Technische Universität München
Fakultät für Informatik
Prof. Dr. M. Broy

WS 2002/2003
1. Februar 2003

Lösungsvorschläge der Klausur zu Einführung in die Informatik III

A

Aufgabe 1 Semaphore (3 + 7 = 10 Punkte)

(a) Folgende Tabelle illustriert die ersten Durchläufe des Petrinetzes:

Durchlauf-Nr.	z	x	y
0	$<>$	$<>$	$<>$
1	$<>$	$< a >$	$< b >$
2	$< aba >$	$< aa >$	$< bb >$
3	$< aabbaa >$	$< aaa >$	$< bbb >$
...	...	...	...
n	$< a^{n-1}b^{n-1}a^{n-1} >$	$< a^n >$	$< b^n >$

Die Variable z nimmt also die Werte $< a^n b^n a^n >, n \in \mathbb{N}_0$ an.

(b) Es ergibt sich folgendes Programm:

```
⌈  var seq char x, y, z := <>,<>,<>;
   sema bool s1, s2, s3, s4 := true, true, false, false;
   ⟦  while true do
      P(s1); P(s2); z := x ∘ y ∘ x; V(s3); V(s4) od
   ||  while true do
      P(s3); x := <a> ∘ x; V(s1) od
   ||  while true do
      P(s4); y := <b> ∘ y; V(s2) od ⟧⌋
```

Aufgabe 2 await auf der MI (5 + 5 = 10 Punkte)

(a) **`await`** *b* **`then`** *c* **`endwait`** wird simuliert durch:

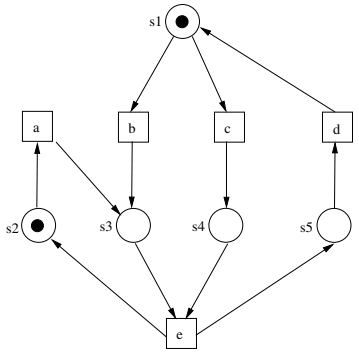
```
sema bool mutex := true;
var wait := true;
while (wait) do
  P(mutex);
  if b then wait := false else V(mutex) fi
od ;
c
V(mutex)
```

(b) **`await`** *b* **`then`** *c* **`endwait`** wird auf der MI simuliert durch:

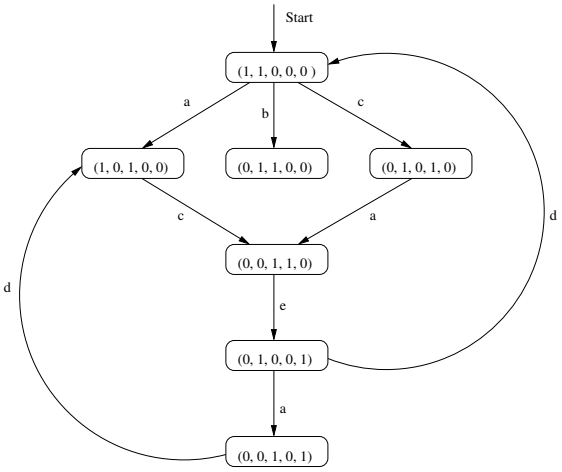
```
b:          RES 1

mutex:      DD B 1                      -- sema bool mutex := true;

await:      JBCCI I 7, mutex, await     -- P(mutex);

              -- b berechnen

            CMP W I 0, b
            JNE continue
            OR B I 1, mutex             -- V(mutex);
            JUMP await
continue:
              -- c ausführen

            OR B I 1, mutex             -- V(mutex);
```

Aufgabe 3 Invarianten in Petrinetzen (3 + 1 + 3 + 3 = 10 Punkte)

Betrachten Sie das folgende Boolesche Petrinetz:



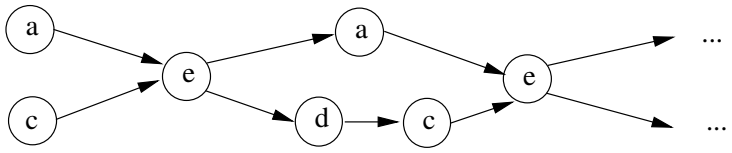
(a) Bestimmen Sie die aus der Anfangsbelegung erreichbaren Belegungen (Zustände) und zeichen Sie das Zustandsübergangsdiagramm. Markieren Sie die Kanten mit den entsprechenden Transitionen.



(b) Gibt es Verklemmungszustände ?

Der Zustand $(0,1,1,0,0)$ ist ein Verklemmungszustand.

(c) Zeichnen Sie ein Aktionsdiagramm, das die unendlichen Abläufe des Netzes wiedergibt.



(d) Geben Sie eine Invariante des Systems an, d.h. eine Eigenschaft, die das System in jedem erreichbaren Zustand erfüllt und weisen Sie die Invarianz dieser Eigenschaft mit Hilfe des Beweisprinzips für Invarianten nach.

Es sind immer genau zwei Marken im System.
$q(m) \equiv_{def} m(s_1)+m(s_2)+m(s_3)+m(s_4)+m(s_5) = 2$

Beweis:

(1) Im Anfangszustand $m_0 = (1,1,0,0,0)$ gilt die Invariante:
$q(m_0) = m_0(s_1)+m_0(s_2)+m_0(s_3)+m_0(s_4)+m_0(s_5) = 2$.

(2) Falls für eine Belegung m die Bedingung $q(m)$ gilt und $m \xrightarrow{a} m'$, dann gilt $q(m')$, denn:
$m'(s_3) = m(s_3)+1$ und $m'(s_2) = m(s_2)-1$.

Falls für eine Belegung m die Bedingung $q(m)$ gilt und $m \xrightarrow{b} m'$, dann gilt $q(m')$, denn:
$m'(s_3) = m(s_3)+1$ und $m'(s_1) = m(s_1)-1$.

Falls für eine Belegung m die Bedingung $q(m)$ gilt und $m \xrightarrow{c} m'$, dann gilt $q(m')$, denn:
$m'(s_4) = m(s_4)+1$ und $m'(s_1) = m(s_1)-1$.

Falls für eine Belegung m die Bedingung $q(m)$ gilt und $m \xrightarrow{d} m'$, dann gilt $q(m')$, denn:
$m'(s_1) = m(s_1)+1$ und $m'(s_5) = m(s_5)-1$.

Falls für eine Belegung m die Bedingung $q(m)$ gilt und $m \xrightarrow{e} m'$, dann gilt $q(m')$, denn:
$m'(s_2) = m(s_2)+1$, $m'(s_5) = m(s_5)+1$ und $m'(s_3) = m(s_3)-1$, $m'(s_4) = m(s_4)-1$.

Aufgabe 4 Dinierende Philosophen in Java (1 + 6 + 3 = 10 Punkte)

(a) Die Lösung ist verklemmungsfrei, da kein Prozess (Philosoph) ein Betriebsmittel (eine Gabel) belegt (für sich reserviert) und dann auf ein anderes Betriebsmittel (Gabel) wartet. Zyklisches warten ist damit ausgeschlossen. 1/2

Verhungern ist möglich. Wenn sich z. B. das abwechselnde Essen der Philosopen 1 und 3 stets überlappt, dann gibt es keine Garantie, dass der wartende Philosoph 2 beide Gabeln irgendwann erhält. 1/2

(b) kurze Lösungsvariante:

```
class Tisch {                                          // 1/2
    private boolean g[] = {true,true,true,true,true }; // 1/2 g & init

    public synchronized void awaitTakeForks (int i) {  // 1/2 sync
        // Teil 1: await E
        while (!(g[i] && g[(i+1) % 5])) {
```

```
            try { wait(); } catch(Exception ex) {};     // 1
        }
        // Teil 2: then S
        g[i] = false;                                   // 1/2
        g[(i+1) % 5] = false;                           // 1/2
    }   // Teil 3: endwait

    public synchronized void awaitPutForks (int i) {    // 1/2 sync
        g[i] = true;                                    // 1/2
        g[(i+1) % 5] = true;                            // 1/2
        notifyAll();                                    // 1
    }
}
```

(c)
```
class Philosoph extends Thread {                    // 1/2 extends Thread
    private Tisch t;    // Tisch mit den Gabeln     // 1/2 korrekte
    private int nr;     // Nummer des Philosophen   //     Attribute

    Philosoph(Tisch t, int nr) {                    // 1/2 Konstruktor &
        this.t  = t;                                //     Param-Übergabe
        this.nr = nr;
    }

    public void run() {                             // 1/2
        while (true) {                              // 1/2
            // think
            t.awaitTakeForks(nr);                   // 1/2 take und put
            // eat
            t.awaitPutForks(nr);
        }
    }
}
```

Syntaktische Fehler bei b) und c): einmaliger Abzug 1P, wobei Strichpunkte und Kommentare ignoriert werden.