Name: | Vorname: | Matr.–Nr.:

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

WS 2002/2003
1. Februar 2003

Klausur zu Einführung in die Informatik III

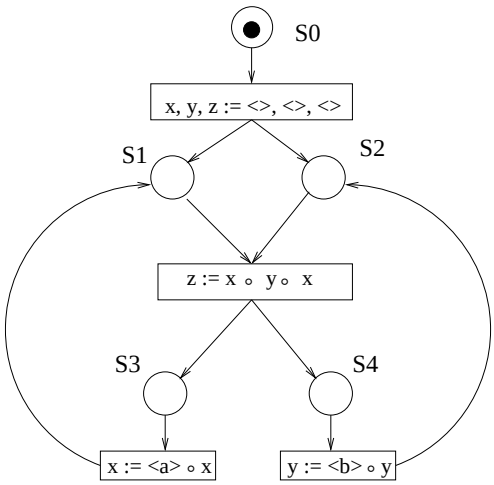
(Gruppe A)

Aufgabe 1 Semaphore (5 + 5 = 10 Punkte)

Das folgende Programm arbeitet auf Zeichenreihen:

```
⌈  var seq char x, y, z := <>,<>,<>;
   ‖  while true do  z := x ∘ y ∘ x od
   ||  while true do  x := <a> ∘ x od
   ||  while true do  y := <b> ∘ y od ‖⌋
```

Die Zuweisungen erscheinen als Transitionen in dem rechts angegebenen Booleschen Petrinetz.



(a) Welche Werte kann die Variable *z* im Petrinetz annehmen ?

(b) Ergänzen Sie das Programm durch Sempahore, so dass es das gleiche Verhalten zeigt, wie das Petrinetz.

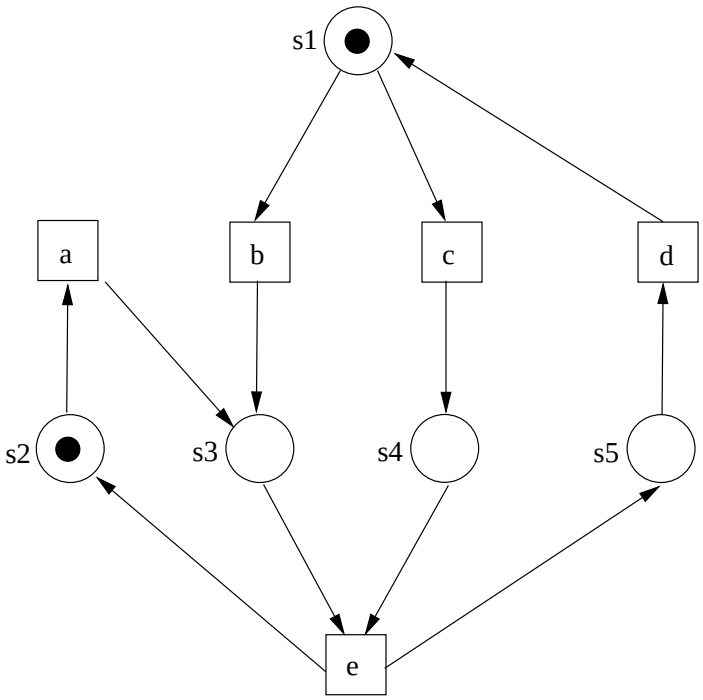
Aufgabe 2 await auf der MI (4 + 6 = 10 Punkte)

(a) Geben Sie eine mögliche Realisiserung des Befehls **`await`** *b* **`then`** *c* **`endwait`** mit Hilfe von Semaphoren an.

(b) Geben Sie eine Realisierung des Befehls **`await`** *b* **`then`** *c* **`endwait`** auf der MI an. Sie können davon ausgehen, dass die MI-Programm-Teile zur Berechnung von *b* und zur Ausführung von *c* gegeben sind.

Aufgabe 3 **(2 + 2 + 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 ?

(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.

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

In der Vorlesung „Einführung in die Informatik III“ wurde u.a. folgende Lösung für das Problem der dinierenden Philosophen diskutiert:

```
var [0:4] array bool g;
g[0],g[1],g[2],g[3],g[4] := true, true, true, true, true;

proc Philosoph = (nat i):
⌈ while true do „think“;
        await g[i] ∧ g[i+1 mod 5] then
              g[i], g[i+1 mod 5] := false, false endwait ;
        „eat“;
        await true then
              g[i], g[i+1 mod 5] := true, true endwait ;
  od  ⌋

⟦ Philosoph(0) ‖...‖ Philosoph(4) ⟧
```

(a) Beantworten Sie folgende Fragen und geben Sie eine kurze Begründung an:
Ist diese Lösung verklemmungsfrei? Ist ein Verhungern eines Prozesses möglich?

(b) Implementieren Sie die kritischen Bereiche des Algorithmus zusammen mit den 5 Gabeln (Feld `g`) in einer Java Hilfsklasse `Tisch` (Hinweis: verwenden Sie das Java Monitorkonzept analog zur Übungsaufgabe 31).

(c) Implementieren Sie die Prozedur „Philosoph“, der parallel essenden und denkenden Philosophen, in einer Java Klasse `Philosoph`.