# 7 Modellierung von Abläufen
# 7.1 Endliche Automaten

**Endlicher Automat**:

Formaler Kalkül zur **Spezifikation von realen oder abstrakten Maschinen**. Sie

- reagieren auf **äußere Ereignisse**,
- ändern ihren **inneren Zustand**,
- produzieren ggf. **Ausgabe**.

Endliche Automaten werden **eingesetzt**, um

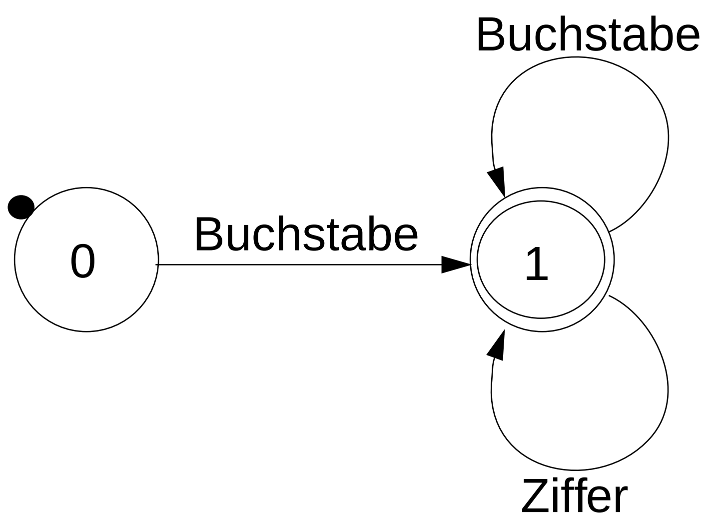
- das **Verhalten realer Maschinen** zu spezifizieren, z. B. Getränkeautomat,
- das **Verhalten von Software-Komponenten** zu spezifizieren,
  z. B. Reaktionen von Benutzungsoberflächen auf Bedienereignisse,
- **Sprachen zu spezifizieren**: Menge der Ereignis- oder Symbolfolgen, die der Automat akzeptiert, z. B. Schreibweise von Bezeichnern und Zahlwerten in Programmen

Zunächst definieren wir nur die **Eingabeverarbeitung** der Automaten;
das Erzeugen von **Ausgabe** fügen wir **später** hinzu.

# Zwei einführende Beispiele

Endlicher Automat definiert eine **Sprache**, d. h. eine Menge von Wörtern.
Ein Wort ist eine Folge von Zeichen.

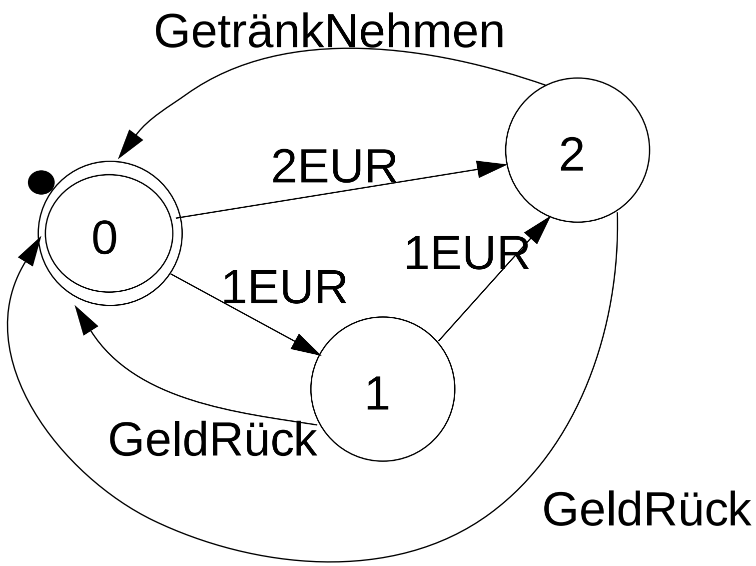
Hier: **Bezeichner** in Pascal-Programmen:




**Akzeptiert** Folgen von Buchstaben und Ziffern beginnend mit einem Buchstaben.

Endlicher Automat spezifiziert das **Verhalten einer Maschine**.

Hier: einfacher **Getränkeautomat**:




**Akzeptiert** Folgen von Ereignissen zur Bedienung eines Getränkeautomaten

Endliche Automaten können durch **gerichtete, markierte Graphen** dargestellt werden, **Ablaufgraphen**.

# Alphabete

**Alphabet**:

Eine **Menge von Zeichen** zur Bildung von Zeichenfolgen,
häufig mit $\Sigma$ bezeichnet.

Wir betrachten hier nur endliche Alphabete, z. B.

{0, 1}
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
{a, b, ..., z}

**Ein Wort über einem Alphabet** $\Sigma$ ist eine **Zeichenfolge** aus $\Sigma^*$

statt $(a_1, a_2, ..., a_n) \in \Sigma^*$ schreiben wir $a_1\, a_2\, ...\, a_n$,
z. B. $10010 \in \{0, 1\}^*$

für die leere Folge schreiben wir auch $\varepsilon$ (epsilon)

# Reguläre Ausdrücke

**Reguläre Ausdrücke** beschreiben **Mengen von Worten**, die nach bestimmten Regeln aufgebaut sind. Seien F und G reguläre Ausdrücke, dann gilt

| **regulärer Ausdruck** | **Menge von Worten** | **Erklärung** |
|---|---|---|
| a | { a } | Zeichen a als Wort |
| $\varepsilon$ | $\{ \varepsilon \}$ | das leere Wort |
| F \| G | $\{ f \mid f \in F \} \cup \{ g \mid g \in G \}$ | Alternativen |
| F G | $\{ f g \mid f \in F, g \in G \}$ | Zusammenfügen von Worten |
| $F^n$ | $\{ f_1 f_2 \dots f_n \mid \forall i \in \{1,..n\}: f_i \in F \}$ | n Worte aus F |
| $F^*$ | $\{ f_1 f_2 \dots f_n \mid n \geq 0 \text{ und } \forall i \in \{1,..n\}: f_i \in F \}$ | Folgen von Worten aus F |
| $F^+$ | $\{ f_1 f_2 \dots f_n \mid n \geq 1 \text{ und } \forall i \in \{1,..n\}: f_i \in F \}$ | nicht-leere Folgen von Worten aus F |
| ( F ) | F | Klammerung |

**Beispiele:** $1^3 ( 1 \mid 0 )^* 0^3$

Bezeichner = B ( B | D )* mit B = a | b | ... | z und D = 0 | 1 | ... | 9

# Deterministischer endlicher Automat

**Deterministischer endlicher Automat** (engl.: deterministic finite automaton, DFA):

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

| | |
|---|---|
| $\Sigma$ | endliches **Eingabealphabet** |
| Q | endliche **Menge von Zuständen** |
| $\delta$ | **Übergangsfunktion** aus $Q \times \Sigma$ -> Q |
| $q_0 \in Q$ | **Anfangszustand** |
| $F \subseteq Q$ | **Menge der Endzustände** (akzeptierend) |

Wir nennen r = $\delta$(q, a) **Nachfolgezustand von q unter a**.

A heißt **deterministisch, weil** es **zu jedem Paar (q, a)**, mit $q \in Q$, $a \in \Sigma$, **höchstens einen Nachfolgezustand** $\delta$(q, a) gibt, d. h. $\delta$ ist eine **Funktion in Q.**

A heißt **vollständig, wenn** die **Übergangsfunktion** $\delta$ eine **totale** Funktion ist.

# Gerichteter Graph zu endlichem Automaten

**Knoten**: **Zustände** des Automaten; Anfangszustand und Endzustände werden speziell markiert
**Kanten**: **Übergangsfunktion**, q -> r markiert mit a, genau dann wenn $\delta(q, a) = r$
Es gibt Kanten, die sich nur durch ihre Markierung unterscheiden, deshalb: **Multigraph**

Beispiele von Mod-7.2:

$\Sigma$ := Menge der ASCII-Zeichen

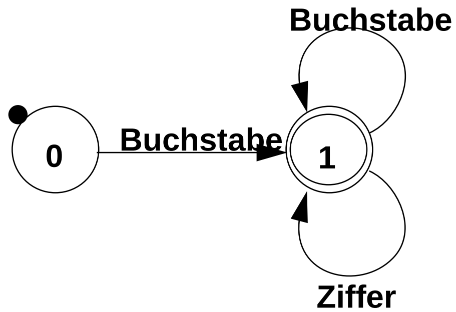
$Q := \{0, 1\}$

$\delta$ :=

| | a...zA...Z | 0...9 | sonstige |
|---|---|---|---|
| 0 | 1 | | |
| 1 | 1 | 1 | |

$q_0 = 0$

$F = \{1\}$




Buchstabe, Ziffer
sind Namen reg. Ausdrücke

$\Sigma$ := {**1EUR, 2EUR, GeldRück, GetränkNehmen**}

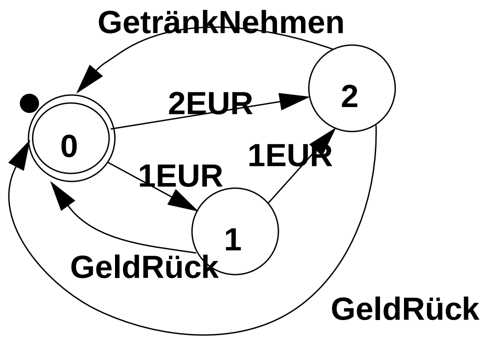
$Q := \{0, 1, 2\}$

$\delta$ :=

| | 1EUR | 2EUR | GeldRück | GetränkNehmen |
|---|---|---|---|---|
| **0** | **1** | **2** | | |
| **1** | **2** | | **0** | |
| **2** | | | **0** | **0** |

$q_0 = 0$

$F = \{0\}$



# Akzeptierte Sprache

Die Zeichen einer Zeichenfolge bewirken nacheinander Zustandsübergänge in Automaten.
**Zustandsübergangsfunktion erweitert für Zeichenfolgen**:

Sei $\delta: Q \times \Sigma \rightarrow Q$ eine **Übergangsfunktion für Zeichen**,
dann ist $\hat{\delta}: Q \times \Sigma^* \rightarrow Q$ eine **Übergangsfunktion für Wörter**, rekursiv definiert:

- Übergang mit dem **leeren Wort**: $\hat{\delta}(q, \varepsilon) = q$ für alle $q \in Q$
- Übergang mit dem **Wort wa**: $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$ für alle $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$

Statt $\hat{\delta}$ schreiben wir meist auch $\delta$.

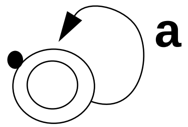
Sei $A = (\Sigma, Q, \delta, q_0, F)$ ein deterministischer endlicher Automat und $w \in \Sigma^*$.
**A akzeptiert das Wort w** genau dann, wenn $\delta(q_0, w) \in F$.
Die Menge $L(A) := \{ w \in \Sigma^* \mid \delta(q_0, w) \in F \}$ heißt die **von A akzeptierte Sprache**.

**Beispiele** für Sprachen, die von endlichen Automaten akzeptiert werden können:

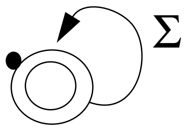
$L_1 = a^+ b^+ = \bigcup_{n, m \in \mathbb{N}} a^n b^m \qquad L_2 = \Sigma^*$

Es gibt keinen endlichen Automaten, der $L_3 = \bigcup_{n \in \mathbb{N}} a^n b^n$ akzeptiert.

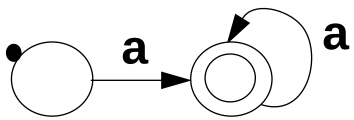
# Beispiele: Endliche Automaten und ihre Sprachen

1. 

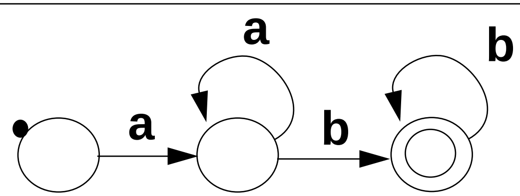

$a^*$

2. 

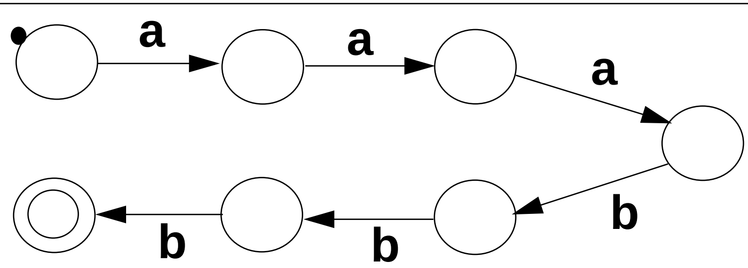

$\Sigma^*$

3. 

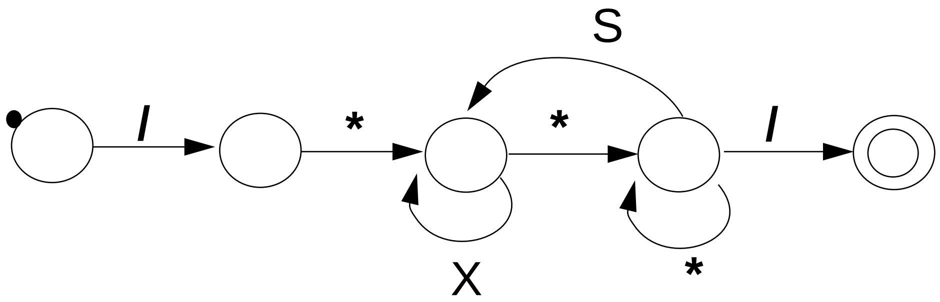

$a^+$

4. 


$a^+b^+$

5. 


$a^3b^3$

6. 


$X = \Sigma \setminus \{ * \}$, $S = \Sigma \setminus \{ /, * \}$

Sprache: $/* \ \Sigma^* \ */$

# Nicht-deterministischer Automat

**Nicht-deterministisch** (allgemein) :
Es gibt mehrere Möglichkeiten der Entscheidung bzw. der Fortsetzung,
es ist aber nicht festgelegt, welche gewählt wird.

**Nicht-deterministischer endlicher Automat**:
Die **Übergangsfunktion** $\delta$ kann einen Zustand q und ein Eingabezeichen a auf **mehrere Nachfolgezustände** abbilden $\delta : Q \times \Sigma$ -> Pow (Q).
Welcher gewählt wird, ist nicht festgelegt.
$\Sigma$ , Q, $q_0$, F sind wie für deterministische endliche Automaten definiert.

**Erweiterung von $\delta$ auf Zeichenfolgen**:

Sei A = ( $\Sigma$ , Q, $\delta$ , $q_0$, F) ein nicht-deterministischer endlicher Automat; dann ist $\hat{\delta}$ definiert:

- Übergang mit dem **leeren Wort**: $\hat{\delta}(q, \varepsilon) = \{ q \}$ für alle $q \in Q$
- Übergang mit dem **Wort wa**: $\hat{\delta}(q, wa) = \{q' \in Q \mid \exists p \in \hat{\delta}(q, w): q' \in \delta(p, a)\}$
  für alle $q \in Q$, $w \in \Sigma^*$ , $a \in \Sigma$,
  d. h. **die Menge aller Zustände, die man von q mit wa erreichen kann**

Wir schreiben meist $\delta$ für $\hat{\delta}$

Ein nicht-deterministischer endlicher Automat A **akzeptiert** ein Wort w gdw. $\delta(q_0, w) \cap F \neq \emptyset$

$L(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset \}$ ist **die von A akzeptierte Sprache**.

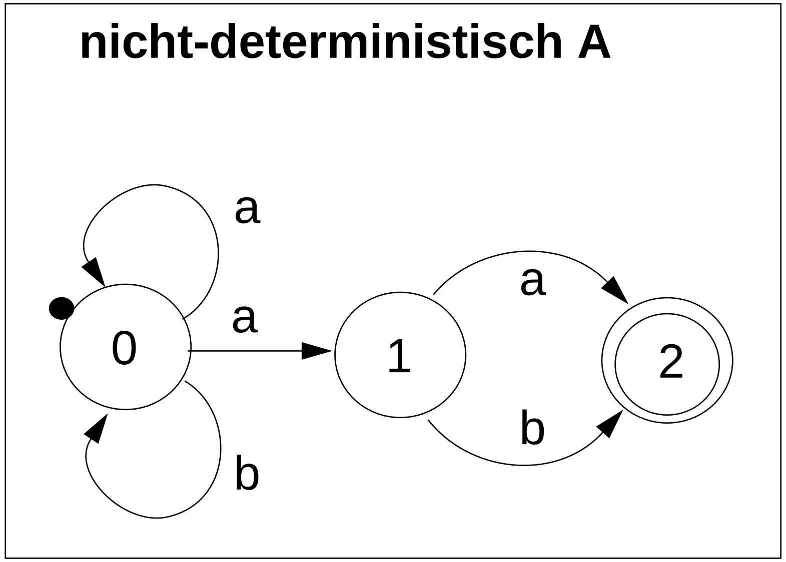
# Nicht-deterministische und deterministische Automaten

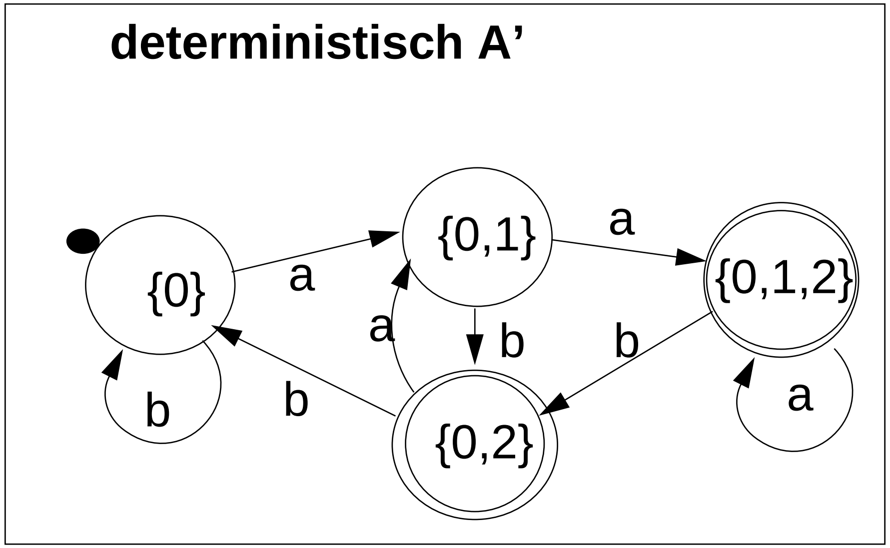
**Satz**: Sei L(A) die Sprache eines nicht-deterministischen Automaten.
Dann gibt es einen deterministischen Automaten, der L(A) akzeptiert.

Man kann **aus einem nicht-deterministischen Automaten A** = ($\Sigma$ , Q , $\delta$, $q_0$ , F) einen **deterministischen A'** = ($\Sigma$, Q', $\delta$', $q_0$', F') systematisch **konstruieren**:

Jeder **Zustand aus Q' repräsentiert eine Menge von Zuständen aus Q, d. h. Q'** $\subseteq$ **Pow(Q)**

**Beispiel**:




Die Zahl der Zustände kann sich dabei **exponentiell** vergrößern.

# Konstruktion deterministischer Automaten

Sei A ein **nicht-deterministischer Automate A** = ($\Sigma$ , Q , $\delta$, $q_0$ , F) daraus wird ein **deterministischer Automat A'** = ($\Sigma$, Q', $\delta$', $q_0$', F') systematisch **konstruiert**:

Jeder **Zustand aus Q' repräsentiert eine Menge von Zuständen aus Q, d. h. Q' $\subseteq$ Pow(Q)**

**Konstruktionsschritte**:

1. **Anfangszustand**: $q_0' = \{q_0\}$

2. Wähle einen schon konstruierten Zustand $q' \in Q'$
   wähle ein Zeichen $a \in \Sigma$

   berechne $r' = \delta'(q', a) = \bigcup_{\mathbf{q} \in \mathbf{q'}} \delta(q, a)$

   d. h. r' repräsentiert die Vereinigung aller Zustände, die in A von q unter a erreicht werden.
   r' wird **Zustand in Q'** und $\delta'(q', a) = r'$ wird **Übergang in $\delta$'**.

3. **Wiederhole (2) bis keine neuen Zustände oder Übergänge** mehr konstruiert werden können.

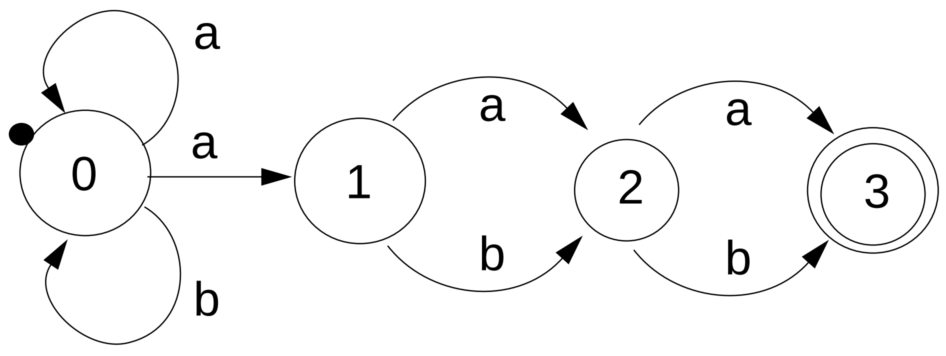
4. **Endzustände**: $F' = \{q' \in Q' \mid q' \cap F \neq \emptyset\}$
   d. h. q' ist Endzustand, wenn seine Zustandsmenge einen Endzustand von A enthält.

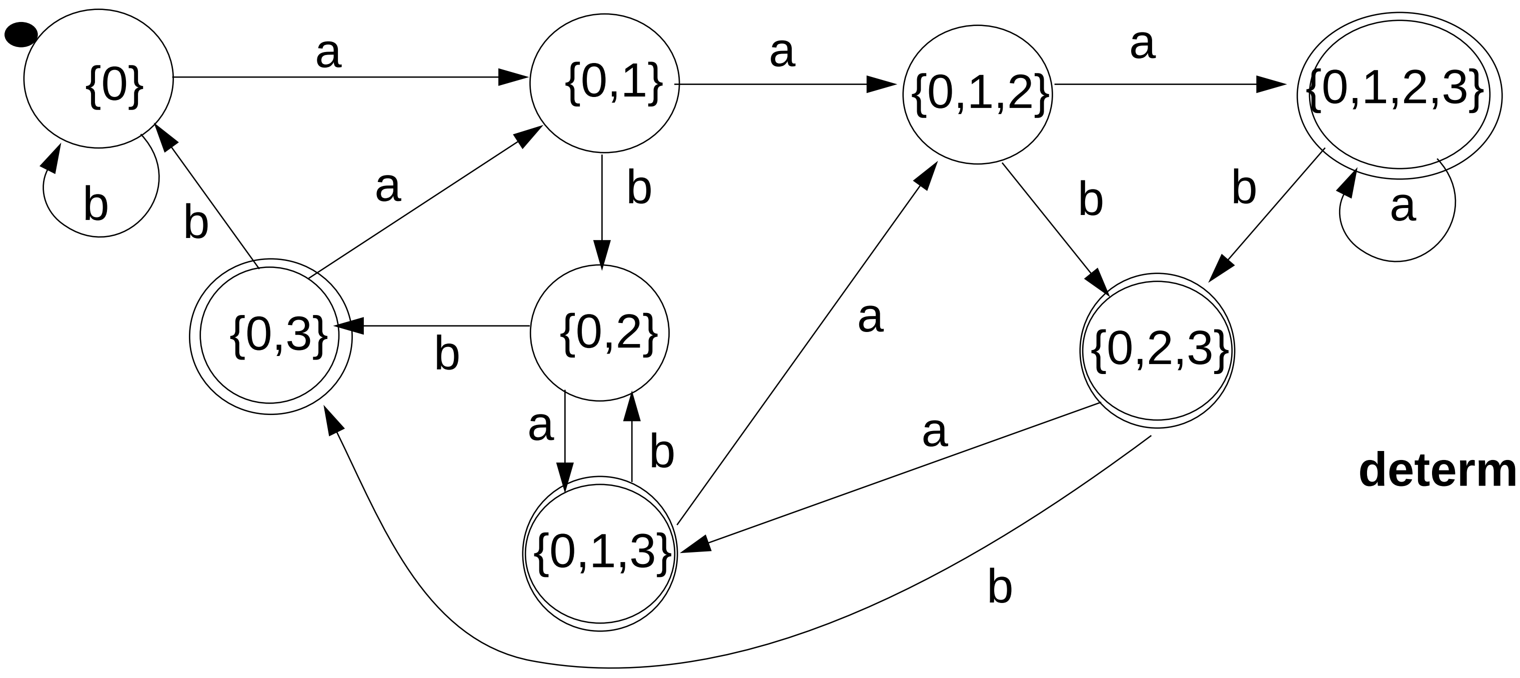
# Beispiel zur Konstruktion NDEA -> DEA

**Sprache: (a | b)* a (a | b)$^2$**

**Worte w über {a, b} mit |w| > 2 und drittletztes Zeichen ist ein a**

**nicht-deterministisch A**




**deterministisch A'**

# Endliche Automaten mit Ausgabe

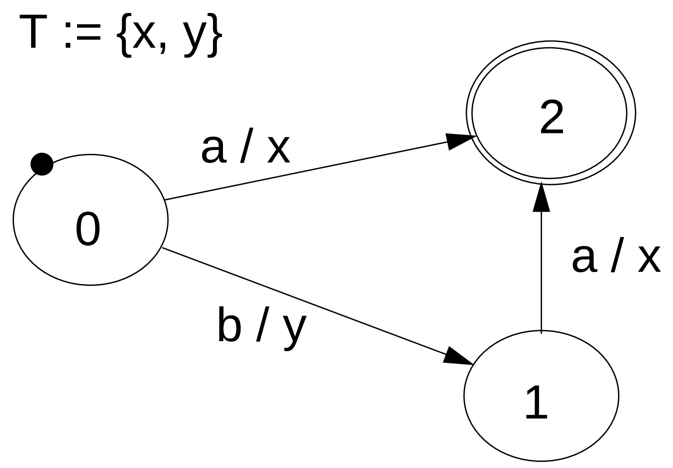
Man kann mit endlichen Automaten auch **Reaktionen der modellierten Maschine** spezifizieren: **Automaten mit Ausgabe**.

Wir erweitern den Automaten um ein **endliches Ausgabealphabet T** und um eine Ausgabefunktion. Es gibt 2 Varianten für die Ausgabefunktion:

**Mealy-Automat**:

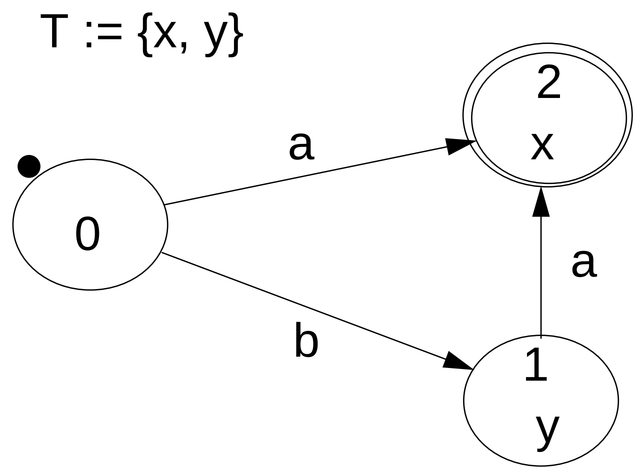
Eine Ausgabefunktion $\lambda : Q \times \Sigma \rightarrow T^*$ ordnet den **Zustandsübergängen** jeweils ein **Wort über dem Ausgabealphabet** zu.

Graphische Notation:




**Moore-Automat**:

Eine Ausgabefunktion $\mu : Q \rightarrow T^*$ ordnet den **Zuständen** jeweils ein **Wort über dem Ausgabealphabet** zu. Es wird bei Erreichen des Zustands ausgegeben.



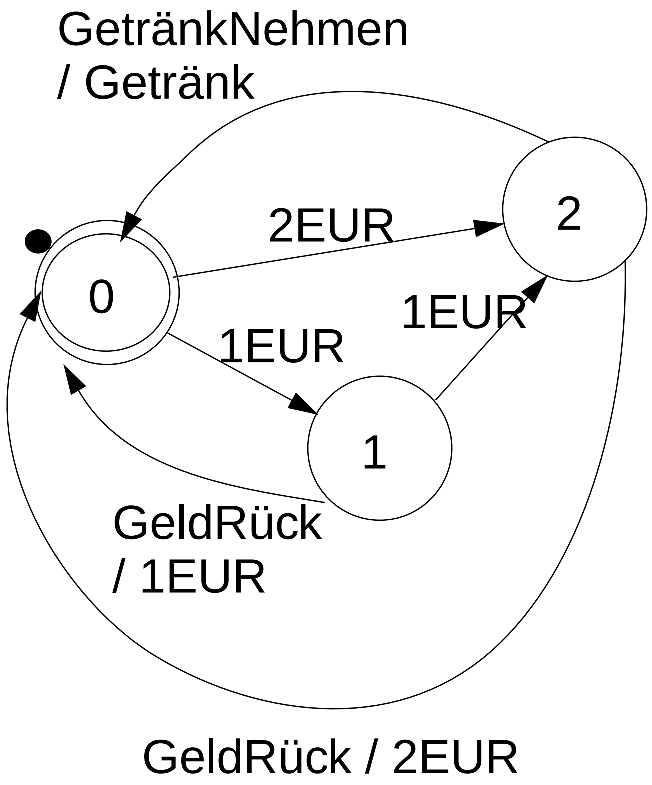

Ein **Mealy-Automat kann die Ausgabe feiner differenzieren** als ein Moore-Automat.

# Beispiele für endliche Automaten mit Ausgabe

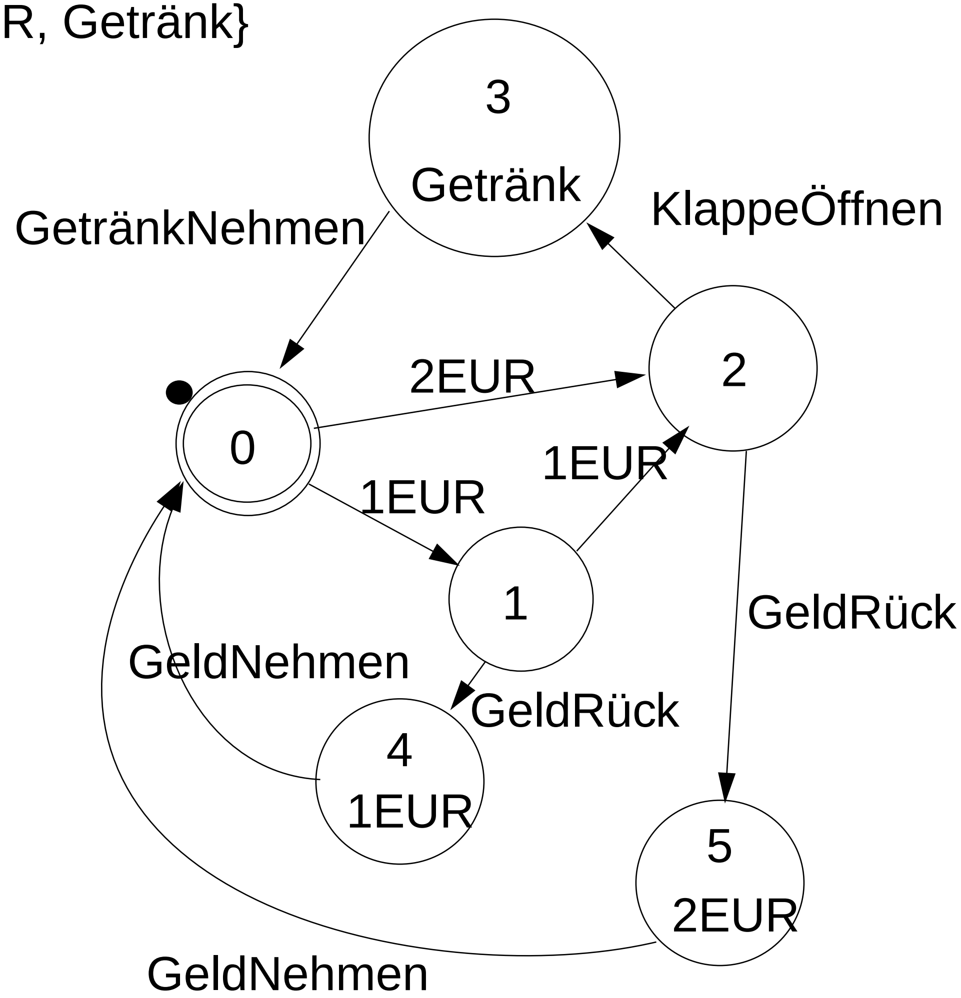
Die Spezifikation des Getränkeautomaten aus Mod-7.2 wird mit Ausgabe versehen:

T = {1EUR, 2EUR, Getränk}

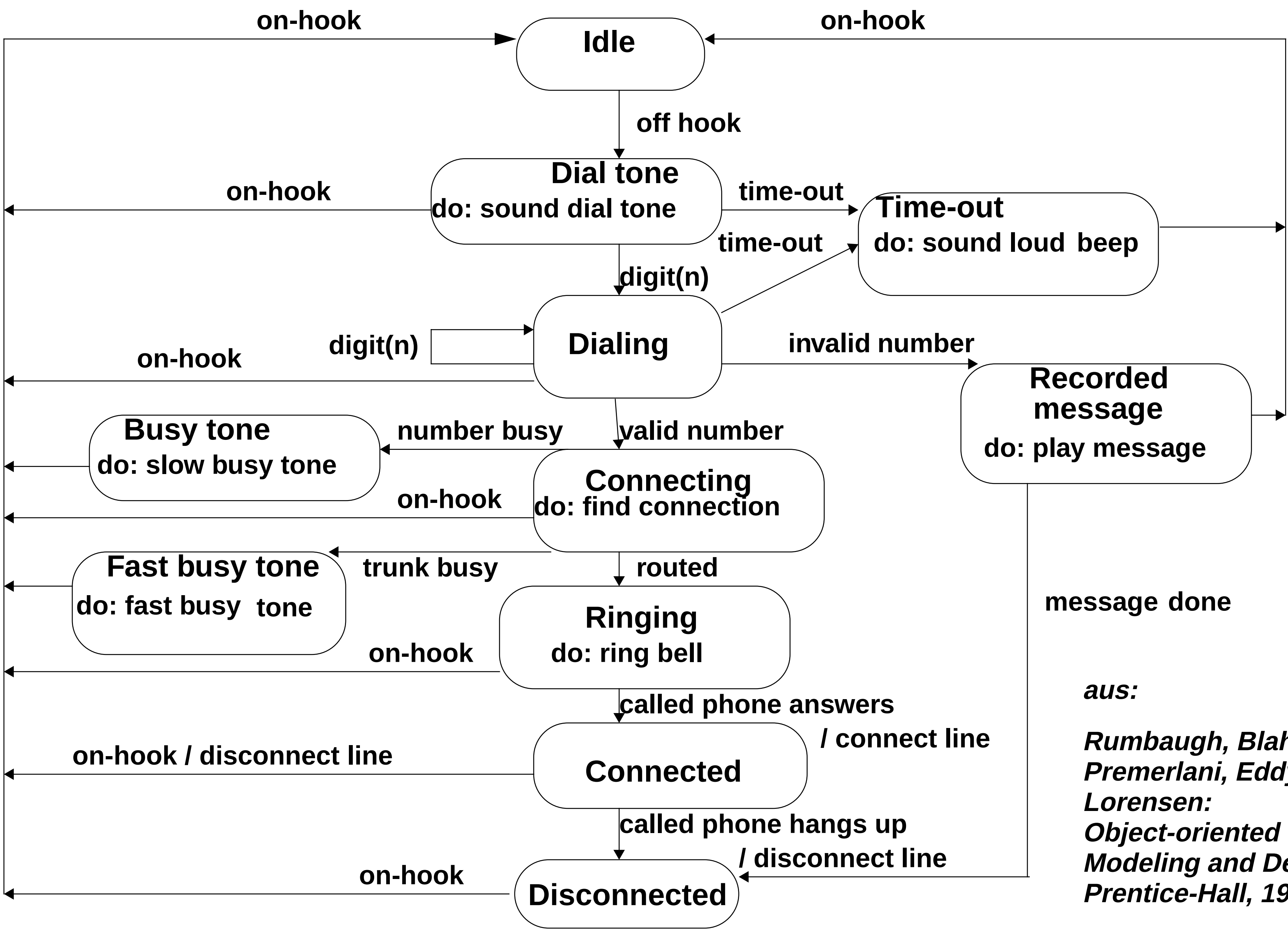
**Mealy-Automat**




**Moore-Automat**



# Endlicher Automat zur Telefonbedienung




*aus:*

***Rumbaugh, Blaha, Premerlani, Eddy, Lorensen: Object-oriented Modeling and Design, Prentice-Hall, 1991***

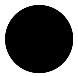
# Endliche Automaten in UML: Modell einer Uhr

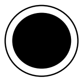
## UML Diagrammtyp Statecharts: Modellierung von Abläufen

Konzeptuelle Grundlage:
**Endliche Automaten**

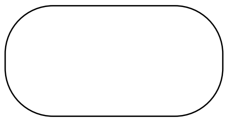
Zustände können **hierarchisch zu Teilautomaten verfeinert** werden.

Mehrere Teilautomaten können „quasi-gleichzeitig“ Übergänge ausführen - zur **Modellierung von Nebenläufigkeit**.

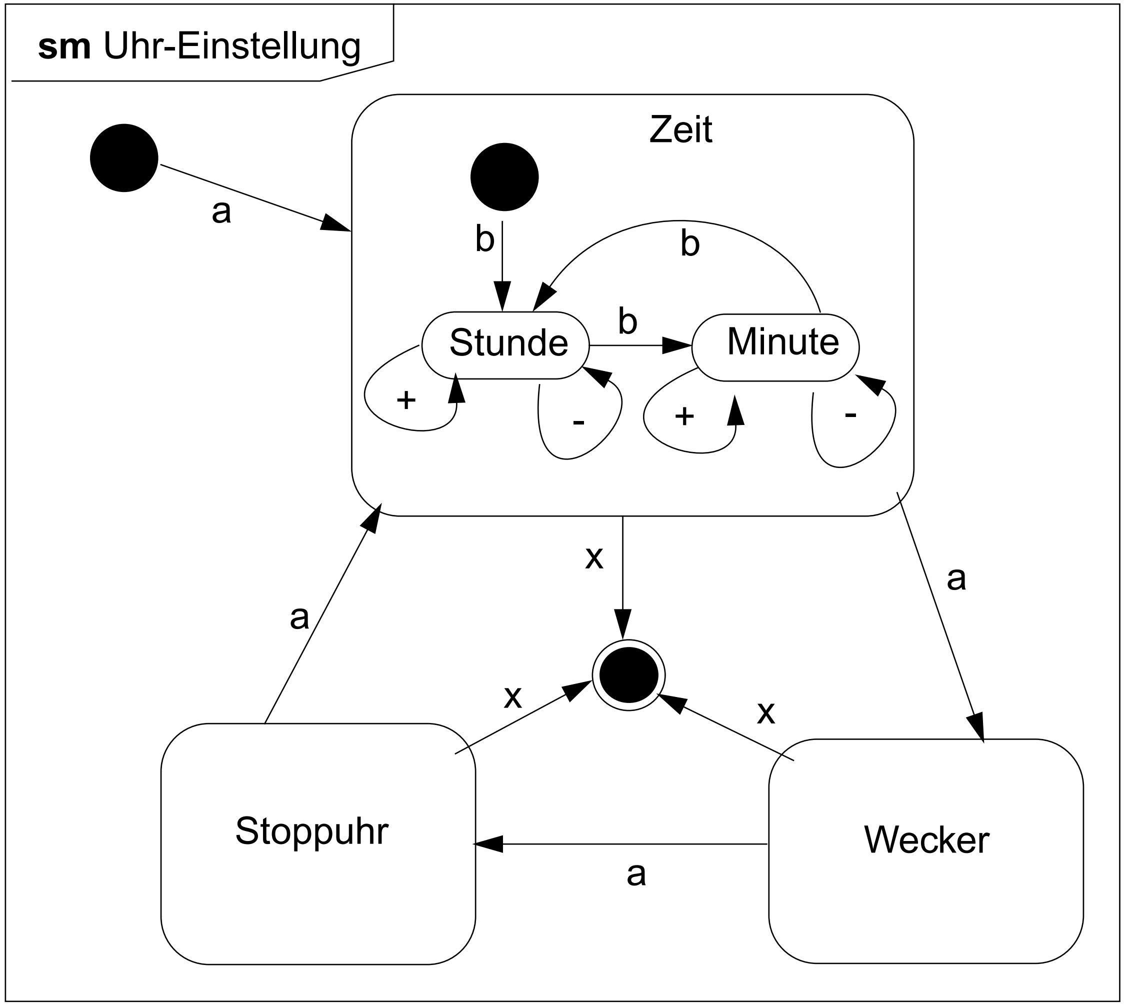

Anfangszustand


Endzustand

elementarer Zustand

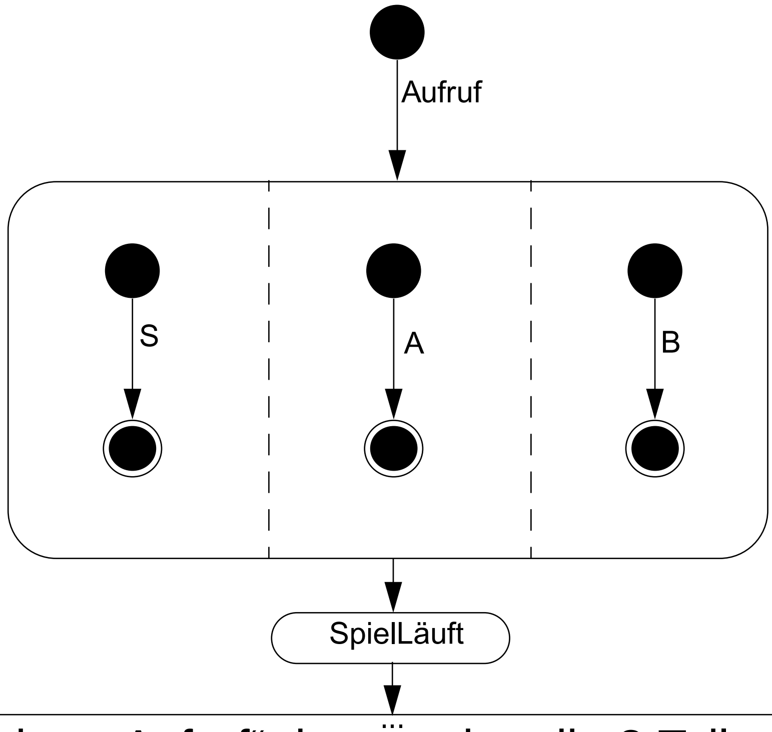

Teilautomat

## Bedienung einer Uhr
## Einstellen von Zeit, Wecker, Stoppuhr

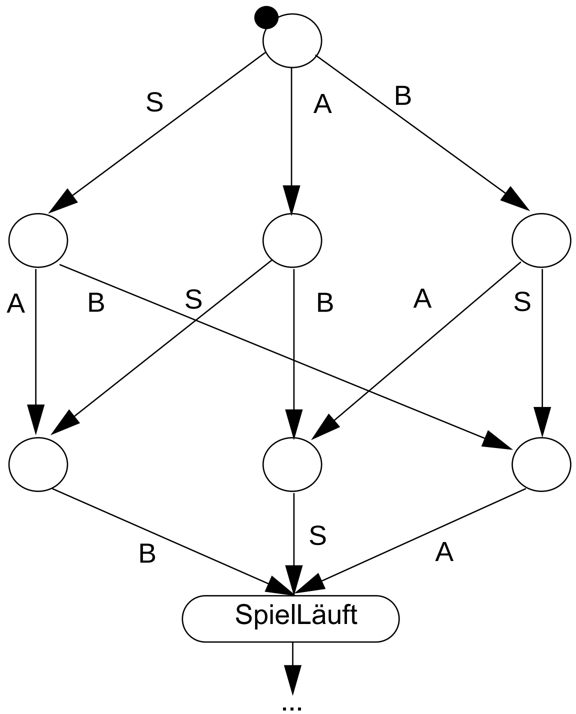


# Modellierung von Nebenläufigkeit: Beginn eines Tennisspieles

## UML Statechart




## Det. endlicher Automat




Mit dem „Aufruf“ des werden die 3 Teilautomaten des mittleren Zustandes „gleichzeitig“ aktiviert.

Sie führen jeweils einen Übergang aus (Ankunft von Schiedsrichter, Spieler A, Spieler B).

Wenn sie ihre Endzustände erreicht haben, wird der zusammengesetzte Zustand verlassen.

Der gleichbedeutende **endliche Automat** modelliert **alle Reihenfolgen der Übergänge** S, A, B.

Das **Statechart abstrahiert** davon.

# 7.2 Petri-Netze

**Petri-Netz** (auch Stellen-/Transitions-Netz):
Formaler Kalkül zur **Modellierung von Abläufen mit nebenläufigen Prozessen** und kausalen Beziehungen

Basiert auf **bipartiten gerichteten Graphen**:

- **Knoten** repräsentieren **Bedingungen**, Zustände bzw. **Aktivitäten**.
- **Kanten** verbinden **Aktivitäten** mit ihren **Vor- und Nachbedingungen**.
- **Knotenmarkierung** repräsentiert den veränderlichen **Zustand des Systems**.
- **graphische Notation**.

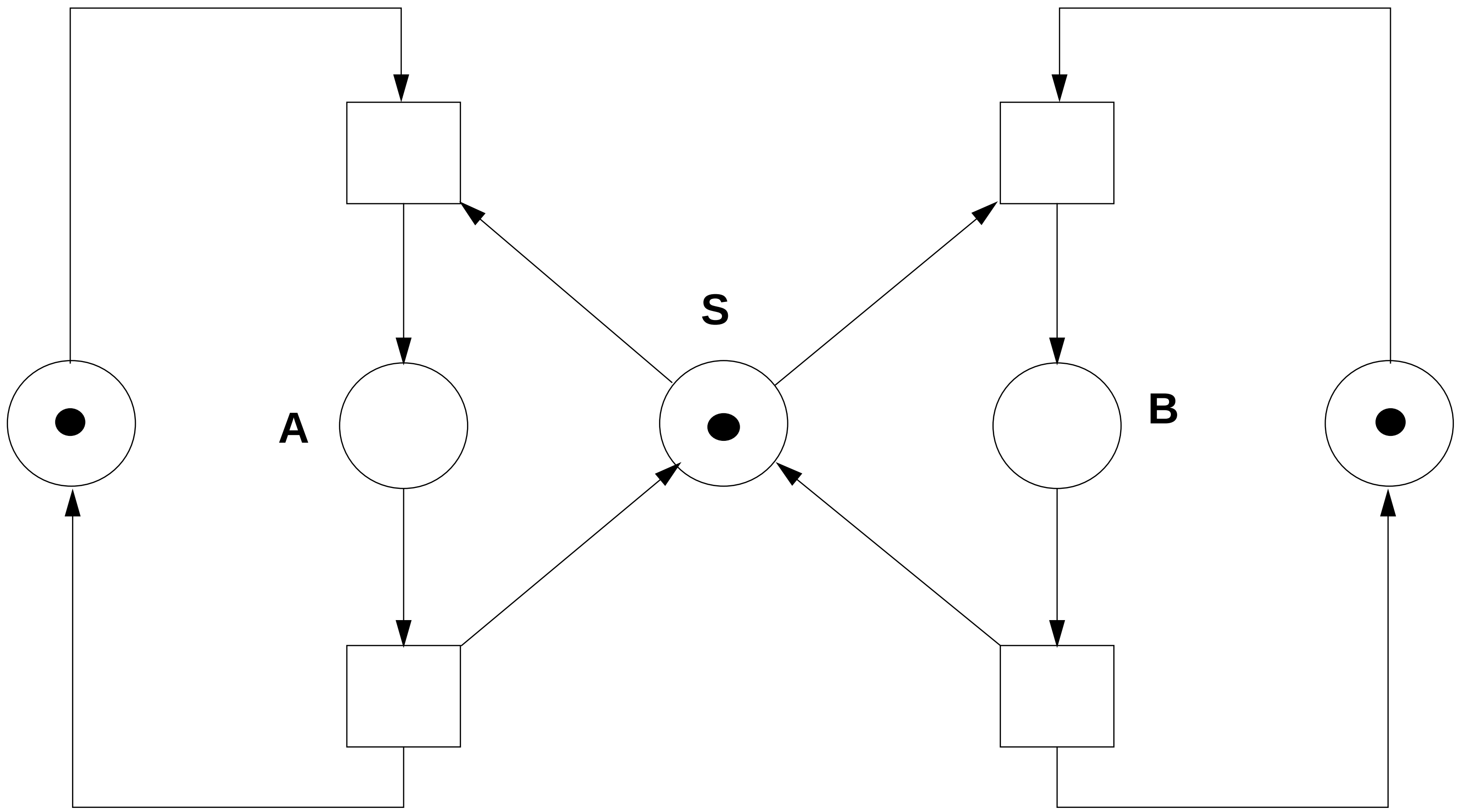
C. A. Petri hat sie 1962 eingeführt.
Es gibt zahlreiche Varianten und Verfeinerungen von Petri-Netzen. Hier nur die Grundform.

**Anwendungen** von Petri-Netzen zur Modellierung von

- realen oder abstrakten Automaten und Maschinen
- kommunizierenden Prozessen in der Realität oder in Rechnern
- Verhalten von Hardware-Komponenten
- Geschäftsabläufe
- Spielpläne

# Einführendes Beispiel

Das Petri-Netz modelliert zwei **zyklisch ablaufende Prozesse**.
Die mittlere Stelle synchronisiert die beiden Prozesse,
so dass sie sich **nicht zugleich in den Zuständen A und B** befinden können.
Prinzip: **gegenseitiger Ausschluss** durch **Semaphor**



# Definition von Petri-Netzen

Ein **Petri-Netz** ist ein Tripel P = (S, T, F) mit

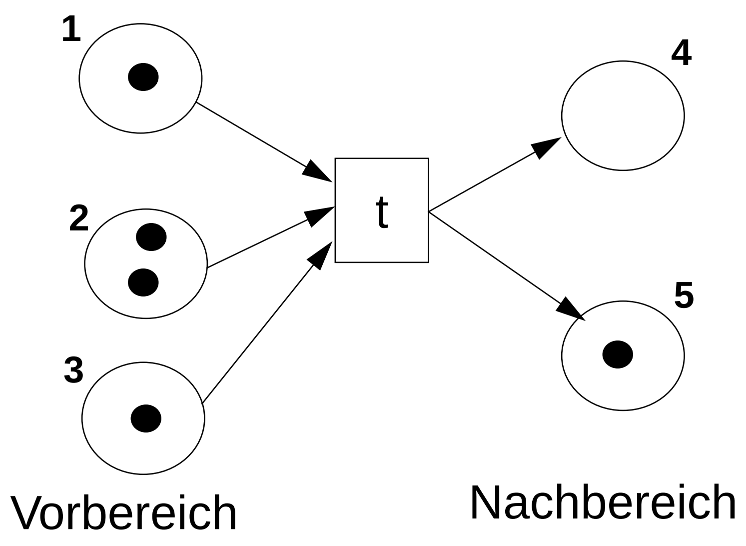
S **Menge von Stellen**,
repräsentieren Bedingungen, Zustände; graphisch Kreise

T **Menge von Transitionen** oder Übergänge,
repräsentieren Aktivitäten; graphisch Rechtecke

F **Relation** mit $F \subseteq S \times T \cup T \times S$
repräsentieren kausale oder zeitliche Vor-, Nachbedingungen von Aktivitäten aus T

P bildet einen **bipartiten, gerichteten Graphen** mit den Knoten $S \cup T$ und den Kanten F.

Zu einer **Transition t** in einem Petri-Netz P sind folgende Stellenmengen definiert
**Vorbereich (t)** $:= \{ s \mid (s, t) \in F\}$
**Nachbereich (t)** $:= \{ s \mid (t, s) \in F\}$




Der **Zustand des Petri-Netzes** wird durch eine **Markierungsfunktion** angegeben, die jeder Stelle eine **Anzahl von Marken** zuordnet:
$M_P: S \rightarrow \mathbb{N}_0$
Sind die Stellen von 1 bis n nummeriert, so kann man $M_P$ als Folge angeben, z. B. (1, 2, 1, 0, 1)

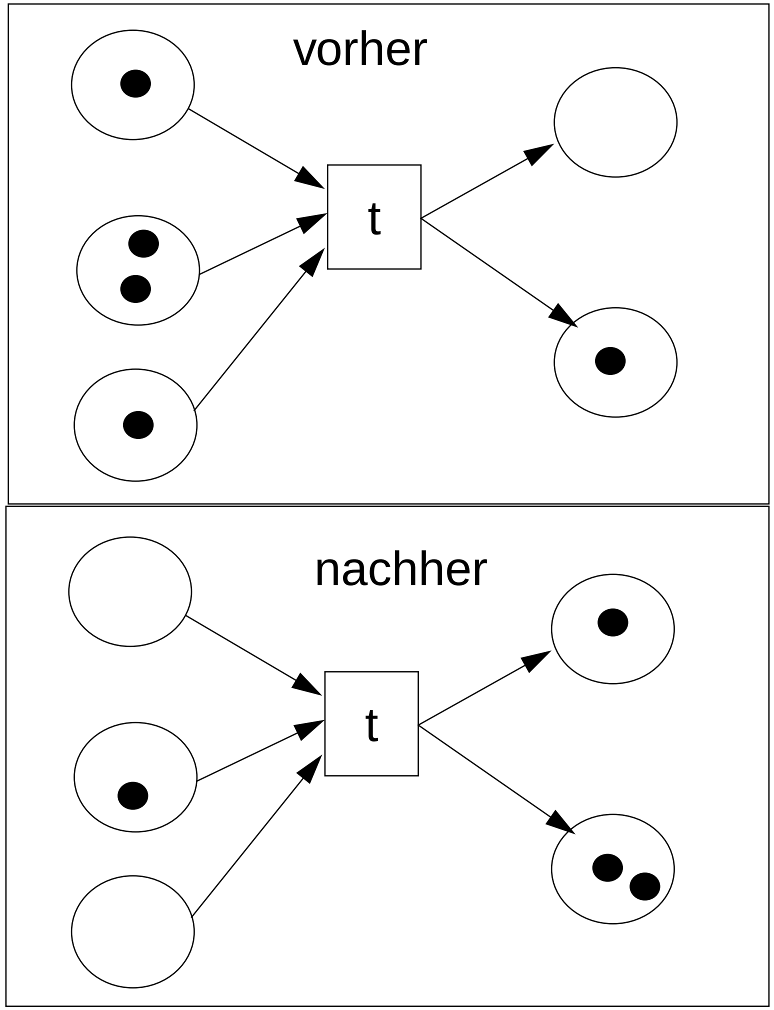
# Schaltregel für Petri-Netze

Das **Schalten einer Transition** t überführt eine Markierung M in eine Markierung M‘.

Eine **Transition t kann schalten**, wenn für alle Stellen $s \in$ Vorbereich (t) gilt $M(s) \geq 1$.

Wenn eine Transition t **schaltet,** gilt für die **Nachfolgemarkierung M‘**:

| | |
|---|---|
| $M'(v) = M(v) - 1$ | für alle $v \in$ Vorbereich(t) \ Nachbereich(t) |
| $M'(n) = M(n) + 1$ | für alle $n \in$ Nachbereich(t) \ Vorbereich(t) |
| $M'(s) = M(s)$ | sonst |




Wenn in einem Schritt **mehrere Transitionen schalten können**, wird eine davon **nicht-deterministisch ausgewählt**.

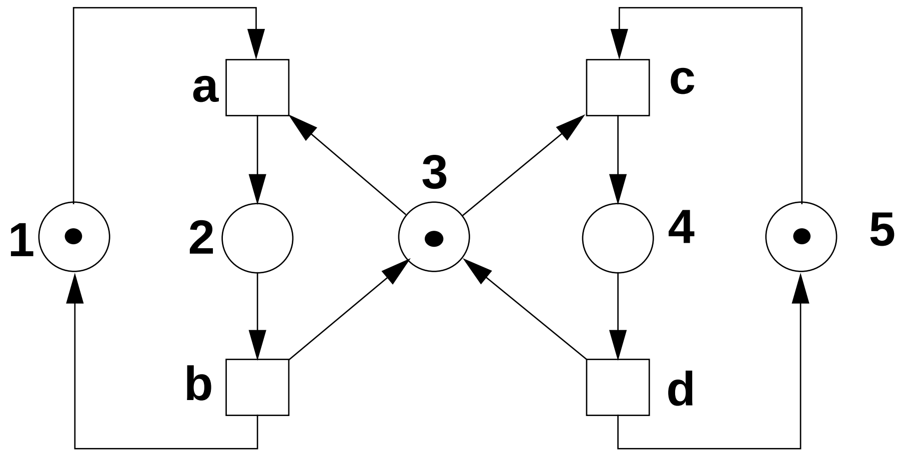
**In jedem Schritt schaltet genau eine Transition**
- auch wenn das Petri-Netz parallele Abläufe modelliert!

Zwei Transitionen mit gemeinsamen Stellen im Vorbereich können (bei passender Markierung) im **Konflikt** stehen:
Jede kann schalten, aber nicht beide nacheinander.

# Markierungen

Zu jedem Petri-Netz wird eine **Anfangsmarkierung $M_0$** angeben.

z. B. $\mathbf{M_0} = (1, 0, 1, 0, 1)$

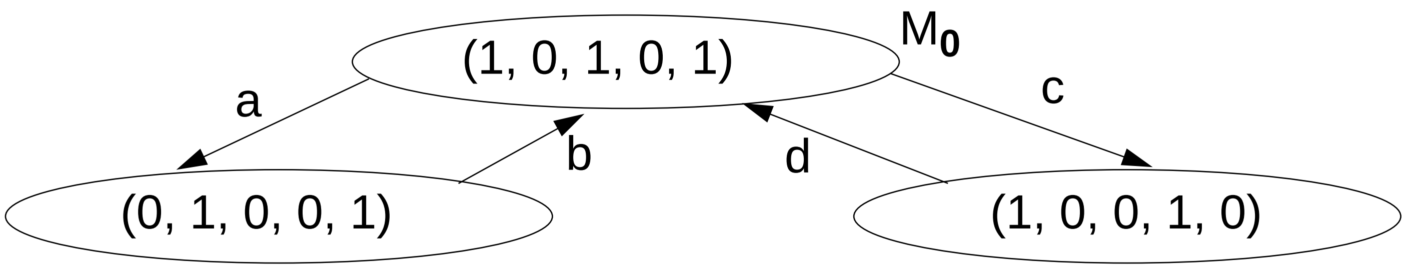



Wir sagen, eine **Markierung $M_2$ ist von einer Markierung $M_1$ aus erreichbar**, wenn es ausgehend von $M_1$ eine Folge von Transitionen gibt, die nacheinander schalten und $M_1$ in $M_2$ überführen können.

Die Markierungen eines Petri-Netzes kann man als gerichteten **Markierungsgraphen** darstellen:

- Knoten: erreichbare Markierung
- Kante x->y: Die Markierung x kann durch Schalten einer Transition in y übergehen.

**Markierungsgraph**



# Schaltfolgen

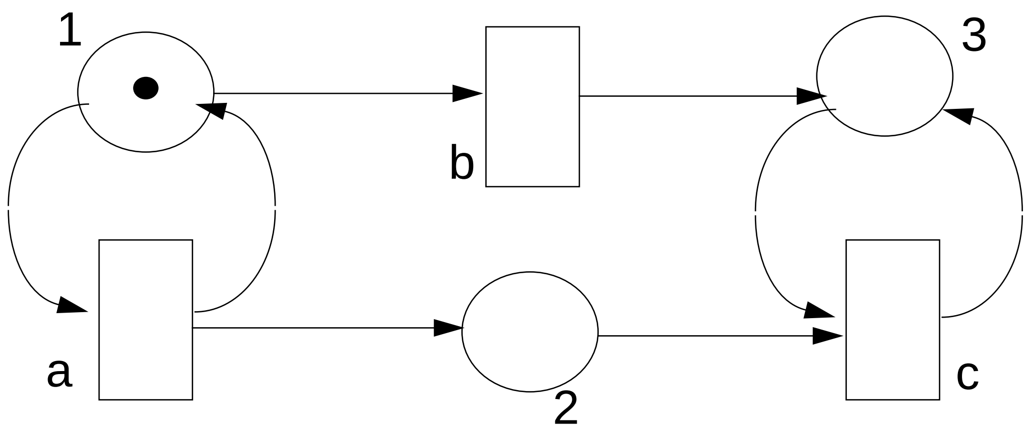
**Schaltfolgen** kann man angeben als

- Folge von Markierungen
- Folge der geschalteten Transitionen

Beispiel für eine **Schaltfolge**
zum Petri-Netz auf Mod-7.19:

| | |
|---|---|
| (1, 0, 1, 0, 1) | a |
| (0, 1, 0, 0, 1) | b |
| (1, 0, 1, 0, 1) | c |
| (1, 0, 0, 1, 0) | d |
| (1, 0, 1, 0, 1) | |

**Schaltfolgen können als Wörter einer Sprache** aufgefasst werden.




alle Schaltfolgen ohne Nachfolgemarkierung haben die Form:

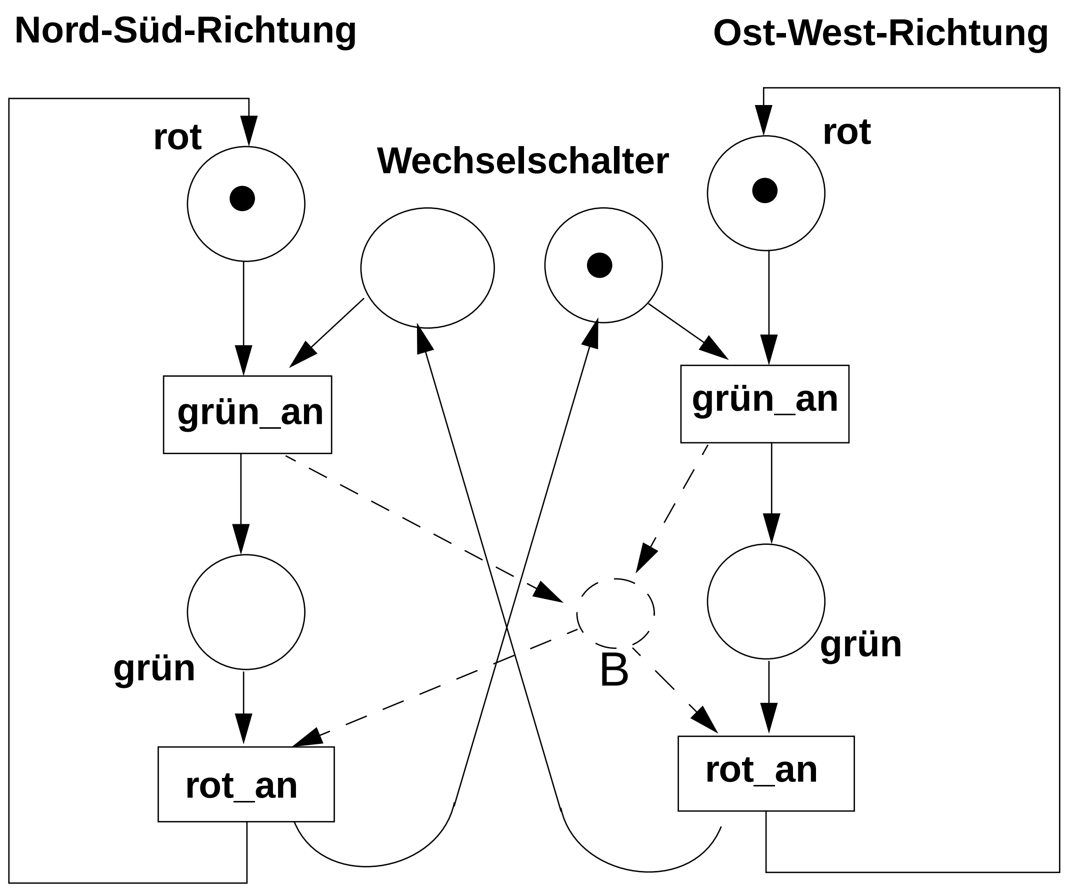
$a^n\, b\, c^n$

Petri-Netze können unbegrenzt zählen: Anzahl der Marken auf einer Stelle.

# Modellierung alternierender zyklischer Prozesse

**Beispiel**: Einfache Modellierung einer Ampelkreuzung:

- 2 sich zyklisch wiederholende Prozesse
- Die beiden Stellen „Wechselschalter“ koppeln die Prozesse, sodass sie alternierend fortschreiten.
- Alle Stellen repräsentieren Bedingungen: 1 oder 0 Marken
- „Beobachtungsstelle“ B modelliert, wieviele Richtungen „grün“ haben

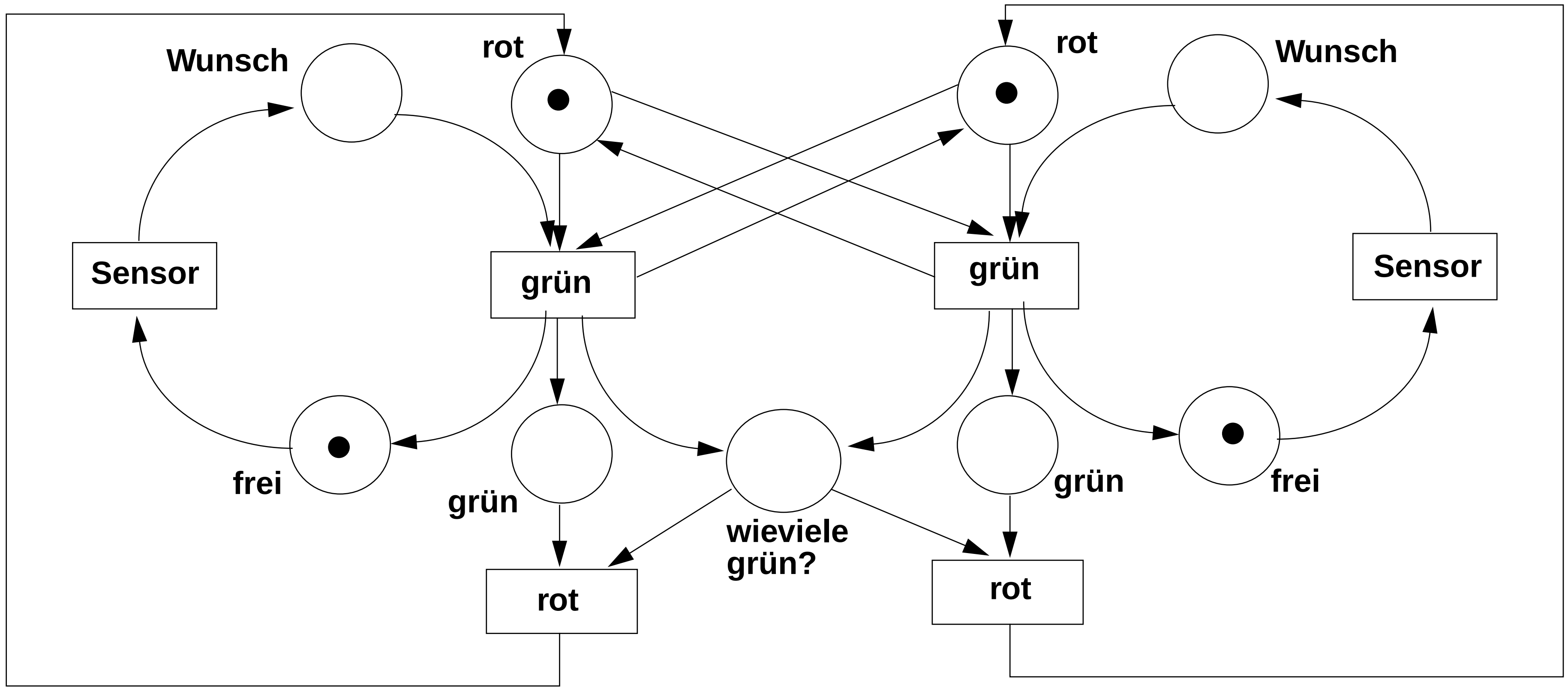


# Beispiel für ein binäres Netz

Ein Petri-Netz heißt **binär (sicher)**, wenn für alle aus $M_0$ erreichbaren Markierungen M und für alle Stellen s gilt $M(s) \leq 1$.

Petri-Netze, deren **Stellen Bedingungen repräsentieren** müssen binär sein.

**Beispiel**: Modellierung einer Sensor-gesteuerten Ampelkreuzung:




aus: B. Baumgarten: Petri-Netze, Bibliographisches Institut & F. A. Brockhaus AG, 1990

# Lebendige Petri-Netze

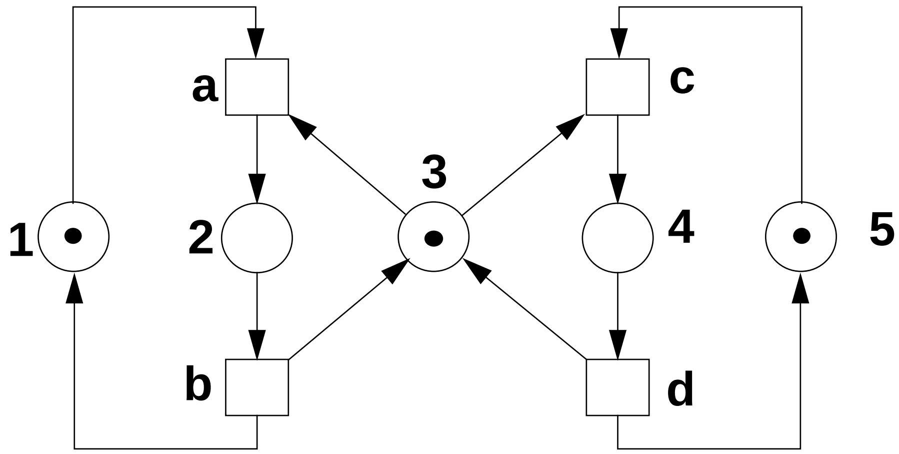
Petri-Netze modellieren häufig **Systeme, die nicht anhalten** sollen.

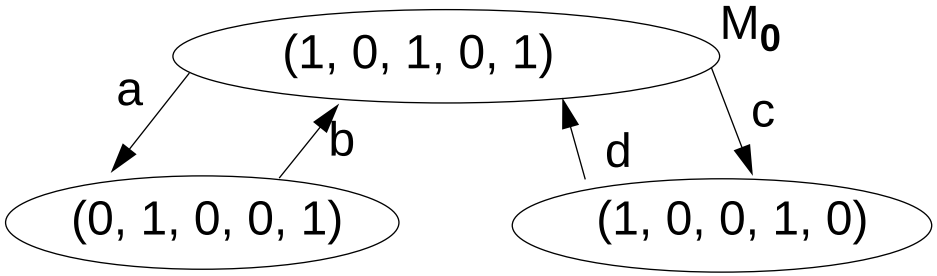
Ein Petri-Netz heißt **schwach lebendig**, wenn es zu jeder von $M_0$ erreichbaren Markierung eine Nachfolgemarkierung gibt.

Eine **Transition t heißt lebendig**, wenn es zu jeder von $M_0$ erreichbaren Markierung M' eine Markierung M" gibt, die von M' erreichbar ist, und in der t schalten kann.

Ein **Petri-Netz heißt lebendig**, wenn alle seine Transitionen lebendig sind.

Beispiel für ein **lebendiges Petri-Netz** (Mod-7.19):



# Verklemmungen

**Verklemmung**: Ein System kann unerwünscht anhalten,
weil das **Schalten einiger Transitionen zyklisch voneinander abhängt**.

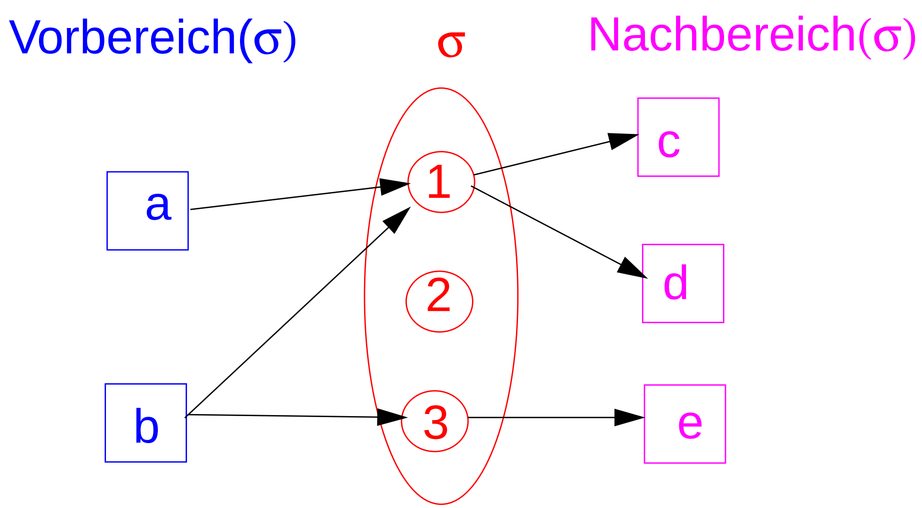
Sei: $\sigma \subseteq S$ eine Teilmenge der Stellen eines Petri-Netzes und

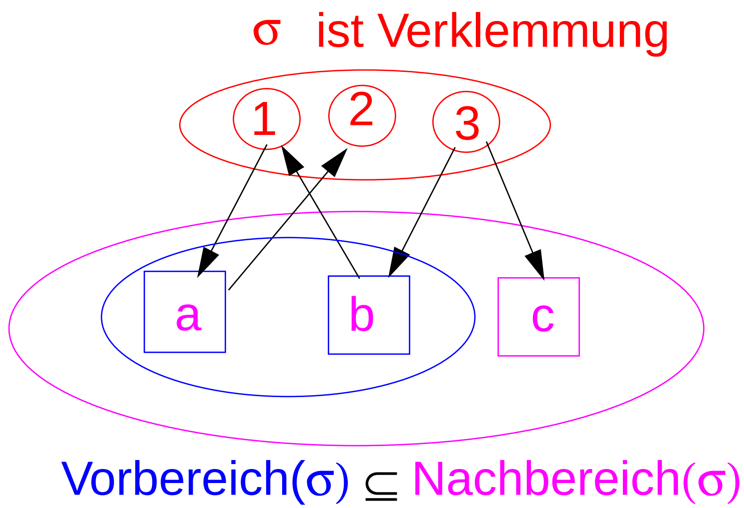
Vorbereich ($\sigma$) := $\{t \mid \exists\, s \in \sigma : (t, s) \in F\}$,
d. h. die Transitionen, die auf Stellen in $\sigma$ wirken

Nachbereich ($\sigma$) := $\{t \mid \exists\, s \in \sigma : (s, t) \in F\}$,
d. h. die Transitionen, die Stellen in $\sigma$ als Vorbedingung haben

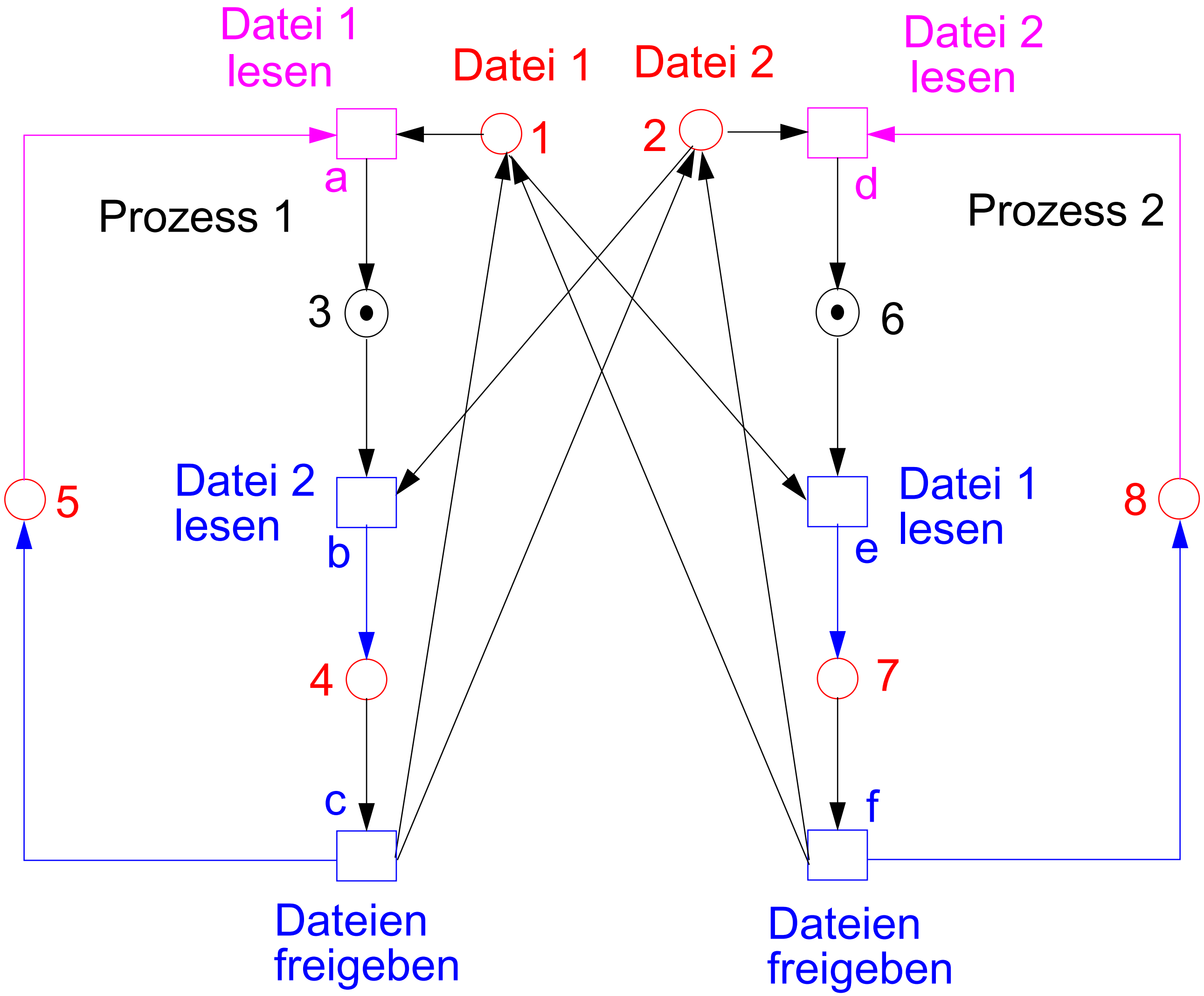
Dann ist $\sigma$ eine **Verklemmung, wenn Vorbereich** ($\sigma$) $\subseteq$ **Nachbereich** ($\sigma$).

Wenn **für alle s** $\in$ **$\sigma$ gilt M (s) = 0**, dann kann es **keine Marken auf Stellen in** $\sigma$ in einer Nachfolgemarkierung von M geben.



# Verklemmung beim Lesen von Dateien




s = {1, 2, 4, 5, 7, 8}

Vorbereich (s)
= {b, c, e, f}

Nachbereich (s)
= {a, b, c, d, e, f}

M (s) = 0

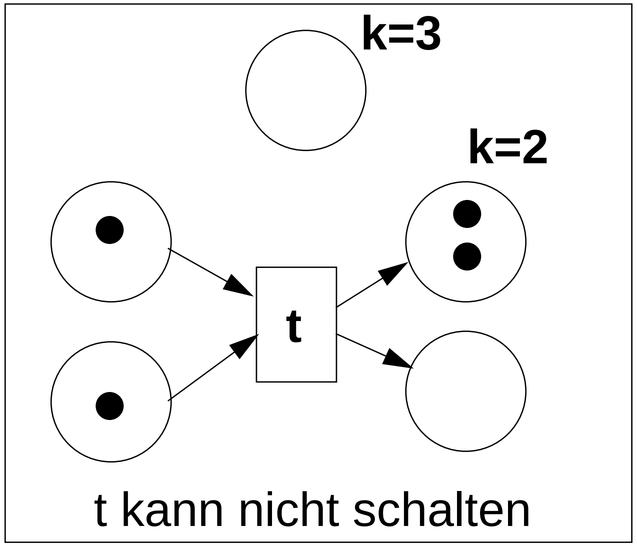
Anfangsmarkierung:
(1, 1, 0, 0, 1, 0, 0, 1)

# Kapazitäten und Gewichte

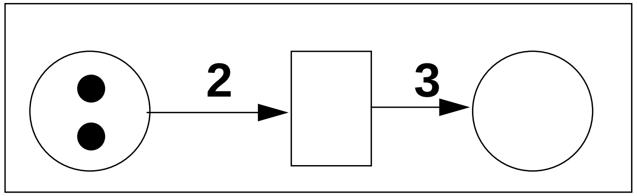
Man kann **Stellen eine begrenzte Kapazität** von $k \in \mathbb{N}$ Marken zuordnen.

Die Bedingung, dass eine **Transition t schalten kann**, wird erweitert um:

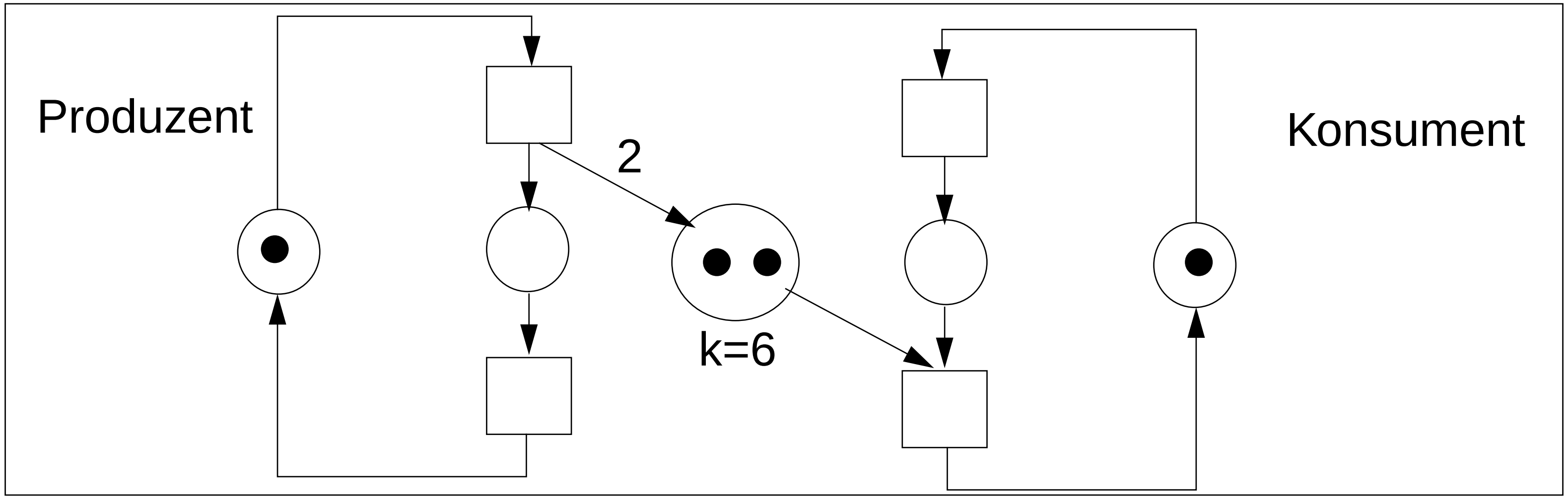
> Die **Kapazität keiner der Stellen im Nachbereich von t darf überschritten** werden.




**Kanten** kann ein **Gewicht** $n \in \mathbb{N}$ zugeordnet werden: sie bewegen **beim Schalten n Marken**.




Beispiel: **Beschränkter Puffer**



# Beispiel: Leser-Schreiber-System

**n Leser-Prozesse** und **m Schreiber-Prozesse** operieren **auf derselben Datei**.

Mehrere **Leser können zugleich lesen**.

Ein **Schreiber** darf nur dann schreiben, wenn **kein anderer Leser oder Schreiber aktiv** ist.

Modellierung: ein **Schreiber entzieht der Synchronisationsstelle alle n Marken**.