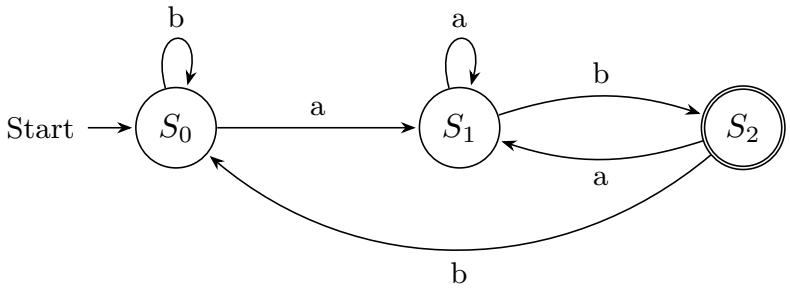
ARCHENHOLD

GYMNASIUM

Sprachen und Automaten

Begleitskript zum Informatikunterricht

Endliche Automaten · Minimieren · Grammatiken · Kellerautomaten · Turingmaschine



Ein Automat, der alle Wörter erkennt, die auf `ab` enden

Inhaltsverzeichnis

1 Endliche Automaten **3**

1.1 Was ist ein Automat? . . . 3

1.2 Moore-Automaten . . . 3

1.3 Mealy-Automaten . . . 5

2 Der DEA – Automat ohne Ausgabe **6**

2.1 Definition und Abgrenzung . . . 6

2.2 Startbeispiel: ein Barcodescanner . . . 6

2.3 Zweites Beispiel: Strichbreiten lesen . . . 7

2.4 Akzeptoren und ihre Sprache . . . 9

3 Einen DEA minimieren **10**

4 Nichtdeterminismus und reguläre Sprachen **11**

4.1 Formale Sprachen . . . 11

4.2 Nichtdeterministische Automaten (NEA) . . . 11

4.3 Vom NEA zum DEA: die Teilmengenkonstruktion . . . 11

4.4 Zweites Beispiel: Wörter, die *abb* enthalten . . . 12

4.5 Grenzen endlicher Automaten . . . 13

5 Grammatiken **14**

5.1 Beispiele . . . 14

5.2 Die Chomsky-Hierarchie . . . 14

5.3 Vom Automaten zur regulären Grammatik . . . 15

6 Kellerautomaten **17**

6.1 Beispiel $a^n b^n$ . . . 17

6.2 Beispiel Klammerterme . . . 18

6.3 Aus einer Grammatik wird ein Kellerautomat . . . 18

7 Die Turingmaschine **19**

7.1 Aufbau . . . 19

7.2 Beispiel: unärer Inkrementer . . . 19

7.3 Beispiel: binärer Inkrementer . . . 19

7.4 Beispiel: Akzeptor für $a^n b^n$ . . . 20

7.5 Was die Turingmaschine bedeutet . . . 21

8 Überblick **22**

9 Übungen **23**

10 Lösungen **24**

1 Endliche Automaten

So arbeiten Sie mit dem Skript: Jeder Automatentyp wird erklärt und an Beispielen durchgerechnet. Die Kästen *„auf einen Blick“* fassen ihn zusammen. Abschnitt 9 enthält Übungen, Abschnitt 10 die Lösungen. Alle Diagramme und Tabellen sind mit einem Programm aus denselben Übergangstabellen erzeugt und an allen kurzen Wörtern geprüft. Zum Ausprobieren eignet sich **FLACI** im Browser: `https://flaci.com`

1.1 Was ist ein Automat?

Ein Automat steuert sein Verhalten selbst: Er hat *innere Zustände*, und seine Reaktion hängt von der Eingabe *und* vom aktuellen Zustand ab (ein Schaltnetz dagegen nur von der Eingabe). Formal ist ein Automat ein Tupel $A = (X, Y, S, s_0, \delta, \omega)$:

X	Eingabealphabet (endliche, nicht leere Menge von Zeichen)
Y	Ausgabealphabet
S	endliche, nicht leere Menge von Zuständen, $s_0 \in S$ ist der Startzustand
$\delta : S \times X \to S$	Übergangsfunktion: aus Zustand und Zeichen wird der Folgezustand
ω	Ausgabefunktion: $\omega : S \to Y$ (*Moore*) bzw. $\omega : S \times X \to Y$ (*Mealy*)

Automaten mit Ausgabe gibt es in zwei Formen:

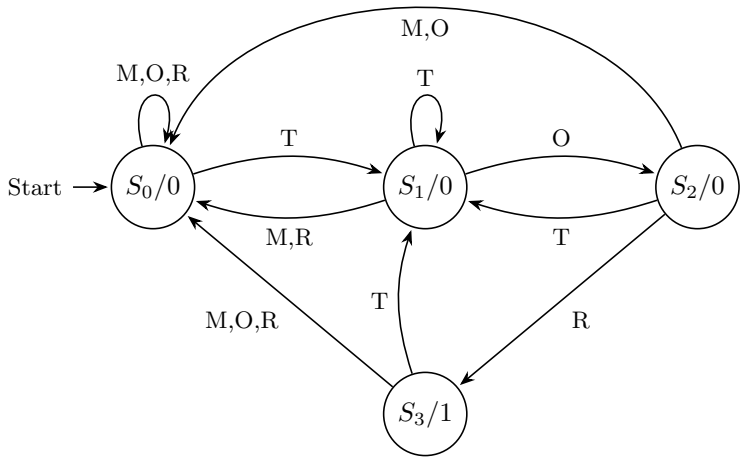
Moore-Automaten geben in jedem *Zustand* etwas aus. Die Ausgabe steht im Diagramm im Zustand (z. B. $S_3/1$).

Mealy-Automaten geben bei jedem *Übergang* etwas aus, die Ausgabe hängt also auch von der Eingabe ab. Sie steht an der Kante (*Eingabe/Ausgabe*).

Zu jedem Mealy-Automaten gibt es einen gleichwertigen Moore-Automaten und umgekehrt; der Moore-Automat braucht meist mehr Zustände. Der wichtigste Automat dieses Skripts hat dagegen *gar keine* Ausgabe: der DEA (Abschnitt 2).

1.2 Moore-Automaten

Beispiel TOR: Auf einer Leitung kommen die Buchstaben M, O, T, R an. Der Automat soll jedes Auftreten von `TOR` melden, also 1 ausgeben, wenn die Eingabe zuletzt auf `TOR` endete. Die Ausgabe steht im Zustand.

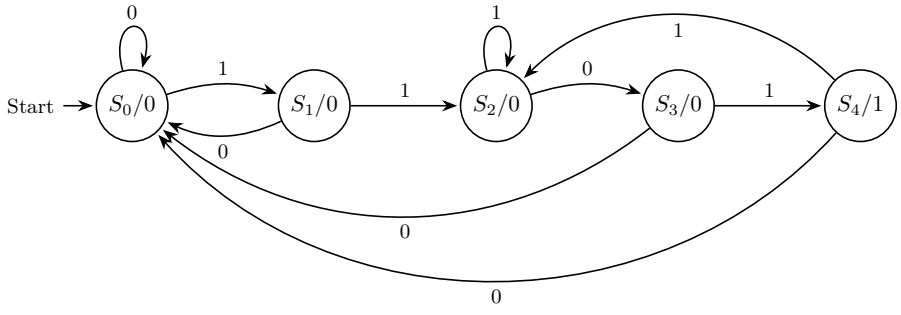


δ	M	O	T	R	Ausgabe
S_0 (Start)	S_0	S_0	S_1	S_0	0
S_1	S_0	S_2	S_1	S_0	0
S_2	S_0	S_0	S_1	S_3	0
S_3	S_0	S_0	S_1	S_0	1

Formal: $A = (X, Y, S, s_0, \delta, \omega)$ mit $X = \{M, O, T, R\}$; $S = \{S_0, S_1, S_2, S_3\}$; $s_0 = S_0$; $Y = \{0, 1\}$; δ und $\omega : S \to Y$: siehe Tabelle.

Aus S_3 führt ein T sofort nach S_1, denn es könnte ein neues `TOR` beginnen. Alle anderen Zeichen führen zurück nach S_0.

Beispiel 1101: Erkannt werden soll das Bitmuster 1101 (die Zahl 13) am Ende der Eingabe. Mit Startzustand braucht man fünf Zustände (S_k: die letzten Zeichen stimmen mit den ersten k Zeichen von 1101 überein).



δ	0	1	Ausgabe
S_0 (Start)	S_0	S_1	0
S_1	S_0	S_2	0
S_2	S_3	S_2	0
S_3	S_0	S_4	0
S_4	S_0	S_2	1

Ein Lauf mit Zustands- und Ausgabefolge: Die Ausgabe 1 erscheint genau nach jedem `1101`.

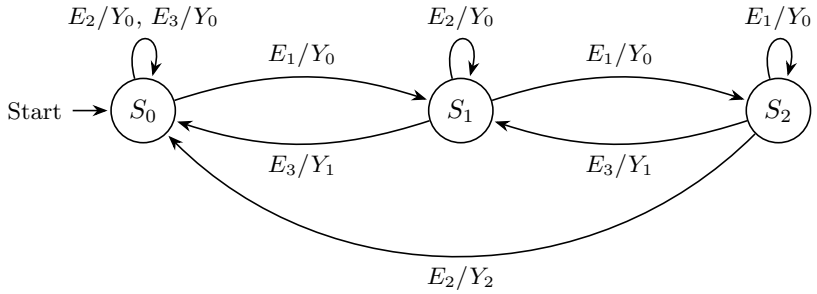
gelesen	–	0	1	1	0	1	0	1	1	0
Zustand	S_0	S_0	S_1	S_2	S_3	S_4	S_0	S_1	S_2	S_3
Ausgabe	0	0	0	0	0	1	0	0	0	0

Beachten Sie die Übergänge zurück: Nach 1101 und einer weiteren 1 stehen am Ende `11`, das sind schon zwei passende Zeichen ($S_4 \xrightarrow{1} S_2$). Man fragt immer: *Welcher längste Anfang des Musters steht gerade am Ende?*

1.3 Mealy-Automaten

Beispiel Kaffeeautomat: Der Automat nimmt nur 1-Euro-Münzen an, höchstens 2 Euro. Taste KK gibt einen kleinen Kaffee für 1 Euro, Taste KG einen großen für 2 Euro. Die Ausgabe hängt vom Guthaben (Zustand) *und* von der gedrückten Taste ab – also ein Mealy-Automat.

Zustände	Eingaben	Ausgaben
S_0: 0 Euro eingeworfen	E_1: 1-Euro-Münze	Y_0: keine Ausgabe
S_1: 1 Euro eingeworfen	E_2: Taste KG (2 Euro)	Y_1: Kaffee klein
S_2: 2 Euro eingeworfen	E_3: Taste KK (1 Euro)	Y_2: Kaffee groß



$\delta \mid \omega$	E_1	E_2	E_3
S_0	$S_1 \mid Y_0$	$S_0 \mid Y_0$	$S_0 \mid Y_0$
S_1	$S_2 \mid Y_0$	$S_1 \mid Y_0$	$S_0 \mid Y_1$
S_2	$S_2 \mid Y_0$	$S_0 \mid Y_2$	$S_1 \mid Y_1$

Formal: $M = (X, Y, S, s_0, \delta, \omega)$ mit $X = \{E_1, E_2, E_3\}$; $S = \{S_0, S_1, S_2\}$; $s_0 = S_0$; $Y = \{Y_0, Y_1, Y_2\}$; δ und $\omega : S \times X \to Y$: siehe Tabelle.

Ein typischer Denkfehler: Bei einer Ausgabe wird auch das Geld abgezogen, der Zustand ändert sich also mit. Ein Lauf:

Eingabe	–	E_1	E_1	E_3	E_1	E_2
Zustand	S_0	S_1	S_2	S_1	S_2	S_0
Ausgabe	–	Y_0	Y_0	Y_1	Y_0	Y_2

2 Der DEA – Automat ohne Ausgabe

2.1 Definition und Abgrenzung

Ein *deterministischer endlicher Automat* (DEA, auch *Akzeptor* oder *erkennender Automat*) gibt unterwegs nichts aus. Er liest ein Wort Zeichen für Zeichen und entscheidet erst *am Ende*: akzeptiert oder verworfen. Statt einer Ausgabefunktion hat er eine Menge von *Endzuständen*:

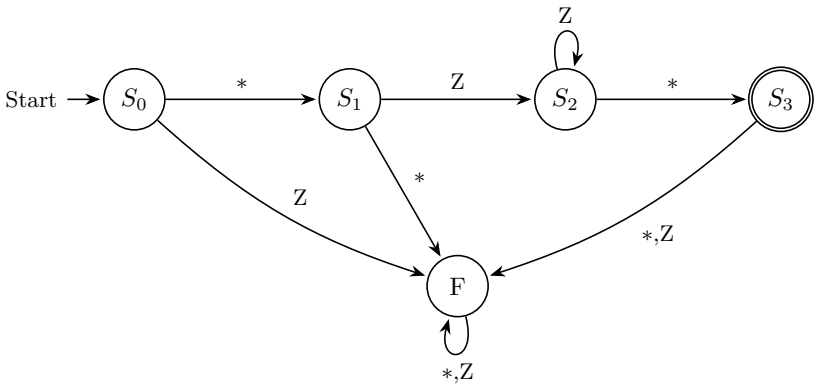
Ein DEA ist ein Tupel $A = (X, S, s_0, \delta, E)$ mit Eingabealphabet X, endlicher Zustandsmenge S, Startzustand $s_0 \in S$, Übergangsfunktion $\delta : S \times X \to S$ und Endzustandsmenge $E \subseteq S$.
Deterministisch: Zu jedem Paar (Zustand, Zeichen) gibt es *genau einen* Folgezustand.
Akzeptiert: Ein Wort wird akzeptiert, wenn der Automat nach dem letzten Zeichen in einem Endzustand steht.

	DEA (Akzeptor)	**Moore-Automat**	**Mealy-Automat**
Tupel	(X, S, s_0, δ, E)	$(X, Y, S, s_0, \delta, \omega)$	$(X, Y, S, s_0, \delta, \omega)$
Ausgabe	keine – nur *ja/nein* am Wortende	in jedem Zustand, $\omega : S \to Y$	bei jedem Übergang, $\omega : S \times X \to Y$
im Diagramm	Doppelkreis = Endzustand	Zustand/Ausgabe	Kante Eingabe/Ausgabe
Leitfrage	Gehört das Wort zur Sprache?	Was meldet der Zustand?	Was passiert beim Übergang?
Beispiel	Barcode prüfen	TOR-Melder	Kaffeeautomat

Ein DEA ist ein besonderer Moore-Automat mit der Ausgabe 1 in den Endzuständen und 0 sonst – aber man schaut nur auf die *letzte* Ausgabe. Genau dadurch wird er zum Werkzeug für *Sprachen*: Die Menge aller akzeptierten Wörter ist die Sprache des Automaten. Darum geht es in allen weiteren Abschnitten.

2.2 Startbeispiel: ein Barcodescanner

Viele Strichcodes (z. B. Code 39 auf Etiketten und Ausweisen) beginnen und enden mit einem Start-/Stoppzeichen `*`. Der Scanner soll nur Codes der Form `*Ziffern*` annehmen, also `*`, mindestens eine Ziffer, `*`. Zur Vereinfachung steht Z für *irgendeine Ziffer* 0–9; das Alphabet ist $X = \{*, Z\}$. Die Zustände merken sich, wie weit der Code schon gelesen ist: S_0 (noch nichts), S_1 (Startzeichen gelesen), S_2 (mindestens eine Ziffer), S_3 (Stoppzeichen gelesen, Endzustand). Jedes falsche Zeichen führt in den *Fangzustand* F, aus dem es kein Zurück gibt.



δ	$*$	Z	Endzustand?
S_0 (Start)	S_1	F	nein
S_1	F	S_2	nein
S_2	S_3	S_2	nein
S_3	F	F	ja
F	F	F	nein

Formal: $A = (X, S, s_0, \delta, E)$ mit $X = \{*, Z\}$ (Z = beliebige Ziffer 0–9); $S = \{S_0, S_1, S_2, S_3, \mathrm{F}\}$; $s_0 = S_0$; $E = \{S_3\}$; δ: siehe Tabelle.

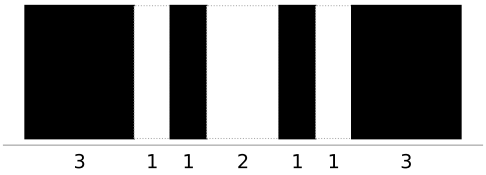
Vier Läufe (jede Ziffer zählt als Z):

Code	Zustandsfolge	Ergebnis
`*4711*`	$S_0\ S_1\ S_2\ S_2\ S_2\ S_2\ S_3$	akzeptiert
`*4711`	$S_0\ S_1\ S_2\ S_2\ S_2\ S_2$	verworfen
`**`	$S_0\ S_1$ F	verworfen
`4711*`	S_0 F F F F F	verworfen

Bei `*4711` fehlt das Stoppzeichen: Der Automat steht am Ende in S_2, kein Endzustand. Der Fangzustand macht den DEA *vollständig* – zu jedem Zustand und jedem Zeichen gibt es genau einen Folgezustand.

2.3 Zweites Beispiel: Strichbreiten lesen

Ein echter Scanner sieht keine Zeichen, sondern Striche. Er misst nacheinander die Breite jedes schwarzen Strichs und jeder weißen Lücke (in *Moduln*, der schmalsten Breite) und liefert eine Zahlenfolge mit Trennzeichen:

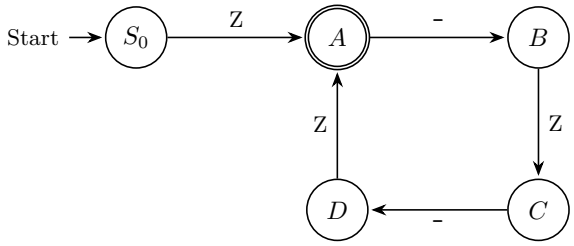


Scannerausgabe: `3-1-1-2-1-1-3`

Erst diese Folge prüft der DEA. Gültig ist sie, wenn

- jede Breite eine einzelne Ziffer 1–4 ist (Z) und zwischen zwei Breiten genau ein Trennzeichen `-` steht,
- der Code mit einem *schwarzen* Strich beginnt und endet. Da sich Strich und Lücke abwechseln, ist die Anzahl der Breiten *ungerade*.

Der Automat muss sich deshalb zweierlei merken: ob er gerade eine Breite oder ein Trennzeichen gelesen hat *und* ob die letzte Breite zu einem Strich oder zu einer Lücke gehörte. A: Strich gelesen (Endzustand), B: Trennzeichen, als Nächstes kommt eine Lücke, C: Lücke gelesen, D: Trennzeichen, als Nächstes kommt ein Strich. Damit das Diagramm übersichtlich bleibt, ist der Fangzustand F nicht eingezeichnet: *Jeder fehlende Übergang führt nach F* (in der Tabelle vollständig).



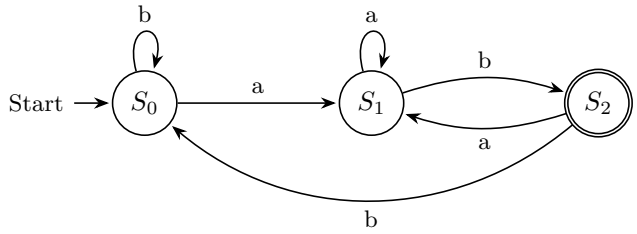
δ	Z	-	Endzustand?
S_0 (Start)	A	F	nein
A	F	B	ja
B	C	F	nein
C	F	D	nein
D	A	F	nein
F	F	F	nein

Breitenfolge	Zustandsfolge	Ergebnis
`3-1-1-2-1-1-3`	S_0 A B C D A B C D A B C D A	akzeptiert
`3-1-1-2`	S_0 A B C D A B C	verworfen
`3--1-1`	S_0 A B F F F F	verworfen
`3-12-1`	S_0 A B C F F F	verworfen
`-3-1-1`	S_0 F F F F F F	verworfen

Als Sprache: $L = \{\, Z(\texttt{-}Z\texttt{-}Z)^n \mid n \geq 0 \,\}$ – eine, drei, fünf, … Breiten. Ein echter EAN-Code prüft zusätzlich Randzeichen und eine Prüfziffer; auch das kann ein (größerer) DEA.

2.4 Akzeptoren und ihre Sprache

Beispiel: Ein Automat liest Wörter aus a und b und soll akzeptieren, wenn das Wort auf ab endet. Die Zustände merken sich, wie viel vom Muster schon passt: S_0 (nichts), S_1 (zuletzt a), S_2 (zuletzt ab, Endzustand).



δ	a	b	Endzustand?
S_0 (Start)	S_1	S_0	nein
S_1	S_1	S_2	nein
S_2	S_1	S_0	ja

Der Startzustand hat einen Pfeil „Start“, Endzustände einen Doppelkreis. Ein *Lauf* zeigt, welche Zustände ein Wort nacheinander besucht, hier für `abaab`:

$$S_0 \xrightarrow{a} S_1 \xrightarrow{b} S_2 \xrightarrow{a} S_1 \xrightarrow{a} S_1 \xrightarrow{b} S_2$$

Der Automat endet in S_2, das Wort wird akzeptiert.

Ein Wort wird also *akzeptiert*, wenn sein Lauf in einem Endzustand endet, sonst *verworfen*. (In Büchern schreibt man für den Lauf auch kurz $\delta^*(S_0, abaab) = S_2$.) Die Menge aller akzeptierten Wörter ist die *Sprache des Automaten*, hier:

$$L(A) = \{\, w \in \{a,b\}^* \mid w \text{ endet auf } ab \,\}.$$

Quelltext 1: Einen DEA simulieren: Übergänge als Wörterbuch

```
delta = {("S0", "a"): "S1", ("S0", "b"): "S0",
         ("S1", "a"): "S1", ("S1", "b"): "S2",
         ("S2", "a"): "S1", ("S2", "b"): "S0"}
ende = ["S2"]

def akzeptiert(wort):
    zustand = "S0"
    for zeichen in wort:
        zustand = delta[(zustand, zeichen)]
    return zustand in ende

print(akzeptiert("abaab"))     # True
print(akzeptiert("aba"))       # False
```

Endlicher Automat (DEA) auf einen Blick	
Speicher	nur der aktuelle Zustand (endlich viele)
Arbeitsweise	liest Zeichen für Zeichen, $\delta(\text{Zustand}, \text{Zeichen}) = \text{Folgezustand}$
Erkennt	reguläre Sprachen (Typ 3)
Typischer Fehler	fehlende Übergänge; bei Mustererkennung nach einem Fehlzeichen zu weit zurückspringen

3 Einen DEA minimieren

Ein selbst entworfener Automat hat oft mehr Zustände als nötig. Zwei Zustände p, q heißen *äquivalent*, wenn sie sich von außen nicht unterscheiden lassen: *Jedes* Wort, dessen Lauf von p aus in einem Endzustand endet, endet auch von q aus in einem Endzustand – und umgekehrt. Äquivalente Zustände darf man verschmelzen. Der Automat ohne äquivalente und ohne unerreichbare Zustände ist der *Minimalautomat*; er ist bis auf die Namen eindeutig.

Tabellenverfahren (Markierungsverfahren)

0. Nicht erreichbare Zustände streichen.
1. Alle Paare $\{p, q\}$ in eine Dreiecksmatrix eintragen.
2. **Runde 0:** Paare markieren, bei denen *genau einer* ein Endzustand ist (sie unterscheidet das leere Wort).
3. **Runde 1, 2, . . . :** Ein unmarkiertes Paar markieren, wenn es ein Zeichen x gibt, sodass $\{\delta(p, x), \delta(q, x)\}$ schon markiert ist. Wiederholen, bis sich nichts mehr ändert.
4. Unmarkierte Paare ($\circ$) sind äquivalent: verschmelzen, Übergänge umlenken.

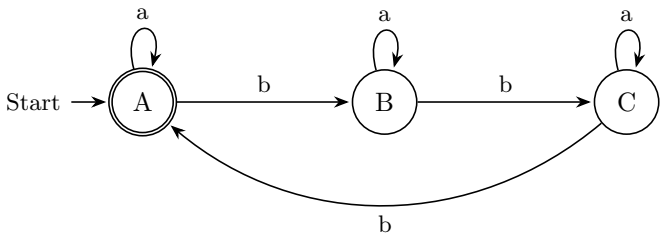
Beispiel: Ein DEA mit sechs Zuständen über $\{a, b\}$, Endzustände Z_0 und Z_3:

δ	a	b	Endzustand?
Z_0 (Start)	Z_3	Z_1	ja
Z_1	Z_4	Z_2	nein
Z_2	Z_2	Z_3	nein
Z_3	Z_0	Z_4	ja
Z_4	Z_1	Z_5	nein
Z_5	Z_5	Z_0	nein

Runde 0: Markiert werden alle Paare aus einem End- und einem Nicht-Endzustand: $\{Z_0, Z_1\}$, $\{Z_0, Z_2\}$, $\{Z_0, Z_4\}$, $\{Z_0, Z_5\}$, $\{Z_1, Z_3\}$, $\{Z_2, Z_3\}$, $\{Z_3, Z_4\}$, $\{Z_3, Z_5\}$.
Runde 1: $\{Z_1, Z_2\}$: mit b nach $\{Z_2, Z_3\}$; $\{Z_1, Z_5\}$: mit b nach $\{Z_2, Z_0\}$; $\{Z_2, Z_4\}$: mit b nach $\{Z_3, Z_5\}$; $\{Z_4, Z_5\}$: mit b nach $\{Z_5, Z_0\}$ – jeweils ein in Runde 0 markiertes Paar, also wird markiert.
Runde 2: Für die übrigen Paare führt kein Zeichen auf ein markiertes Paar. Es ändert sich nichts mehr.

	Z_0	Z_1	Z_2	Z_3	Z_4
Z_1	0				
Z_2	0	1			
Z_3	$\circ$	0	0		
Z_4	0	$\circ$	1	0	
Z_5	0	1	$\circ$	0	1

Die Zahl in der Zelle ist die Runde der Markierung. Unmarkiert bleiben $\{Z_0, Z_3\}$, $\{Z_1, Z_4\}$ und $\{Z_2, Z_5\}$. Mit $A = \{Z_0, Z_3\}$, $B = \{Z_1, Z_4\}$, $C = \{Z_2, Z_5\}$ entsteht der Minimalautomat:



Jetzt sieht man, was der Automat tut: a ändert nichts, b zählt weiter. Er erkennt die Wörter, in denen die Anzahl der b *durch 3 teilbar* ist.

Merke: Ein Endzustand ist nie äquivalent zu einem Nicht-Endzustand. Verschmolzen werden immer *ganze Klassen*; die Äquivalenz ist transitiv. Bleibt in der Matrix kein Paar unmarkiert, war der DEA schon minimal.

Zur Kontrolle mit FLACI: Automat unter *Abstrakte Automaten* (`https://flaci.com`) eingeben → *DEA minimieren*. Vergleichen Sie das Ergebnis mit Ihrem Minimalautomaten – das Verfahren selbst üben Sie aber von Hand.

Schritt für Schritt nachvollziehen: Die Simulation auf `g-ymnasium.de/index.php/automaten` zeigt das Tabellenverfahren Runde für Runde mit Dreiecksmatrix und Begründung; eigene Automaten lassen sich eintippen.

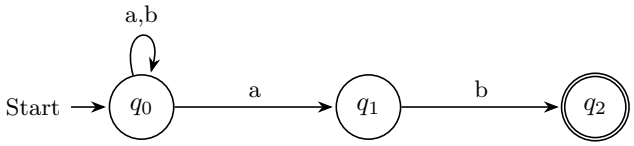
4 Nichtdeterminismus und reguläre Sprachen

4.1 Formale Sprachen

Jeder Akzeptor legt eine Menge von Wörtern fest – eine *Sprache*. Ein *Alphabet* Σ ist eine endliche Menge von Zeichen, ein *Wort* eine endliche Zeichenfolge ($|w|$ ist die Länge, ε das leere Wort), Σ^* die Menge aller Wörter. Eine *formale Sprache* ist eine Teilmenge $L \subseteq \Sigma^*$. Sprachen, die ein DEA erkennt, heißen *regulär*.

4.2 Nichtdeterministische Automaten (NEA)

Bei einem NEA darf es zu (Zustand, Zeichen) *mehrere oder keinen* Folgezustand geben. Ein Wort wird akzeptiert, wenn es *mindestens einen* Weg in einen Endzustand gibt. NEA sind oft leichter zu entwerfen. Beispiel „endet auf *ab*“: Der Automat „rät“, wann das letzte *ab* beginnt.



Man verfolgt alle möglichen Wege gleichzeitig, also die *Menge* der erreichbaren Zustände. Für `abab`: $\{q_0\} \to \{q_0, q_1\} \to \{q_0, q_2\} \to \{q_0, q_1\} \to \{q_0, q_2\}$ – die letzte Menge enthält den Endzustand q_2, das Wort wird akzeptiert.

4.3 Vom NEA zum DEA: die Teilmengenkonstruktion

Genau diese Mengen werden die Zustände eines DEA. Das Verfahren:

Teilmengenkonstruktion

1. Startzustand des DEA ist die Menge $\{q_0\}$.
2. Für eine noch nicht bearbeitete Menge M und jedes Zeichen x: Sammle *alle* Zustände, die von irgendeinem Zustand aus M mit x erreichbar sind. Diese Menge ist der Folgezustand.
3. Ist die Folgemenge neu, kommt sie als neue Zeile in die Tabelle. Wiederholen, bis keine neue Menge mehr entsteht.
4. Endzustände des DEA sind alle Mengen, die mindestens einen Endzustand des NEA enthalten.

Schritt für Schritt für den NEA „endet auf ab“:

1. $\{q_0\}$: mit a nach $\{q_0, q_1\}$ **(neu)**, mit b nach $\{q_0\}$.
2. $\{q_0, q_1\}$: mit a nach $\{q_0, q_1\}$, mit b nach $\{q_0, q_2\}$ **(neu)**.
3. $\{q_0, q_2\}$: mit a nach $\{q_0, q_1\}$, mit b nach $\{q_0\}$.

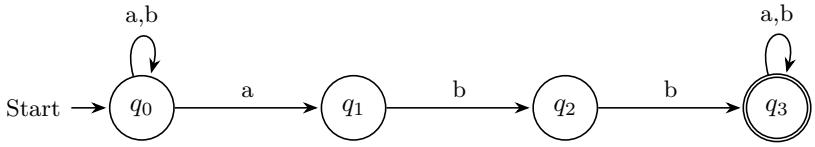
Es entstehen keine weiteren Mengen. Die Tabelle des DEA:

DEA-Zustand	a	b	Endzustand?
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$	nein
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$	nein
$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$	ja

Das ist – bis auf die Namen – genau der DEA aus Abschnitt 2.4: $\{q_0\} = S_0$, $\{q_0, q_1\} = S_1$, $\{q_0, q_2\} = S_2$. NEA erkennen also *keine* anderen Sprachen als DEA. Im ungünstigsten Fall hat der DEA aber 2^n Zustände bei n NEA-Zuständen.

4.4 Zweites Beispiel: Wörter, die *abb* enthalten

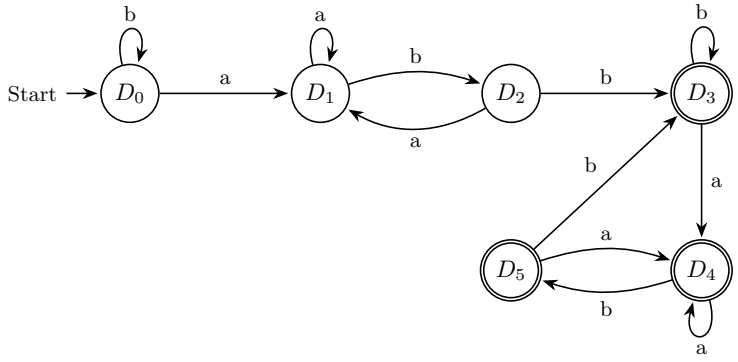
Der NEA rät, wo das Teilwort *abb* beginnt, und bleibt danach im Endzustand q_3:



1. $\{q_0\}$: mit a nach $\{q_0, q_1\}$ **(neu)**, mit b nach $\{q_0\}$.
2. $\{q_0, q_1\}$: mit a nach $\{q_0, q_1\}$, mit b nach $\{q_0, q_2\}$ **(neu)**.
3. $\{q_0, q_2\}$: mit a nach $\{q_0, q_1\}$, mit b nach $\{q_0, q_3\}$ **(neu)**.
4. $\{q_0, q_3\}$: mit a nach $\{q_0, q_1, q_3\}$ **(neu)**, mit b nach $\{q_0, q_3\}$.
5. $\{q_0, q_1, q_3\}$: mit a nach $\{q_0, q_1, q_3\}$, mit b nach $\{q_0, q_2, q_3\}$ **(neu)**.
6. $\{q_0, q_2, q_3\}$: mit a nach $\{q_0, q_1, q_3\}$, mit b nach $\{q_0, q_3\}$.

DEA-Zustand	a	b	Endzustand?
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$	nein
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$	nein
$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_3\}$	nein
$\{q_0, q_3\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_3\}$	ja
$\{q_0, q_1, q_3\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_2, q_3\}$	ja
$\{q_0, q_2, q_3\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_3\}$	ja

Mit den Namen $D_0 = \{q_0\}$, $D_1 = \{q_0, q_1\}$, $D_2 = \{q_0, q_2\}$, $D_3 = \{q_0, q_3\}$, $D_4 = \{q_0, q_1, q_3\}$, $D_5 = \{q_0, q_2, q_3\}$ ergibt sich ein DEA mit 6 Zuständen:



Er ist korrekt, aber nicht minimal: Die drei Endzustände D_3, D_4, D_5 sind äquivalent (wer *abb* schon gelesen hat, bleibt akzeptiert). Das Tabellenverfahren aus Abschnitt 3 verschmilzt sie; übrig bleibt ein Minimalautomat mit 4 Zuständen. Typischer Ablauf in der Praxis: *NEA entwerfen* → *Teilmengenkonstruktion* → *minimieren* – genau so wie in FLACI.

4.5 Grenzen endlicher Automaten

Nicht jede Sprache ist regulär. Für $L = \{a^n b^n \mid n \geq 1\} = \{ab, aabb, aaabbb, \dots\}$ müsste der Automat die Anzahl der a *unbegrenzt* mitzählen. Ein DEA mit k Zuständen kann aber nur k verschiedene Situationen unterscheiden: Liest er $a^1, a^2, \dots, a^{k+1}$, so landet er für zwei verschiedene Anzahlen a^i und a^j im selben Zustand – und kann $a^i b^i$ (richtig) nicht mehr von $a^j b^i$ (falsch) unterscheiden. Dafür braucht man einen Speicher (Abschnitt 6).

NEA auf einen Blick	
Speicher	Menge der möglichen Zustände
Arbeitsweise	alle Wege gleichzeitig verfolgen; akzeptiert, wenn ein Weg im Endzustand endet
Erkennt	genau die regulären Sprachen (wie der DEA)
Typischer Fehler	Teilmengen vergessen, die erst später entstehen; leere Menge $\emptyset$ als Zustand nicht eintragen

5 Grammatiken

Ein Automat *prüft* Wörter, eine Grammatik *erzeugt* sie: Man beginnt mit einem Startsymbol und ersetzt nach Regeln, bis nur noch Zeichen des Alphabets übrig sind.

Grammatik $G = (N, T, P, S)$: N Nichtterminale (Variablen, hier Großbuchstaben), T Terminale (die Zeichen der Wörter), P Produktionen (Regeln) $\alpha \to \beta$, $S \in N$ das Startsymbol.
$u \Rightarrow v$: v entsteht aus u durch *eine* Regelanwendung. Die erzeugte Sprache ist $L(G) = \{ w \in T^* \mid S \Rightarrow^* w \}$. Mehrere Regeln mit gleicher linker Seite schreibt man mit „|“: $A \to a \mid bA$.

5.1 Beispiele

$a^n b^n$ mit $S \to aSb \mid ab$: $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$. Jede Anwendung von $S \to aSb$ setzt außen ein a *und* ein b – genau das Mitzählen, das ein DEA nicht kann.

Klammerterme mit $S \to (S)S \mid \varepsilon$:
$S \Rightarrow (S)S \Rightarrow ((S)S)S \Rightarrow (()S)S \Rightarrow (())S \Rightarrow (())(S)S \Rightarrow (())()S \Rightarrow (())()$.

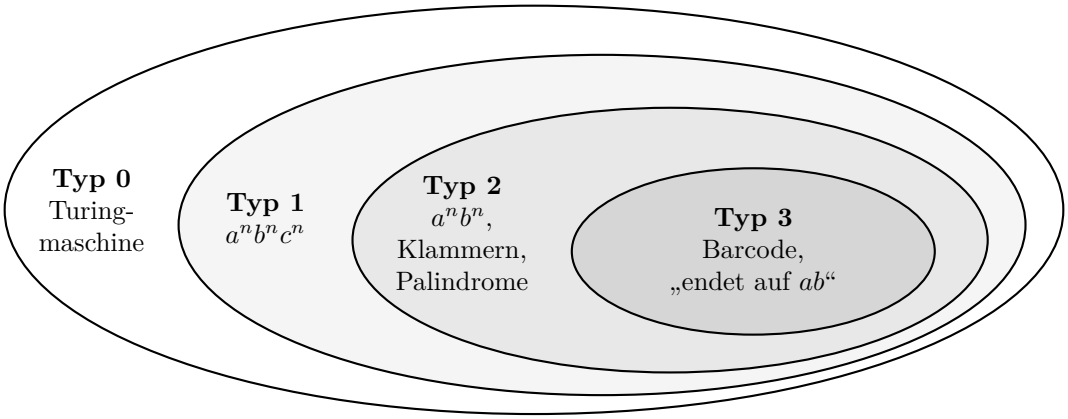
Palindrome mit $S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$: $S \Rightarrow aSa \Rightarrow abSba \Rightarrow abbba$.

5.2 Die Chomsky-Hierarchie

Noam Chomsky ordnete Grammatiken 1956 nach den erlaubten Regeln in vier Typen. Je strenger die Regeln, desto einfacher der Automat, der die Sprache erkennt – und desto weniger Sprachen lassen sich beschreiben. Die Beispiele stammen alle aus diesem Skript:

Typ	Name	erlaubte Regeln	Automat	Beispiele im Skript
3	regulär	$A \to a$ oder $A \to aB$ (rechts höchstens ein Nichtterminal, ganz hinten)	DEA / NEA	Barcodescanner (2.2), Strichbreiten (2.3), „endet auf ab“ (2.4), „enthält abb“ (4.4)
2	kontextfrei	$A \to \beta$ (links genau ein Nichtterminal, rechts beliebig)	Kellerautomat	$a^n b^n$, Klammerterme, Palindrome (5.1)
1	kontextsensitiv	$\alpha \to \beta$ mit $\lvert\alpha\rvert \le \lvert\beta\rvert$ (nichts wird kürzer)	linear beschränkte TM	$a^n b^n c^n$
0	rekursiv aufzählbar	keine Einschränkung	Turingmaschine	alles, was überhaupt berechenbar ist (7)

Jede Regel vom Typ 3 ist auch vom Typ 2, jede vom Typ 2 (ohne ε-Regeln) auch vom Typ 1 usw. Die Sprachklassen liegen deshalb *ineinander*:



Die Grenzen sind echt: $a^n b^n$ ist kontextfrei, aber *nicht* regulär (ein DEA kann nicht beliebig weit zählen, Abschnitt 4.5); $a^n b^n c^n$ ist kontextsensitiv, aber nicht kontextfrei (ein einziger Keller reicht nicht). **Faustregel:** Muss sich ein Automat nur endlich viel merken → Typ 3. Muss er paarweise zählen oder klammern → Typ 2. *Ausblick:* Auch Programmiersprachen werden mit kontextfreien Grammatiken beschrieben; der Compiler prüft die Syntax eines Programms mit einem Kellerautomaten.

5.3 Vom Automaten zur regulären Grammatik

Ein Automat *prüft* die Wörter einer Sprache, eine Grammatik *erzeugt* sie. Für reguläre Sprachen lässt sich das eine direkt in das andere übersetzen – so, wie es FLACI beim Export einer Grammatik macht:

endlicher Automat	**Grammatik**
Eingabezeichen	Terminalzeichen T
Zustände	Nichtterminale N
Startzustand S_0	Startsymbol S_0
Übergang $P \xrightarrow{x} Q$	Produktion $P \to x\,Q$
Übergang $P \xrightarrow{x} Q$ in einen *Endzustand* Q	zusätzlich $P \to x$ (hier darf das Wort aufhören)
Fangzustand	fällt weg (aus ihm entsteht nie ein Wort)

Beispiel 1: der Barcodescanner aus 2.2. Übersetzt man jeden Übergang außer denen zum Fangzustand F:

$$\begin{aligned} S_0 &\to *S_1 \\ S_1 &\to ZS_2 \\ S_2 &\to *S_3 \mid * \mid ZS_2 \end{aligned}$$

S_3 hat keine eigenen Regeln mehr (aus S_3 führen nur Übergänge nach F). Die Regel $S_2 \to *\,S_3$ kann also nie zu einem Wort führen und wird gestrichen. Übrig bleibt

$$\begin{aligned} S_0 &\to *S_1 \\ S_1 &\to ZS_2 \\ S_2 &\to * \mid ZS_2 \end{aligned}$$

Die Ableitung des Codes `*47*` (also $*ZZ*$) entspricht genau dem Lauf durch den Automaten:

$$S_0 \Rightarrow *S_1 \Rightarrow *ZS_2 \Rightarrow *ZZS_2 \Rightarrow *ZZ*$$

Beispiel 2: „endet auf ab“ aus 2.4. Hier ist S_2 der Endzustand; jeder Übergang *nach* S_2 bekommt die zusätzliche kurze Regel:

$$S_0 \to aS_1 \mid bS_0 \qquad S_1 \to aS_1 \mid bS_2 \mid b \qquad S_2 \to aS_1 \mid bS_0$$

Die Regel $S_1 \to b$ ist nötig, weil der Übergang $S_1 \xrightarrow{b} S_2$ in den *Endzustand* führt: Nach diesem b darf das Wort zu Ende sein. Ohne die kurze Regel bliebe am Ende jeder Ableitung ein Nichtterminal stehen – die Grammatik würde gar kein Wort erzeugen. $S_2 \xrightarrow{b} S_0$ führt dagegen nicht in einen Endzustand, deshalb gibt es *kein* $S_2 \to b$.

Ableitung von `babab`: $S_0 \Rightarrow bS_0 \Rightarrow baS_1 \Rightarrow babS_2 \Rightarrow babaS_1 \Rightarrow babab$. In FLACI erscheint dieselbe Grammatik als `S_0 -> a S_1 | b S_0` usw.

Umgekehrt wird aus einer regulären Grammatik ein NEA: jedes Nichtterminal ein Zustand, dazu ein neuer Endzustand E; aus $P \to x\,Q$ wird $P \xrightarrow{x} Q$, aus $P \to x$ wird $P \xrightarrow{x} E$.

So üben Sie mit FLACI: Automat zeichnen → *Grammatik exportieren* und mit der eigenen Übersetzung vergleichen. Kontrollfrage: Hat jeder Übergang in einen Endzustand seine kurze Regel $P \to x$?

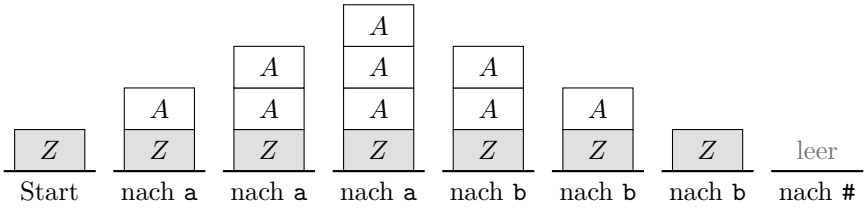
6 Kellerautomaten

Für $a^n b^n$ oder korrekt geklammerte Terme muss ein Automat beliebig weit zählen können. Ein *Kellerautomat* bekommt dazu einen unbegrenzten Speicher, auf den er nur von oben zugreift: den *Keller* (Stapel, last in – first out).

Kellerautomat: Eine *Konfiguration* ist (Zustand, Resteingabe, Keller). Ein Übergang $(z, x, Z) \to (z', \beta)$ bedeutet: Liest der Automat x und liegt Z oben auf dem Keller, so wechselt er nach z' und ersetzt Z durch β (oberstes Zeichen links). $\beta = \varepsilon$: Z wird entfernt (*pop*); $\beta = AZ$: A wird aufgelegt (*push*). Der Keller beginnt mit dem Bodenzeichen Z; die Endemarke # wird wie ein Zeichen gelesen. Akzeptiert ist ein Wort, wenn die Eingabe gelesen und der Keller leer ist.

6.1 Beispiel $a^n b^n$

Jedes a legt ein A auf, jedes b nimmt eines weg; bei # muss das Bodenzeichen oben liegen. Mehr passiert nicht: Der Keller wird *aufgebaut* und wieder *abgeräumt*. So sieht er beim Lesen von `aaabbb#` aus:



Warum schafft das kein DEA? Ein DEA hat nur seinen Zustand als Gedächtnis – und davon gibt es *endlich viele*. Um $a^n b^n$ zu prüfen, müsste er sich die Anzahl der a merken, und die kann beliebig groß sein. Hat er zum Beispiel 5 Zustände, landen zwei der Wörter $a, aa, \ldots, aaaaaa$ zwangsläufig im selben Zustand; dann reagiert er auf die folgenden b gleich und verwechselt richtige mit falschen Wörtern.

Der Keller ist das zusätzliche Gedächtnis: unbegrenzt groß, aber wie ein Tellerstapel nur von oben zugänglich. Für $a^n b^n$ genügt das, denn das zuletzt gezählte a wird als Erstes wieder abgezogen.

Zustand	liest	Kellerspitze	neuer Zustand	neue Kellerzeichen
z_0	a	Z	z_0	AZ
z_0	a	A	z_0	AA
z_0	b	A	z_1	ε
z_1	b	A	z_1	ε
z_1	#	Z	z_1	ε

Schritt	Zustand	Resteingabe	Keller
0	z_0	$aabb\#$	Z
1	z_0	$abb\#$	AZ
2	z_0	$bb\#$	AAZ
3	z_1	$b\#$	AZ
4	z_1	#	Z
5	z_1	ε	ε

Bei `aab#` bliebe ein A übrig (kein Übergang für # auf A), bei `abb#` fehlt für das zweite b ein A. In beiden Fällen bleibt der Automat stecken: nicht akzeptiert.

6.2 Beispiel Klammerterme

Eine öffnende Klammer legt ein K auf, eine schließende nimmt eines weg (wie in der Parsingtabelle: „öffnende Klammern kommen auf den Stapel, schließende räumen eine ab“).

Zustand	liest	Kellerspitze	neuer Zustand	neue Kellerzeichen
z	(	Z	z	KZ
z	(	K	z	KK
z	)	K	z	ε
z	#	Z	z	ε

Rechnung für `(()())#`:
$(z, (()())\#, Z) \to (z, ()())\#, KZ) \to (z,)())\#, KKZ) \to (z, ())\#, KZ) \to (z,))\#, KKZ) \to (z,)\#, KZ)$
$\to (z, \#, Z) \to (z, \varepsilon, \varepsilon)$ – akzeptiert.

6.3 Aus einer Grammatik wird ein Kellerautomat

Zu jeder kontextfreien Grammatik gibt es einen Kellerautomaten mit nur einem Zustand. Der Keller enthält die noch abzuleitende Satzform und beginnt mit dem Startsymbol:

Ersetzen: Für jede Regel $A \to \beta$: liegt A oben, ersetze es durch β (ohne zu lesen).

Abgleichen: Liegt ein Terminal x oben und wird x gelesen, entferne es.

Beispiel $S \to aSb \mid ab$, Eingabe `aabb`:

Schritt	Resteingabe	Keller	Aktion
0	$aabb$	S	
1	$aabb$	aSb	Regel $S \to aSb$
2	abb	Sb	a lesen und entfernen
3	abb	abb	Regel $S \to ab$
4	bb	bb	a lesen und entfernen
5	b	b	b lesen und entfernen
6	ε	ε	b lesen und entfernen

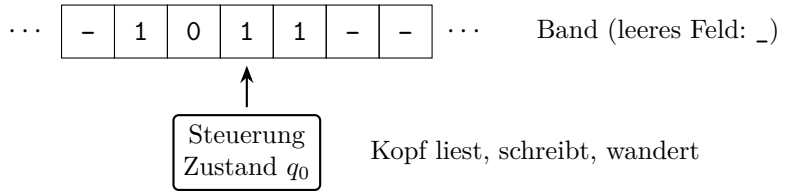
Gibt es zu einem Nichtterminal mehrere Regeln, muss der Automat die richtige *raten*: Er ist *nichtdeterministisch* (NKA). Anders als bei endlichen Automaten sind NKA mächtiger als deterministische: Die Palindrome erkennt nur ein NKA, weil er die Wortmitte nicht kennt. Die kontextfreien Sprachen sind genau die Sprachen, die ein NKA erkennt. Für $a^n b^n c^n$ reicht ein Keller nicht mehr.

Kellerautomat auf einen Blick	
Speicher	ein Keller (Stapel), Zugriff nur oben
Arbeitsweise	Zeichen lesen, oberstes Kellerzeichen ersetzen (push/pop)
Erkennt	kontextfreie Sprachen (Typ 2), z. B. $a^n b^n$, Klammerterme
Typischer Fehler	Keller beim Ende nicht leer; Spitze des Kellers rechts statt links notiert

7 Die Turingmaschine

7.1 Aufbau

Endliche Automaten haben keinen Speicher, Kellerautomaten einen Keller. Die *Turingmaschine* (Alan Turing, 1936) hat ein unendliches *Band* aus Feldern. Ein *Schreib-Lese-Kopf* liest das Feld, auf dem er steht, überschreibt es und geht ein Feld nach links oder rechts – beliebig oft hin und her.



Ein Schritt: (Zustand, gelesenes Zeichen) → (neuer Zustand, geschriebenes Zeichen, Kopf *R* / *L* / *N*).
Die Maschine hält in einem Haltezustand (*halt*, bei Akzeptoren *ja* oder *nein*) – oder sie läuft ewig. Fehlt für (Zustand, Zeichen) ein Übergang, hält sie mit *nein*. Im Diagramm steht an der Kante *gelesen/geschrieben,Richtung*, z. B. `1/0,L`. In den Rechnungen steht der Kopf in eckigen Klammern.

7.2 Beispiel: unärer Inkrementer

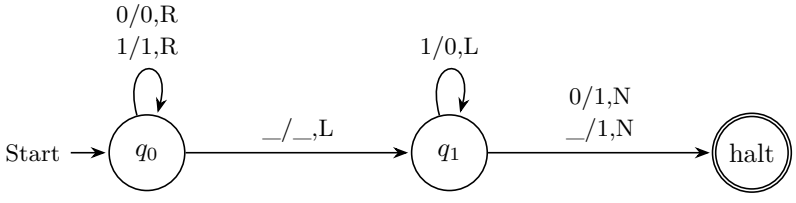
Eine Zahl n steht als n Einsen auf dem Band; die Maschine hängt eine 1 an ($n \to n+1$).

Zustand	liest	neuer Zustand	schreibt	Kopf
q_0	`1`	q_0	`1`	R
q_0	`_`	halt	`1`	N

Rechnung für `111`: `[1]11` → `1[1]1` → `11[1]` → `111[_]` → `111[1]`.

7.3 Beispiel: binärer Inkrementer

Die Maschine addiert 1 zu einer Binärzahl. Sie läuft in q_0 ans Ende, dann in q_1 zurück: Jede 1 wird zur 0 (Übertrag), die erste 0 oder das leere Feld wird zur 1, dann hält sie.



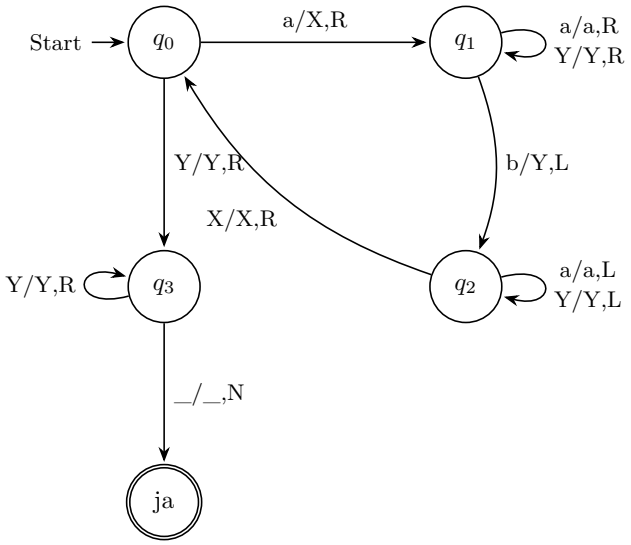
Zustand	liest	neuer Zustand	schreibt	Kopf
q_0	`0`	q_0	`0`	R
q_0	`1`	q_0	`1`	R
q_0	`_`	q_1	`_`	L
q_1	`1`	q_1	`0`	L
q_1	`0`	halt	`1`	N
q_1	`_`	halt	`1`	N

Schritt	Zustand	Band (Kopf in [])
0	q_0	`[1]011`
1	q_0	`1[0]11`
2	q_0	`10[1]1`
3	q_0	`101[1]`
4	q_0	`1011[_]`
5	q_1	`101[1]_`
6	q_1	`10[1]0_`
7	q_1	`1[0]00_`
8	halt	`1[1]00_`

Aus 1011 (11) wird nach 8 Schritten 1100 (12).

7.4 Beispiel: Akzeptor für $a^n b^n$

Die Maschine streicht abwechselnd das erste a (wird `X`) und das erste b (wird `Y`) durch. Sind alle a verbraucht, prüft sie in q_3, ob nur noch `Y` folgen.



Zustand	liest	neuer Zustand	schreibt	Kopf
q_0	`a`	q_1	`X`	R
q_0	`Y`	q_3	`Y`	R
q_1	`a`	q_1	`a`	R
q_1	`Y`	q_1	`Y`	R
q_1	`b`	q_2	`Y`	L
q_2	`a`	q_2	`a`	L
q_2	`Y`	q_2	`Y`	L
q_2	`X`	q_0	`X`	R
q_3	`Y`	q_3	`Y`	R
q_3	`_`	ja	`_`	N

Schritt	Zustand	Band (Kopf in [])
0	q_0	`[a]abb`
1	q_1	`X[a]bb`
2	q_1	`Xa[b]b`
3	q_2	`X[a]Yb`
4	q_2	`[X]aYb`
5	q_0	`X[a]Yb`
6	q_1	`XX[Y]b`
7	q_1	`XXY[b]`
8	q_2	`XX[Y]Y`
9	q_2	`X[X]YY`
10	q_0	`XX[Y]Y`
11	q_3	`XXY[Y]`
12	q_3	`XXYY[_]`
13	ja	`XXYY[_]`

Nach 13 Schritten: *ja*. Bei `aaabb` findet die Maschine zum dritten a kein b mehr (q_1, Kopf liest `_`, kein Übergang) und hält mit *nein*. Mit derselben Idee – noch eine dritte Markierung – erkennt eine Turingmaschine auch $a^n b^n c^n$, was kein Kellerautomat kann.

Quelltext 2: Eine Turingmaschine simulieren

```
def turing(delta, band, zustand="q0"):
    felder = {}
    for i in range(len(band)):
        felder[i] = band[i]
    kopf = 0
    while zustand not in ["halt", "ja", "nein"]:
        zeichen = felder.get(kopf, "_")          # leeres Feld
        if (zustand, zeichen) not in delta:
            return "nein", felder                # kein Übergang
        zustand, schreibt, richtung = delta[(zustand, zeichen)]
        felder[kopf] = schreibt
        if richtung == "R":
            kopf = kopf + 1
        elif richtung == "L":
            kopf = kopf - 1
    return zustand, felder

inkrement = {("q0", "1"): ("q0", "1", "R"), ("q0", "_"): ("halt", "1", "N")}
print(turing(inkrement, "111"))     # ('halt', {0: '1', 1: '1', 2: '1', 3: '1'})
```

7.5 Was die Turingmaschine bedeutet

Church-Turing-These: Alles, was man überhaupt „mechanisch berechnen“ kann – mit Papier und Bleistift, mit einem Computer, mit jeder Programmiersprache –, kann auch eine Turingmaschine berechnen. Ein Computer ist schneller und bequemer, aber nicht mächtiger.

Grenzen: Es gibt Probleme, die *keine* Turingmaschine lösen kann. Das bekannteste ist das *Halteproblem*: Es gibt kein Programm, das für jedes Programm und jede Eingabe entscheidet, ob es irgendwann hält. (Angenommen, es gäbe so ein Programm H; dann baut man ein Programm, das genau dann endlos läuft, wenn H sagt, dass es hält – ein Widerspruch.)

Fleißige Biber: Welche Turingmaschine mit n Zuständen schreibt auf ein leeres Band die meisten Einsen und hält dann? Für 2 Zustände ist es diese Maschine – 4 Einsen nach 6 Schritten:

Zustand	liest	neuer Zustand	schreibt	Kopf
A	_	B	1	R
A	1	B	1	L
B	_	A	1	L
B	1	halt	1	R

Schritt	Zustand	Band (Kopf in [])
0	A	`[_]`
1	B	`1[_]`
2	A	`[1]1`
3	B	`[_]11`
4	A	`[_]111`
5	B	`1[1]11`
6	halt	`11[1]1`

Schon für wenige Zustände werden die Zahlen riesig, und es ist im Allgemeinen nicht berechenbar, welche Maschine gewinnt – ein Beispiel für die Grenzen der Berechenbarkeit.

Turingmaschine auf einen Blick	
Speicher	unendliches Band, Kopf in beide Richtungen
Arbeitsweise	lesen, schreiben, Kopf $R/L/N$, Zustand wechseln
Erkennt	alle berechenbaren Probleme (Typ 0); hält nicht immer
Typischer Fehler	Kopfrichtung vergessen; leeres Feld am Bandende nicht berücksichtigt

8 Überblick

Modell	**Speicher**	**Sprachklasse**	**Beispiel**
DEA / NEA	keiner (nur Zustand)	regulär (Typ 3)	endet auf ab, Anzahl b durch 3 teilbar
Kellerautomat (NKA)	Keller	kontextfrei (Typ 2)	$a^n b^n$, Klammerterme, Palindrome
linear beschr. TM	Band so lang wie die Eingabe	kontextsensitiv (Typ 1)	$a^n b^n c^n$
Turingmaschine	unendliches Band	rekursiv aufzählbar (Typ 0)	alles Berechenbare

Selbstkontrolle

1. Worin unterscheiden sich Moore- und Mealy-Automat?
2. Warum darf man beim Minimieren nie einen End- mit einem Nicht-Endzustand verschmelzen?
3. Warum erkennt ein NEA keine anderen Sprachen als ein DEA?
4. Warum kann kein endlicher Automat $a^n b^n$ erkennen?
5. Was kann eine Turingmaschine, was ein Kellerautomat nicht kann?

Lösungshinweise: 1. Moore: Ausgabe im Zustand; Mealy: am Übergang, abhängig von der Eingabe. 2. Das leere Wort unterscheidet sie. 3. Teilmengenkonstruktion. 4. Endlich viele Zustände können nicht beliebig weit zählen. 5. Auf dem Band frei hin und her gehen und überschreiben, z. B. $a^n b^n c^n$ erkennen.

9 Übungen

Aufgabe 1 (DEA). Entwerfen Sie einen DEA über $\{a, b\}$, der alle Wörter mit einer *geraden Anzahl* von a akzeptiert (auch das leere Wort). Geben Sie Diagramm und Übergangstabelle an.

Aufgabe 2 (Moore-Automat). Ein Moore-Automat liest eine Binärzahl von links nach rechts und gibt 1 aus, wenn die bisher gelesene Zahl durch 3 teilbar ist. Tipp: Merken Sie sich den Rest bei Division durch 3; ein angehängtes Zeichen x macht aus der Zahl z die Zahl $2z + x$. Rechnen Sie die Eingabe `10010` durch.

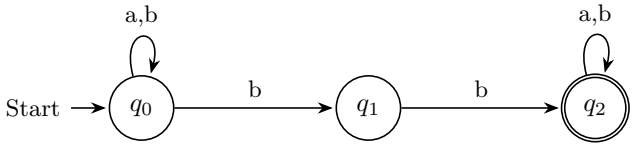
Aufgabe 3 (Mealy-Automat). Ein Mealy-Automat liest Bits und gibt nach jedem Bit aus, ob die Anzahl der bisher gelesenen Einsen *ungerade* ist (Ausgabe 1) oder gerade (Ausgabe 0). Zeichnen Sie ihn mit zwei Zuständen und geben Sie die Ausgabe für `1101` an.

Aufgabe 4 (Minimieren). Minimieren Sie den DEA (Alphabet $\{0, 1\}$, Start A):

δ	0	1	Endzustand?
A (Start)	B	C	nein
B	D	E	nein
C	E	D	nein
D	D	D	ja
E	D	D	ja
F	A	D	nein

Streichen Sie zuerst unerreichbare Zustände. Beschreiben Sie die Sprache des Minimalautomaten.

Aufgabe 5 (NEA). Der NEA akzeptiert alle Wörter, die *bb* enthalten. Wandeln Sie ihn mit der Teilmengenkonstruktion in einen DEA um.



Aufgabe 6 (reguläre Grammatik). Geben Sie zu Ihrem Automaten aus Aufgabe 1 eine reguläre Grammatik an und leiten Sie `abba` ab.

Aufgabe 7 (kontextfreie Grammatik). Geben Sie eine Grammatik für $\{a^n c\, b^n \mid n \geq 0\}$ an und leiten Sie `aaacbbb` ab. Von welchem Typ ist sie? Warum gibt es keinen DEA für diese Sprache?

Aufgabe 8 (Chomsky-Typen). Bestimmen Sie den höchsten möglichen Typ: (a) $S \to aS \mid b$ (b) $S \to aSb \mid ab$ (c) $S \to aB,\ B \to bS \mid b$ (d) $S \to AB,\ AB \to BA,\ A \to a,\ B \to b$

Aufgabe 9 (Kellerautomat). Rechnen Sie mit dem Klammerautomaten aus 6.2 die Eingabe `(())()#` durch (Resteingabe und Keller nach jedem Schritt). Warum wird `())(#` nicht akzeptiert?

Aufgabe 10 (Turingmaschine entwerfen). Entwerfen Sie eine Turingmaschine, die in einer Bitfolge jede 0 durch 1 und jede 1 durch 0 ersetzt und am Ende hält. Rechnen Sie `0110` durch.

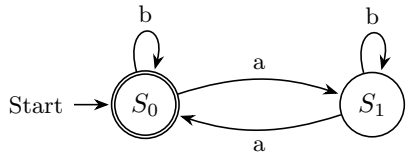
Aufgabe 11 (Turingmaschine rechnen). Rechnen Sie den Akzeptor für $a^n b^n$ aus 7.4 mit dem Wort `abab` durch. Wo und warum hält er?

Aufgabe 12 (formale Beschreibung). Geben Sie die Automaten als Tupel an (Mengen ausschreiben, δ bzw. ω dürfen auf die Tabelle verweisen): (a) den Moore-Automaten für 1101 aus 1.2, (b) den Strichbreiten-Automaten aus 2.3, (c) den DEA „endet auf *ab*“ aus 2.4.

Aufgabe 13 (Minimieren am Barcode). Minimieren Sie den Strichbreiten-Automaten aus 2.3 (vollständige Tabelle mit Fangzustand F) mit dem Tabellenverfahren. Welche Zustände werden verschmolzen? Begründen Sie das Ergebnis auch inhaltlich: Was „weiß“ der Automat in diesen Zuständen über die bisher gelesene Folge?

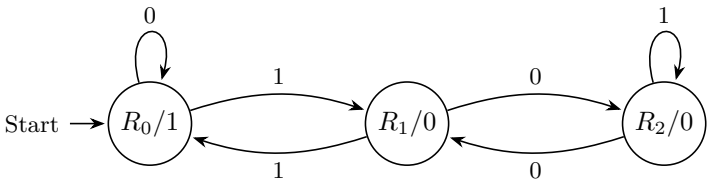
10 Lösungen

Lösung 1. S_0: gerade Anzahl a (Start- und Endzustand), S_1: ungerade. a wechselt, b ändert nichts.



δ	a	b	Endzustand?
S_0 (Start)	S_1	S_0	ja
S_1	S_0	S_1	nein

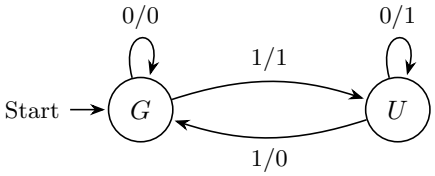
Lösung 2. Zustand R_r = Rest r der bisher gelesenen Zahl; $\delta(R_r, x) = R_{(2r+x) \bmod 3}$.



δ	0	1	Ausgabe
R_0 (Start)	R_0	R_1	1
R_1	R_2	R_0	0
R_2	R_1	R_2	0

`10010`: $R_0 \to R_1 \to R_2 \to R_1 \to R_0 \to R_0$. Am Ende R_0, Ausgabe 1: $10010_2 = 18$ ist durch 3 teilbar.

Lösung 3. G: bisher gerade Anzahl Einsen, U: ungerade (Start G).

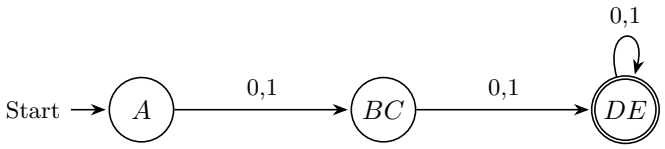


Ausgabe für `1101`: `1001`.

Lösung 4. F ist von A aus nicht erreichbar und wird gestrichen. Matrix (Zahl = Runde der Markierung):

	A	B	C	D
B	1			
C	1	∘		
D	0	0	0	
E	0	0	0	∘

Klassen $\{A\}$, $\{B, C\}$, $\{D, E\}$. Der Minimalautomat akzeptiert alle Wörter mit *mindestens zwei Zeichen*:



Lösung 5. Teilmengentabelle (4 DEA-Zustände):

DEA-Zustand	a	b	Endzustand?
$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$	nein
$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0, q_1, q_2\}$	nein
$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	ja
$\{q_0, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	ja

Lösung 6. Aus dem Automaten mit S_0 (Start, Ende) und S_1: $S_0 \to aS_1 \mid bS_0 \mid b$, $S_1 \to aS_0 \mid a \mid bS_1$. Das leere Wort lässt die Grammatik weg ($S_0 \to \varepsilon$ wäre nur erlaubt, wenn S_0 auf keiner rechten Seite vorkommt).
Ableitung: $S_0 \Rightarrow aS_1 \Rightarrow abS_1 \Rightarrow abbS_1 \Rightarrow abba$.

Lösung 7. $S \to aSb \mid c$, kontextfrei (Typ 2): $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaacbbb$. Ein DEA müsste die Anzahl der a unbegrenzt mitzählen, um sie mit den b zu vergleichen (wie bei $a^n b^n$).

Lösung 8. (a) Typ 3 (regulär) (b) Typ 2 (kontextfrei, S steht in der Mitte) (c) Typ 3 (d) Typ 1 (kontextsensitiv): $AB \to BA$ hat links zwei Zeichen, ist also nicht kontextfrei; keine Regel verkürzt ($|\alpha| \leq |\beta|$).

Lösung 9.

Schritt	Resteingabe	Keller
0	(())()#	Z
1	())()#	KZ
2	))()#	KKZ
3	)()#	KZ
4	()#	Z
5	)#	KZ
6	#	Z
7	ε	ε

Bei `())(#` ist nach den ersten beiden Zeichen der Keller wieder auf Z. Die dritte Klammer `)` trifft auf Z – dafür gibt es keinen Übergang, der Automat bleibt nach 2 Zeichen stecken.

Lösung 10.

Zustand	liest	neuer Zustand	schreibt	Kopf
q_0	`0`	q_0	`1`	R
q_0	`1`	q_0	`0`	R
q_0	`_`	halt	`_`	N

Schritt	Zustand	Band (Kopf in [])
0	q_0	`[0]110`
1	q_0	`1[1]10`
2	q_0	`10[1]0`
3	q_0	`100[0]`
4	q_0	`1001[_]`
5	halt	`1001[_]`

Lösung 11.

Schritt	Zustand	Band (Kopf in [])
0	q_0	`[a]bab`
1	q_1	`X[b]ab`
2	q_2	`[X]Yab`
3	q_0	`X[Y]ab`
4	q_3	`XY[a]b`
5	nein	`XY[a]b`

Nach dem ersten Paar (`XY`) findet q_0 kein `a` mehr, sondern ein `Y` und geht nach q_3, um zu prüfen, ob nur noch `Y` folgen. Dort steht aber noch ein `a` (q_3 liest `a`): kein Übergang, die Maschine hält mit *nein*. Richtig so, denn `abab` hat nicht die Form $a^n b^n$.

Lösung 12. (a)

Formal: $A = (X, Y, S, s_0, \delta, \omega)$ mit $X = \{0, 1\}$; $S = \{S_0, S_1, S_2, S_3, S_4\}$; $s_0 = S_0$; $Y = \{0, 1\}$; δ und $\omega : S \to Y$: siehe Tabelle.

(b)

Formal: $A = (X, S, s_0, \delta, E)$ mit $X = \{Z, \text{-}\}$ (Z = Breite 1–4); $S = \{S_0, \text{A}, \text{B}, \text{C}, \text{D}, \text{F}\}$; $s_0 = S_0$; $E = \{\text{A}\}$; δ: siehe Tabelle.

(c)

Formal: $A = (X, S, s_0, \delta, E)$ mit $X = \{a, b\}$; $S = \{S_0, S_1, S_2\}$; $s_0 = S_0$; $E = \{S_2\}$; δ: siehe Tabelle.

Lösung 13. Alle Zustände sind erreichbar. Matrix (Zahl = Runde der Markierung):

	S_0	A	B	C	D
A	0				
B	1	0			
C	1	0	2		
D	∘	0	1	1	
F	1	0	3	2	1

Runde 0: $\{S_0, \text{A}\}$, $\{\text{A}, \text{B}\}$, $\{\text{A}, \text{C}\}$, $\{\text{A}, \text{D}\}$, $\{\text{A}, \text{F}\}$; Runde 1: $\{S_0, \text{B}\}$, $\{S_0, \text{C}\}$, $\{S_0, \text{F}\}$, $\{\text{B}, \text{D}\}$, $\{\text{C}, \text{D}\}$, $\{\text{D}, \text{F}\}$; Runde 2: $\{\text{B}, \text{C}\}$, $\{\text{C}, \text{F}\}$; Runde 3: $\{\text{B}, \text{F}\}$. In Runde 4 ändert sich nichts mehr. Unmarkiert bleibt nur $\{S_0, D\}$: Beide sind keine Endzustände, beide führen mit Z nach A und mit - nach F. Inhaltlich erwartet der Automat in beiden als Nächstes einen *schwarzen Strich* – ob am Anfang oder nach einer Lücke, spielt für den Rest der Folge keine Rolle. D wird mit S_0 verschmolzen, der Übergang $C \xrightarrow{-} D$ wird zu $C \xrightarrow{-} S_0$. Der Automat in 2.3 ist also korrekt, aber nicht minimal: D macht ihn leichter verständlich, ist aber überflüssig. Minimalautomat mit 5 statt 6 Zuständen (F wie in 2.3 nicht eingezeichnet):

