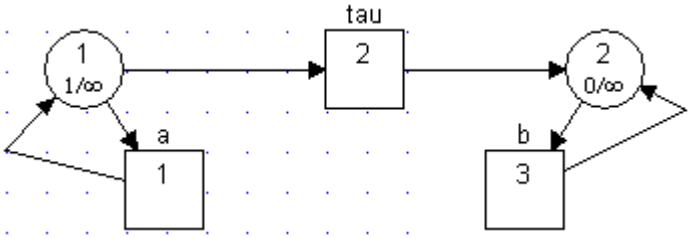
a) ε, a, ab, aba



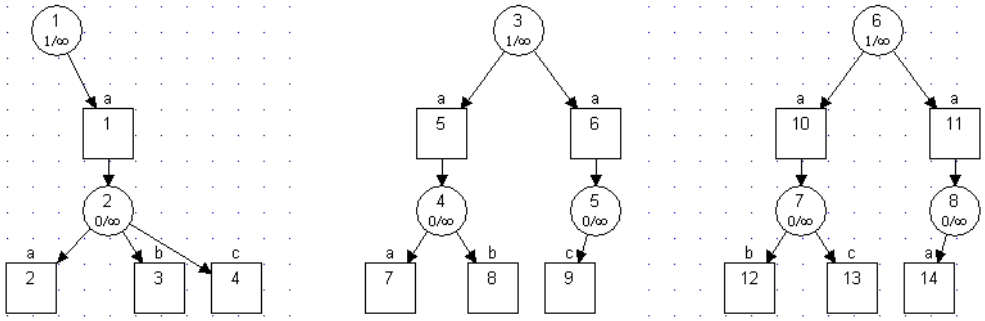
b) ε, a, aa, aaa, aab



c) ε, a, b, aa, ab, bb, aaa, aab, abb, bbb



zu 5)



links: (ε, {a}), (a, {a,b,c}), (aa, ∅), (ab, ∅), (ac, ∅)
mitte: (ε, {a}), (a, {a,b}), (a, {c}), (aa, ∅), (ab, ∅), (ac, ∅)
rechts: (ε, {a}), (a, {b,c}), (a, {a}), (aa, ∅), (ab, ∅), (ac, ∅)

Lösungen zu Kapitel 5

zu 1)

a)

```
if(a==0. && b==0.){
System.out.println("beliebiges x");
}
else{
if(a==0.){
        System.out.println("keine Lösung");
}
else{
        System.out.println("Lösung: "+(-b/a));
}
}
```

Semantische Erweiterung:

< `System.out.println(<Text>);` , z> -> <E, z>

b) <P,z> -> < `if(a==0.){`

```
                                System.out.println("keine Lösung");
                        }
                        else{
                                System.out.println("Lösung: "+(-b/a));
                        }, z>
```

-> < `System.out.println("keine Lösung");` , z> -> <E,z>

c) siehe a), weiterhin müssen Datentypen für Variablen angegeben werden.

zu 2)

Wenn Sem(<Variable> == <Ausdrucki>,z)= wahr und für alle j<i Sem(<Variable> == <Ausdruckj>,z)= falsch, dann < switch ..., z> -> <Befehlssequenzi, z>

Wenn Sem(<Variable> == <Ausdrucki>,z)= falsch und für alle j1<=i<=n , dann <switch ..., z> -> <Befehlssequenzn+1, z>

oder

```
if(<Variable> == <Ausdruck1>){
  <Befehlsequenz1>
}
else{
if(<Variable> == <Ausdruck1>){
           <Befehlsequenz2>
        }
            else{
        ...
        else{
                if(<Variable> == <Ausdruckn>){
                           <Befehlsequenzn>
                        }
                else {
                  <Befehlssequenzn+1>
```

```
                    }
            }
}
}
```

zu 3)

Syntax (Zahlwerte und Variablennamen sind noch zu ergänzen):

<Ausdruck> ::= <Zahl> | <Variable>

<Ausdruck> ::= (<Ausdruck>) | <Ausdruck> <op> <Ausdruck>

<op> ::= + | - | * | /

Semantik:

Sem(<Zahl>, z) = <Zahl>

Sem(<Variable>,z) = z(<Variable>)

Sem((<Ausdruck>), z) = (Sem(<Ausdruck>, z))

Sem(<Ausdruck1> <op> <Ausdruck2>, z)

= Sem(<Ausdruck1>,z) <op> Sem(<Ausdruck2>,z)

Dabei steht <op> für den Operator in der Menge der Zahlen

Für die Division durch Null kann eine zusätzliche Regel angegeben werden, so dass ein spezieller Wert angenommen wird, z.B. wie in Java:

Sem(<Ausdruck1> / <Ausdruck2>, z)

= NaN, falls Sem(<Ausdruck1>)= Sem(<Ausdruck2>)=0

= Infinity, falls Sem(<Ausdruck1>)>0 und Sem(<Ausdruck2>)=0

= -Infinity, falls Sem(<Ausdruck1>)<0 und Sem(<Ausdruck2>)=0

= Sem(<Ausdruck1>,z) / Sem(<Ausdruck2>,z) sonst

Alternativ kann die Semantik um eine Abbruchmöglichkeit ergänzt werden.

zu 4)

a)

Sem (p1,z)= Sem(x*y, z) < Sem(y*y,z)

= Sem(x,z) * Sem(y,z) < Sem(y,z) * Sem(y,z)

= z(x)*z(y) < z(y)*z(y)

= 6*7 < 7*7

= 42 < 49

= wahr

Sem (p2,z)= Sem(x+y, z) = Sem(x-y,z)

= Sem(x,z) + Sem(y,z) = Sem(x,z) - Sem(y,z)

= z(x)+z(y) = z(y)-z(y)

= 6+7 = 6-7

= 13 = -1

= falsch

Sem(p3,z)= Sem(¬(x>3 ∧ y>3) ∨ (x*y>9), z)

= Sem(¬(x>3 ∧ y>3), z) oder Sem((x*y>9), z)

= nicht Sem((x>3 ∧ y>3), z) oder (Sem(x*y,z) > Sem(9,z))

= nicht (Sem(x>3,z) und Sem(y>3,z)) oder (Sem(x,z)*Sem(y,z) > 9)

= nicht (Sem(x,z) > Sem(3,z) und Sem(y,z) > Sem(3,z)) oder (6 *7 > 9)

= nicht (6>3 und 7>3) oder (42>9)

= nicht (true und true) oder true

= nicht (true) oder true

= false oder true

= true

b)

p1[y:=1] ist x*1<1*1 ist x<1

p2[y:=1] ist x+1=x-1 ist 1=-1 ist false

p3[y:=1] ist (x>3 and 1>3) -> x*1>9) ist x>3 -> x>9 ist nicht(x>3) oder x>9 ist x<=3 oder x>9

p1[y:=x] ist x*x<x*x ist false

p2[y:=x] ist x+x=x-x ist 2*x=-0 ist x=0

p3[y:=x] ist (x>3 and x>3) -> x*x>9) ist x>3 -> x*x>9 ist nicht(x>3) oder x*x>9 ist x<=3 oder x*x>9 ist true

c)

Sem(p1) = {z | z(x)<z(y) für y>0} ∪ {z | z(x)>z(y) für y<0}

Sem(p2) = {z | z(y)=0}

Sem(p3) = {z | z beliebig}

zu 5)

SemP(P1,z1)={z} mit z(x)=6, z(y)=10

SemP(P1,z2)={z} mit z(x)=-1, z(y)=-1

SemP(P2,z1)={z} mit z(x)=2, z(y)=2

SemP(P2,z2)={z} mit z(x)=-1, z(y)=2,

SemP(P3,z1)={z} mit z(x)=0, z(y)=4, z(erg)=1

SemP(P3,z2)={}

zu 6)

SemP(P1,p1) = $2x-y=3 \wedge y-x>0$ Nebenrechnung : $x=3 \wedge y>0$ mit [x:=x+y] ergibt $x-y=3 \wedge y>0$; $x-y=3 \wedge y>0$ mit [y :=y+x] ergibt $x-(y-x)=3$ mit $y-x>0$

SemP(P1,p2) = $2x-y>1 \wedge y-x>2$ Nebenrechnung : $x>1 \wedge y>2$ mit [x:=x+y] ergibt $x-y>1 \wedge y>2$; $x-y>1 \wedge y>2$ mit [y :=y+x] ergibt $x-(y-x)>1 \wedge y-x>2$

SemP(P2,p1)= $x=3 \wedge y=1$ Nebenrechnung : $x=3 \wedge y>0$ mit [y=x-2] ergibt $x=3 \wedge y=1$

SemP(P2,p2)= $x>1 \wedge y>-1$

SemP(P3,p1)= $x=0 \wedge y>0 \wedge erg=-1$

SemP(P3,p2) $x=0 \wedge y>0 \wedge (erg=1 \vee erg=-1)$

zu 7) (nur wesentliche Rechenschritte genannt)

a) $(x=9 \wedge y=5)[x:=y+4] \equiv y=5$

$y=5[y:=3+x] \equiv x=2$

$x=2[x:=2] \equiv true$

b) if: $x>1 \wedge x<11[x:=x-5] \equiv x>6 \wedge x<16$, Rest mit

$x>0 \wedge x<16 \wedge x>8 \rightarrow x>6 \wedge x<16$, damit folgt

$\{ x>0 \wedge x<16 \wedge x>8\}$ x=x-5 $\{x>1 \wedge x<11\}$

else: $x>1 \wedge x<11[x:=x+2] \equiv x>-1 \wedge x<9$, Rest mit

$x>0 \wedge x<16 \wedge x \leq 8$ -> $x>-1 \wedge x<9$, damit folgt

$\{ x>0 \wedge x<16 \wedge x \leq 8 \}$ x=x-5 $\{x>1 \wedge x<11\}$

Rest mit Beweisregel

c) if: $x>3 \wedge x \leq 30[x:=x+x] \equiv x+x>3 \wedge x+x \leq 30 \equiv x>1.5 \wedge x \leq 15$, Rest mit

$x>1 \wedge x \leq 15 \wedge x>10 \rightarrow x>1.5 \wedge x \leq 15$

else: $x>3 \wedge x \leq 30$ $[x:=x*3] \equiv 3*x>3 \wedge 3* x \leq 30 \equiv x>1 \wedge x \leq 10$, Rest mit

$x>1 \wedge x \leq 15 \wedge x \leq 10 \rightarrow x>1 \wedge x \leq 10$

Rest wie b)

d) Gegenbeispiel: Zustand z(y)=3 erfüllt true (Sem(true,z) = true), nach Ausführung der Zuweisung wird der Zustand z1 mit z1(y)=4 erreicht, Widerspruch, da Sem(y=5,z1)=false

e) Fehler mit x=10

f) y=5[y:=4] ≡ 4=5 ≡ false

g) Invariante y=erg +x

y=erg +x [x:=x-2] ≡ y= erg+x-2

y=erg+x-2 [erg:=erg+2] ≡ y=erg+2+x-2 ≡ y=erg+x

Die einzige konkrete Möglichkeit für eine Terminierungsfunktion beinhaltet x. Da es aber keine Invariante gibt, die garantieren kann, dass x größer Null bleibt, kann man keine Invariante passend zu einer Terminierungsfunktion finden.

h) if-Zweig: Es wird x=0 → x=0 ∨ y=0 genutzt

x=0[x:=0] ≡ 0=0 ≡ true, mit true ∧ a → true folgt dieser Zweig

else-Zweig: Es wird y=0 → x=0 ∨ y=0 genutzt

y=0[y:=0] ≡ 0=0 ≡ true, mit true ∧ ¬a → true folgt dieser Zweig

i) zunächst inneres if von else, es wird gezeigt:

{¬a} `if (a) {x=42;} else {x=5;}` {x=5}

if: x=5 [x:=5] ≡ 5=5 ≡ true, Rest folgt mit Verstärkung ¬a ∧ ¬a → true

else: x=5 [x:=42] ≡ 42=5 ≡ false, Rest folgt mit Verstärkung ¬a ∧ a → false

Alternativer Beweis: Nutze äußeres if:

Zu zeigen (1) {true ∧ a} `if (b) {x=3;} else {x=4;}` {x=3 ∨ x=4 ∨ x=5}

(2) {true ∧ ¬a} `if (a) {x=42;} else {x=5;}` {x=3 ∨ x=4 ∨ x=5}

zu (1)

if: x=3 ∨ x=4 ∨ x=5 [x:=3] ≡ true, Rest folgt mit true ∧ a ∧ b → true

else: x=3 ∨ x=4 ∨ x=5 [x:=4] ≡ true, Rest folgt mit true ∧ a ∧ ¬b → true

zu (2)

if: : x=3 ∨ x=4 ∨ x=5 [x:=42] ≡ false, Rest folgt mit true ∧ ¬a ∧ a → false

else: x=3 ∨ x=4 ∨ x=5 [x:=5] ≡ true, Rest folgt mit true ∧ ¬a ∧ ¬a → true

j) Invariante p ≡ erg=6*(y-x)+24 ∧ x>3

erg=6*(y-x)+24 ∧ x>3 [x :=x-1] ≡ erg=6*(y-(x-1))+24 ∧ x-1>3

≡ erg=6*(y-x)+6+24 ∧ x>4

erg=6*(y-x)+6+24 ∧ x>4 [erg :=erg+6] ≡ erg+6=6*(y-x)+6 +24∧ x>4

≡ erg=6*(y-x) +24 ∧ x>4

gewünschter Regelteil folgt mit

$p \wedge b \equiv erg=6*(y-x)+24 \wedge x>3 \wedge x!=4 \equiv erg=6*(y-x)+24 \wedge x>4$

Initialisierung der Terminierungsfunktion t=x

$p \rightarrow t>0$ klar, da $erg=6*(y-x)+24 \wedge x>3 \rightarrow x>0$

Terminierungsfunktion

Zu zeigen $\{ p \wedge b \wedge t=z\}$ P $\{t<z\}$

Genutzt wird wieder $t=z-1 \rightarrow t<z$, gezeigt wird $\{ p \wedge b \wedge x=z\}$ P $\{x=z-1\}$

$x=z-1$ [x:=x-1] $\equiv x-1=z-1 \equiv x=z$

$x=z$ [erg:=erg+6] $\equiv x=z$

Rest folgt mit Verstärkung: $p \wedge b \wedge x=z \rightarrow x=z$

Vor der Schleife:

$erg=6*(y-x)+24 \wedge x>3$ [erg :=24] $\equiv 24=6*(y-x)+24 \wedge x>3 \equiv y=x \wedge x>3$,

Rest folgt mit der Verstärkung $x>4 \wedge y=x \rightarrow y=x \wedge x>3$

Nach der Schleife: Abschwächung $erg=6*(y-x)+24 \wedge x>3 \wedge \neg (x!=4)$

$\equiv erg=6*(y-4)+24 \wedge x>3 \wedge x=4 \equiv erg=6*y \wedge x>3 \wedge x=4 \rightarrow \equiv erg=6*y$

k) Invariante: $erg=(y)*(y+1)/2 \wedge x-y \geq 0$

$erg=(y)*(y+1)/2 \wedge x-y \geq 0$ [erg:=erg+y] $\equiv erg+y = (y)*(y+1)/2 \wedge x-y \geq 0$

$erg+y =(y)*(y+1)/2 \wedge x-y \geq 0$ [y:=y+1] $\equiv erg+y+1 = (y+1)*(y+2)/2 \wedge x-y-1 \geq 0$

Nebenrechnung : $(y+1)*(y+2) = y^2+3y+2$

$y2+3y+2 -2*(y+1) = y2+3y+2 -2y-2 = y2+y = (y)*(y+1)$

$erg+y+1 = (y+1)*(y+2)/2 \wedge x-y-1 \geq 0 \equiv erg=(y)*(y+1)/2 \wedge x-y-1 \geq 0$

$p \wedge b \equiv erg=(y)*(y+1)/2 \wedge x-y \geq 0 \wedge y \neq x \equiv erg=(y)*(y+1)/2 \wedge x-y \geq 1$,

damit erster Teil gezeigt

Anmerkung : $x-y \geq 0 \wedge y \neq x$ bedeutet, das y echt kleiner als x ist und

das heißt für ganze Zahlen $x-y \geq 1$.

Die Terminierungsfunktion ist x-y.

Ansatz: x-y+1=z mit Konsequenzregel abgeschwächt zu x-y<z

x-y+1=z [y:=y+1] ≡ x-(y+1) +1 =z ≡ x-y=z

l) Invariante: erg=$(y-x)^2 \wedge x \geq 0 \wedge$ add= (y-x)*2+1,

der Rest kann aus dem komplexeren Beweis zu m) abgeleitet werden.

m)

Vorüberlegungen:

$(y-x)^3 = (y-x) * (y^2 -2xy +x^2) = y^3 - 2xy^2 + x^2y - xy^2 + 2x^2y - x^3$

$= y^3 - 3xy^2 + 3x^2y - x^3$

$(y-(x-1))^3 = y^3 - 3(x-1)y^2 + 3(x-1)^2y - (x-1)^3$

$= y^3 - 3xy^2 + 3y^2 + (3x^2 - 6x + 3)y - (x^3 - 3x^2 + 3x - 1)$

$= y^3 - 3xy^2 + 3y^2 + 3x^2y - 6xy + 3y - x^3 + 3x^2 - 3x + 1$

$(y-(x-1))^3 - (y-x)^3 = 3y^2 - 6xy + 3y + 3x^2 - 3x + 1$

$(y-(x-2))^3 = y^3 - 3(x-2)y^2 + 3(x-2)^2y - (x-2)^3$

$= y^3 - 3xy^2 + 6y^2 + (3x^2 - 12x + 12)y - (x^3 - 6x^2 + 12x - 8)$

$= y^3 - 3xy^2 + 6y^2 + 3x^2y - 12xy + 12y - x^3 + 6x^2 - 12x + 8$

$(y-(x-2))^3 - (y-(x-1))^3 = 3y^2 - 6xy + 9y + 3x^2 - 9x + 7$

Zunächst wird eine Invariante gesucht,

Inv ≡ erg=$(y-x)^3 \wedge x \geq 0 \wedge$ sum= (y-x)*6 $\wedge$ add=$(y-x+1)^3 - (y-x)^3$; es ist zu zeigen:

(1) {Inv $\wedge$ x>0}

```
erg=erg+add;
sum=sum+6;
add=add+sum;
x=x-1;
```

{Inv}

Dies wird mit der Zuweisungsregel gezeigt.

Inv [x:=x-1] ≡ erg=$(y-(x-1))^3 \wedge$ x-1 ≥ 0 $\wedge$ sum= (y-(x-1))*6

$\wedge$ add=$(y-(x-1)+1)^3 - (y-(x-1))^3$

≡ erg=$(y-x+1)^3 \wedge$ x ≥ 1 $\wedge$ sum= (y-x+1)*6 $\wedge$ add=$(y-x+2)^3 - (y-x+1)^3$

erg=$(y-x+1)^3 \wedge$ x ≥ 1 $\wedge$ sum= (y-x+1)*6 $\wedge$ add=$(y-x+2)^3 - (y-x+1)^3$[add:=add+sum]

≡ erg=$(y-x+1)^3 \wedge$ x ≥ 1 $\wedge$ sum= (y-x+1)*6 $\wedge$ add+sum=$(y-x+2)^3 - (y-x+1)^3$

$\equiv erg=(y-x+1)^3 \wedge x \geq 1 \wedge sum= (y-x+1)*6 \wedge add+(y-x+1)*6 =(y-x+2)^3 -(y-x+1)^3$

Betrachten wir den letzten Teil genauer:

$add+(y-x+1)*6 =(y-x+2)^3 -(y-x+1)^3 \equiv add = 3y^2 - 6xy +9y + 3x^2 -9x + 7 – 6*(y-x+1)$

$\equiv add = 3y^2 - 6xy +3y + 3x^2 -3x + 1$

$\equiv add = (y-(x-1))^3 - (y-x)^3$

$erg=(y-x+1)^3 \wedge x \geq 1 \wedge sum= (y-x+1)*6 \wedge add = (y-(x-1))^3 - (y-x)^3$ [sum:=sum+6]

$\equiv erg=(y-x+1)^3 \wedge x \geq 1 \wedge sum+6= (y-x+1)*6 \wedge add = (y-(x-1))^3 - (y-x)^3$

$\equiv erg=(y-x+1)^3 \wedge x \geq 1 \wedge sum= (y-x)*6 \wedge add = (y-(x-1))^3 - (y-x)^3$

$erg=(y-x+1)^3 \wedge x \geq 1 \wedge sum= (y-x)*6 \wedge add = (y-(x-1))^3 - (y-x)^3$[erg:=erg+add]

$\equiv erg+add=(y-x+1)^3 \wedge x \geq 1 \wedge sum= (y-x)*6 \wedge add = (y-(x-1))^3 - (y-x)^3$

$\equiv erg=(y-x+1)^3 –((y-(x-1))^3 - (y-x)^3) \wedge x \geq 1 \wedge sum= (y-x)*6$

$\wedge add = (y-(x-1))^3 - (y-x)^3$

$\equiv erg=(y-x)^3 \wedge x \geq 1 \wedge sum= (y-x)*6 \wedge add = (y-(x-1))^3 - (y-x)^3$

Da $x \geq 0 \wedge x>0$ für ganzzahlige x äquivalent zu $x \geq 1$ ist (1) gezeigt.

Die Terminierungsfunktion ist wieder x.

Für das Programmende reicht es aus, dass $Inv \wedge x \geq 0 \wedge \neg(x > 0)$

direkt $erg=y^3$ folgt.

Für den Programmanfang wird wieder rückwärts gerechnet.

$Inv[add:=1] \equiv erg=(y-x)^3 \wedge x \geq 0 \wedge sum= (y-x)*6 \wedge 1=(y-x+1)^3 -(y-x)^3$

$erg=(y-x)^3 \wedge x \geq 0 \wedge sum= (y-x)*6 \wedge 1=(y-x+1)^3 -(y-x)^3$ [sum:=0]

$\equiv erg=(y-x)^3 \wedge x \geq 0 \wedge 0= (y-x)*6 \wedge 1=(y-x+1)^3 -(y-x)^3$

$\equiv erg=0 \wedge x=y \wedge 1=1^3$

$erg=0 \wedge x=y$ [erg:=0] $\equiv x=y$

Da aus $x>0 \wedge x=y$ auch $x=y$ folgt, ist mit dieser Verstärkung der Anfang bewiesen.

zu 8)

{false} P {false}: wird von jedem Programm erfüllt, da es keinen Zustand gibt, der die Vorbedingung erfüllt

{true} P {false}: beschreibt die Forderung, dass P nie terminieren darf, kann für totale Korrektheit nicht erfüllt werden

{false} P {true}: wird von jedem Programm erfüllt, da es keinen Zustand gibt, der die Vorbedingung erfüllt

{true} P {true}: gilt für partielle Korrektheit immer, fordert bei totaler Korrektheit die Programmterminierung

zu 9) Beweis für Programm mit drei Variablen, einfach auf andere Aufgabenteile übertragbar.

```
public  class Sortieren {

        public static int y1;
        public static int y2;
        public static int y3;

  public static void zweiSortieren(int x1, int x2) {
        if(x1<x2){
              y1=x1;
              y2=x2;
        }else{
              y1=x2;
              y2=x1;
        }
        System.out.println(y1+" # "+y2);
  }

  public static void dreiSortieren(int x1, int x2, int x3) {
        if(x1<=x2){
              if(x2<=x3){
                    y1=x1;
                    y2=x2;        } B1
                    y3=x3;
              }else{
                    if(x1<=x3){
                       y1=x1;
                       y2=x3;     } A1          B
                       y3=x2;
                    }else{                  A
                       y1=x3;
                       y2=x1;     } A2
                       y3=x2;
                    }
              }
        }else{
              if(x3>=x1){
                    y1=x2;
                    y2=x1;
                    y3=x3;
              }else{
                    if(x3<=x2){
                       y1=x3;
                       y2=x2;
                       y3=x1;
                    }else{
                       y1=x2;
                       y2=x3;
                       y3=x1;
                    }
              }
        }
        System.out.println(y1+" # "+y2+ " # "+y3);
  }

  public static void main(String[] args) {
        for(int i1=0; i1<4;i1++)
              for(int i2=0; i2<4;i2++){
                    zweiSortieren(i1, i2);
```

```
                    for(int i3=0; i3<4;i3++)
                         dreiSortieren(i1, i2, i3);
               }
  }
}
```

Vorbedingung: true

Nachbedingung N: $y1 \le y2 \le y3 \wedge ($ $(y1=x1 \wedge y2=x2 \wedge y3=x3)$

$\vee (y1=x1 \wedge y2=x3 \wedge y3=x2)$

$\vee (y1=x2 \wedge y2=x1 \wedge y3=x3)$

$\vee (y1=x2 \wedge y2=x3 \wedge y3=x1)$

$\vee (y1=x3 \wedge y2=x1 \wedge y3=x2)$

$\vee (y1=x3 \wedge y2=x2 \wedge y3=x1))$

Hinweis: Man kann auf Hilfsvariablen verzichten, insofern garantiert wird, dass das Programm x1, x2 und x3 nicht verändert.

Der Beweis erfolgt von innen nach außen. Zunächst wird für A gezeigt:

(1) $\{x1 \le x2 \wedge x3 < x2\}$ A $\{N\}$, dazu muss (2) und (3) gelten, damit (1) aus der Nutzung der Alternativ-Regel folgt:

(2): $\{x1 \le x2 \wedge x3 < x2 \wedge x1 \le x3\}$ A1 $\{N\}$, was äquivalent ist, zu $\{x1 \le x3 < x2\}$ A1 $\{N\}$

genauer wird gezeigt:

$\{x1 \le x3 < x2\}$ A1 $\{ y1 \le y2 \le y3 \wedge (y1=x1 \wedge y2=x3 \wedge y3=x2)\}$, da gilt:

$(y1 \le y2 \le y3 \wedge (y1=x1 \wedge y2=x3 \wedge y3=x2)) \rightarrow N$ (einfache Abschwächung des zweiten Teils)

Nachrechnen mit Zuweisungsregel:

$y1 \le y2 \le y3 \wedge (y1=x1 \wedge y2=x3 \wedge y3=x2)\ [y3:=x2]$

$\equiv y1 \le y2 \le x2 \wedge (y1=x1 \wedge y2=x3 \wedge x2=x2)$

$\equiv y1 \le y2 \le x2 \wedge (y1=x1 \wedge y2=x3)$

$y1 \le y2 \le x2 \wedge (y1=x1 \wedge y2=x3)[y2:=x3]$

$\equiv y1 \le x3 \le x2 \wedge (y1=x1 \wedge x3=x3)$

$\equiv y1 \le x3 \le x2 \wedge y1=x1$

$y1 \le x3 \le x2 \wedge y1=x1\ [y1:=x1]$

$\equiv x1 \le x3 \le x2 \wedge x1=x1$

$\equiv x1 \le x3 \le x2$

da die Verstärkung $x1 \le x3 < x2 \rightarrow x1 \le x3 \le x2$ gilt, ist (1) gültig

(3): $\{x1 \le x2 \wedge x3 < x2 \wedge x1 > x3\}$ A2 $\{N\}$, was äquivalent ist, zu $\{x3 < x1 \le x2\}$ A1 $\{N\}$

genauer wird gezeigt:

$\{ x3 < x1 \leq x2\}$ A2 $\{ y1 \leq y2 \leq y3 \wedge (y1=x3 \wedge y2=x1 \wedge y3=x2)\}$, da gilt:

$(y1 \leq y2 \leq y3 \wedge (y1=x3 \wedge y2=x1 \wedge y3=x2)) \rightarrow N$ (einfache Abschwächung des zweiten Teils)

Nachrechnen mit Zuweisungsregel analog zu (2), damit ist (1) gezeigt.

Im nächsten Schritt wird das äußere if betrachtet. Es wird gezeigt:

(4) $\{x1 \leq x2\}$ B $\{N\}$, dazu muss (5) und (6) gelten:

(5) $\{x1 \leq x2 \wedge x2 \leq x3\}$ B1 $\{N\}$ was äquivalent ist, zu $\{x1 \leq x2 \leq x3\}$ B1 $\{N\}$

genauer wird gezeigt:

$\{x1 \leq x2 \leq x3\}$ B1 $\{ y1 \leq y2 \leq y3 \wedge (y1=x1 \wedge y2=x2 \wedge y3=x3)\}$, was analog zu (2) gezeigt werden kann.

(6) $\{x1 \leq x2 \wedge x2 > x3\}$ A $\{N\}$, was aber als (1) schon gezeigt wurde.

Der Rest des Beweises lässt sich analog zu (2) bzw. (4) komponieren.

zu 10) nur Programm, Lösungsaufbau wie in 9)

```
public class Alter {

    public int maxJahre(int alter, boolean geschlecht, boolean raucher){
        int ergebnis=85-alter;
        if(!geschlecht){
            if(raucher){
                ergebnis-=9;
            }
            else{
                ergebnis-=5;
            }
        }
        else {
            if(raucher){
                ergebnis-=3;
            }
        }
        return ergebnis;
    }

    public static void main(String[] args) {
        Alter a=new Alter();
        int alter=60;
        boolean raucher=true;
        boolean geschlecht=true;
        System.out.println("Alter:"+alter+" Geschlecht:"+geschlecht+" Raucher:"
                +raucher+" = "+a.maxJahre(alter,geschlecht,raucher));
        geschlecht=false;
        System.out.println("Alter:"+alter+" Geschlecht:"+geschlecht+" Raucher:"
                +raucher+" = "+a.maxJahre(alter,geschlecht,raucher));
        raucher=false;
        System.out.println("Alter:"+alter+" Geschlecht:"+geschlecht+" Raucher:"
                +raucher+" = "+a.maxJahre(alter,geschlecht,raucher));
        geschlecht=true;
        System.out.println("Alter:"+alter+" Geschlecht:"+geschlecht+" Raucher:"
                +raucher+" = "+a.maxJahre(alter,geschlecht,raucher));
```

```
    }
}
```

zu 11)

a) folgende Ausführungsreihenfolgen sind möglich:

P1 P2 P3: x=8

P1 P3 P2: x=12

P2 P1 P3: x=6

P2 P3 P1: x=6

P3 P1 P2: x=12

P3 P2 P1: x=10

b) nur Idee:

{y=x} P { x=((y+1)*3)+2 ∨ x=(y+3)*3 ∨ x= (y*3)+3 ∨ x= ((y+2)*3+1}

c) mit b) für x<10 folgt am Ende (für y den Wert 10 einsetzen):

x<35 ∨ x<39 ∨ x<33 ∨ x<37

zu 12)

a) Angenommen P1 und P2 betreten immer direkt nacheinander ihre while-Schleifen und führen dann jeweils den Rumpf der Schleife aus, dann ist am Ende der Schleifen wieder der Ausgangszustand erreicht.

b) Da die Schleifen wegen a) nicht durchlaufen werden dürfen, ist Folgendes möglich: {x<0 ∧ y<0} P {true}

c)

{s=x+y+pos2 && pos1=0}

```
P1 is {
    pos1=0;
```

{inv:s=x+y+pos2 && pos1=0}

```
  while(x>0){
```

{s=x+y+pos2 && pos1=0}

```
    atomic{x=x-1; pos1=1;}
```

{s=x+y+1+pos2 && pos1=1}

```
    atomic{y=y+1; pos1=0;}
   }
}
```

{s=x+y+pos2 && pos1=0}

{s=x+y+pos1 && pos2=0}

```
P2 is {
    pos2=0;
```

{inv:s=x+y+pos1 && pos2=0}

```
  while(x>0){
```

{s=x+y+pos1 && pos2=0}

```
    atomic{y=y-1; pos2=1;}
```

{s=x+y+1+pos1 && pos2=1}

```
      atomic{x=x+1; pos2=0;}
    }
}
```

{s=x+y+pos1 && pos2=0}

p	pre(Q)	p && pre(Q)	Q	p
{s=x+y+pos2 && pos1=0}	{s=x+y+pos1 && pos2=0}	s=x+y && pos1=0 && pos2=0	pos2=0;	{s=x+y+pos2 && pos1=0}
{s=x+y+pos2 && pos1=0}	{s=x+y+pos1 && pos2=0}	s=x+y && pos1=0 && pos2=0	atomic{y=y-1; pos2=1;}	{s=x+y+pos2 && pos1=0}
{s=x+y+pos2 && pos1=0}	{s=x+y+1+pos1 && pos2=1}	s=x+y+1 && pos1=0 &&pos2=1	atomic{x=x+1; pos2=0;}	{s=x+y+pos2 && pos1=0}
{s=x+y+1+pos2 && pos1=1}	{s=x+y+pos1 && pos2=0}	s=x+y+1 && pos1=1 && pos2=0	pos2=0;	{s=x+y+1+pos2 && pos1=1}
{s=x+y+1+pos2 && pos1=1}	{s=x+y+pos1 && pos2=0}	s=x+y+1 && pos1=1 && pos2=0	atomic{y=y-1; pos2=1;}	{s=x+y+1+pos2 && pos1=1}
{s=x+y+1+pos2 && pos1=1}	{s=x+y+1+pos1 && pos2=1}	s=x+y+2 && pos1=1 && pos2=1	atomic{x=x+1; pos2=0;}	{s=x+y+1+pos2 && pos1=1}

Jeweils gezeigt durch Rückwärtsrechnung für die Zuweisung. Im letzten Fall z.B.

(s=x+y+1+pos2 && pos1=1)[pos2:=0][x:=x+1] ergibt (s=x+1+y+1+0 && pos1=1) ist (s=x+y+2 && pos1=1)

mit (s=x+y+2 && pos1=1 && pos2=1) -> (s=x+y+2 && pos1=1) folgt die Behauptung.

Rest ist Symmetrie.