UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Software Engineering im Wintersemester 2021/2022

Prof. Dr. Martin Leucker, Malte Schmitz, Stefan Benox, Julian Schulz, Benedikt Stepanek, Friederike Weilbeer, Tom Wetterich

Übungszettel 3 (Lösungsvorschlag)

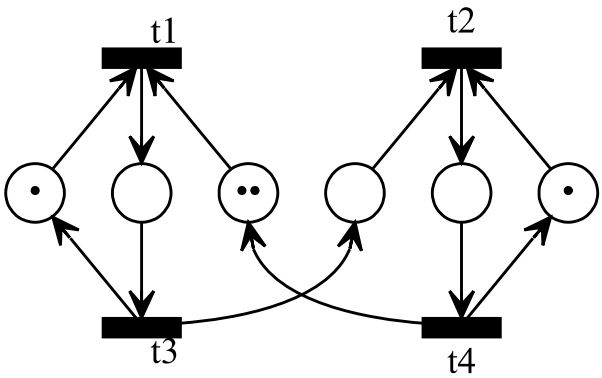
24.11.2021

Abgabe bis Donnerstag, 11. November um 23:59 Uhr online im Moodle.

Aufgabe 3.1: Erreichbarkeitsgraph eines Petri-Netzes

4 Punkte, mittel

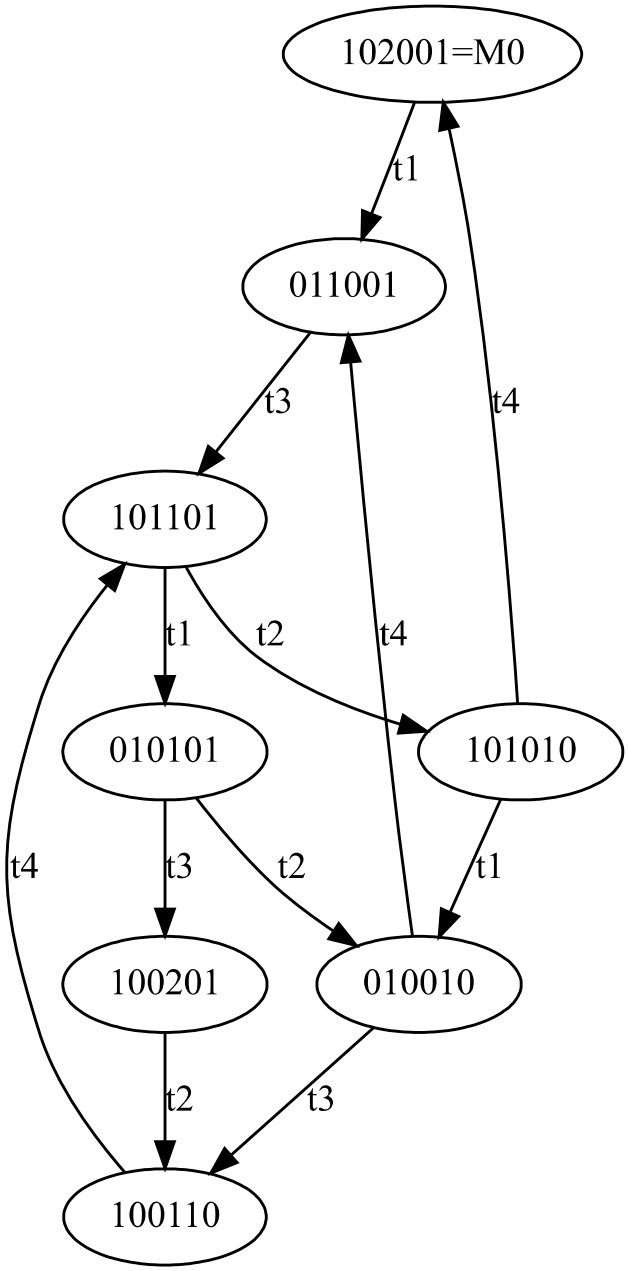
Betrachten Sie folgendes Petri-Netz:



▶ Quelltext des Diagramms

1. Zeichnen Sie den Erreichbarkeitsgraphen des Petri-Netzes. Verwenden Sie dabei zur Darstellung der Knoten die Vektorisierung der Markierungen. Zum Beispiel für die initiale Markierung M_0 ergibt sich somit der Vektor $(1, 0, 2, 0, 0, 1)$, den Sie vereinfacht als 102001 notieren können.

▼ **Lösungsvorschlag**



▶ Quelltext des Diagramms

2. Besitzt das Petri-Netz eine Verklemmung? Begründen Sie Ihre Antwort kurz.

▼ **Lösungsvorschlag**

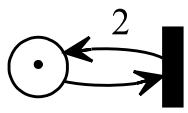
Nein, denn alle Knoten im Erreichbarkeitsgraphen haben ausgehende Kanten. Eine Verklemmung ist aber gerade als eine initial erreichbar Konfiguration, also ein Knoten im Erreichbarkeitsgraphen, definiert, unter der keine Transition aktiviert ist, also ein Knoten ohne ausgehende Kante.

3. Ist das Petri-Netz lebendig? Begründen Sie Ihre Antwort kurz.

▼ **Lösungsvorschlag**

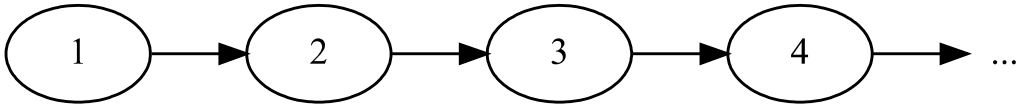
Ja, das Petri-Netz ist lebendig. Ein Petri-Netz ist lebendig, wenn für jede initial erreichbare Markierung und für jede Transition t immer eine Markierung erreichbar ist, unter der t aktiviert ist. Am Erreichbarkeitsgraphen kann man sehen, dass von der initialen Konfiguration aus für jede Transition eine Konfigurationen erreicht werden kann, unter der diese Transition aktiviert ist. Weiter ist der Erreichbarkeitsgraph stark zusammenhängend, das heißt alle Konfigurationen sind untereinander erreichbar. Der Erreichbarkeitsgraph ist der Graph aller initial erreichbarer Konfigurationen. Also kann in diesem Petri-Netz jede initial erreichbare Konfiguration immer wieder erreicht werden.

Dieser Zusammenhang ist übrigens umgekehrt nicht gegeben: Der Erreichbarkeitsgraph eines lebendigen Petri-Netzes muss nicht stark zusammenhängend sein. Als Gegenbeispiel sei folgendes Petri-Netz gegeben:



▶ Quelltext des Diagramms

Der zugehörige Erreichbarkeitsgraph ist unendlich und nicht stark zusammenhängend, da jede einmal erreichte Konfiguration nie wieder erreicht werden kann, denn die Anzahl Token kann nur größer werden:



▶ Quelltext des Diagramms

Trotzdem ist dieses Petri-Netz lebendig, denn die einzige vorhandene Transition ist jederzeit aktiviert, sodass trivial immer eine Konfiguration erreichbar ist, unter der diese Transition aktiviert ist.

4. Ist es möglich, dass eine Stelle zu einem Zeitpunkt drei Marken besitzt? Begründen Sie kurz.

▼ **Lösungsvorschlag**

Nein, denn es gibt im Erreichbarkeitsgraphen keinen Knoten mit einer Zahl größer 2 im Vektor.

Aufgabe 3.2: Entwurf eines Petri-Netzes

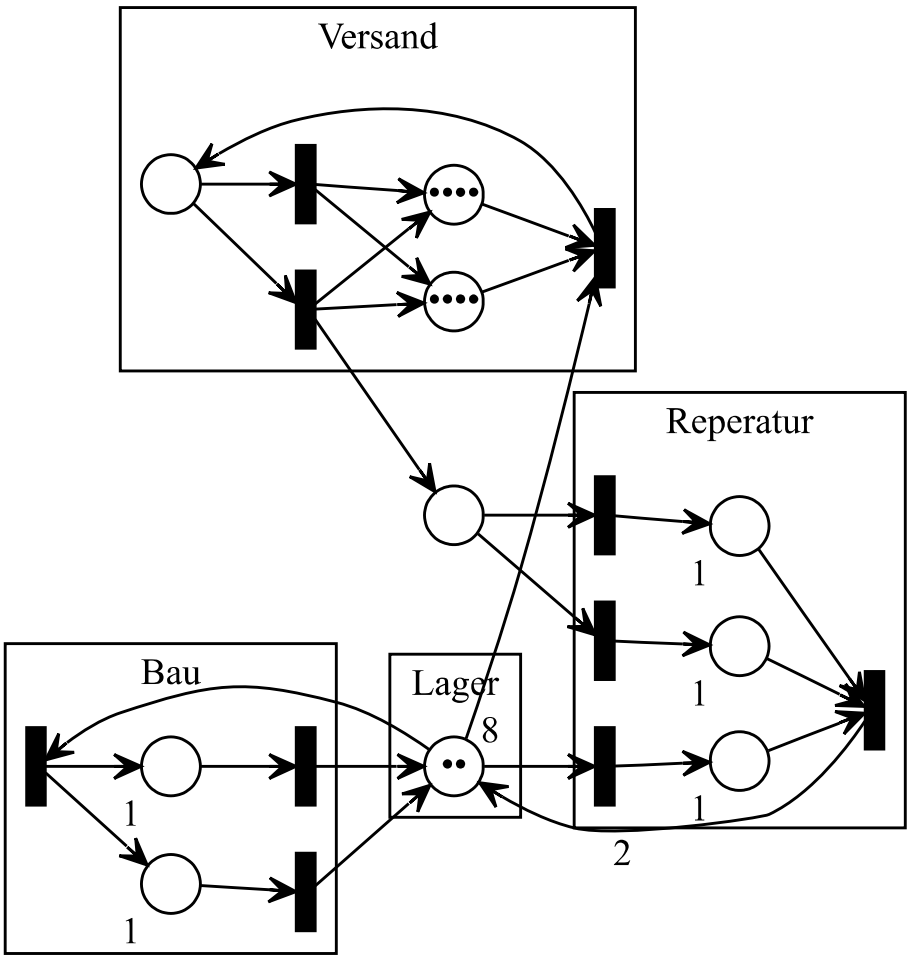
4 Punkte, mittel

Folgender Text beschreibt die Abläufe eines Fahrradhändlers:

> Die Fahrräder werden erstellt, indem eines aus dem Lager genommen und dieses kopiert wird. Aus diesem einen Fahrrad werden dann also zwei. Diese beiden werden kurz in zwei Zwischenlagern einzeln gelagert und dann in das Lager übernommen. Diese Prozedur kann erst wieder ausgeführt werden, sobald die Zwischenlager leer sind. Das Lager fasst maximal acht Fahrräder und die ersten beiden Fahrräder sind wie durch ein Wunder in das Lager gekommen und initial alleine dort.
>
> Außerdem gibt es zwei Pferdeherden mit jeweils vier Pferden. Will nun der Fahrradverkäufer oder einer seiner Helfer ein Fahrrad versenden, so wird ein Fahrrad aus dem Lager und jeweils aus jeder Pferdeherde genau ein Pferd, die das Fahrrad transportieren, genommen. Danach gilt das Fahrrad als versandt. Nun gibt es zwei Fälle: Zum einen kann der böse Gegenspieler des Fahrradverkäufers das Fahrrad stehlen. Das Fahrrad ist dann im Besitz des Gegenspielers und die Pferde kommen wieder in ihre Herden. Andernfalls gilt das Fahrrad als ausgeliefert und die Pferde kommen in ihre Herden zurück.
>
> Zuletzt können ausgelieferte Fahrräder, die defekt sind, noch wieder repariert werden. Dies funktioniert wie folgt: Ist ein Fahrrad defekt, so landet es wie durch Zauberhand an einem der beiden Reparaturplätze, die jeweils genau ein Fahrrad fassen. Außerdem gibt es noch eine Ersatzteilstelle mit Kapazität 1, die mit Fahrrädern aus dem Lager gefüllt werden kann. Sind die Reparaturplätze und die Ersatzteilstelle gefüllt, so können aus den zwei defekten und dem einen nicht-defekten Fahrrad wieder zwei funktionierende erstellt und ins Lager überführt werden.

1. Modellieren Sie den dargestellten Zusammengang als Petri-Netz. Machen Sie dabei deutlich, was in welchem Teil des Petri-Netzes modelliert wurde. (2 Punkte)

▼ Lösungsvorschlag



▶ Quelltext des Diagramms

2. Besitzt Ihr Petri-Netz eine Verklemmung? Begründen Sie Ihre Antwort kurz. (1 Punkt)

▼ Lösungsvorschlag

Ja, zum Beispiel wenn die ersten beiden Fahrräder direkt von dem bösen Gegenspieler geklaut werden.

3. Ist Ihr Petri-Netz lebendig? Begründen Sie Ihre Antwort kurz. (1 Punkt)

▼ Lösungsvorschlag

Nein, weil es eine Verklemmung gibt.

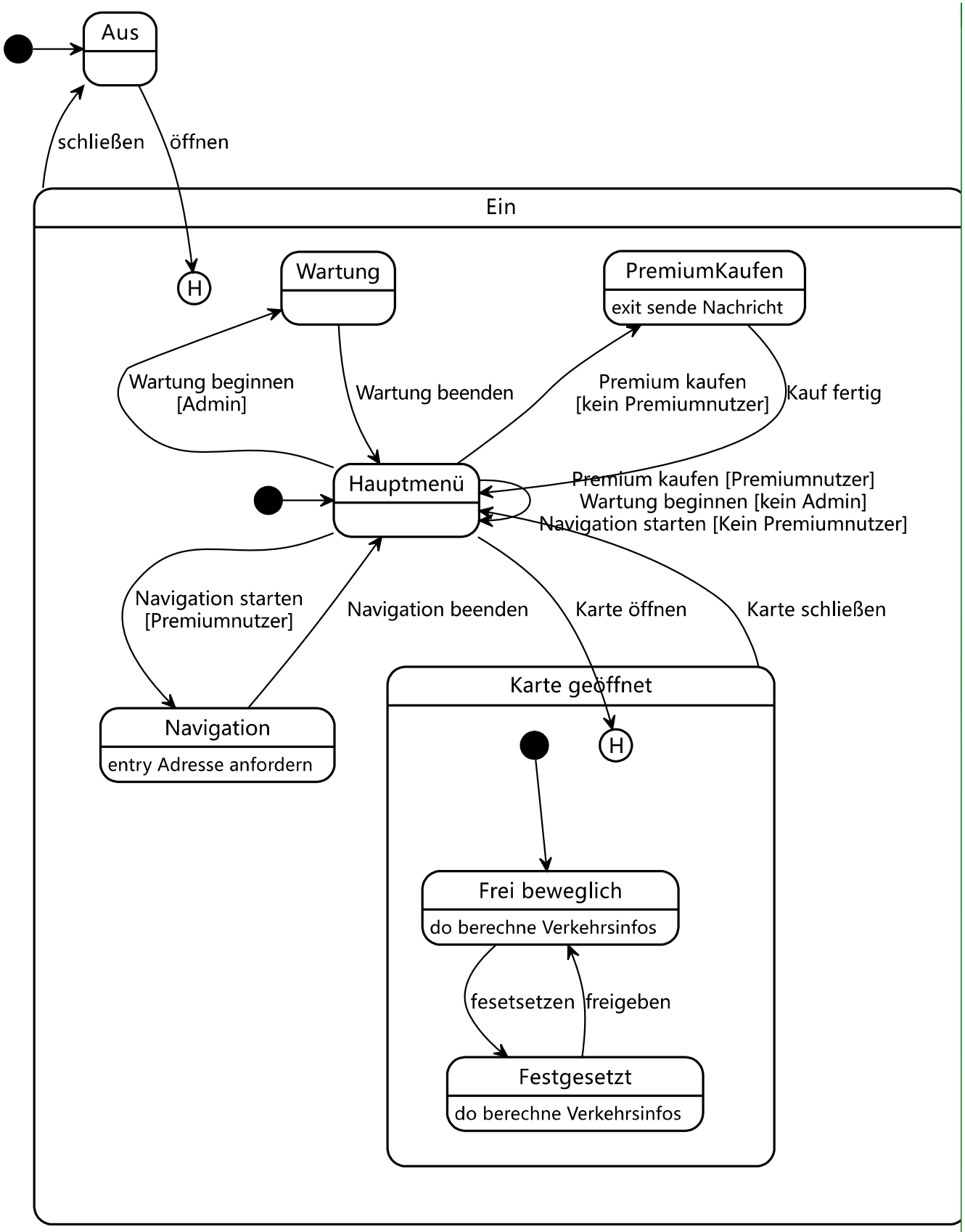
Aufgabe 3.3: UML-Zustandsdiagramm

4 Punkte, mittel

In dieser Aufgabe soll ein UML-Zustandsdiagramm für eine Straßen-Navigationsapp erstellt werden. Die App beginnt immer im Zustand *Aus*. Des Weiteren gilt folgendes:

- Wird die App geöffnet, so ist sie im Zustand *Ein*.
- Nach dem Einschalten öffnet sich das *Hauptmenü* der App.
- Vom *Hauptmenü* aus kann der Benutzer die *Karte öffnen*. Zu Beginn befindet sich die Karte irgendwo und der Benutzer kann die Karte frei bewegen. Der Benutzer kann sich allerdings entscheiden, die Karte auf seiner aktuellen Position festzusetzen und dann auch wieder freizugeben, so dass er wieder frei scrollen kann. Solange die *Karte offen* ist, kann der Benutzer zwischen den beiden Modi *frei* und *festgesetzt* beliebig wechseln. Der Benutzer kann die Karte jederzeit schließen und zum *Hauptmenü* zurückgehen. Wenn die Karte wieder geöffnet wird, wechselt das System wieder in den selben Modus (*frei* oder *festgesetzt*) von dem aus die Karte verlassen wurde. Während die *Karte offen* ist, werden die ganze Zeit über aktuelle Verkehrsinformationen berechnet.
- Vom *Hauptmenü* aus kann der Benutzer in einen Zustand *PremiumKaufen* wechseln, wenn er kein Premiumnutzer ist. Ist er mit dem Kauf fertig, wird beim Verlassen des Zustandes eine Nachricht an den Benutzer gesendet und die App wechselt wieder ins *Hauptmenü* zurück.
- Als Premiumnutzer kann der Nutzer eine *Navigation* starten. Beim Erreichen des Zustandes fordert das System eine Adresse an. Der Benutzer kann die *Navigation* jederzeit beenden und zum *Hauptmenü* zurückkehren.
- Wenn der Benutzer ein Administrator ist, kann er in einen *Wartungszustand* wechseln. Diesen kann er auch jederzeit wieder verlassen und ins *Hauptmenü* zurückkehren.
- Wenn die App ausgeschaltet wird, merkt sich das System den vorherigen Zustand und kehrt bei erneutem Einschalten in den entsprechenden Zustand zurück.

▼ **Lösungsvorschlag**

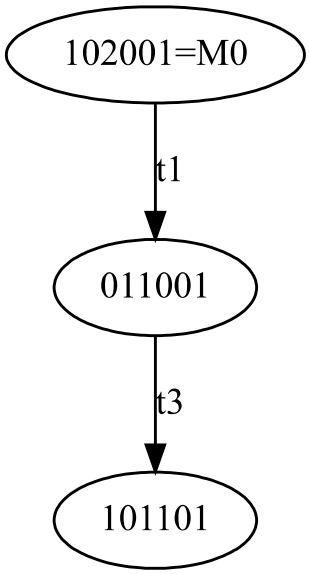


▶ Quelltext des Diagramms

Hinweise zum Erstellen der Diagramme

Erreichbarkeitsgraph

Ein Erreichbarkeitsgraph ist ein gerichteter Graph mit beschrifteten Knoten und Kanten. Entsprechend kann er leicht mit GraphViz gesetzt werden. GraphViz ist in PlantUML enthalten, was wiederum in CodiMD (oder HedgeDoc) direkt eingebunden ist.



▶ Quelltext des Diagramms

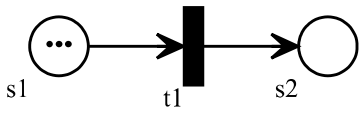
In LaTeX können Sie statt GraphViz auch Ti*k*Z verwenden und erhalten mit etwas mehr Code ein deutlich schöneres Ergebnis und haben mehr Kontrolle über die Positionierung der Knoten:

```
\usepackage{tikz}
\usetikzlibrary{positioning,shapes.geometric}

\begin{tikzpicture}[auto, state/.style={draw,shape=ellipse}]
  \node[state] (102001) {$102001=M_0$};
  \node[state,below=of 102001] (011001) {011001};
  \node[state,below=of 011001] (101101) {101101};
  \draw[->]
    (102001) edge node {$t_1$} (011001)
    (011001) edge node {$t_3$} (101101);
\end{tikzpicture}
```

Petri-Netze

Petri-Netze können ebenfalls mit GraphViz gezeichnet werden:



▶ Quelltext des Diagramms

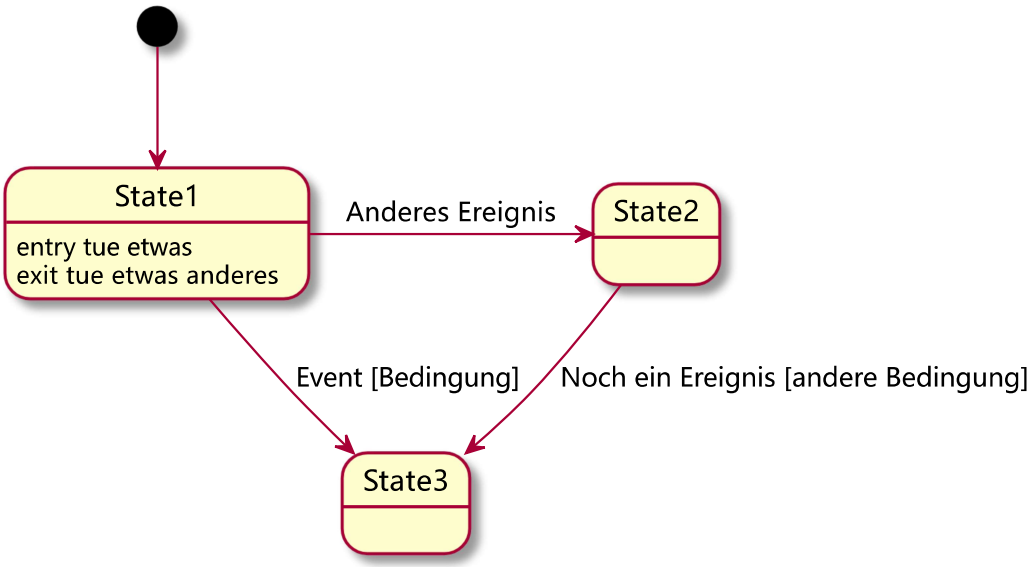
In TikZ gibt es eine eigene Bibliothek für Petri-Netze, die im Kapitel *64 Petri-Net Drawing Library* der Anleitung ausführlich beschrieben wird:

```
\usepackage{tikz}
\usetikzlibrary{positioning,petri}

\begin{tikzpicture}[auto, on grid, every transition/.style={fill=black, minimum height=8mm, minimum w
  \node[place,tokens=3,label=$s_1$] (s1) {};
  \node[place,label=$s_2$,right=2 of s1] (s2) {};
  \node[transition, right=of s1] {} edge[pre] (s1) edge[post] (s2);
\end{tikzpicture}
```

UML-Zustandsdiagramm

Zustandsdiagramme lassen sich sehr elegant mit PlantUML zeichnen.



▶ Quelltext des Diagramms