7. Modellieren von Abläufen

In diesem Kapitel geht es darum, das dynamische Verhalten von Systemen zu beschreiben, z.B.

- die Wirkung von Bedienoperationen auf reale Automaten oder auf die Benutzungsoberflächen von Software-Systemen
- Schaltfolgen von Ampelanlagen
- Abläufe von Geschäftsprozessen in Firmen
- Steuerung von Produktionsanlagen

Solche Abläufe werden modelliert, indem man die Zustände angibt, die ein System annehmen kann, und beschreibt, unter welchen Bedingungen es aus einem Zustand in einen anderen übergehen kann (vgl. das Flussüberquerungs-Problem aus Beispiel 1.1).

In diesem Kapitel werden zwei grundlegende Kalküle vorgestellt, mit denen man solche Abläufe beschreiben kann:

1) endliche Automaten, die sich gut für Modellierung sequentieller Abläufe eignen, und
2) Petri-Netze, mit denen nebenläufige Prozesse beschrieben werden können, bei denen Ereignisse gleichzeitig an mehreren Stellen des Systems Zustandsänderungen bewirken können.

7.1 Endliche Automaten

Endliche Automaten sind ein formaler Kalkül, der zur Spezifikation von realen oder abstrakten Maschinen genutzt werden kann. Endliche Automaten

- reagieren auf äußere Ereignisse
- ändern ggf. seinen "inneren Zustand"
- produzieren ggf. eine Ausgabe.

Sie werden z.B eingesetzt um

- das Verhalten realer Maschinen zu spezifizieren (Bsp: Getränkeautomat)
- das Verhalten von Software-Komponenten zu beschreiben (Bsp: das Wirken von Bedienoperationen auf Benutzungsoberflächen von Software-Systemen)
- Sprachen zu spezifizieren, d.h. die Menge aller Ereignisfolgen, die den Automat von seinem "Startzustand" in einen "akzeptierenden Zustand" überführen (Bsp: "Flüssigkeitsregelung" aus Bsp 1.1: alle Folgen von "Flüssigkeitsregelungsschritten", mit denen man vom "Startzustand" 

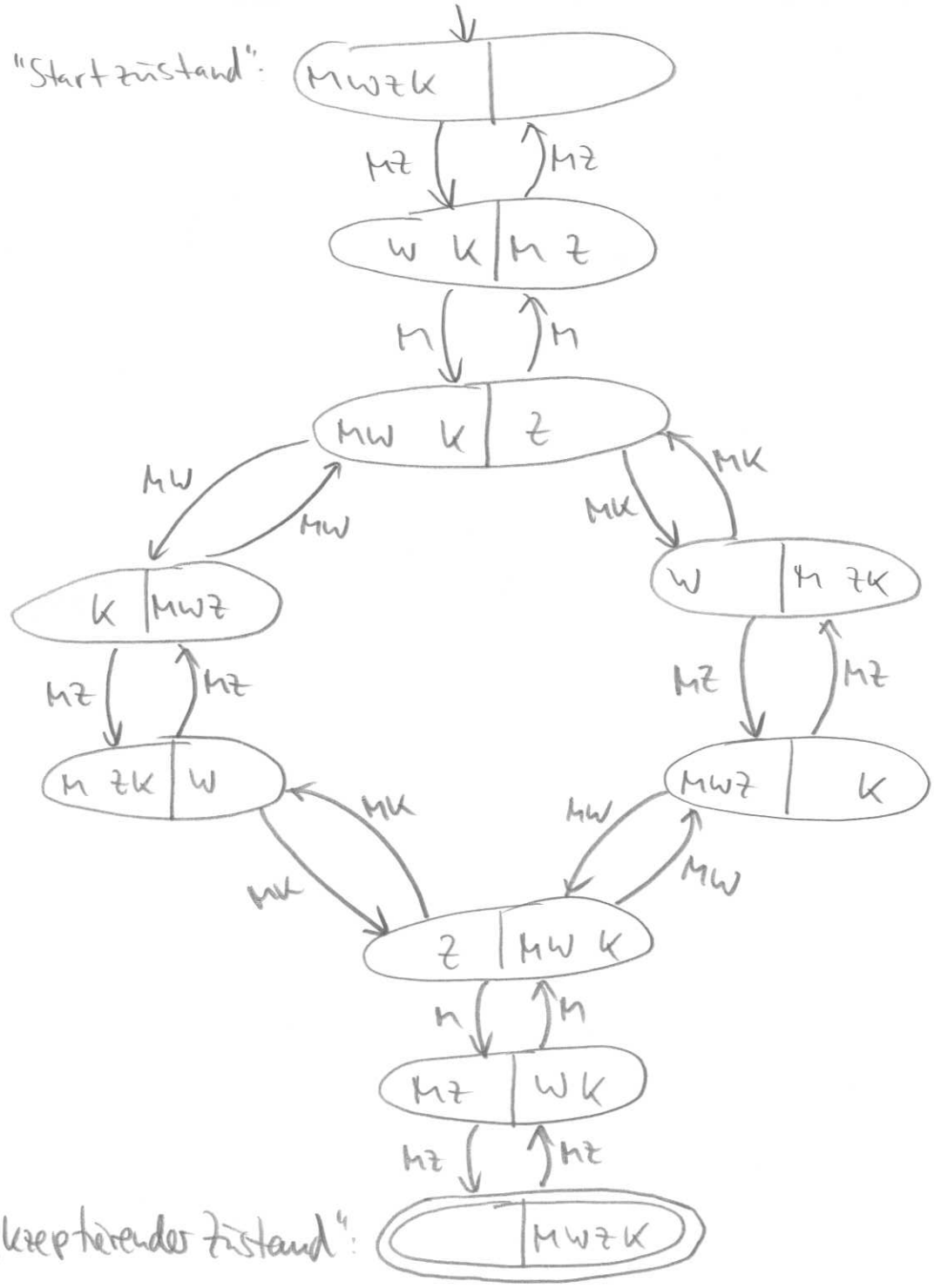
zum "Zielzustand" 

gelangen kann)

Vor der formalen Definition endlicher Automaten zunächst zwei einführende Beispiele:

Beispiel 7.1

Graphische Darstellung eines endlichen Automaten zum Flussüberquerungs-Problem aus Bsp 1.1:



Dieser endliche Automat "akzeptiert" genau diejenigen Folgen von einzelnen Flussüberquerungen, die vom Startzustand in den akzeptierenden Zustand führen

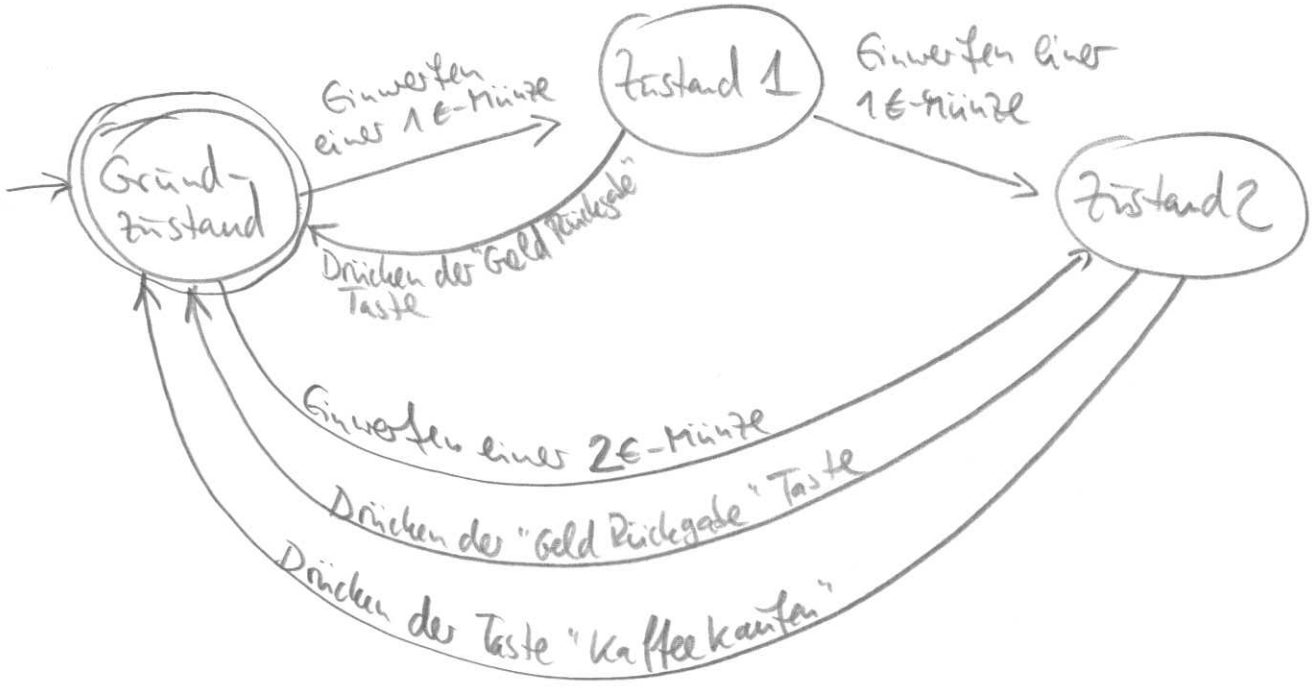
Beispiel 7.2

Betrachte einen einfachen Getränkeautomat, der folgende Bedienoptionen hat:

- Einwerfen einer 1€-Münze
- Einwerfen einer 2€-Münze
- Taste "Geld Rückgabe" drücken
- Taste "Kaffee kaufen" drücken

und bei dem man ein einziges Getränk kaufen kann, das 2€ kostet.

Dieser Getränkeautomat kann durch folgenden endlichen Automaten modelliert werden:



Dieser endliche Automat "akzeptiert" genau diejenigen Folgen von Bedienoperationen, die vom Grundzustand aus wieder in den Grundzustand führen.

Deterministische endliche Automaten

(engl.: deterministic finite automaton,
kurz: DFA)

Definition 7.3 (DFA):

Ein deterministischer endlicher Automat

$$A = (\Sigma, Q, \delta, q_0, F)$$

besteht aus

- einer endlichen Menge Σ, dem so genannten Eingabealphabet
- einer endlichen Menge Q, der so genannten Zustandsmenge (die Elemente aus Q werden Zustände genannt)
- einer partiellen Funktion δ von $Q \times \Sigma$ nach Q, der so genannten (Zustands-) Übergangsfunktion (oder Überführungsfunktion)
- einem Zustand $q_0 \in Q$, dem sog. Startzustand
- einer Menge $F \subseteq Q$, der so genannten Menge der Endzustände bzw. akzeptierenden Zustände

(der Buchstabe "F" steht für "final states", also "Endzustände")

Graphische Darstellung endlicher Automaten

Endliche Automaten lassen sich anschaulich durch beschriftete Graphen darstellen (vgl Bsp 7.1 und 7.2):

- Für jeden Zustand $q \in Q$ gibt es einen durch

  (q) dargestellten Knoten.

- Der Startzustand q_0 wird durch einen in ihn hinein führenden Pfeil markiert, d.h. $\rightarrow (q_0)$

- Jeder akzeptierende Zustand $q \in F$ wird durch eine doppelte Umrandung markiert, d.h. ((q))

- Ist $q \in Q$ ein Zustand und $a \in \Sigma$ ein Symbol aus dem Eingabealphabet, so dass $(q,a) \in \mathrm{Def}(\delta)$ liegt, so gibt es in der graphischen Darstellung von A einen mit dem Symbol a beschrifteten Pfeil von Knoten (q) zu Knoten $(\delta(q,a))$, d.h.: $(q) \xrightarrow{a} (\delta(q,a))$.

Beispiel 7.4

Die graphische Darstellung aus Bsp 7.2 repräsentiert den DFA $A = (\Sigma, Q, \delta, q_0, F)$ mit

- $\Sigma = \{$ Einwerfen einer 1€-Münze,
  Einwerfen einer 2€-Münze,
  Drücken der "Geld Rückgabe" Taste,
  Drücken der Taste "Kaffee kaufen" $\}$
- $Q = \{$ Grundzustand, Zustand 1, Zustand 2 $\}$
- $q_0 =$ Grundzustand
- $F = \{$ Grundzustand $\}$
- δ ist die partielle Funktion von $Q \times \Sigma$ nach Q mit

  δ(Grundzustand, Einwerfen einer 1€-Münze) = Zustand 1,
  δ(Grundzustand, Einwerfen einer 2€-Münze) = Zustand 2,
  δ(Zustand 1, Einwerfen einer 1€-Münze) = Zustand 2,
  δ(Zustand 1, Drücken der "Geld Rückgabe" Taste) = Grundzustand,
  δ(Zustand 2, Drücken der "Geld Rückgabe" Taste) = Grundzustand,
  δ(Zustand 2, Drücken der Taste "Kaffee kaufen") = Grundzustand,

Die von einem DFA akzeptierte Sprache:

Ein DFA $A = (\Sigma, Q, \delta, q_0, F)$ erhält als Eingabe ein Wort $w \in \Sigma^*$, das eine Folge von "Aktionen" oder "Bedienoperationen" repräsentiert.

Ist das Eingabewort w von der Form $a_1 \cdots a_n$ mit $n \geq 0$ und $a_1 \in \Sigma, \ldots, a_n \in \Sigma$, so geschieht bei der "Verarbeitung" von w durch A folgendes:

A wird im "Startzustand" q_0 gestartet. Durch Lesen des ersten Buchstabens von w, also a_1, geht der Automat über in den Zustand $q_1 := \delta(q_0, a_1)$. In der graphischen Darstellung von A wird der Zustand $\to (q_0)$ durch die mit a_1 beschriftete Kante verlassen, und q_1 ist der Endknoten dieser Kante, d.h. $\to (q_0) \xrightarrow{a_1} (q_1)$.

Dies ist allerdings nur möglich, wenn $(q_0, a_1) \in \mathrm{Def}(\delta)$ liegt — d.h. wenn es in der graphischen Darstellung von A eine mit a_1 beschriftete Kante gibt, die

aus Zustand q_0 herausführt. Falls es keine solche Kante gibt, d.h. falls $(q_0, a_1) \notin \mathrm{Def}(\delta)$, so "stürzt A ab", und die Verarbeitung des Worts w ist beendet. Ansonsten ist A nach Lesen des ersten Symbols von w im Zustand $q_1 := \delta(q_0, a_1)$. Durch Lesen des zweiten Symbols von w, also a_2, geht A nun in den Zustand $q_2 := \delta(q_1, a_2)$ über — bzw "stürzt ab", falls $(q_1, a_2) \notin \mathrm{Def}(\delta)$. In der graphischen Darstellung wird (q_1) durch die mit a_2 beschriftete Kante verlassen (falls eine solche Kante existiert); und $q_2 := \delta(q_1, a_2)$ ist der Endknoten dieser Kante, d.h. $(q_1) \xrightarrow{a_2} (q_2)$.

Auf diese Weise wird nach und nach das gesamte Eingabewort $w = a_1 \cdots a_n$ abgearbeitet und — ausgehend vom Startzustand q_0 — werden nacheinander Zustände $q_1, \ldots, q_n$ erreicht.
In der graphischen Darstellung von A entspricht dies gerade dem Durchlaufen eines Weges der Länge n, der im Knoten $\rightarrow (q_0)$ startet und dessen Kanten mit den Buchstaben $a_1, \ldots, a_n$ beschriftet sind. Der

Knoten (q_n), der am Ende dieses Weges erreicht wird (falls der Automat nicht zwischendurch abstürzt, d.h. falls es überhaupt einen mit $a_1 \dots a_n$ beschrifteten in (q_0) startenden Weg gibt),

ist der von A bei Eingabe w erreichte Zustand,

kurz: $q_n = \hat{\delta}(q_0, w)$.

(Im Fall, dass A bei Eingabe von w zwischendurch abstürzt, sagen wir: „$\hat{\delta}(q_0, w)$ ist undefiniert").

Präzise Definition von $\hat{\delta}$:

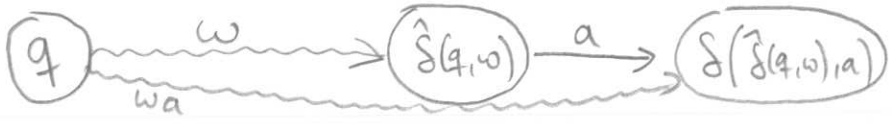
Definition 7.6:

Sei $A := (\Sigma, Q, \delta, q_0, F)$ ein DFA.

Die partielle Funktion $\hat{\delta}$ von $Q \times \Sigma^*$ nach Q ist rekursiv wie folgt definiert:

- f.a. $q \in Q$ ist $\hat{\delta}(q, \varepsilon) := q$
- f.a. $q \in Q$, $w \in \Sigma^*$ und $a \in \Sigma$ gilt:
  Falls $(q, w) \in \mathrm{Def}(\hat{\delta})$ und $(\hat{\delta}(q, w), a) \in \mathrm{Def}(\delta)$,
  so ist $\hat{\delta}(q, wa) := \delta(\hat{\delta}(q, w), a)$.

Graphische Darstellung:

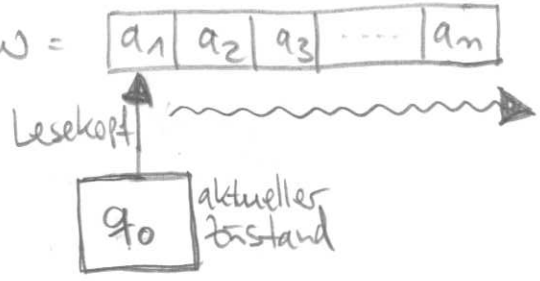


Insgesamt gilt:

Falls $(q_0, w) \in \mathrm{Def}(\hat{\delta})$ so ist $\hat{\delta}(q_0, w)$ der Zustand, der durch Verarbeiten des Worts w erreicht wird; falls $(q_0, w) \notin \mathrm{Def}(\hat{\delta})$, so stürzt der Automat beim Verarbeiten des Worts w ab.

Das Eingabewort w wird vom DFA A akzeptiert, falls e bei Eingabe von w nicht abstürzt und der durch Verarbeiten des Worts w erreichte Zustand zur Menge F der akzeptierenden Zustände gehört. In der graphischen Darstellung von A heißt das für ein Eingabewort $w = a_1 \cdots a_n$, dass es einen in $\to(q_0)$ startenden Weg der Länge n gibt, dessen Kanten mit den Symbolen $a_1, \ldots, a_n$ beschriftet sind, und der in einem akzeptierenden Zustand ◎ endet.

Verarbeitung eines Eingabeworts durch einen DFA A:



Definition 7.7: (Die von einem DFA A akzeptierte Sprache $L(A)$)

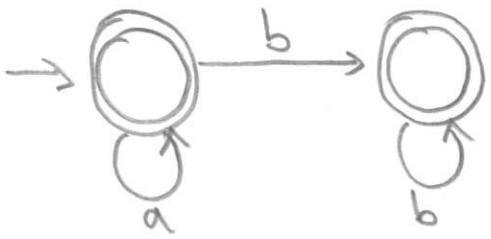
Die von einem DFA $A = (\Sigma, Q, \delta, q_0, F)$ akzeptierte Sprache $L(A)$ ist

$$L(A) := \{ w \in \Sigma^* : \hat{\delta}(q_0, w) \in F \}.$$

D.h. ein Wort $w \in \Sigma^*$ gehört genau dann zur Sprache $L(A)$, wenn es vom DFA A akzeptiert wird.

Beispiel 7.8: Der Einfachheit halber betrachten wir das Eingabealphabet $\Sigma := \{a, b\}$.

(a) Sei A_1 ein DFA mit folgender graphischer Darstellung:



- A_1 akzeptiert z.B. folgende Worte: ε, a, b, aaa, aaab, aaaaabbbb, bbb, ...
- A_1 "stirbt ab" z.B. bei Eingabe von ba, aabbba

Insgesamt gilt: $L(A_1) = \{ a^n b^m : n \in \mathbb{N}, m \in \mathbb{N} \}$

(Notation: $a^n b^m$ bezeichnet das Wort

$\underbrace{a \cdots a}_{n} \underbrace{b \cdots b}_{m}$ der Länge $n+m$, das aus

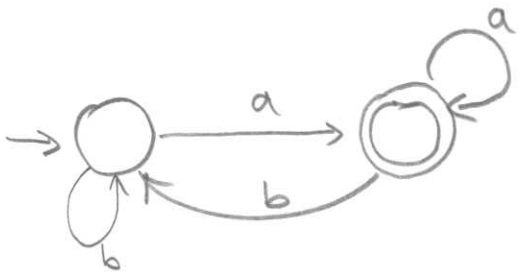
n a's gefolgt von m b's besteht,

z.B. ist $a^3 b^4$ das Wort aaabbbb)

(b) Ein DFA A_2 mit

$L(A_2) = \{ w \in \{a,b\}^* :$ der letzte Buchstabe von w ist ein a $\}$:

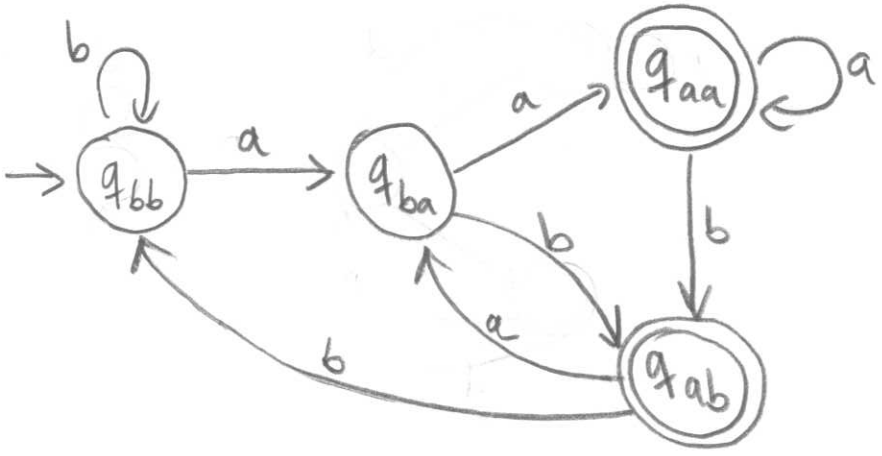
graphische Darstellung von A_2:



(c) Ein DFA A_3 mit

$L(A_3) = \{ w \in \{a,b\}^* :$ der vorletzte Buchstabe von w ist ein a $\}$:

graphische Darstellung von A_3



Bemerkung 7.9

- Die in Definition 7.3 eingeführten DFAs $A = (\Sigma, Q, \delta, q_0, F)$ heißen <u>deterministisch</u>, weil es zu jedem Paar $(q,a) \in Q \times \Sigma$ höchstens einen "Nachfolgezustand" $\delta(q,a)$ gibt (da δ eine partielle Funktion von $Q \times \Sigma$ nach Q ist).

  Beim Verarbeiten eines Eingabewort ist daher zu jedem Zeitpunkt klar, ob A "abstürzt" oder nicht – und falls nicht, welchen eindeutig festgelegten Nachfolgezustand A annimmt.

- Ein DFA A heißt <u>vollständig</u>, wenn die Übergangsfunktion δ eine totale Funktion $\delta: Q \times \Sigma \to Q$ ist.

  <u>Bsp</u>: Die DFAs A_2 und A_3 aus Bsp 7.8 sind vollständig; der DFA A_1 nicht, und auch die DFAs aus Bsp 7.1 und 7.2 sind nicht vollständig.

  Für die graphische Darstellung eines DFAs gilt: Der DFA ist genau dann vollständig, wenn für jeden Zustand q gilt: Für jedes Symbol $a \in \Sigma$ gibt es genau eine aus (q) herausführende Kante, die mit a beschriftet ist.

Beachte: In manchen Büchern weicht die Definition von DFAs von Definition 7.3.9b, indem gefordert wird, dass DFAs grundsätzlich vollständig sein müssen.

Nicht-deterministische endliche Automaten:

(engl. non-deterministic finite automaton, kurz: NFA)

Für manche Modellierungsaufgaben ist die Forderung, dass es für jeden Zustand q und jedes Eingabesymbol a höchstens einen Nachfolgezustand $\delta(q,a)$ gibt, zu restriktiv, da man in manchen Zuständen für den Übergang mit einem Symbol a mehrere Möglichkeiten angeben will, ohne festzulegen, welche davon gewählt wird. Solche Entscheidungsfreiheiten in der Modellierung von Abläufen nennt man nicht-deterministisch. Nicht-deterministische Modelle sind häufig einfacher aufzustellen und leichter zu verstehen als deterministische.

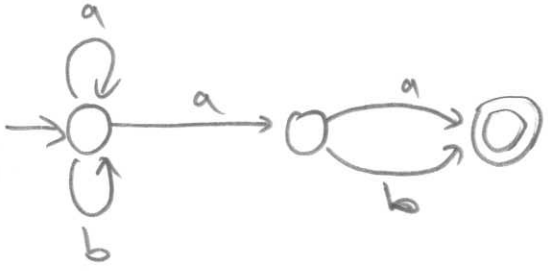
Beispiel 7.10:

In Beispiel 7.8 (c) haben wir einen (recht komplizierten) DFA A_3 kennengelernt mit

$L(A_3) = \{ w \in \{a,b\}^* :$ der vorletzte Buchstabe von w ist ein $a \}$.

Dieselbe Sprache auch vom folgenden, deutlich einfacheren nicht-deterministischen endlichen Automaten (kurz: NFA) A_4 akzeptiert:

graphische Darstellung von A_4:



Generell gilt: Ein Eingabewort w wird von dem NFA A_4 genau dann akzeptiert, wenn es in der graphischen Darstellung (mindestens) einen Weg gibt, der im Startzustand $\to\bigcirc$ beginnt, dessen Kanten mit w beschriftet sind und der im akzeptierenden Zustand ⓞ endet.

Präzise Definition von NFAs:

Definition 7.11 (NFA)

Ein **nicht-deterministischer endlicher Automat**

$A = (\Sigma, Q, \delta, q_0, F)$ besteht aus

- einer endlichen Menge Σ, dem so genannten **Eingabealphabet**
- einer endlichen Menge Q, der sog. **Zustandsmenge**
- einer Funktion $\delta: Q \times \Sigma \to \mathcal{P}(Q)$, der sog. **Übergangsfunktion**, die jedem Zustand $q \in Q$ und jedem Symbol $a \in \Sigma$ eine **Menge $\delta(q,a)$ von möglichen Nachfolgezuständen** zuordnet
  (Beachte: möglicherweise ist $\delta(q,a) = \emptyset$ — dann "stirbt" der Automat ab, wenn er im Zustand q ist und das Symbol a liest).
- einem Zustand $q_0 \in Q$, dem sog. **Startzustand**
- einer Menge $F \subseteq Q$, der sog. Menge der **Endzustände** bzw. **akzeptierenden Zustände**

Graphische Darstellung von NFAs:

Wie bei DFAs. Ist $q \in Q$ ein Zustand und ist $a \in \Sigma$ ein Eingabesymbol, so gibt es für jeden Zustand $q' \in \delta(q,a)$ in der graphischen Darstellung des NFAs einen mit dem Symbol a beschrifteten Pfeil von Knoten (q) zu Knoten (q'), d.h. $(q) \xrightarrow{a} (q')$.

Die von einem NFA A akzeptierte Sprache L(A):

Definition 7.12

Sei $A = (\Sigma, Q, \delta, q_0, F)$ ein NFA

(a) Sei $n \in \mathbb{N}$ und sei $w = a_1 \cdots a_n$ ein Eingabewort der Länge n.
Das Wort w wird genau dann vom NFA A akzeptiert, wenn es in der graphischen Darstellung von A einen im Startzustand $\rightarrow (q_0)$ beginnenden Weg der Länge n gibt, dessen Kanten mit den Symbolen $a_1, \ldots, a_n$ beschriftet sind und der

in einem akzeptierenden Zustand endet.

(b) Die von A akzeptierte Sprache $L(A)$ ist

$$L(A) := \{ w \in \Sigma^* : A \text{ akzeptiert } w \}$$

Ein Anwendungsbeispiel: Stichwort-Suche in Texten

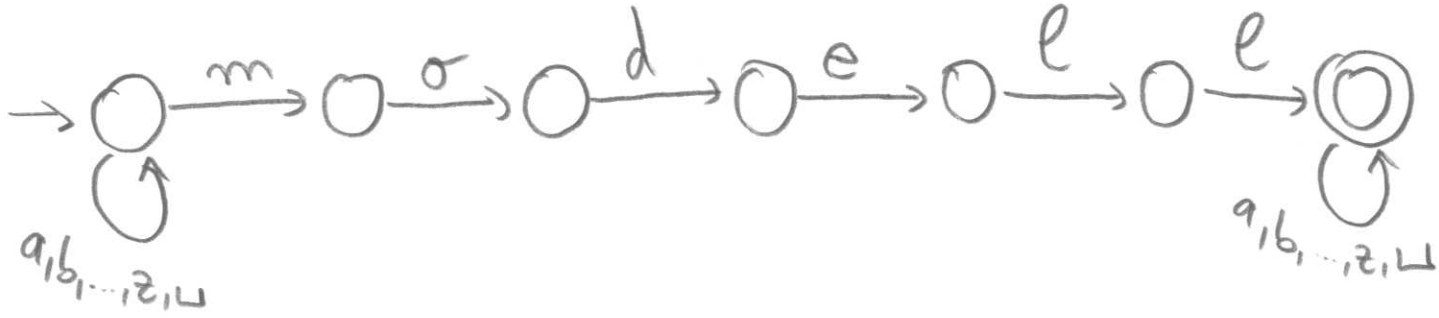
Gegeben: Ein Stichwort, z.B. "modell"

Eingabe: Ein Text, der aus den Buchstaben $a, \dots, z, \sqcup$ besteht

Frage: Kommt das Stichwort "modell" irgendwo im Eingabetext vor?

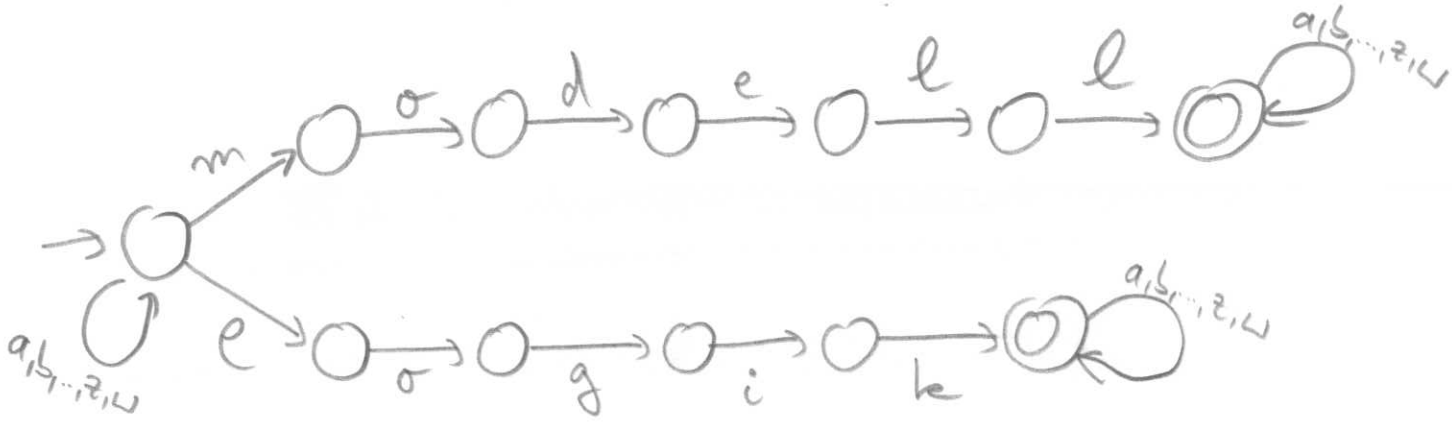
— Der Eingabetext soll genau dann akzeptiert werden, wenn er das Stichwort "modell" enthält.

Graphische Darstellung eines NFAs, der dies bewerkstelligt:



Variante: Kommt mindestens eins der Stichworte "modell" bzw. "logik" im Eingabetext vor?

Graphische Darstellung eines NFAs, der dies Lewerkstelligt:



NFAs vs. DFAs

Frage: Können NFAs wirklich "mehr" als DFAs?

Antwort: Nein:

Satz 7.13:
Für jeden NFA $A = (\Sigma, Q, \delta, q_0, F)$ gibt es einen DFA $A' = (\Sigma, Q', \delta', q_0', F')$ mit $L(A') = L(A)$.

D.h: NFAs und DFA können genau dieselben Sprachen akzeptieren.

Beweis: In der Vorlesung GL-2.

Bemerkung 7.15

Typische Fragen bzgl. DFAs bzw. NFAs:

(a) Welche Sprachen können prinzipiell durch DFAs (bzw. äquivalent dazu, NFAs) akzeptiert werden; welche nicht?

Bsp: Die Sprache $L_1 := \{ a^n b^m : n \in \mathbb{N}, m \in \mathbb{N} \}$ wird von dem DFA A_1 aus Beispiel 7.8(a) akzeptiert, d.h. $L_1 = L(A_1)$.

Satz: Es gibt keinen DFA, der genau die Sprache $L_2 := \{ a^n b^n : n \in \mathbb{N} \}$ akzeptiert.

Beweis: In der Vorlesung GL-2.

(b) Gegeben sei ein DFA oder NFA A.
Wie kann man herausfinden, ob $L(A) \neq \emptyset$ ist, d.h. ob es (mind.) ein Eingabewort gibt, das von A akzeptiert wird (↗ vgl. das Flüssigkeitsübergießungsproblem aus Beispiel 1.1).
— Antwort: Teste, ob es in der graphischen Darstellung von A einen Weg gibt, der vom Startzustand

zu einem akzeptierenden Zustand führt.

(c) Wie schwierig ist es, zu einem gegebenen NFA einen DFA zu finden, der dieselbe Sprache akzeptiert?

Antwort(en): In der Vorlesung GL-2.

Reguläre Ausdrücke

Reguläre Ausdrücke beschreiben Mengen von Worten, die nach bestimmten Regeln bzw "Mustern" aufgebaut sind.

Beispiel 7.16:

Die Menge aller Wörter über dem Alphabet $\{a,b\}$, deren vorletzter Buchstabe ein a ist, wird durch den folgenden regulären Ausdruck beschrieben:

$$(a|b)^* a (a|b)$$

Definition 7.17 (Reguläre Ausdrücke – Syntax)

Sei Σ ein endliches Alphabet.

Die Menge aller regulären Ausdrücke über Σ ist rekursiv wie folgt definiert:

Basisregeln:

- $\emptyset$ ist ein regulärer Ausdruck über Σ ("leere Menge")
- ε ist ein regulärer Ausdruck über Σ ("leeres Wort")
- für jedes $a \in \Sigma$ gilt:
  a ist ein regulärer Ausdruck über Σ

Rekursive Regeln:

- Ist R ein regulärer Ausdruck über Σ, so ist auch R^* ein regulärer Ausdruck über Σ ("Kleene-Stern")
- Sind R und S reguläre Ausdrücke über Σ, so ist auch
  - $(R \cdot S)$ ein regulärer Ausdruck über Σ ("Konkatenation")
  - $(R \mid S)$ ein regulärer Ausdruck über Σ ("Vereinigung")

Definition 7.18 (Reguläre Ausdrücke – Semantik)

Sei Σ ein endliches Alphabet.

Jeder reguläre Ausdruck R über Σ beschreibt (oder: definiert) eine Sprache $L(R) \subseteq \Sigma^*$, die induktiv wie folgt definiert ist:

- $L(\emptyset) := \emptyset$
- $L(\varepsilon) := \{\varepsilon\}$
- für jedes $a \in \Sigma$ gilt $L(a) := \{a\}$
- Ist R ein regulärer Ausdruck über Σ, so ist

  $L(R^*) := \{\varepsilon\} \cup \{w_1 \cdots w_k : k \in \mathbb{N}_{>0},\ w_1 \in L(R), \ldots, w_k \in L(R)\}$

- Sind R und S reguläre Ausdrücke über Σ, so ist
  - $L((R \cdot S)) := \{wu : w \in L(R),\ u \in L(S)\}$
  - $L((R \mid S)) := L(R) \cup L(S)$.

Notation 7.19

Zur vereinfachten Schreibweise und Lesbarkeit von regulären Ausdrücken vereinbaren wir folgende Konventionen:

- Den "Punkt" bei der Konkatenation $(R \cdot S)$ darf man weglassen
- Bei Ketten gleichartiger Operatoren darf man Klammern weglassen: z.B. schreiben wir kurz $(R_1 | R_2 | R_3 | R_4)$ statt $(((R_1 | R_2) | R_3) | R_4)$ und $(R_1 R_2 R_3 R_4)$ statt $(((R_1 R_2) R_3) R_4)$
- "Präzedenzregeln":
  1) $*$ bindet stärker als $\cdot$
  2) $\cdot$ bindet stärker als $|$
- äußere Klammern, die einen regulären Ausdruck umschließen, dürfen weggelassen werden.
- Zur besseren Lesbarkeit dürfen zusätzliche Klammern benutzt werden.

Beispiel 7.20

(a) $a|bc^*$ ist eine verkürzte Schreibweise für den regulären Ausdruck $(a|(b \cdot c^*))$.

Die von diesem regulären Ausdruck beschriebene Sprache ist $L(a|bc^*) = \{a\}$

$\cup \{w \in \{a,b,c\}^* :$ der erste Buchstabe von w ist ein b und alle weiteren Buchstaben von w sind c's $\}$

(b) $L((a|b)^*) = \{a,b\}^*$

(c) Die Menge aller Worte über dem Alphabet $\{a,b,c\}$, in denen abb als Teilwort vorkommt, wird durch den folgenden regulären Ausdruck beschrieben: $(a|b|c)^* abb (a|b|c)^*$

(d) Die Menge aller Worte über $\{a,b,c\}$, deren letzter oder vorletzter Buchstabe ein a ist, wird durch den folgenden regulären Ausdruck beschrieben:

$$(a|b|c)^* a (\varepsilon|a|b|c)$$

Frage: Welche Arten von Sprachen können durch reguläre Ausdrücke beschrieben werden?

Antwort: Genau dieselben Sprachen, die durch (deterministische oder nicht-deterministische) endliche Automaten akzeptiert werden können.

— Diese Sprachen werden reguläre Sprachen genannt.

Details: In der Veranstaltung GL-2.

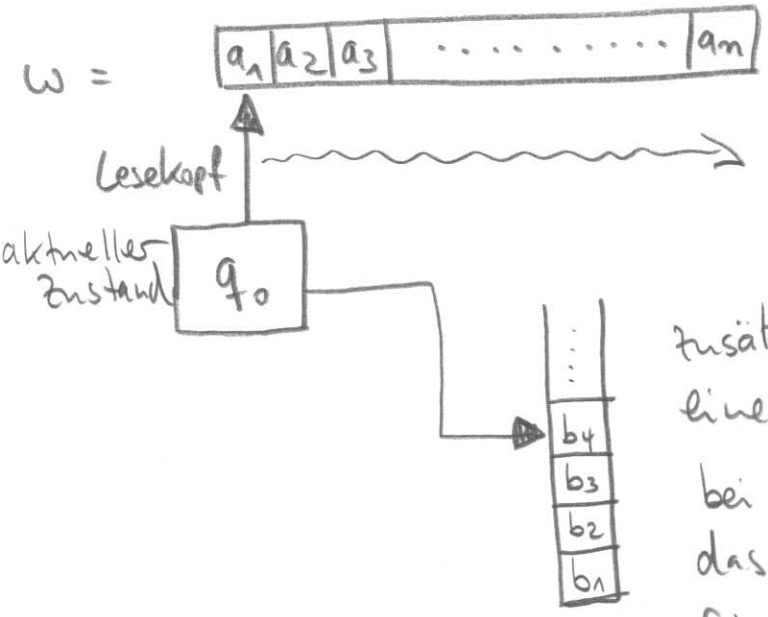
Ausblick:

Abgesehen von DFAs, NFAs und regulären Ausdrücken kann man die regulären Sprachen auch durch bestimmte Grammatiken erzeugen: so genannte reguläre Grammatiken — das sind kontextfreie Grammatiken, die von einer besonders einfachen Form sind. Generell gilt: Für jede reguläre Sprache L gibt es eine kontextfreie Grammatik, die die Sprache L erzeugt. Aber es gibt kontextfreie Grammatiken, die nicht-reguläre Sprachen erzeugen.

Analog zu DFAs und NFAs gibt es auch ein erweitertes Automatenmodell, das genau diejenigen Sprachen akzeptiert, die von kontextfreien Grammatiken erzeugt werden: so genannte

Kellerautomaten

Schematische Darstellung der Verarbeitung eines Eingabeworts durch einen Kellerautomaten:



zusätzlicher Speicher in Form eines Kellers ("Stapel"), bei dem immer nur auf das oberste Element des Stapels zugegriffen werden kann.

Details: In der Vorlesung GL-2. Dort werden auch allgemeinere Arten von Grammatiken betrachtet, z.B. so genannte kontext-sensitive Grammatiken.

7.2 Petri-Netze

Petri-Netze:

- eingeführt von C.A. Petri, 1962
- formaler Kalkül zur Modellierung von Abläufen, an denen mehrere Prozesse beteiligt sind
- modelliert werden die Interaktionen zwischen Prozessen und die Effekte, die sich daraus ergeben, dass Operationen prinzipiell gleichzeitig ausgeführt werden können (Stichwort: "Nebenläufigkeit").

Typische Anwendungsbeispiele für Petri-Netze:

zur Modellierung von

- realen oder abstrakten Automaten und Maschinen
- kommunizierenden Prozessen (z.B. in Rechnern)
- Verhalten von Software- oder Hardware-Komponenten
- Geschäftsabläufe
- Spiele (Spielregeln)

Der Kalkül der Petri-Netze basiert auf bipartiten gerichteten Graphen:

- 2 Sorten von Knoten, die
  - Bedingungen oder Zustände (sog. "Stellen") bzw.
  - Aktivitäten (sog. "Transitionen")

  repräsentieren
- Kanten verbinden "Aktivitäten" mit ihren "Vorbedingungen" und ihren "Nachbedingungen"
- Knotenmarkierungen repräsentieren den veränderlichen "Zustand" des Systems.

Definition 7.21 (Petri-Netz)

Ein Petri-Netz $P = (S, T, F)$ besteht aus

- einer endlichen Menge S, den sog. Stellen von P
- einer endlichen Menge T, den sog Transitionen von P
- einer Relation $F \subseteq (S \times T) \cup (T \times S)$, den sog. Kanten von P.

Die Mengen S und T sind disjunkt, d.h. $S \cap T = \emptyset$.

Ein Petri-Netz P bildet einen bipartiten gerichteten Graphen mit Knotenmenge $S \cup T$ und Kantenmenge F.

Graphische Darstellung:

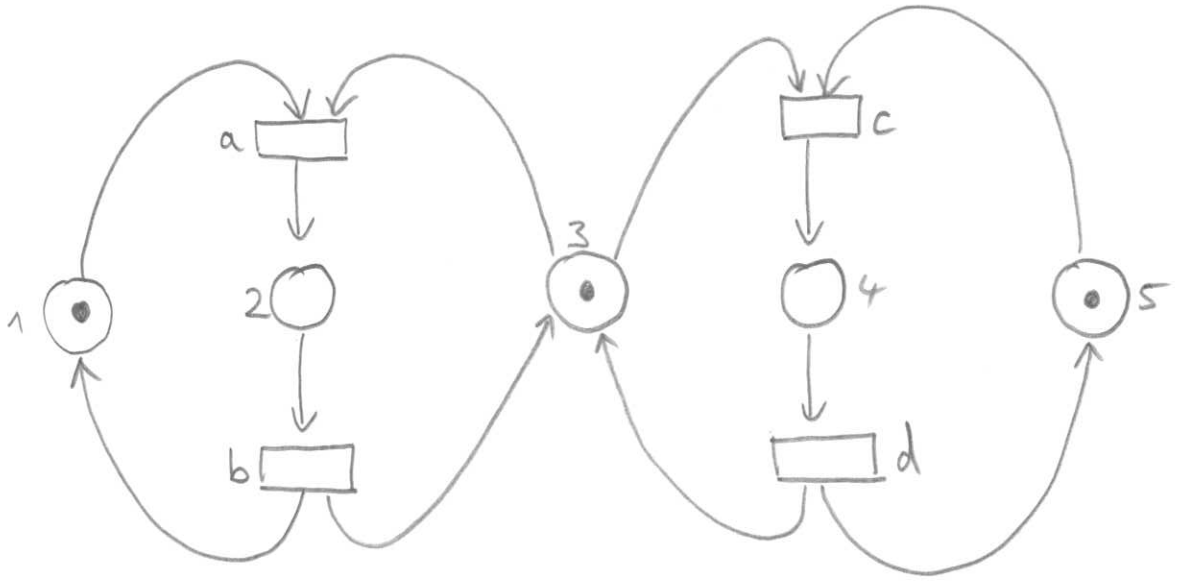
- "Stellen", d.h. Knoten in S, werden durch Kreise dargestellt
- "Transitionen", d.h. Knoten in T, werden durch Rechtecke dargestellt

Definition 7.22

Der "Zustand" eines Petri-Netzes $P = (S, T, F)$ wird durch eine Markierungsfunktion (kurz: Markierung) $M : S \to \mathbb{N}$, die jeder Stelle $s \in S$ eine Anzahl $M(s)$ von sog. Marken zuordnet, repräsentiert.

Beispiel 7.23:

Graphische Darstellung eines Petri-Netzes $P = (S, T, F)$ und einer Markierung M: die einzelnen "Marken", die M einer Stelle $s \in S$ zuordnet, werden durch Punkte • in den die Stelle s repräsentierenden Knoten dargestellt.



$S = \{1, 2, 3, 4, 5\}$

$T = \{a, b, c, d\}$

$F = \{ (1,a), (3,a), (a,2), (2,b), (b,1), (b,3), (3,c), (5,c), (c,4), (4,d), (d,3), (d,5) \}$

$M : S \to \mathbb{N}$ mit $M(1) = M(3) = M(5) = 1$ und $M(2) = M(4) = 0$.

Definition 7.24

Sei $P = (S, T, F)$ ein Petri-Netz und sei $t \in T$ eine Transition von P.

Wir setzen

- $\text{Vorbereich}(t) := \{ s \in S : (s,t) \in F \}$

  = "Menge aller Stellen, von denen aus eine Kante in t hineinführt"

- $\text{Nachbereich}(t) := \{ s \in S : (t,s) \in F \}$

  = "Menge aller Stellen, zu denen eine von t ausgehende Kante hinführt"

Petri-Netze verändern ihren "Zustand", indem Transitionen "schalten" und dadurch die "Markierung" des Petri-Netzes ändern. Das wird folgendermaßen präzisiert:

Schaltregel:

Sei $P = (S, T, F)$ ein Petri-Netz und sei $M: S \to \mathbb{N}$ eine Markierung.

Das "Schalten einer Transition $t \in T$" überführt die Markierung M in eine Markierung $M': S \to \mathbb{N}$.

Die Transition t kann schalten, wenn gilt:

f.a. Stellen $s \in$ Vorbereich(t) ist $M(s) \geq 1$,

d.h.: Jede Stelle s in Vorbereich(t) hat mindestens eine Marke.

Wenn die Transition t schaltet, so gilt für die sog. Nachfolgemarkierung M': f.a. $s \in S$:

$$M'(s) := \begin{cases} M(s) - 1 & \text{falls } s \in \text{Vorbereich}(t) \setminus \text{Nachbereich}(t) \\ M(s) + 1 & \text{falls } s \in \text{Nachbereich}(t) \setminus \text{Vorbereich}(t) \\ M(s) & \text{sonst} \end{cases}$$

D.h.: Das "Schalten von Transition t" bewirkt, dass in jeder Stelle in Vorbereich(t) eine Marke entfernt wird und dass in jeder Stelle in Nachbereich(t) eine Marke hinzugefügt wird.

Wenn mehrere Transitionen schalten können, so wird eine davon "nicht-deterministisch ausgewählt"

In jedem Schritt schaltet genau eine Transition.

Durch schrittweises Schalten von Transitionen wird der Ablauf von Prozessen modelliert

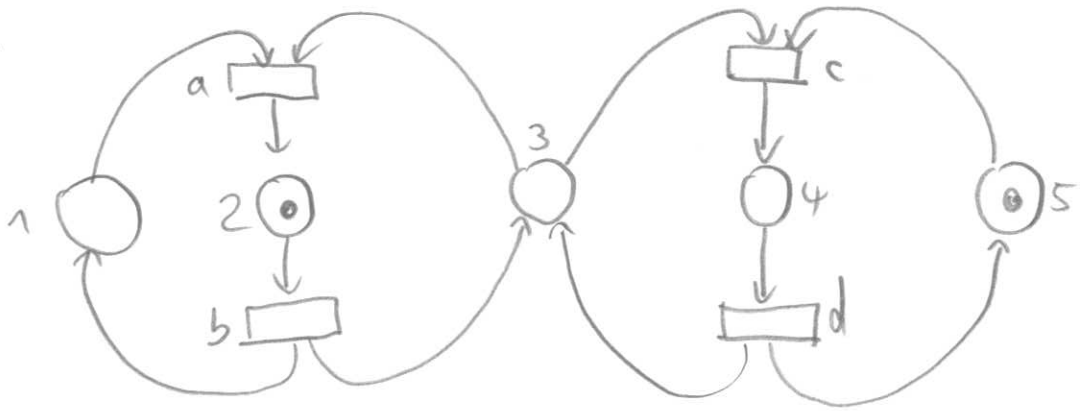
Beispiel 7.25:

Seien $P = (S, T, F)$ und $M: S \to \mathbb{N}$ das Petri-Netz und die Markierung aus Beispiel 7.23.

Bei der angegebenen Markierung M können die Transitionen a und c schalten.

Wenn wir a schalten lassen ergibt sich als Nachfolgemarkierung die Markierung $M': S \to \mathbb{N}$ mit $M'(1) = 0$, $M'(3) = 0$, $M'(2) = 1$, $M'(4) = 0$, $M'(5) = 1$,

d.h.:



Als nächstes kann Transition c nicht schalten, da die Stelle 3 keine Marke trägt.
Die einzige Transition, die jetzt schalten kann, ist Transition b, deren Schalten bewirkt, dass die Marke bei 2 verschwindet und stattdessen Marken bei 1 und 3 erzeugt werden. Nach dem Schalten von b ist das System also wieder in seinem "ursprünglichen Zustand", d.h. es trägt die Markierung M.

Insgesamt gilt: Das Petri-Netz P aus Bsp. 7.23 (zusammen mit der Startmarkierung M) modelliert zwei zyklisch ablaufende Prozesse.
Die Stelle 3 "synchronisiert" die beiden Prozesse, so dass sich nie gleichzeitig in beiden Stellen 2 und 4 eine Marke befinden kann.

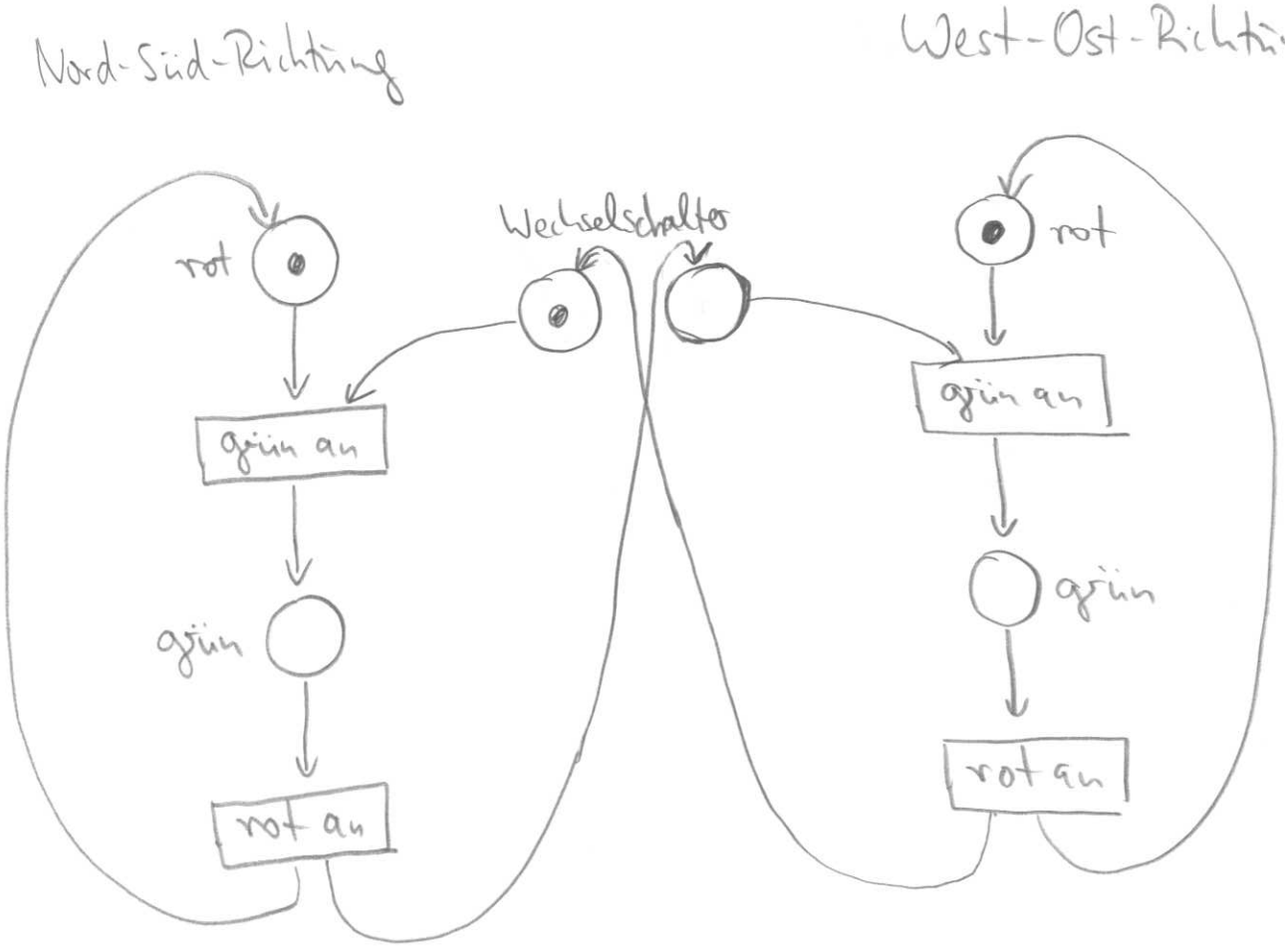
Auf diese Weise könnte man z.B. beschreiben, wie Autos eine 1-spurige Brücke von 2 Seiten überqueren, so dass sich immer nur 1 Auto auf der Brücke befindet.

Beispiel 7.26

Modellierung einer Ampel an einer Kreuzung

- 2 sich zyklisch wiederholende Prozesse:
  - "grün" in Nord-Süd-Richtung
  - "grün" in West-Ost-Richtung
- die beiden Prozesse sollen sich immer abwechseln

Petri-Netz incl. Anfangs-Markierung:



Die beiden Stellen "Wechselschalter" koppeln die Prozesse, so dass abwechselnd in West-Ost- bzw. in Nord-Süd-Richtung die Ampel "grün" ist.