Technische Universität München
Fakultät für Informatik
Dr. M. Schneider, G. Bauer,
Dr. M. Pizka, Dr. B. Rumpe,
Dr. R. Steinbrüggen

WS 2002/2003
Lösungsvorschläge zu Blatt 4
15. November 2002

Übungen zur Vorlesung Einführung in die Informatik III

Aufgabe 12 Agenten, Abläufe

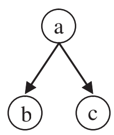
(a) Sei $E = \{e_0, e_1, e_2, ...\}$ die zugrunde liegende Ereignismenge ("Universum"). Die Abläufe eines Agenten sind (bis auf Isomorphie) alle vollständigen Abläufe sowie alle deren Präfixe (auch der leere Prozess).

(i) Der Agent $t_0 =_{def}\ a;(b\ \textbf{or}\ c)$ hat zwei vollständige Abläufe:

- $p_0 = (E_0, \leq_0, \alpha_0)$ mit
  $E_0 = \{e_0, e_1\}$,
  $\alpha_0(e_0) = a,\ \alpha_0(e_1) = b$,
  $\leq_0 = \{(e_0, e_1)\}*$
- $p_1 = (E_1, \leq_1, \alpha_1)$ mit
  $E_1 = \{e_0, e_1\}$,
  $\alpha_0(e_0) = a,\ \alpha_0(e_1) = c$,
  $\leq_0 = \{(e_0, e_1)\}*$

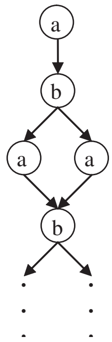
(ii) Der Agent $t_1 =_{def}\ a;(b \,||\, c)$ besitzt den folgenden Prozess als vollständigen Ablauf.

- $p_0 = (E_0, \leq_0, \alpha_0)$ mit
  $E_0 = \{e_0, e_1, e_2\}$,
  $\alpha_0(e_0) = a,\ \alpha_0(e_1) = b,\ \alpha_0(e_2) = c$,
  $\leq_0 = \{(e_0, e_1), (e_0, e_2)\}*$



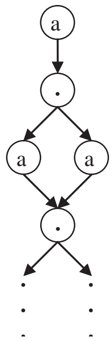
(iii) Der Agent $t_2 =_{def}\ (x :: a; b; x) \,||_{\{b\}}\, (y :: b; a; y)$ besitzt den folgenden Prozess als vollständigen Ablauf:

- $p_0 = (E_0, \leq_0, \alpha_0)$ mit
  $E_0 = E$,
  $\alpha_0(e_{3i}) = a,\ \alpha_0(e_{3i+1}) = b,\ \alpha_0(e_{3i+2}) = a$ für alle $i \in \mathbb{N}_0$,
  $\leq_0 = \{(e_{3i}, e_{3i+1}), (e_{3i+1}, e_{3i+2}), (e_{3i+1}, e_{3i+3}), (e_{3i+2}, e_{3i+4}) : i \in \mathbb{N}_0\}*$



(iv) Die vollständigen Abläufe des Agenten $t_3 =_{def}\ (x :: a; (b\ \mathbf{or}\ c); x)\ ||_{\{b,c\}} (y :: (b\ \mathbf{or}\ c); a; y)$ sind alle Prozesse p_0 mit der folgenden Eigenschaft.

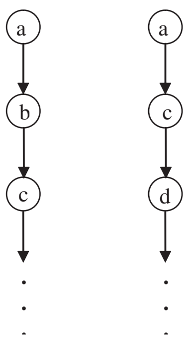
- $p_0 = (E_0, \leq_0, \alpha_0)$ mit
  $E_0 = E$,
  $\alpha_0(e_{3i}) = a,\ \alpha_0(e_{3i+1}) \in \{b, c\},\ \alpha_0(e_{3i+2}) = a$ für alle $i \in \mathbb{N}_0$,
  $\leq_0 = \{(e_{3i}, e_{3i+1}), (e_{3i+1}, e_{3i+2}), (e_{3i+1}, e_{3i+3}), (e_{3i+2}, e_{3i+4}) : i \in \mathbb{N}_0\}*$



wobei $\bullet \in \{b, c\}$

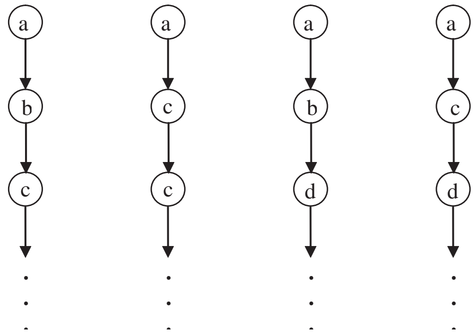
(v) Die vollständigen Abläufe des Agenten $t_4 =_{def}\ (x :: a; (b; c)\ \mathbf{or}\ (c; d)); x)$ sind alle Prozesse p_0 mit der folgenden Eigenschaft.

- $p_0 = (E_0, \leq_0, \alpha_0)$ mit
  $E_0 = E$,
  $\alpha_0(e_{3i}) = a,\ (\alpha_0(e_{3i+1}) = b \wedge \alpha_0(e_{3i+2}) = c) \vee (\alpha_0(e_{3i+1}) = c \wedge \alpha_0(e_{3i+2}) = d)$
  für alle $i \in \mathbb{N}_0$,
  $\leq_0 = \{(e_i, e_{i+1}) : i \in \mathbb{N}_0\}*$



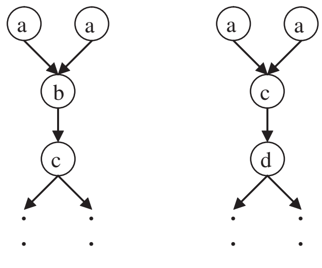
Dagegen sind die Abläufe des Agenten $(y :: a; (b \textbf{ or } c); (c \textbf{ or } d); y)$ alle Prozesse p_1 mit der folgenden Eigenschaft.

- $p_0 = (E_0, \leq_0, \alpha_0)$ mit
  $E_0 = E$,
  $\alpha_0(e_{3i}) = a,\ \alpha_0(e_{3i+1}) \in \{b, c\},\ \alpha_0(e_{3i+2}) \in \{c, d\}$ für alle $i \in \mathbb{N}_0$,
  $\leq_0 = \{(e_i, e_{i+1}) : i \in \mathbb{N}_0\}*$



Durch die parallele Komposition werden die Aktionen b, c, d synchron und a asynchron ausgeführt.
Der Agent $t_4 =_{def} (x :: a; ((b; c) \textbf{ or } (c; d)); x) \mid\mid_{\{b,c,d\}} (y :: a; (b \textbf{ or } c); (c \textbf{ or } d); y)$ besitzt also die folgenden Abläufe.

- $p_2 = (E_2, \leq_2, \alpha_2)$ mit
  $E_2 = E$,
  $\alpha_2(e_{4i}) = a,\ \alpha_2(e_{4i+1}) = a$,
  $(\alpha_2(e_{4i+2}) = b\ \wedge\ \alpha_2(e_{4i+3}) = c)\ \vee\ (\alpha_2(e_{4i+2}) = c\ \wedge\ \alpha_2(e_{4i+3}) = d)$ für alle $i \in \mathbb{N}_0$,
  $\leq_2 = \{(e_{4i}, e_{4i+2}), (e_{4i+1}, e_{4i+2}), (e_{4i+2}, e_{4i+3}), (e_{4i+3}, e_{4i+4}), (e_{4i+3}, e_{4i+5}) : i \in \mathbb{N}_0\}*$

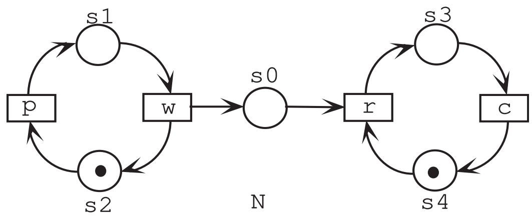


oder

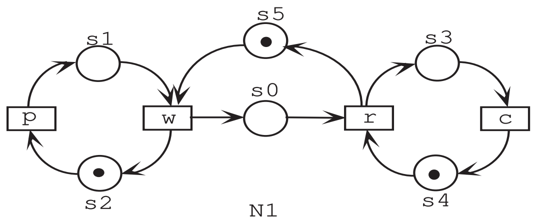
In den Agenten t_3 und t_4 können Verklemmungen auftreten, nämlich genau dann wenn sich ein Teilagent die Aktion b und der andere die Aktion c ausführen will.

(b) Der folgende Agent beschreibt den Prozess P aus Aufgabe 2.
$t_5 =_{def} ((x :: a_0; a_2; x) \mid\mid (y :: a_1; a_3; y)) \mid\mid_{\{a_0, a_3\}} (z :: a_0; a_3; z)$

Aufgabe 13 Petri-Netze, Agenten, Prozessprädikate



Mit der oben angegebenen (erreichbaren) Belegung kann `w` nicht schalten, weil `s0` belegt ist (Konvention für Boolesche Petri-Netze).
Ein Ablauf ist ein Prozess. Die Menge der Abläufe eines Petri-Netzes oder eines Agenten wurde in der Vorlesung induktiv definiert.



a) Der Graph `N1` ist ein ganzzahliges Petri-Netz, das zu `N` ablaufäquivalent ist. (`N1` besitzt die gleiche Menge vollständiger Abläufe wie `N` .) In `N1` sind `w` und auch `r` genau dann schaltbereit, wenn sie es in `N` sind, wie man leicht nachprüft. (Die Konstruktion ist verallgemeinerungsfähig auf Stellen mit mehreren Ein- und Ausgängen, vgl. Hausaufgabe.)

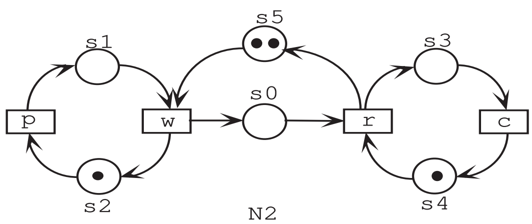
b) Im Petri-Netz `N1` lassen sich drei Agenten erkennen:

```
P = ( prod::p;w;prod )
C = ( cons::r;c;cons )
B = ( buff::w;r;buff )
```

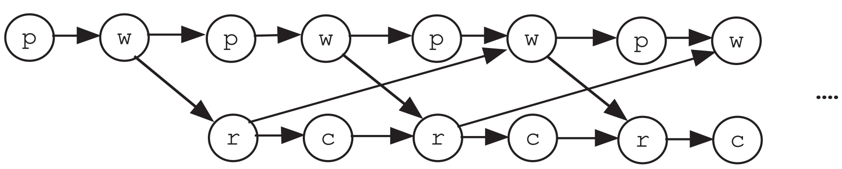
Diese sind über die gemeinsamen Aktionen `w` und `r` koordiniert:

$t = (P||C)||_{\{w,r\}}B$

In Bezug auf die zugehörigen Abläufe ist synchronisierende Parallelisierung dadurch definiert, dass paarweise die gleich markierten Ereignisse identifiziert werden, und zwar unter Wahrung der Kausalitätsbeziehungen.



c) Das Petrinetz `N2` beschreibt ein Erzeuger/Verbraucher-Problem mit Puffergröße 2; darunter ein vollständiger Ablauf `A2` zu `N2`. Nach der Sequenz p, w, p (z.B.) kann auf den Puffer gleichzeitig lesend und schreibend zugegriffen werden.



d) Es gelten die folgenden Beziehungen (Es wird die totale Differenz auf `IN` benutzt):

(1) für $q = A$

$\forall e \in E : \alpha(e) = r \Rightarrow \#(r, A_e) = \#(w, A_e)$

$\forall e \in E : \alpha(e) = w \Rightarrow \#(r, A_e) = \#(w, A_e) - 1$

(2) für $q = A2$

$\forall e \in E : \alpha(e) = r \Rightarrow \#(r, A_e) = \#(w, A_e)$

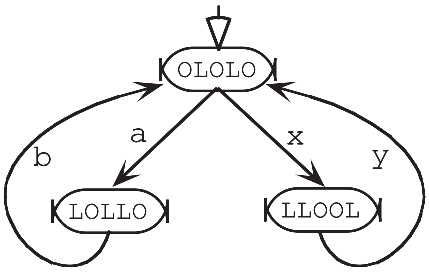
$\forall e \in E : \alpha(e) = w \Rightarrow \#(r, A_e) = \#(w, A_e) - 2$

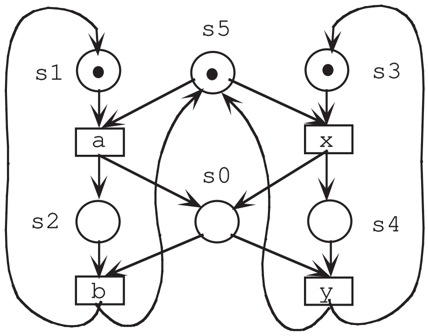
Aufgabe 14 (H) Petri-Netze

a) Unten links steht ein Zustandsübergangsgraph zum vorgelegten Booleschen Petri-Netz.

b) Blockierend ist die Stelle `s0`.

c) Rechts ist ein ablaufäquivalentes natürlichzahliges Petri-Netz angegeben. Die zusätzliche Stelle `s5` mit ihren Kanten und ihrer Anfangsbelegung modelliert die blockierende Wirkung von `s0` im Booleschen Petri-Netz. Diese Wirkung ist als „Racing Condition" interpretierbar. Das Netz läßt sich vereinfachen, denn die Stellen `s0`, `s1`, `s3` mit ihren Kanten sind jetzt überflüssig.





Aufgabe 15 (P) Erreichbarkeit von Zuständen

a)

```
class BoolState {
   private boolean s[];

   BoolState(int n) {
       s = new boolean[n];
       for (int i = 0; i < n; i++)
           s[i] = false;
   }

   public int num() {
       return s.length;
   }

   public void setStateVar(int i, boolean b) {
       if (i >= 0 && i < s.length)
           s[i] = b;
```

```
    }

    public boolean getStateVar(int i) {
        return (i >= 0 && i < s.length) ? s[i] : false;
    }

    public boolean equals(Object x) {
        BoolState o = (BoolState) x;
        if (o == null)
            return false;
        if (num() != o.num())
            return false;
        for (int i = 0; i < s.length; i++)
            if (getStateVar(i) != o.getStateVar(i))
                return false;
        return true;
    }

    public String toString() {
        String r = "";
        for (int i = 0; i < s.length; i++)
            r += s[i] ? "L" : "O";
        return r;
    }
}
```

b-d)

```
 import java.util.Vector;

public class BPetriNetReachable extends BPetriNet {

    public BoolState getState() {
        BoolState state = new BoolState(places.length);
        for (int i = 0; i < places.length; i++)
            state.setStateVar(i,getPlace(i).get());
        return state;
    }

    public void setState(BoolState state) {
        for (int i = 0; i < state.num(); i++)
            getPlace(i).set(state.getStateVar(i));
    }

    public void reachable() {
        BoolState s1,s2;
        Vector R = new Vector(), T = new Vector();
        T.add(getState());

        System.out.println("T: "+T+"\n"+"R: "+R+"\n");
        while (!T.isEmpty()) {
            s1 = (BoolState) T.firstElement();
            for (int i = 0; i < transitions.length; i++) {
```

```
                setState(s1);
                if (getTransition(i).enabled())
                    {
                        getTransition(i).fire();
                        s2 = getState();
                        if (!(T.contains(s2) || R.contains(s2)))
                            T.add(s2);
                    }
            }
            T.remove(s1);
            R.add(s1);
            System.out.println("T: "+T+"\n"+"R: "+R+"\n");
        }
    }
}
```