ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

HS 2009

Kai Lampka
Jasmin Smula
Raphael Eidenbenz

Discrete Event Systems
Solution to Exercise 9

1 Strukturelle Eigenschaften von PN und Token Game

a) Die Prä- und Postmengen einer Transition t sind folgendermassen definiert (siehe Folie 21):

- Prämenge: $\bullet t := \{p \mid (p,t) \in C\}$
- Postmenge: $t\bullet := \{p \mid (t,p) \in C\}$,

die Prä- und Postmengen der Stellen sind analog definiert.

Dementsprechend erhalten wir die folgenden Mengen für das Pertinetz N_1:

$$\begin{array}{rr} \bullet t_2 = \{p_2\}, & t_2\bullet = \{p_3, p_9\} \\ \bullet t_5 = \{p_5, p_9\}, & t_5\bullet = \{p_6\} \\ \bullet t_6 = \{p_6\}, & t_6\bullet = \{p_7\} \\ \bullet p_3 = \{t_2\}, & p_3\bullet = \{t_3\} \\ \bullet p_4 = \{t_3\}, & p_4\bullet = \{t_4\} \\ \bullet p_{10} = \{t_8\}, & p_{10}\bullet = \{t_3\} \end{array}$$

b) Eine Transition ist aktiviert (enabled), wenn alle Stellen in ihrer Prämenge genügend Token enthalten. In diesem Fall, wo alle Kanten ungewichtet sind, genügt ein Token pro Stelle. Wenn t_2 feuert, wird aus jeder Stelle aus der Prämenge von t_2 ein Token konsumiert und in jeder Stelle in der Postmenge von t_2 ein Token erzeugt (siehe Folie 24). Durch das Feuern von t_2 wird also jeweils ein Token auf p_3 und p_9 erzeugt, das Token auf p_2 wird konsumiert. Anschliessend ist t_5 aktiviert, weil sowohl auf p_9 als auch auf p_5 ein Token liegt. t_3 ist hingegen nicht aktiviert, da zwar p_3 ein Token enthält, aber p_{10} nicht.

c) Vor dem Feuern von t_2 gab es zwei Tokens im Netz: Auf p_2 und p_5. Unmittelbar nach dem Feuern von t_2 liegen Tokens auf den Stellen p_3, p_9 und p_5.

d) Ein Token durchläuft den oberen Kreis, bis Transition t_2 feuert. Dann bleibt ein Token auf p_3 liegen und wartet, ein anderes wird in p_9 erzeugt und aktiviert dadurch t_5. Nun kann ein Token durch den unteren Kreis laufen, bis schliesslich t_8 aktiviert wird. Jetzt bleibt ein Token auf p_5 liegen und „wartet“; Ein anderes aktiviert t_3, da ja noch ein Token auf p_3 liegt. Nun läuft wieder durch den oberen Kreis ein Token, bis Transition t_2 aktiviert wird usw.

Es wird also ein Ablauf von zwei Prozessen modelliert, die sich immer gegenseitig abwechseln, beispielsweise die Korrespondenz zweier Brieffreunde, wo immer einer den Brief des anderen erhält, ihn liest, einen Antwortbrief schreibt, diesen abschickt und auf die erneute Antwort wartet.

Der Erreichbarkeitsgraph (auch *reachability graph*) $RG(P,\vec{s}_0)$ zu einem Petrinetz P ist ein 4-Tupel $(\mathbb{S},\mathbb{S}_0,Act,\mathbb{E})$, wobei gilt

- $\mathbb{S}$ ist die Menge der von $\vec{s}_0$ aus erreichbaren Zustände von P
- $\mathbb{S}_0 := \{\vec{s}_0\}$ ist der Anfangszustand des Petrinetzes P
- Act ist die Menge der Transitionslabels
- $\mathbb{E} \subseteq \mathbb{S} \times Act \times \mathbb{S}$ ist die Kantenmenge, so dass gilt $\mathbb{E} = \{(\vec{s},t,\delta(\vec{s},t)) \mid \vec{s} \in \mathbb{S} \wedge t \in T \wedge \vec{s} \triangleright t\}$

(siehe auch Folie 26). Die Zustände des Petrinetzes werden als Vektoren geschrieben, so dass die i-te Position im Vektor die Anzahl der Tokens auf Stelle p_i des Petrinetzes angibt. Wir können also zum Beispiel den Anfangszustand $\vec{s}_0$ des Petrinetzes N_1, wo auf den Stellen p_1 und p_5 jeweils ein Token liegt, schreiben als $\vec{s}_0 = (1,0,0,0,1,0,0,0,0,0)$. Als Erreichbarkeitsgraph ergibt sich:

$$\begin{aligned}
\mathbb{S} = \{ \quad & (1,0,0,0,1,0,0,0,0,0), (0,1,0,0,1,0,0,0,0,0), (0,0,1,0,1,0,0,0,1,0), \\
& (0,0,1,0,0,1,0,0,0,0), (0,0,1,0,0,0,1,0,0,0), (0,0,1,0,0,0,0,1,0,0), \\
& (0,0,1,0,1,0,0,0,0,1), (0,0,0,1,1,0,0,0,0,0) \quad \},
\end{aligned}$$

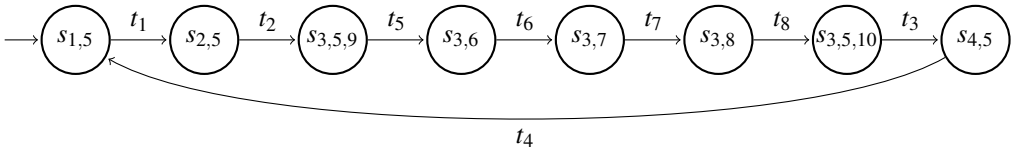
$$\mathbb{S}_0 = \quad \{(1,0,0,0,1,0,0,0,0,0)\},$$

$$Act = \quad \{t_1,t_2,t_3,t_4,t_5,t_6,t_7,t_8,t_9,t_{10}\},$$

$$\begin{aligned}
\mathbb{E} = \{ \quad & \big((1,0,0,0,1,0,0,0,0,0),t_1,(0,1,0,0,1,0,0,0,0,0)\big), \\
& \big((0,1,0,0,1,0,0,0,0,0),t_2,(0,0,1,0,1,0,0,0,1,0)\big), \\
& \big((0,0,1,0,1,0,0,0,1,0),t_5,(0,0,1,0,0,1,0,0,0,0)\big), \\
& \big((0,0,1,0,0,1,0,0,0,0),t_6,(0,0,1,0,0,0,1,0,0,0)\big), \\
& \big((0,0,1,0,0,0,1,0,0,0),t_7,(0,0,1,0,0,0,0,1,0,0)\big), \\
& \big((0,0,1,0,0,0,0,1,0,0),t_8,(0,0,1,0,1,0,0,0,0,1)\big), \\
& \big((0,0,1,0,1,0,0,0,0,1),t_3,(0,0,0,1,1,0,0,0,0,0)\big), \\
& \big((0,0,0,1,1,0,0,0,0,0),t_4,(1,0,0,0,1,0,0,0,0,0)\big) \quad \}.
\end{aligned}$$

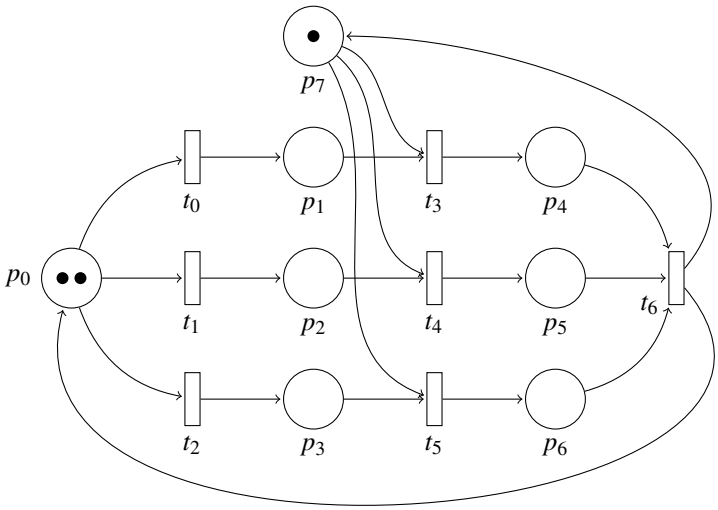
Der besseren Lesbarkeit halber benennen wir die Zuständen so, dass im Index die Stellen stehen, die in diesem Zustand ein Token enthalten, also zum Beispiel $\vec{s}_0 = (1,0,0,0,1,0,0,0,0,0) = s_{1,5}$.

Dann können wir den Erreichbarkeitsgraphen auch folgendermassen schreiben:



2 Carsharing

Ein Petrinetz, das die Situation widergibt, kann ganz unterschiedlich aussehen, je nachdem, wieviele Informationen man mit dem Netz modellieren möchte. Folgendes Petrinetz enthält alle beschriebenen Sachverhalte:

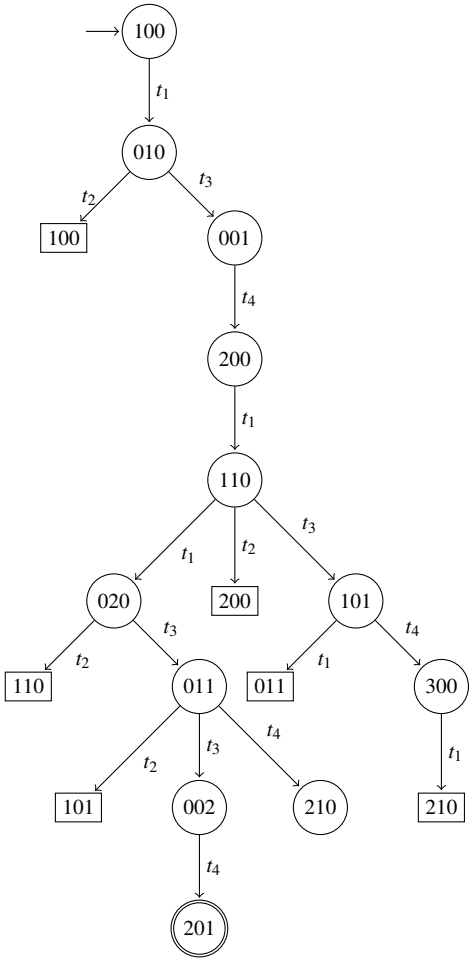


Das Petrinetz funktioniert folgendermassen: Im Anfangszustand liegen beide Schlüssel an ihrem Platz (2 Tokens auf p_0) und das Auto steht vor dem Haus (ein Token auf p_7). Michael, Johannes oder Christoph können sich einen Schlüssel nehmen (t_0, t_1, t_2) und ihn dann bei sich tragen (p_1, p_2, p_3). Wenn einer von ihnen einen Schlüssel bei sich trägt und das Auto vor der Tür steht, kann er losfahren (t_3, t_4, t_5). Dabei wird sein Schlüsseltoken und das Autotoken konsumiert, und ein neues wird auf p_4, p_5 bzw. p_6 erzeugt, wodurch angezeigt wird, dass derjenige unterwegs ist. Sobald er zurückkommt (t_6), stellt er das Auto wieder ab und legt den Schlüssel zurück an seinen Platz, was dadurch modelliert wird, dass auf p_7 eine neues Autotoken erzeugt wird und auf p_0 ein neues Schlüsseltoken.

Freilich könnte man das Modell noch weiter spezifizieren, indem man z.B. noch die Möglichkeit miteinbezieht, dass jemand den Schlüssel wieder ans Brett hängt bevor er das Auto benützt, oder auch, dass jemand seine Mitbewohner nach dem Schlüssel fragt, wenn keiner am Brett hängt etc.

3 Erreichbarkeitsanalyse

a) Der aufgebaute Erreichbarkeitsgraph sieht folgendermassen aus:



Die Zustände sind durch die Anzahl Tokens in den entsprechenden Stellen codiert. In Zustand (200) bspw. befinden sich 2 Tokens in $P1$, $P2$ und $P3$ sind leer. Bereits besuchte Zustände werden als Rechtecke dargestellt.

Nach 5 Schritten entlang des Erreichbarkeitsbaumes enthält $\mathbb{S}_{tmp}$ die Zustände (020) und (101). Im nächsten Schritt wird (020) rausgenommen und (011) eingefügt. Weil $\mathbb{S}_{tmp}$ FIFO ist, wird im darauffolgenden Schritt (101), der Geschwisterknoten von (020), bearbeitet und nicht etwa (011), das Kind von (020). Es handelt sich also um eine Breitensuche.

b) Mit einer LIFO-Liste macht der Erreichbarkeitsalgorithmus eine Tiefensuche. Das bedeutet z.B., dass der Teilbaum an (020) zuerst komplett angeschaut wird, bevor der Teilbaum an (101) erkundet wird.

c) Petrinetze können unendliche Erreichbarkeitsgraphen beschreiben, z.B. N_2 mit $K \geq 2$. Falls der gefragte Zustand in einem solchen Petrinetz tatsächlich erreichbar ist, wird es der Erreichbarkeitsalgorithmus zwar irgendwann herausfinden, falls er aber nicht erreichbar ist, wird der Algorithmus das nie mit Sicherheit sagen können. (Vgl. Halteproblem)

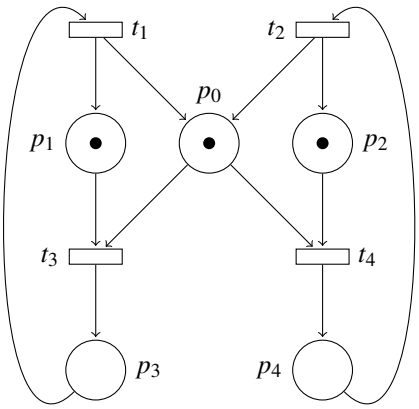
d) Wenn das Gleichungssystem $\mathbf{A} \cdot \vec{f} = \vec{s} - \vec{s_0}$ keine Lösung hat, wissen wir, dass Zustand $\vec{s}$ nicht vom Initialzustand $\vec{s_0}$ erreicht werden kann.

$$\mathbf{A} \cdot \vec{f} = \begin{pmatrix} -1 & 1 & 0 & 2 \\ 1 & -1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix} = \begin{pmatrix} 100 \\ 99 \\ 4 \end{pmatrix} = \vec{s} - \vec{s_0}$$

Dieses Gleichungssystem ist aber erfüllbar. Somit nützt es uns leider nicht viel, weil wir daraus nicht folgern können, dass $\vec{s}$ erreichbar ist. Um zu beweisen, dass $\vec{s}$ erreichbar ist müssen wir eine Feuersequenz angeben, mit der man von $\vec{s_0}$ nach $\vec{s}$ kommt. Hierbei könnte uns die Lösung des Gleichungssystems Hinweise geben. Bspw. wissen wir aus der letzten Zeile, dass $f_3 = f_4 + 4$, also t_3 in der gesuchten Feuersequenz 4 mal mehr gefeuert wird als t_4. Über die Feuer-Reihenfolge aber sagt $\vec{f}$ nichts aus. Schaut man sich das Petrinetz an, so sieht man, dass ausgehend von $\vec{s_0}$ die Anzahl Tokens in $P1$ jeweils um eins erhöht werden kann, wenn man nacheinander t_1, t_3 und t_4 feuert. Wiederholt man das 203 mal, gelangt man in den Zustand $(204, 0, 0)$. Nun feuere t_1 103 mal, anschliessend t_3 4 mal und das System befindet sich in Zustand $\vec{s}$. Somit haben wir die Erreichbarkeit von $\vec{s}$ bewiesen.

4 Gegenseitiger Ausschluss

a) Wir fügen für jeden Prozess zwei Stellen ein (p_1, p_2, p_3 und p_4), die den Prozess jeweils während seiner normalen Programmausführung (p_1, p_2) und in seinem kritischen Abschnitt (p_3, p_4) darstellen. Für jeden Prozess gibt es ein Token, das angibt, in welchem Programmabschnitt sich der Prozess gerade befindet. Ausserdem fügen wir eine Stelle p_0 ein, an der ein weiteres Token liegt, wenn die Semaphore $\neq 0$ ist. Wir müssen sicherstellen, dass ein Prozess nur dann in seinen kritischen Programmabschnitt wechseln kann, wenn das Semaphorentoken auf p_0 liegt. Das Petrinetz, das sich ergibt, sieht dann wie folgt aus:



Angenommen, zu Anfang sind die beiden Prozesse in einem unkritischen Abschnitt. Wenn nun einer der beiden in seinen kritischen Abschnitt wechseln will, kann er das nur tun, wenn sich ein Token auf p_0 befindet. In diesem Fall wird dieses Token beim Eintritt in den kritischen Abschnitt konsumiert. Erst wenn der Prozess den kritischen Abschnitt wieder verlässt, wird wieder ein Semaphorentoken auf p_0 erzeugt. So schliessen sich die beiden Prozesse gegenseitig vom zeitgleichen Zugriff auf den kritischen Abschnitt aus.

b) Es darf zu jedem Zeitpunkt auf höchstens einer der beiden Stellen p_3 und p_4 ein Token liegen. Für die Zustände im Erreichbarkeitsgraphen muss also gelten:

$$(\vec{s}[p_3] = 1 \Rightarrow \vec{s}[p_4] = 0) \ \wedge \ (\vec{s}[p_4] = 1 \Rightarrow \vec{s}[p_3] = 0) \quad \forall \vec{s} \in \mathbb{S}. \tag{1}$$

Eine entsprechender Anfangszustand wäre zum Beispiel $\vec{s}_0 = (1,1,1,0,0)$, wo sich beide Prozesse in einem unkritischen Abschnitt ihres Programms befinden. Die einzigen beiden anderen Zustände im Erreichbarkeitsgraphen sind dann $\vec{s}_1 = (0,0,1,1,0)$ und $\vec{s}_2 = (0,1,0,0,1)$. Dann gilt für alle Zustände die oben stehende Forderung (1).

5 Euklidscher Algorithmus

Siehe Musterlösung zu Exercise 10.

6 Boundedness und Deadlock Freeness

Ein Petrinetz ist k-beschränkt, wenn es keine Feuersequenz gibt, die die Anzahl Tokens in einer Stelle über k anwachsen lässt. Im Petrinetz N_2 sieht man schnell, dass das Netz 1-beschränkt ist, wenn $K \leq 1$ ist. Dies ist deshalb so, weil dann keine Transition die Gesamtzahl der Tokens erhöht. Da im Initialzustand nur ein Token vorhanden ist, wird es also nie mehr als ein Token in einer Stelle haben. Ist $k \geq 2$, kann die Anzahl Tokens in $P1$ durch wiederholtes Feuern der Sequenz t_1, t_3, t_4 unendlich anwachsen. Das Petrinetz ist also unbeschränkt.

Ein Petrinetz ist deadlockfrei, falls man durch keine Feuersequenz in einen Zustand gelangt, in dem keine Transition aktiviert ist. Ist K gleich 0, so ist N_2 nicht deadlockfrei. Die Sequenz t_1, t_3, t_4 führt nämlich dazu, dass kein Token mehr im Petrinetz vorhanden ist und somit keine Transition mehr aktiviert ist. Für $K \geq 1$ hingegen kann nie ein Deadlock entstehen.