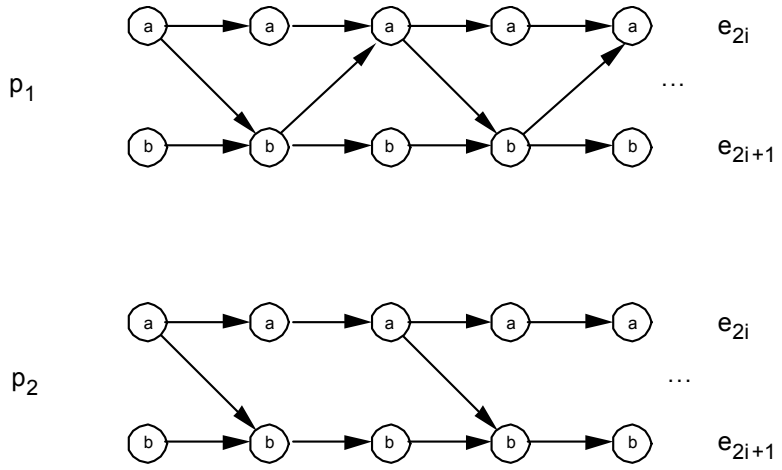
Technische Universität München
Fakultät für Informatik
Prof. Dr. M. Broy

WS 2002
Lösungsblatt 3
8. November 2002

Übungen zur Einführung in die Informatik III

Aufgabe 9 Fairness

Im Folgenden betrachten wir die Prozesse

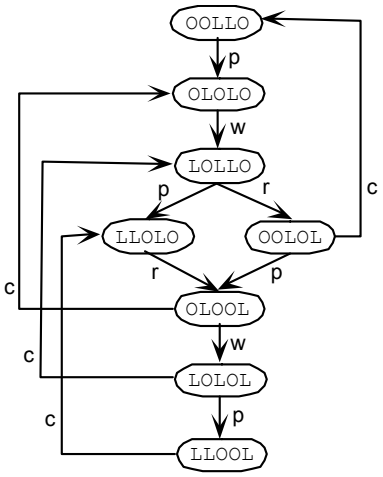


Die Menge der unfairen Spuren, d.i. die Menge `MaxSpuren(p)\Spuren(p)`, besteht aus Spuren von Prozessen, die nicht alle Ereignisse von p enthalten. Die Definition von `MaxSpuren(p)` ist dem topologischen Sortieren nachgebildet: Jedes Element aus S(p) beginnt mit der Aktion a eines Ereignisses e, das keinen kausalen Vorgänger besitzt und hat einen Rest t aus $S(p|_{E\setminus\{e\}})$. Für dieses t ist analog fortzufahren. Die Frage ist, ob es dafür eine unendliche Folge von Ereignissen gibt, die echt kleiner ist als die Gesamtmenge der Ereignisse.

Bei p_1 ist das nicht der Fall, denn vor jedem e_{2i+2} und jedem e_{2i+1} ($i=1,2,...$) ist jeweils der gesamte Präfix des Ereignisses berücksichtigt . Dagegen besitzt p_2 z.B. die unfaire Spur, die aus lauter `a` besteht.

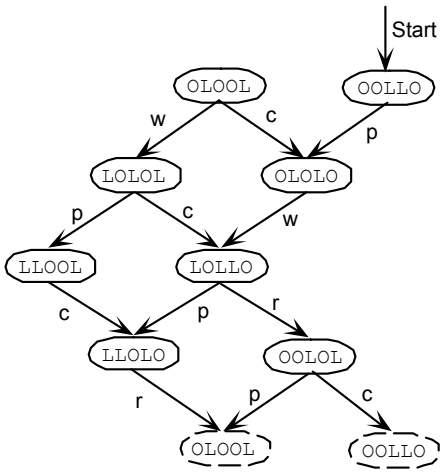
Aufgabe 10 Petrinetze

a) Um alle erreichbaren Belegungen zu ermitteln, sind alle möglichen Schaltvorgänge von der Startbelegung aus zu untersuchen; für die Folgebelegungen ist diese Überlegung fortzusetzen. Dies ist gleichwertig mit dem Aufstellen des zugehörigen Zustandsübergangsautomaten für das Petri-Netz. Boolesche Belegungen stellen wir durch 5-Tupel $b \in \mathbb{B}^5$ mit den Werten für die 5 Stellen dar: $b = \langle b_0\ b_1\ b_2\ b_3\ b_4 \rangle$. $b_i = L$ genau dann, wenn Stelle s_i belegt ist. Wichtig zu beachten ist, dass in Booleschen Petri-Netzen Plätze mit höchstens einer Marke belegt werden können.

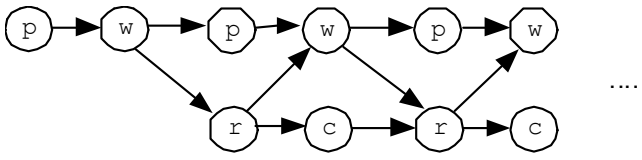


b) Eine Verklemmung ist nicht erreichbar. In jeder Belegung gibt es wenigstens eine Transition, die schalten kann, wie man an dem Automaten leicht sieht. Aus der modifizierten Anfangsbelegung <OOLOO> wird nach den Transitionen p w p eine Verklemmung erreicht, da keine weitere Transition mehr schalten kann.

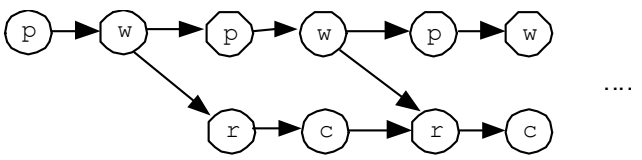
c) Durch das parallele Schalten von p und r existieren nichtsequentielle Abläufe. Ebenso können w und c sowie p und c parallel schalten. Man sieht dies deutlicher, wenn man den Automaten von a) anders anordnet, wobei die Zustände OLOOL und OOLLO doppelt auftreten. Die doppelten Zustände sind zu identifizieren. Man kann sich dazu das Blatt geeignet zusammengerollt denken. Die durch parallele Transitionen bedingten „Rauten“ treten hier deutlich hervor. Es sind vier (!) Stück.



d) Eine unendliche Aktionsstruktur ist ein vollständiger Ablauf eines Petri-Netzes N zur Anfangsbelegung b, wenn jeder endliche Präfix ein Ablauf von N zu b ist. Der rechts angegebene Prozess stellt einen vollständigen Ablauf des Petri-Netzes N dar. Jede (teilweise oder vollständige) Sequentialisierung davon ist natürlich auch ein vollständiger Ablauf von N. Beachte, dass parallel auftretende Ereignisse wie etwa p und r die möglichen sequentiellen Abläufe $p \rightarrow r$ und $r \rightarrow p$ darstellen (Raute im obigen Automatendiagramm). Man erhält diese Struktur am einfachsten, indem man zunächst die beiden Kreise in N separat betrachtet und dann s_0 mit einbezieht.



e) Die Transition w kann nun mehrfach schalten ohne auf eine Transition r warten zu müssen. Der Prozess rechts stellt einen vollständigen Ablauf des Petri-Netzes N′ dar.



f) Da die obigen Prozesse jeweils alle Abläufe darstellen, hat N keine unfairen Abläufe, N′ jedoch schon. (Vgl. die Hausaufgabe)

Aufgabe 11 (H) Präfixe

Ausgehend von der Ereignismenge $E = \{e_i: i \in \mathbb{N}_0\}$ und einer Aktionenmenge A sei ein Prozess $p = (E, \leq, \alpha)$ gegeben. Für $i \in \mathbb{N}_0$ sei $p_i = (E_i, \leq_i, \alpha_i)$ der kleinste Präfix von p, der e_i enthält. Dann gilt insbesondere für alle $i \in \mathbb{N}_0$

$E_i = \{e \in E: e \leq e_i\}$
$\leq_i = \leq|_{E_i \times E_i}$
$\alpha_i = \alpha|_{E_i}$

Wir betrachten den Prozess $p' = (E', \subseteq, \alpha')$ mit $E' = \{E_i: i \in \mathbb{N}_0\}$ und

$\alpha': E' \to A$ gegeben durch $\alpha'(E_i) = \alpha(e_i)$ für $i \in \mathbb{N}_0$

Die Abbildung, die E_i den Wert e_i zuordnet für $i \in \mathbb{N}_0$, ist ein Prozessisomorphismus, denn

- sie ist bijektiv,
- $E_j \subseteq E_k$ gdw. $e_j \in E_k$ gdw. $e_j \leq e_k$ für $j, k \in \mathbb{N}_0$
- $\alpha'(E_i) = \alpha(e_i)$ gilt nach Definition für $i \in \mathbb{N}_0$.

Aufgabe 12 Simulation von booleschen Petri-Netzen in Java

a) Petri-Netzen lassen sich auf unterschiedliche Weise in Java realisieren. Aus Sicht der Objektorientierung ist es nahe liegend, einzelne Stellen und Transitionen jeweils als individuelle Objekte zu realisieren und mit entsprechenden Zugriffsoperationen zu kapseln. Dieser Ansatz wird in den folgenden Teilaufgaben verfolgt.

Eine technisch effizientere Alternative der Realisierung eines booleschen Petri-Netzes mit n Stellen und m Transitionen wäre z.B. die Repräsentation der Transitionen mit zwei Adjazenzmatrizen $In \in \{\text{true, false}\}^{n \times m}$ und $Out \in \{\text{true, false}\}^{n \times m}$ zzgl. der Realisierung der aktuellen Belegung des Netzes mit einem Belegungsvektor $B \in \{\text{true, false}\}^n$.

b)

```
public class Place {

  private boolean token = false;

  public void set(boolean value) {
    token = value;
  }

  public boolean get() {
    return token;
  }

}
```

c)

```
public class Transition {

  private Place[] inPlaces;
  private Place[] outPlaces;

  // Ueberpruefung der Ein-/Ausgabestellen
```

```
  public boolean enabled() {
    for(int i = 0; i < inPlaces.length; i++)
      if(!inPlaces[i].get()) return false;
    for(int i = 0; i < outPlaces.length; i++)
      if(outPlaces[i].get()) return false;
    return true;
  }

  // Transition ausfuehren
  public void fire() {
    if(enabled()) {
      for(int i = 0; i < inPlaces.length; i++)
        inPlaces[i].set(false);
      for(int i = 0; i < outPlaces.length; i++)
        outPlaces[i].set(true);
    } else System.out.println("Tranisition disabled!");
  }

  // Eingabestellen definieren
  public void setIn(Place[] in) {
    inPlaces = in;
  }

  // Ausgabestellen definieren
  public void setOut(Place[] out) {
    outPlaces = out;
  }

}
```

d)

```
public class BPetriNet {

  private Place[] places;
  private Transition[] transitions;

  public void createPlaces(int i) {
    places = new Place[i];
    for(int j = 0; j < places.length; j++)
      places[j] = new Place();
  }

  public Place getPlace(int i) {
    return places[i];
  }

  public void createTransitions(int i) {
    transitions = new Transition[i];
    for(int j = 0; j < transitions.length; j++)
      transitions[j] = new Transition();
  }

  public Transition getTransition(int i) {
    return transitions[i];
  }

  public void simulate() {
    boolean live = true;

    System.out.println(toString());
    while (live) {
      live = false;
      for(int i = 0; i < transitions.length; i++)
          if (transitions[i].enabled())
              {
```

```
                    transitions[i].fire();
                    live = true;
                    System.out.println(toString());
                }
            }
        }

        public String toString() {
            String statePlaces = "";
            for(int i = 0; i < places.length; i++) {
                statePlaces += i+":" + places[i].get() + " ";
            }
            return statePlaces;
        }
    }
```

e)

```
    public class ErzVerb {

        public static void main(String[] args) {

            BPetriNet net = new BPetriNet();
            net.createPlaces(5);
            net.createTransitions(4);

            Place[] inP, inW, inR, inC;
            Place[] outP, outW, outR, outC;

            // Zuordnung der Ein-/Ausgabestellen:
            inP = new Place[1];
            inP[0] = net.getPlace(1);
            outP = new Place[1];
            outP[0] = net.getPlace(0);

            inW = new Place[1];
            inW[0] = net.getPlace(0);
            outW = new Place[2];
            outW[0] = net.getPlace(1);
            outW[1] = net.getPlace(4);

            inR = new Place[2];
            inR[0] = net.getPlace(2);
            inR[1] = net.getPlace(4);
            outR = new Place[1];
            outR[0] = net.getPlace(3);

            inC = new Place[1];
            inC[0] = net.getPlace(3);
            outC = new Place[1];
            outC[0] = net.getPlace(2);

            // Markierung der Stellen im Startzustand
            net.getPlace(1).set(true);
            net.getPlace(2).set(true);

            net.getTransition(0).setIn(inP);
            net.getTransition(0).setOut(outP);
            net.getTransition(1).setIn(inW);
            net.getTransition(1).setOut(outW);
            net.getTransition(2).setIn(inR);
            net.getTransition(2).setOut(outR);
            net.getTransition(3).setIn(inC);
            net.getTransition(3).setOut(outC);

            net.simulate();
        }
    }
```