Lehrstuhl für Connected Mobility
School of Computation, Information and Technology
Technische Universität München

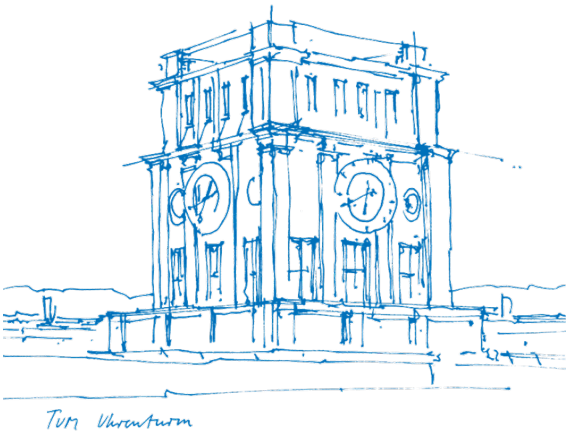
Grundlagen: Betriebssysteme und Systemsoftware

Tutorübung

Mario Delic

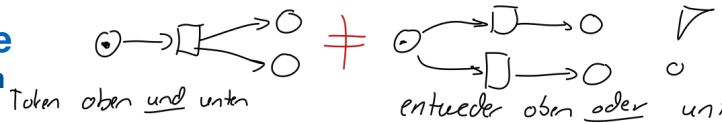
Lehrstuhl für Connected Mobility
School of Computation, Information and Technology
Technische Universität München

Übungswoche 6



Petrinetze

Formalitäten



Petrinetz **P = (S,T,F)** is ein Tripel aus Mengen von Stellen S, Transitionen T und Kanten F.

Stellen: Modellieren passive Einheiten (Speicherzellen o.ä.). Dargestellt durch Kreise.

Transitionen: Modellieren aktive Einheiten (Prozesse o.ä.). Dargestellt durch Rechtecke.

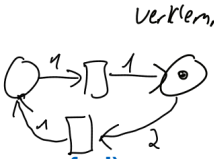
Kanten: $F \subseteq (S \times T) \cup (T \times S) \rightarrow$ Kanten sind gerichtet und führen entweder von einer Stelle S zu einer Transition T, oder von einer T zu einer S. $\rightarrow (S \times S), (T \times T) \notin F$

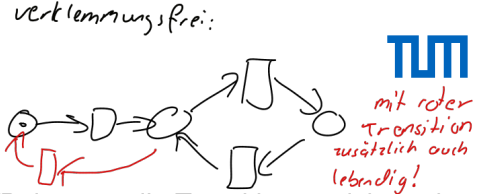
Kapazität: $c : S \rightarrow \mathbb{N}_0 \cup \{\infty\}$. c(s) = Maximale Anzahl an Tokens, die Stelle s aufnehmen kann.

Belegung: $M : S \rightarrow \mathbb{N}_0 \cup \{\infty\}$. M(s) = Anzahl der Tokens in Stelle s.

Kantengewicht: $w : F \rightarrow \mathbb{N}_0$. w(s,t) = Anzahl der Tokens, die die Kante bzw. Transition t aus der Stelle s erwartet. w(t,s) = Anzahl der Tokens, die die Transition t in Stelle s reinschreibt.

Petrinetze
Eigentschaften





Verklemmt ($\leftrightarrow$ Verklemmungsfrei)

Ein Petrinetz ist verklemmt, wenn ab einem Zeitpunkt/Belegung alle Transitionen nicht mehr schalten können.

$\hookrightarrow \exists\, M\ \forall\, t \in T$: t kann nie wieder Schalten ($\leftrightarrow \forall\, M\ \exists\, t \in T$: t kann Schalten)

nicht Lebendig ($\leftrightarrow$ Lebendig)

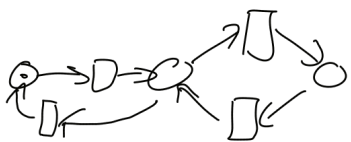
Ein Petrinetz ist nicht lebendig, wenn ab einem Zeitpunkt/Belegung irgendeine Transition nie wieder schalten können wird.

$\hookrightarrow \exists\, M\ \exists\, t \in T$: t kann nie wieder Schalten ($\leftrightarrow \forall\, M\ \forall\, t \in T$: t kann in Zukunft wieder Schalten)

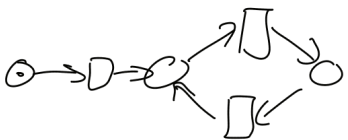
Verklemmt $\Longrightarrow$ **nicht Lebendig** und **Lebendig** $\Longrightarrow$ **Verklemmungsfrei**

Fair ($\leftrightarrow$ Unfair)

Ein Petrinetz ist fair, wenn keine Transition verhungern kann.
(Erinnerung Verhungern: Ausführung zwar möglich, könnte aber unendlich lange hinausgezögert/vermieden werden $\rightarrow$ Transition t_x ist schaltbereit, aber es schaltet immer nur $t_y \hookrightarrow t_x$ kann verhungern.)



- verklemmungsfrei
- lebendig
- unfair



- verklemmungsfrei
- nicht lebendig
- fair

Petrinetze
Funktionalität

- Eine Transition kann nur schalten, wenn alle ihre Vor- und Nachbedingungen erfüllbar sind!
$\rightarrow$ Die Stellen aller eingehenden Kanten haben genügend Tokens **&&** die Stellen aller ausgehenden Kanten haben genug freien Platz.
- Es schaltet immer nur eine Transition zu einer Zeit $\rightarrow$ Mehrere Transitionen schalten nie gleichzeitig.
- Sonderfall Bool'sches Netz: Alle Kapazitäten und Kantengewichte = 1!

- Synchronisation in Petrinetzen:
Kapazitäten und Belegungen reichen i.d.R. nicht aus um ein bestehendes Netz effektiv zu Synchronisieren.
Stattdessen: 'Deklarieren' einer extra Mutex/Semaphor-Stelle + Verbinden mit allen Transitionen die in den den kritischen Bereich rein/raus führen. Anschließendes Synchronisieren durch Anpassung der Kapazität/Belegung und der Kantenrichtungen.

Aufgabe 2

Objekte

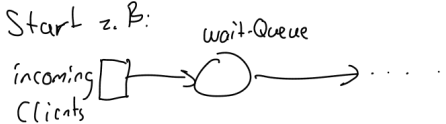
Modelliert Zustände/Orte als **Stelle (Kreis)** und Aktionen/Tätigkeiten als **Transition (Rechteck)**.

Könnte Zustand haben/Objekt sein

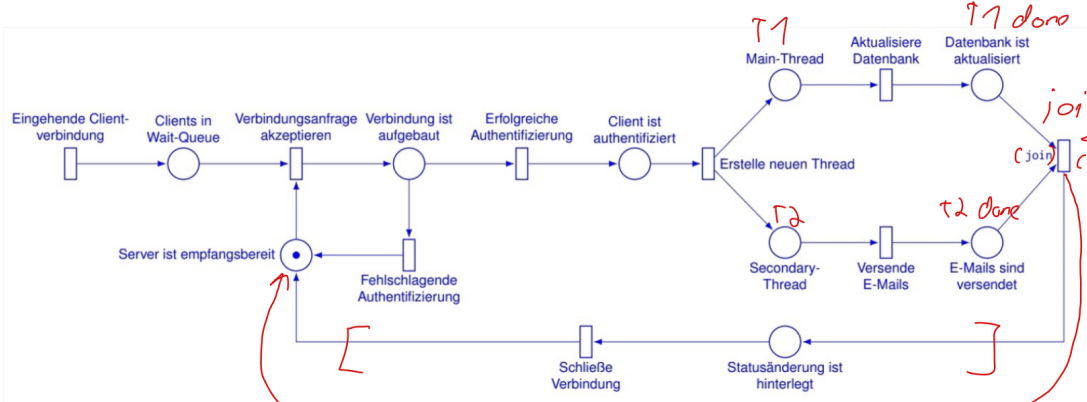
- Wartenschlange
- Client
- Server
- Threads (Main/Second)
- Datenbank
- Emails
- Verbindungszustand

Mögliche Aktionen:

- Verbindugsanfrage
- Verbindungsakzeptierung
- Cient-Authentifizierung
- Client-Ablehnung
- Thread-Erstellung
- Emails senden
- Datenbank updaten
- Thread-join
- Verbindung schließen



Aufgabe 2

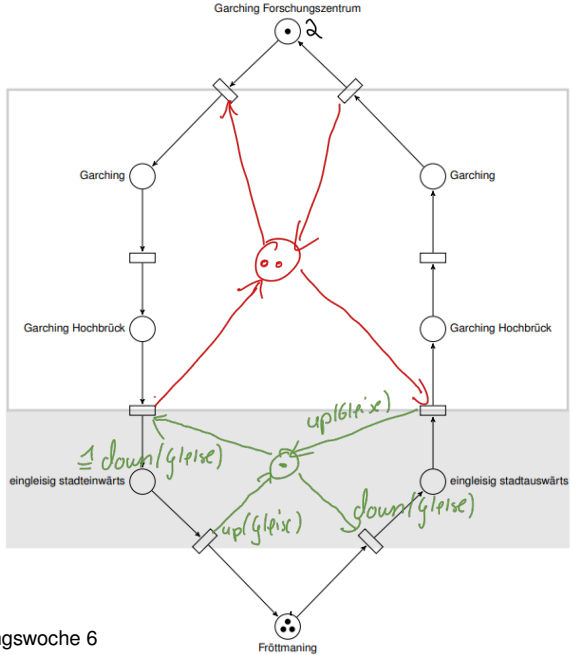

Eingehende Client-verbindung
Clients in Wait-Queue
Verbindungsanfrage akzeptieren
Verbindung ist aufgebaut
Erfolgreiche Authentifizierung
Client ist authentifiziert
Erstelle neuen Thread
T1
Main-Thread
Aktualisiere Datenbank
Datenbank ist aktualisiert
T1 done
T2
Secondary-Thread
Versende E-Mails
E-Mails sind versendet
T2 done
join
join & close
Server ist empfangsbereit
Fehlschlagende Authentifizierung
Schließe Verbindung
Statusänderung ist hinterlegt

Petrinetze
Synchronisation

Kapazitäten und Belegungen reichen i.d.R. nicht aus um ein bestehendes Netz effektiv zu Synchronisieren. Stattdessen:

1. ’Deklarieren’ einer extra Mutex/Semaphor-Stelle
2. Verbinden mit allen Transitionen die in den den kritischen Bereich rein/raus führen.
3. Anschließendes Synchronisieren durch Anpassung der Kapazität/Belegung und der Kantenrichtungen.

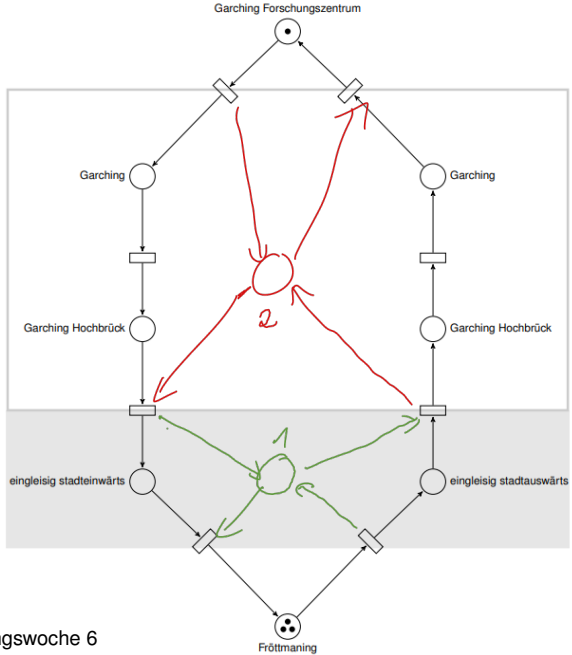
Aufgabe 3



3b)
max. 2

3a)
max. 1

3c)


Garching Forschungszentrum
Garching
Garching
Garching Hochbrück
Garching Hochbrück
2
1
eingleisig stadteinwärts
eingleisig stadtauswärts
Fröttmaning

Erreichbarkeitsgraph

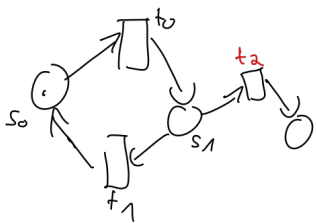
Erreichbarkeitsgraphen sind **Zustandsautomaten**, d.h. jeder Knoten modelliert einen **Belegungszustand** des Petrinetzes.

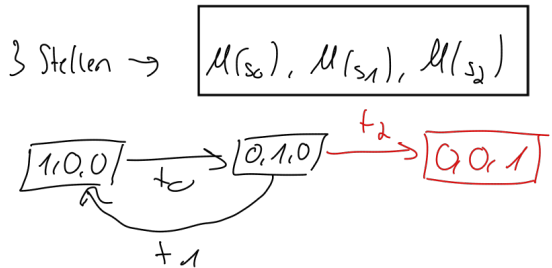
- Zustände: Tupel $(M(s_0), ..., M(s_n))$ für alle n Stellen des Petrinetzes.

- Übergangsfunktion: Kanten t_0,t_1,t_2... welche schaltende Transitionen darstellen.
$\hookrightarrow$ ÜF ist partiell, d.h. eine Kante t_x aus einem Zustand existiert nur, wenn die Transition t_x aus der Belegung des Zustands heraus schalten kann.

- Ein Blatt bzw. ein Zustand aus dem keine Kante herausführt bezeichnet man auch als Fangzustand (eng.: trap state). In diesem Zustand **verklemmt** das Petrinetz.

Erreichbarkeitsgraph
Beispiele





Petrinetze
Synchronisation

Kapazitäten und Belegungen reichen i.d.R. nicht aus um ein bestehendes Netz effektiv zu Synchronisieren. Stattdessen:
1. 'Deklarieren' einer extra Mutex/Semaphor-Stelle
2. Verbinden mit allen Transitionen die in den den kritischen Bereich rein/raus führen.
3. Anschließendes Synchronisieren durch Anpassung der Kapazität/Belegung und der Kantenrichtungen.

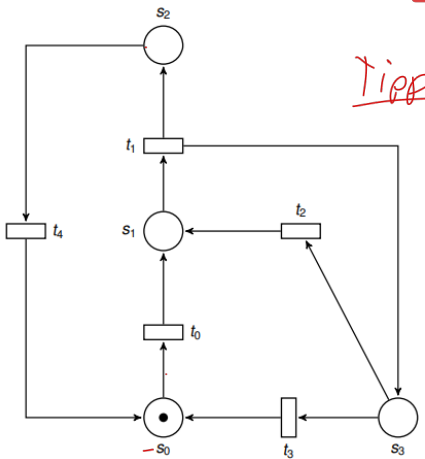
Deadlock - was tun?:
1. Betrachte den Erreichbarkeitsgraphen und die Fangzustände.
2. Führe neue Transition(en) ein, die aus dem Fangzustand in einen existierenden Zustand führen.
3. Setze diese Transition(en) im Petrinetz um (ggf. mit einer neuen Hilfsstelle).

Aufgabe 4

a) Geben Sie den Erreichbarkeitsgraphen (zu folgendem boolschen Netz) an.

4 Stellen

(s_0, s_1, s_2, s_3)



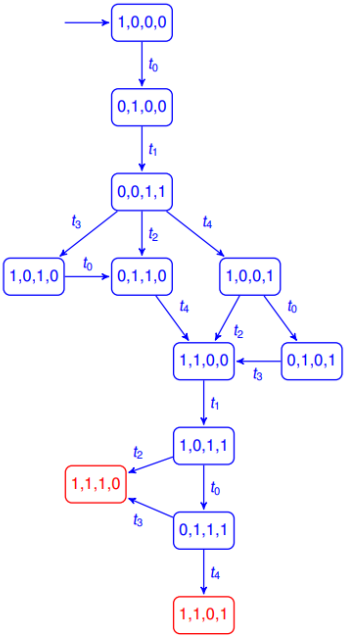
Tipp: Genau 12 Zustände!

(1, 0, 0, 0)

↓ t_0

(0, 1, 0, 0)

↓ t_1 → (0, 0, 1, 1) usw. ...


1,0,0,0
t_0
0,1,0,0
t_1
0,0,1,1
t_3
t_2
t_4
1,0,1,0
t_0
0,1,1,0
1,0,0,1
t_4
t_2
t_0
1,1,0,0
t_3
0,1,0,1
t_1
1,0,1,1
t_2
1,1,1,0
t_0
t_3
0,1,1,1
t_4
1,1,0,1

Aufgabe 4

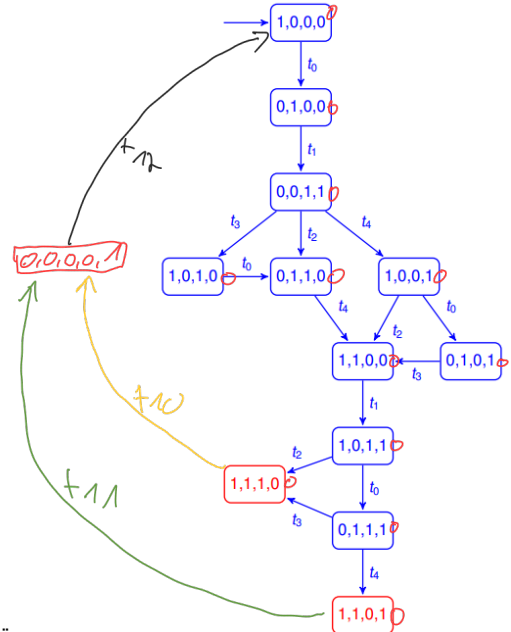
b) Ist das Netz verklemmungsfrei? Argumentieren Sie anhand des Erreichbarkeitsgraphen.

Aufgabe 4

b) Ist das Netz verklemmungsfrei? Argumentieren Sie anhand des Erreichbarkeitsgraphen.

Nein. Falls der Erreichbarkeitsgraph einen Fangzustand besitzt, so ist das Petrinetz nicht verklemmungsfrei.

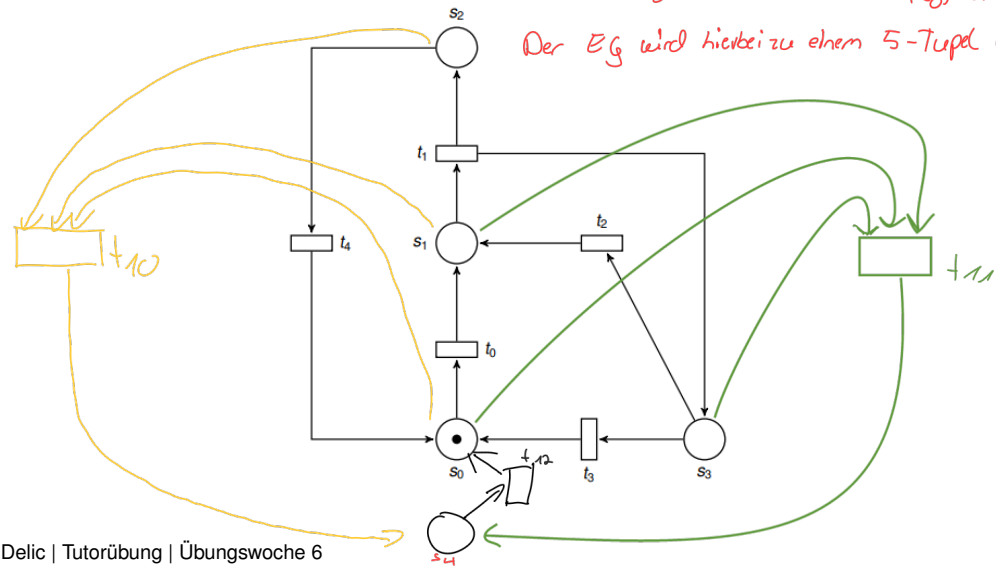
c) Beseitigen Sie die ggf. vorhandene Verklemmung durch Einführen einer neuen Stelle und dazugehörigen Transitionen. Können Sie anhand des Erreichbarkeitsgraphen ableiten, welche Transitionen hinzugefügt werden müssen?


1,0,0,0
t0
0,1,0,0
t1
0,0,1,1
t3
t2
t4
1,0,1,0
t0
0,1,1,0
1,0,0,1
t4
t2
t0
1,1,0,0
t3
0,1,0,1
t1
1,0,1,1
t2
1,1,1,0
t0
t3
0,1,1,1
t4
1,1,0,1
0,0,0,0,1
t12
t10
t11

Aufgabe 4

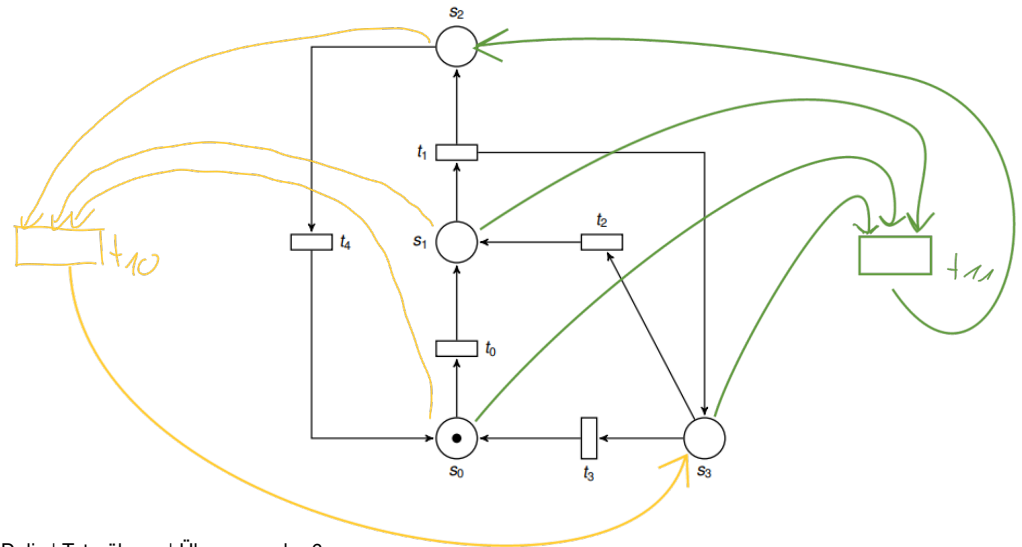
Die Hilfsstelle ist notwendig, da man sonst nicht in den Startzustand kann, da $M(s_0)=1$ und auch $c(s_0)=1$!

Der EG wird hierbei zu einem 5-Tupel wegen s_4!



Aufgabe 4

Alternative Deadlock-Lösung ohne Hilfsstelle



Neuer Eg ohne Hilfsstelle!

Beachte die nun neuen Zustände, die vorher nicht möglich waren!