

# Automaten und Sprachen

Prof. Dr. Carsten Lutz

Paulinum, Raum P810

[clu@informatik.uni-leipzig.de](mailto:clu@informatik.uni-leipzig.de)



# Organisatorisches



# Zeit + Ort

**Vorlesung:**

Mi 17–19, HS 1 (+ Stream)

**Übungen:**

Mo 13-15 S313 (c,d)

Mo 17-19 S310 (a, b)

Di 11-13 S310 (e,f)

Mi 11-15 S310 (e, f)

(Gehalten von Maurice Funk und Moritz Schönherr)

**Position im Curriculum:**

3. Semester

Pflichtmodul B.Sc. Informatik

Verwendbar für Lehramt Informatik und Digital Humanities



# Vorlesungsmaterial

**Skript in Moodle. Diese VL: Teile I + II der insgesamt 4 Teile**

Bei Interesse und Bedarf außerdem:

1. John Hopcroft, Rajeev Motwani, Jeff Ullmann: [Introduction to Automata Theory, Languages, and Computation](#) (3rd edition). Pearson Education, 2014 (*dt. Version erhältlich*)
2. Dexter Kozen: [Automata and Computability](#). Springer 2007
3. Uwe Schöning: [Theoretische Informatik – kurzgefasst](#). Spektrum Akademischer Verlag, 2008
4. Ingo Wegener: [Theoretische Informatik – Eine algorithmenorientierte Einführung](#). Teubner, 2005

Die Vorlesungen werden aufgezeichnet und in Moodle bereitgestellt  
(knapp 1 Woche Verzögerung!)



## Hausaufgaben + Klausur:

- Hausaufgaben jede zweite Woche, insgesamt 6 Blätter
- Werden in Gruppen (**2 Personen**) bearbeitet, abgegeben und korrigiert (bitte Vor- und Nachnamen auf Abgabe!)
- Es müssen **50%** erreicht werden (gemittelt über alle Blätter), um zur Klausur zugelassen zu werden
- Werden **mindestens 85% erreicht**, so verbessert sich die Klausurnote um eine Stufe (+0,3)
- 1-stündige **Klausur** am Ende des Semesters

Auf jedem Blatt sind **Übungsaufgaben** (werden in Übung gelöst) und **Hausaufgaben** (werden in 2er-Gruppe gelöst und abgegeben)

# Abgabe

- Aufgabenblätter erscheinen in Moodle
- Abgabe der Lösungen ebenfalls in Moodle bis jeweils Montag 12h als PDF, jeweils nur eine Datei pro Gruppe
- Die Noten erfahrt Ihr ebenfalls über Moodle
- Erste Abgabe am 10.11.
- Zweiergruppen in Moodle eintragen!

Übungsbetrieb startet ab 27.10.!

Keine Vorlesung am 19.11. (Buß- und Betttag) und  
28.1. (Tag der Lehre)

Keine Übung am Di, 02.12. (Dies Academicus)  
Bitte Alternativtermin wählen!



## Beim Lösen der Übungsaufgaben gilt:

- Die Lösungen müssen **von Euch selbst** erarbeitet worden sein (in der Gruppe).
- Ihr dürft Euch natürlich in Lehrbüchern und im Netz belesen
- Ihr müsst dann aber zu einer **eigenen** Lösung kommen.
- Solltet Ihr Teile aus externen Quellen übernehmen, dann müsst Ihr das angeben – sonst ist es ein Täuschungsversuch!
- Dann wird aber nur Euer Eigenanteil gewertet

# Theoretische Informatik: Eine kurze Einführung



# Theoretische Informatik

... schafft **formale** und **kulturelle Grundlage** für die Informatik.

## Kultur:

- Gemeinsames Grundwissen:  
Welche allgemeinen **Konzepte und Methoden**  
sind zentral für die Disziplin und sind „Common Knowledge“?
- Gemeinsame Sprache:  
Welche **zentralen Begriffe** werden von allen verstanden?



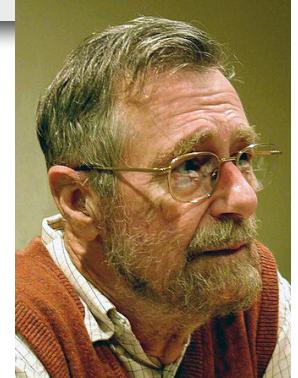
Clipart: helix84, Wikipedia (CC BY-SA 3.0)



# Theoretische Informatik

In der Informatik geht es genauso wenig um Computer wie in der Astronomie um Teleskope.

(Edsger W. Dijkstra, 1930–2002, niederländ. Informatiker, Turing-Award 1972)



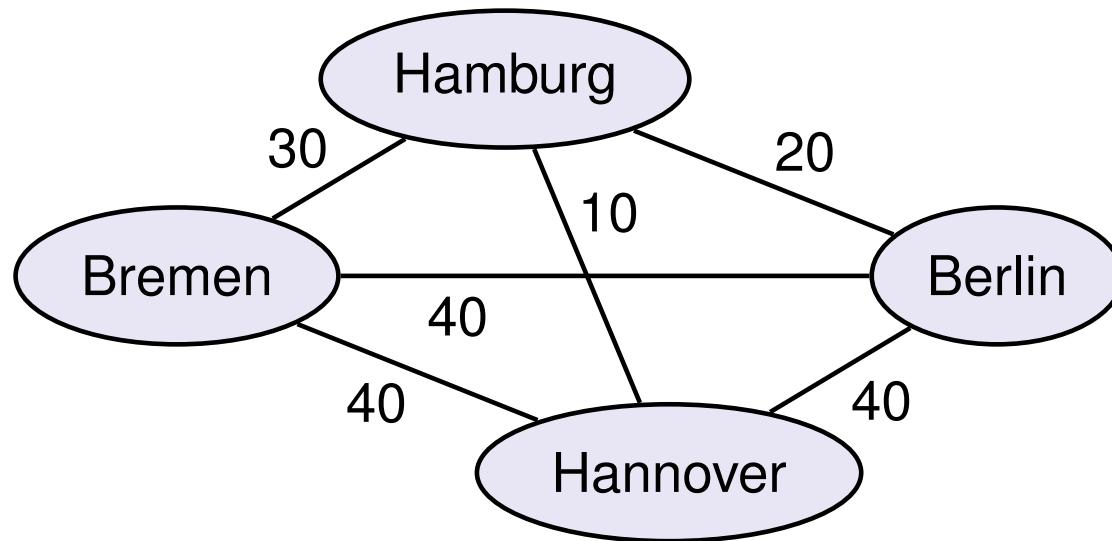
Edsger W. Dijkstra  
Wikipedia, CC BY-SA 3.0  
Autor: Hamilton Richards

## Schwerpunkte:

- Schaffen von mathematischen Modellen und Abstraktionen
  - Unwichtiges ausblenden, Wesentliches klar herausstellen
  - Grundlage für exakte math. Behandlung informatischer Fragestellungen
- Berechnungsmodelle und algorithmischen Techniken
  - Abstrakte Modelle von Computern, Programmiersprachen etc.
  - Wie unterscheiden sich PC, Tablet, Quantencomputer, DNA-Computer?
- Verständnis der Grenzen der (effizienten) Berechenbarkeit
  - Kann man alles berechnen, was man beschreiben kann (bei vollständiger Information)? Wie effizient kann man Dinge berechnen?

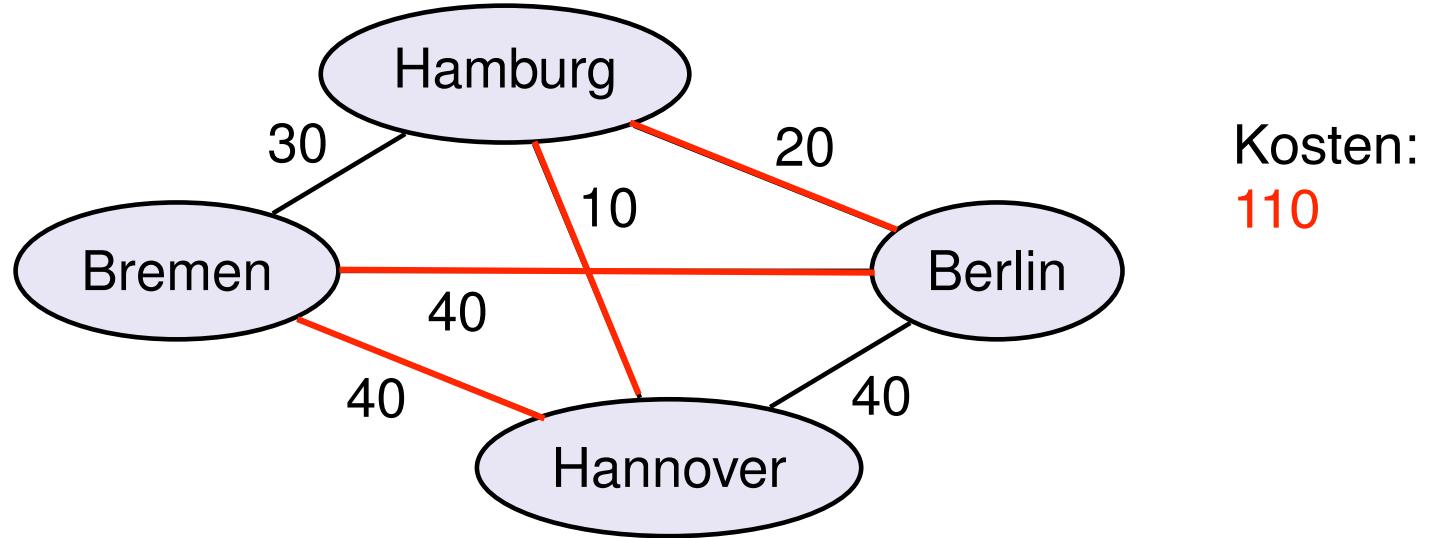
# Theoretische Informatik – Beispiel 1

**Aufgabe:** Finde den günstigsten Weg für eine Rundreise



# Theoretische Informatik – Beispiel 1

**Aufgabe:** Finde den günstigsten Weg für eine Rundreise



Das Programm ist nicht sehr effizient?

Das liegt nicht am Programmierer/der Programmiererin!

Denn:

- Es ist unbekannt, ob dieses Problem (**Traveling Salesman**) effizient lösbar ist.
- Frage ist äquivalent zum **wichtigsten offenen Problem** in der Informatik/Mathematik: dem P vs NP Problem.



„Ich kann keinen effizienten Algorithmus finden, bin wohl zu dumm.“



„Ich kann keinen effizienten Algorithmus finden, aber  
all diese anderen berühmten Leute können das auch nicht.“

©

M. R. Garey, D. S. Johnson.  
*Computers and Intractability: A Guide to the Theory of NP-Completeness.*  
Freeman, New York, 1979.

# Theoretische Informatik – Beispiel 2

**Aufgabe:** Entwurf eine Raketensteuerung



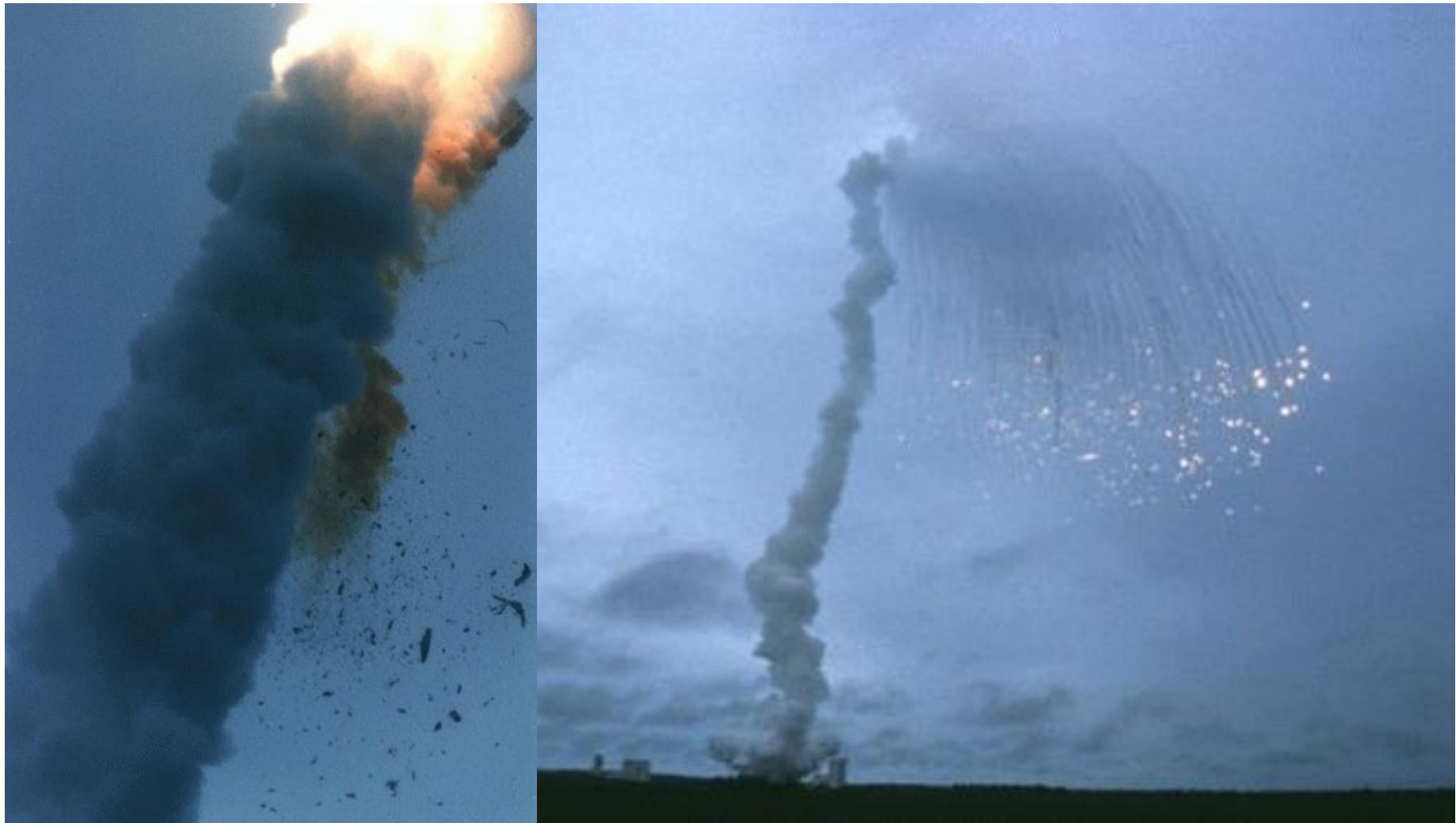
© ESA (European Space Agency)



UNIVERSITÄT  
LEIPZIG

# Theoretische Informatik – Beispiel 2

**Aufgabe:** Entwurf eine Raketensteuerung



© ESA (European Space Agency)

**Problem:** Fehler in der Steuersoftware



UNIVERSITÄT  
LEIPZIG

# Theoretische Informatik – Beispiel 2

Klassische Methode zum Finden von Bugs:

**Testen!** (nach Möglichkeit systematisch)

**Problem:** in der Regel zu viele mögliche Eingaben, um alle zu testen!

In kritischen Anwendungen viel besser: **Verifikation**

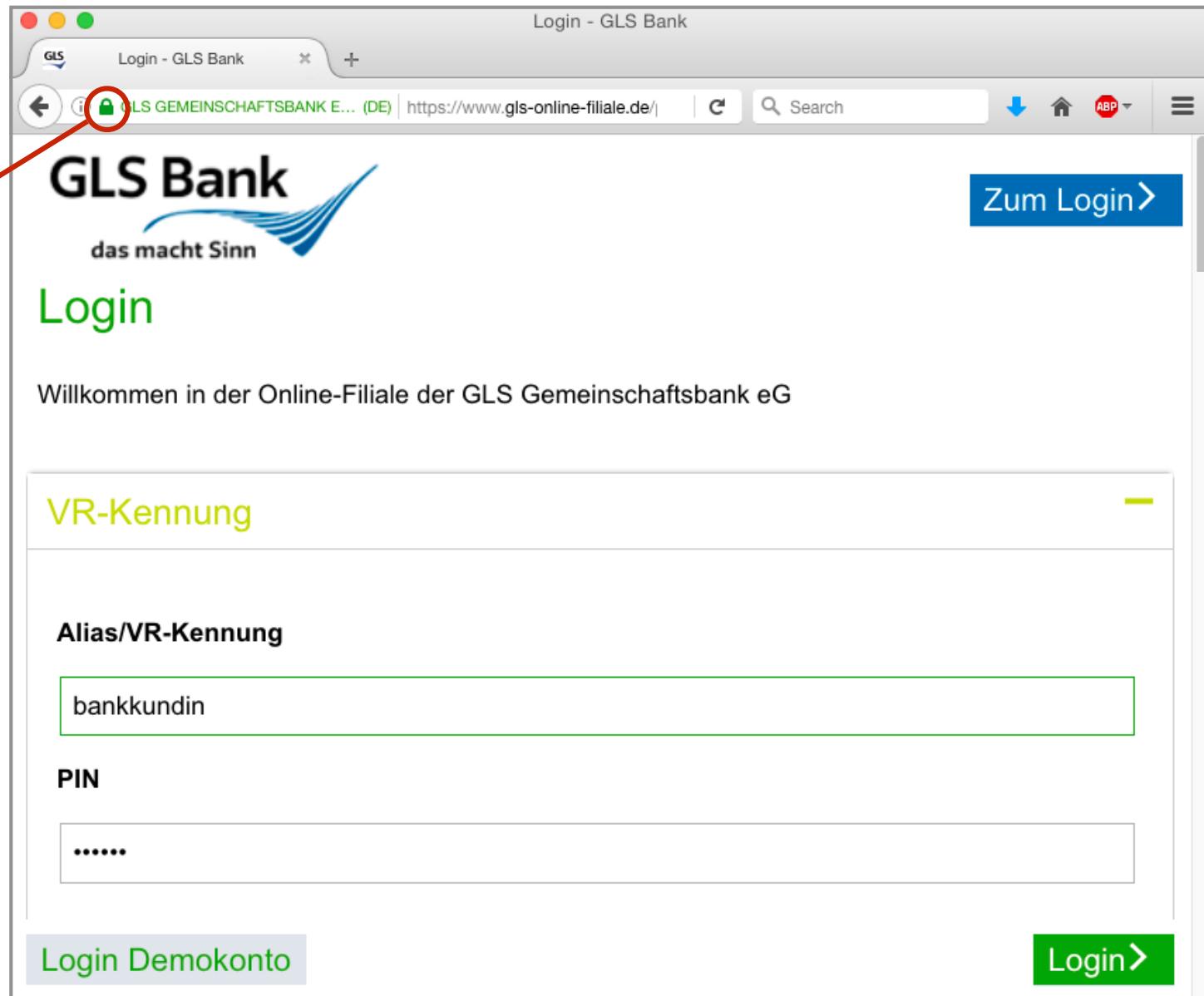
- erlaubt einen formalen Beweis der Korrektheit  
(automatische Analyse des Programms, kein Testen)
- basiert auf mathematischen Methoden, insbesondere Logik
- Teilgebiet der theoretischen Informatik



# Theoretische Informatik – Beispiel 3

**Aufgabe:** Verwende Online-Banking ohne bestohlen zu werden

Was genau  
bedeutet das?



# Theoretische Informatik – Beispiel 3

Das Schloss bedeutet natürlich **verschlüsselte Übertragung**.

Aber es bleiben berechtigte Fragen:

- ➊ Kann jemand den Schlüssel **abfangen**?  
**Sehr leicht sogar, aber das macht nichts.**
- ➋ Kann man **ganz sicher sein**, dass niemand einen Trick gefunden hat, die Verschlüsselung zu brechen?  
**Nein, kann man nicht!**
- ➌ Kann ich der Verschlüsselung **vertrauen**?  
**Ja, durchaus! (für den Hausgebrauch)**

Diese Fragen werden in der **Kryptographie** studiert.



... besteht aus **vielen Teilgebieten:**

- Automaten und formale Sprachen
- Algorithmentheorie
- Berechenbarkeits- und Komplexitätstheorie
- Verifikation und mathematische Logik
- Kryptographie
- Datenbanktheorie
- etc.

**Diese VL**

1. Semester

4. Semester



# Zur Rolle der Mathematik

Exakte und formale Beschreibungen essentiell in der theoretischen Informatik

Lasst Euch davon nicht abschrecken:

- Mit etwas gutem Willen erkennt man leicht, warum die mathematischen Definitionen und Resultate **wichtig für die Informatik** sind.
- Mathe ist nicht gleich Mathe:
  - ▶ Wir werden hier oft **andere mathematische Methoden** benötigen, als Ihr aus der Schule kennt.
  - ▶ Wir wollen nicht rechnen (= langweilig), sondern Dinge **beweisen (= verstehen)**.

Bei mathematischen Themen gilt stets: Übung ist extrem wichtig!



# Automaten und formale Sprachen: Eine Einführung



# Grundlegende Definitionen

## Alphabet:

- ist eine **endliche** Menge von **Symbolen**
- wir verwenden meist  $\Sigma$  für Alphabete;  $a, b, c$  etc. für Symbole
- Beispiele:
  - \*  $\Sigma = \{a, b, \dots, z\}$        $\Sigma = \{0, 1\}$        $\Sigma = \{0, \dots, 9\} \cup \{,\}$
  - \*  $\Sigma$  ist das standardisierte Alphabet ISO-8859-1 (aka Latin-1)
  - \*  $\Sigma = \{ \text{program, const, var, label, procedure, function, type, begin, end, if, then, else, case, of, repeat, until, while, do, for, to} \}$   
 $\cup \{ \text{VAR, VALUE, FUNCTION} \}$



# Grundlegende Definitionen

## Wort:

- ist eine endliche **Folge** von Symbolen
- $w = a_1 \cdots a_n$  mit  $a_i \in \Sigma$  heißt **Wort über dem Alphabet  $\Sigma$**
- Beispiele:
  - \*  $w = abc$        $w = 1000110$        $w = „,10,0221,4292,,$
  - \* Jedes Pascal-Programm kann als Wort über
$$\Sigma = \{ \text{program, const, var, label, procedure, function, type, begin, end, if, then, else, case, of, repeat, until, while, do, for, to} \}$$
$$\cup \{ \text{VAR, VALUE, FUNCTION} \}$$
betrachtet werden  
(wenn man von Variablennamen, Werten, Funktionsnamen abstrahiert)
  - \* Die leere Folge von Buchstaben ist auch ein Wort: das **leere Wort  $\varepsilon$**



# Grundlegende Definitionen

## Formale Sprache:

- ist eine **endliche oder unendliche** Menge von Wörtern
- $\Sigma^*$  ist die Menge aller Wörter über  $\Sigma$  
$$\Sigma^+ = \underbrace{\Sigma^*}_{\text{(d. h. alle Wörter außer dem leeren)}} \setminus \{\epsilon\}$$
- $\underbrace{L \subseteq \Sigma^*}$  heißt **Sprache über dem Alphabet  $\Sigma$**   
d. h. beliebige Menge von Wörtern
- Beispiele:
  - \*  $L = \emptyset$  (die leere Sprache)  $L = \{abc\}$   $L = \{a, b, c, ab, ac, bc\}$
  - \*  $L = \{w \in \{a, \dots, z\}^+ \mid w \text{ ist ein Wort der deutschen Sprache}\}$
  - \* Die Menge aller Wörter über
$$\Sigma = \{ \text{program, const, var, label, procedure, function, type, begin, end, if, then, else, case, of, repeat, until, while, do, for, to } \} \cup \{ \text{VAR, VALUE, FUNCTION } \},$$

die **wohlgeformte Pascal-Programme** beschreiben



# Formale Sprachen in der Informatik

Formale Sprachen sind in der Informatik ein wichtiger  
**Abstraktionsmechanismus:**

- Menge aller wohlgeformten Pascal-/Java-/Python-/...Programme
- Menge aller wohlgeformten Eingaben für ein Programm/Website
  - z. B. Menge aller Kontonummern, Menge aller Geburtsdaten
- Jeder Suchausdruck definiert eine formale Sprache
  - z. B. finde alle Dokumente, die die Wörter „Universität“ und „Leipzig“ enthalten
- Kommunikationsprotokolle
  - z. B. Menge aller wohlgeformten TCP-Pakete
- „Erlaubtes Verhalten“ von Software- und Hardwaresystemen
  - z. B. die erlaubten Verhaltensweisen eines Moduls zur Raketensteuerung ...



# Wichtige Fragestellungen

## Fragestellung 1: Charakterisierung

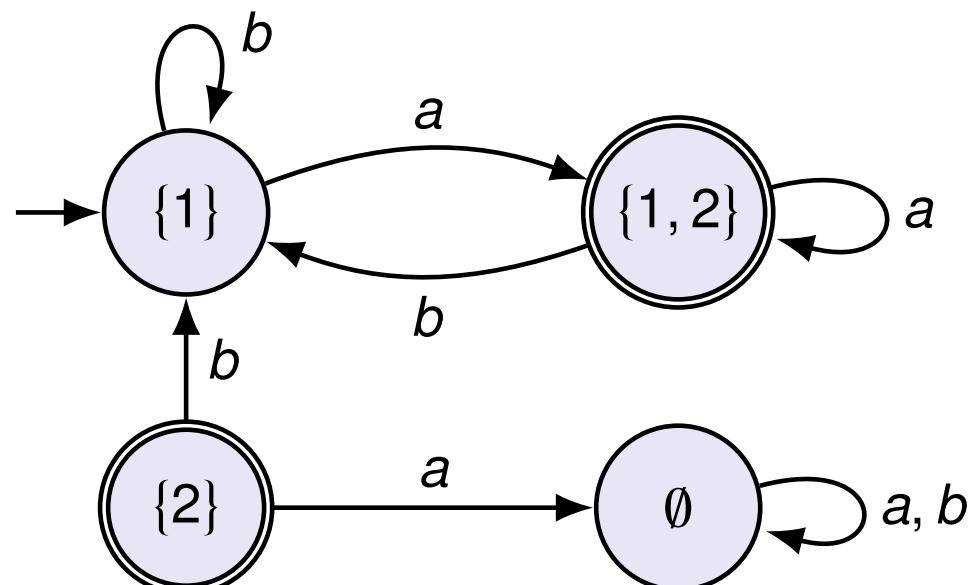
Interessante Sprachen sind meist **unendlich**.

Wie **beschreibt** man unendliche Sprachen **mit endlichen Mitteln**?

- mittels **Automaten**:

abstrakte Maschinen, die ein Wort als Eingabe erhalten und dieses genau dann **akzeptieren**, wenn es zur Sprache gehört (z. B. endliche Automaten, Kellerautomaten, Turing-Maschinen)

Beispiel:



# Wichtige Fragestellungen

## Fragestellung 1: Charakterisierung

Interessante Sprachen sind meist **unendlich**.

Wie **beschreibt** man unendliche Sprachen **mit endlichen Mitteln**?

- mittels **Grammatiken**:

Regelsysteme, die genau die Wörter der Sprache **generieren**

(z. B. kontextfreie Grammatiken für die Syntax von  
Programmiersprachen)

$S$

$\underline{aSBc}$

$aabc\underline{Bc}$

$aab\underline{Bcc}$

$aabbcc$

Beispiel:

$G = (N, \Sigma, P, S)$  mit

$N = \{S, B\}$

$\Sigma = \{a, b, c\}$

$P = \{ \begin{array}{l} S \rightarrow aSBc, \\ S \rightarrow abc, \\ cB \rightarrow Bc, \\ bB \rightarrow bb \end{array} \}$



# Wichtige Fragestellungen

## Fragestellung 1: Charakterisierung

Interessante Sprachen sind meist **unendlich**.

Wie **beschreibt** man unendliche Sprachen **mit endlichen Mitteln**?

- mittels **Ausdrücken**:

beschreiben, wie man die Sprache aus Basissprachen  
mittels geeigneter Operationen (z.B. Vereinigung)  
generieren kann

(z.B. reguläre Ausdrücke)

Beispiel:

$$(a + b)^* ab (a + b)^*$$



# Die Chomsky-Hierarchie

Wir werden verschiedene **Typen von Sprachen** unterscheiden; diese haben recht unterschiedliche Eigenschaften.

Klasse	Automatenmodell	Grammatikart
Typ 0	Turingmaschine (TM)	allgemeine Chomsky-Grammatik
Typ 1	TM mit linearer Bandbeschränkung	monotone Grammatik
Typ 2	Kellerautomat	kontextfreie Grammatik
Typ 3	endlicher Automat	einseitig lineare Grammatik

Die in dieser Vorlesung behandelten Typen sind 2 und 3:

**Typ 3:** werden z. B. als Suchpattern bei der Textsuche verwendet

**Typ 2:** können weitgehend die Syntax von Programmiersprachen beschreiben



# Wichtige Fragestellungen

## Fragestellung 2: Algorithmische Probleme

Was sind die relevanten Berechnungsprobleme für formale Sprachen?  
Wie löst man sie algorithmisch für die jeweilige Sprachklasse?

- **Wortproblem**

Gegeben: eine Beschreibung der Sprache  $L$  (als Automat, Grammatik, . . . )  
und ein Wort  $w$

Frage: Gilt  $w \in L$  ? (d. h. gehört  $w$  zu  $L$  ?)

### Anwendungsbeispiele:

- \* Programmiersprache, deren Syntax durch eine Grammatik beschrieben ist.  
Entscheide, ob ein gegebenes Stück Code syntaktisch korrekt ist.
- \* Suchpattern als regulärer Ausdruck.  
Suche die Dateien (= Wörter), die das Suchpattern enthalten (= zu der durch den Ausdruck beschriebenen Sprache gehören).



# Wichtige Fragestellungen

## Fragestellung 2: Algorithmische Probleme

Was sind die relevanten Berechnungsprobleme für formale Sprachen?

Wie löst man sie algorithmisch für die jeweilige Sprachklasse?

- **Leerheitsproblem**

Gegeben: eine Beschreibung der Sprache  $L$

Frage: Gilt  $L \neq \emptyset$ ? (d. h. ist die Menge  $L$  leer?)

**Anwendungsbeispiel:**

Wenn ein Suchpattern die leere Sprache beschreibt,  
so braucht man die Dateien gar nicht zu durchsuchen.

- **Äquivalenzproblem**

Beschreiben zwei Beschreibungen dieselbe Sprache?

**Anwendungsbeispiel:**

Ein komplexes Suchpattern soll durch ein einfacheres ersetzt werden.  
Sind die beiden Patterns äquivalent?



# Wichtige Fragestellungen

## Fragestellung 3: Abschlusseigenschaften

Unter welchen Operationen auf Sprachen  
(wie Schnitt  $L_1 \cap L_2$ , Vereinigung  $L_1 \cup L_2$ , Komplement  $\bar{L}$ )  
ist die Sprachklasse abgeschlossen?

Sind mathematisch von sehr großem Interesse

Anwendungsbeispiele:

Suche nach Dokumenten, die

- \* ein Suchpattern **nicht** enthalten (**Komplement**)
- \* zwei Pattern **gleichzeitig** enthalten (**Schnitt**)



## Teil I: Endliche Automaten und reguläre Sprachen



0. Grundbegriffe
1. Endliche Automaten
2. Nachweis der Nichterkennbarkeit
3. Abschlusseigenschaften
4. Entscheidungsprobleme
5. Reguläre Ausdrücke und Sprachen
6. Minimale DEAs und die Nerode-Rechtskongruenz

## Teil II: Grammatiken, kontextfreie Sprachen und Kellerautomaten

7. Die Chomsky-Hierarchie
8. Rechtslineare Grammatiken und reguläre Sprachen
9. Normalformen und Entscheidungsprobleme
10. Abschlusseigenschaften und Pumping-Lemma
11. Kellerautomaten
12. Die Struktur kontextfreier Sprachen

# Zur Erinnerung

**Alphabet:** endliche Menge von Symbolen; geschrieben  $\Sigma$

z. B.  $\Sigma = \{a, b\}$

**Wort:** endliche Folge von Symbolen; geschrieben  $w = a_1 \cdots a_n$

z. B.  $abba, bab, \varepsilon$

**Menge aller Wörter** über  $\Sigma$ : geschrieben  $\Sigma^*$

**Formale Sprache:** Teilmenge  $L \subseteq \Sigma^*$  (endlich oder unendlich)

z. B.:  $\{\varepsilon, abba\}$

alle Wörter über  $\Sigma = \{a, b\}$ , die mindestens ein  $a$  enthalten  
 $\Sigma^*$



# Operationen auf Wörtern und Sprachen

## Präfix, Infix, Suffix

$u$  ist **Präfix** von  $v$  wenn  $v = \textcolor{red}{u}w$  mit  $u, w \in \Sigma^*$

$u$  ist **Suffix** von  $v$  wenn  $v = w\textcolor{red}{u}$  mit  $u, w \in \Sigma^*$

$u$  ist **Infix** von  $v$  wenn  $v = w_1 \textcolor{red}{u} w_2$  mit  $u, w_1, w_2 \in \Sigma^*$

Statt „Infix“ sagen wir auch „**Teilwort**“.

### Beispiel

$ab$  ist Präfix von  $abba$

$a$  ist Suffix von  $abba$

$b$  ist Infix von  $abba$

$abba$  ist Präfix/Suffix/Infix von  $abba$

$\varepsilon$  ist Präfix/Suffix/Infix von  $abba$



# Operationen auf Wörtern und Sprachen

## Konkatenation

... kann auf Wörter **und** Sprachen angewendet werden:

Auf Wörtern:       $u \cdot v := uv$

Auf Sprachen:       $L_1 \cdot L_2 := \{u \cdot v \mid u \in L_1 \text{ und } v \in L_2\}$

Konkatenationspunkt wird häufig weggelassen.

## Beispiele

$$ab \cdot ba = abba$$

$$\varepsilon \cdot abba = abba$$

Wenn  $L_1 = \{ab, ac\}$  und  $L_2 = \{ba, ca\}$ ,

dann  $L_1 \cdot L_2 = \{abba, abca, acba, acca\}$



# Operationen auf Wörtern und Sprachen

## Boolesche Operationen:

die üblichen Booleschen Mengenoperationen, angewendet auf Sprachen

Vereinigung     $L_1 \cup L_2 := \{w \mid w \in L_1 \text{ oder } w \in L_2\}$

Schnitt             $L_1 \cap L_2 := \{w \mid w \in L_1 \text{ und } w \in L_2\}$

Komplement         $\overline{L_1} := \{w \mid w \in \Sigma^* \text{ und } w \notin L_1\}$

## Beispiele

Wenn  $L_1 = \{ab, ac\}$  und  $L_2 = \{ba, ca\}$ ,

dann  $L_1 \cup L_2 = \{ab, ac, ba, ca\}$  und  $L_1 \cap L_2 = \emptyset$

Wenn  $\Sigma = \{a\}$  und  $L_1 = \{w \in \Sigma^* \mid w \text{ hat ungerade Länge}\}$ ,

dann  $\overline{L_1} = \{w \in \Sigma^* \mid w \text{ hat gerade Länge}\}$



# Operationen auf Wörtern und Sprachen

**Kleene-Stern:** Beliebig (aber nur endlich) oft iterierte Konkatenation von  $L$

Wir definieren zunächst die unendliche Sprachfolge  $L^0, L^1, L^2, \dots$ :

$$L^0 := \{\varepsilon\}$$

$$L^{n+1} := L^n \cdot L \text{ für alle } n \geq 0$$

Anwendung des Kleene-Sterns auf  $L$ :  $L^* := \bigcup_{n \geq 0} L^n$

<b>Beispiel</b>	Wenn $L = \{ab, ba\}$ ,
	dann $L^0 = \{\varepsilon\}$
	$L^1 = \{\varepsilon\} \cdot L = \{ab, ba\}$
	$L^2 = \{ab, ba\} \cdot L = \{abab, abba, baab, baba\}$
	$L^3 = \dots$
	$\vdots$
	$L^* = \{\varepsilon, ab, ba, abab, abba, \dots\}$



# Operationen auf Wörtern und Sprachen

## Kleene-Stern, anders formuliert

$$L^* = \{\varepsilon\} \cup \{w \mid \exists u_1, u_2, \dots, u_n \in L : w = u_1 \cdot u_2 \cdot \dots \cdot u_n\}$$

## Rechengesetze zum Beispiel

$$(L^*)^* = L^*$$

$$L^* \cdot L^* = L^*$$

## Variante, die nicht (zwingend) das leere Wort enthält:

$$L^+ := \bigcup_{n \geq 1} L^n$$

Beachte:  $L^+$  enthält  $\varepsilon$  gdw.  $\varepsilon \in L$



## Ein bisschen zusätzliche Notation

- $|w|$  bezeichnet die Länge des Wortes  $w$  (= Anzahl Symbole)  
z. B.  $|aabba| = 5$ ,  $|\varepsilon| = 0$
- $a^n$  bezeichnet das Wort, das aus  $n$ -mal dem Symbol  $a$  besteht,  
z. B.  $a^3 = aaa$ ,  $b^5 = bbbbb$ ,  $a^0 = \varepsilon$
- $w^n$  bezeichnet das Wort, das aus  $n$ -mal dem Wort  $w$  besteht,  
z. B.  $(abc)^3 = abcabcabc$  (aber  $abc^3 = abccc$ )
- $|w|_a$  bezeichnet die Anzahl Vorkommen des Symbols  $a$  im Wort  $w$ ,  
z. B.  $|abbaa|_a = 3$ ,  $|abbaa|_b = 2$ ,  $|\varepsilon|_a = |\varepsilon|_b = 0$



## Teil I: Endliche Automaten und reguläre Sprachen

- 0. Grundbegriffe
- 1. Endliche Automaten
- 2. Nachweis der Nichterkennbarkeit
- 3. Abschlusseigenschaften
- 4. Entscheidungsprobleme
- 5. Reguläre Ausdrücke und Sprachen
- 6. Minimale DEAs und die Nerode-Rechtskongruenz

## Teil II: Grammatiken, kontextfreie Sprachen und Kellerautomaten

- 7. Die Chomsky-Hierarchie
- 8. Rechtslineare Grammatiken und reguläre Sprachen
- 9. Normalformen und Entscheidungsprobleme
- 10. Abschlusseigenschaften und Pumping-Lemma
- 11. Kellerautomaten
- 12. Die Struktur kontextfreier Sprachen



# §1 Endliche Automaten



- sind **endliches Mittel** zum Beschreiben von (potentiell unendlichen) **formalen Sprachen**
- **Abstraktion** eines Systems (z. B. Software- oder Hardwaresystem) basierend auf **Zuständen und Zustandsübergängen**

## Zustand:

Abstrakte Beschreibung des momentanen Systemzustands,  
Zustand als Name ( $q_0, q_1$  usw.) ohne Beschreibung der Systemdetails

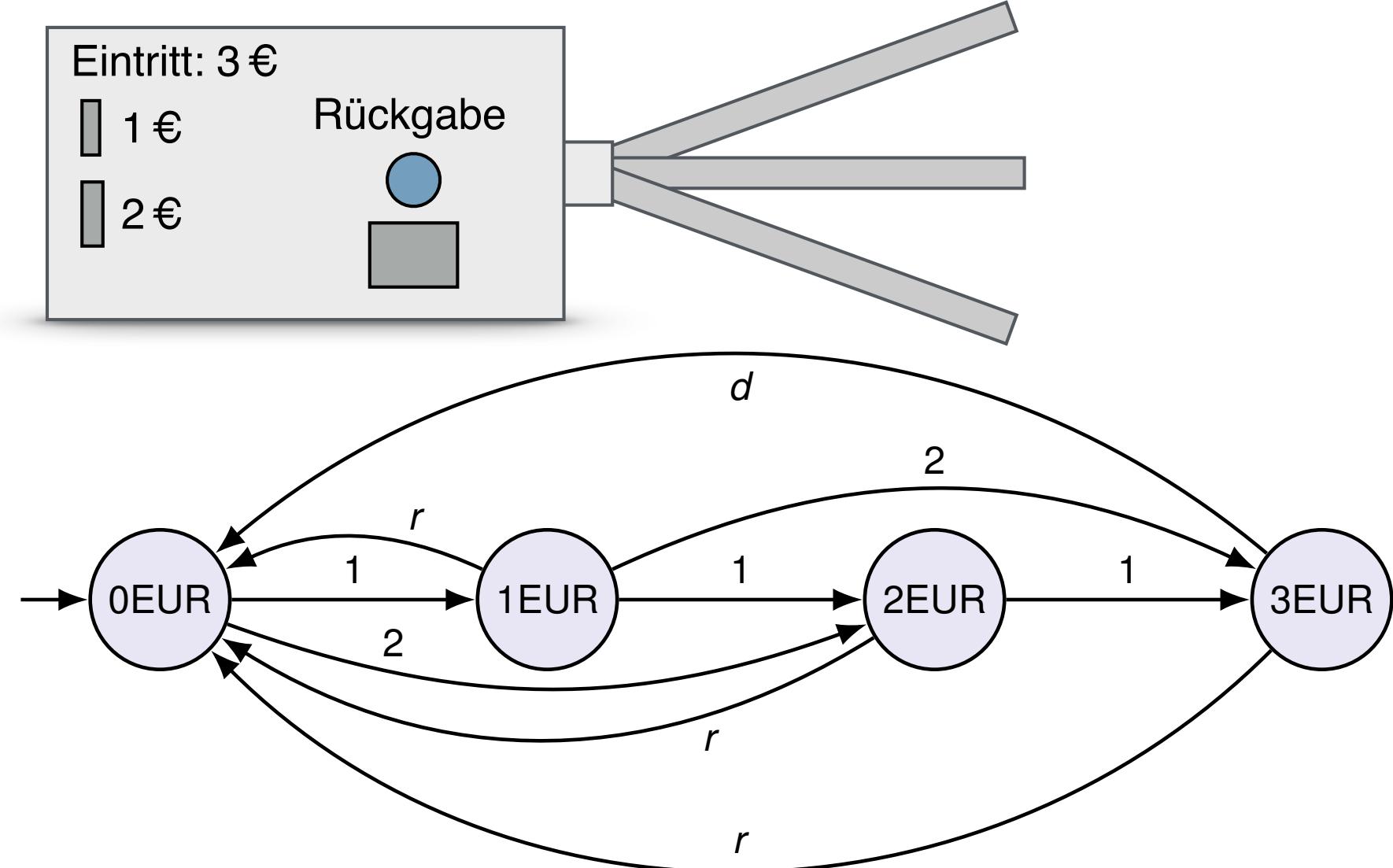
## Zustandsübergänge:

Wechsel zwischen Zuständen, werden als augenblicklich angenommen.  
Ein Lauf des Systems ist also eine Folge von Zuständen.



# Ein Beispiel

## Eintrittsautomat mit Drehsperre



Welche Eingabe führt zum Zustand 3EUR (man darf durchs Drehgitter)?



# Automaten und formale Sprachen

Automaten beschreiben (wir sagen: **erkennen**) formale Sprachen:

1. erhalten **Wörter als Eingabe**, lesen diese von links nach rechts
2. befinden sich nach jedem Leseschritt in genau einem von  
**endlich vielen** möglichen **Zuständen**
3. arbeiten ein „festes Programm“ ab
4. **verwerfen** oder **akzeptieren** Eingabe über den zum Schluss  
erreichten Zustand

Wir werden **mehrere Versionen** von endlichen Automaten kennen lernen.

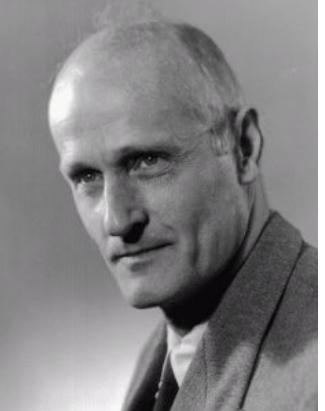


## § 1.1 Deterministische endliche Automaten



# Deterministische endliche Automaten

Stephen C. Kleene  
© Univ. of Wisconsin-Madison



## Definition 1.1 (DEA)

Ein deterministischer endlicher Automat (DEA) ist von der Form  $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ , wobei

- $Q$  eine **endliche** Menge von **Zuständen** ist,
- $\Sigma$  ein **Alphabet** ist (das **Eingabealphabet**),
- $q_0 \in Q$  der **Anfangszustand** ist,
- $\delta : Q \times \Sigma \rightarrow Q$  die **Übergangsfunktion** ist,
- $F \subseteq Q$  eine Menge von **akzeptierenden Zuständen** ist.

Zur Erinnerung:

- $Q \times \Sigma = \{(q, a) \mid q \in Q, a \in \Sigma\}$
- $\delta$  bildet also Paare  $(q, a)$  auf Zustände  $q'$  ab.

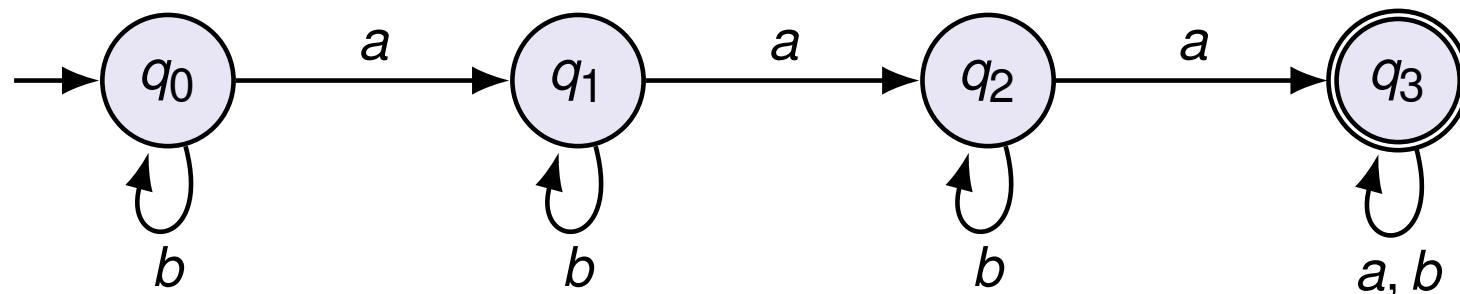


# Ein Beispiel-DEA

## Beispiel 1.2

$\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$  mit

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_0, a) = q_1 \quad \delta(q_1, a) = q_2 \quad \delta(q_2, a) = \delta(q_3, a) = q_3$   
 $\delta(q_i, b) = q_i \text{ für } i \in \{0, 1, 2, 3\}$
- $F = \{q_3\}$



Beispieleingaben:     $aaa$      $aabbabb$      $babbabb$      $\varepsilon$

# Fortsetzung der Übergangsfunktion auf Wörter

## Definition 1.3

Die **kanonische Fortsetzung** von  $\delta : Q \times \Sigma \rightarrow Q$  von einzelnen Symbolen auf Wörter beliebiger Länge ist eine Funktion

$$\hat{\delta} : Q \times \Sigma^* \rightarrow Q,$$

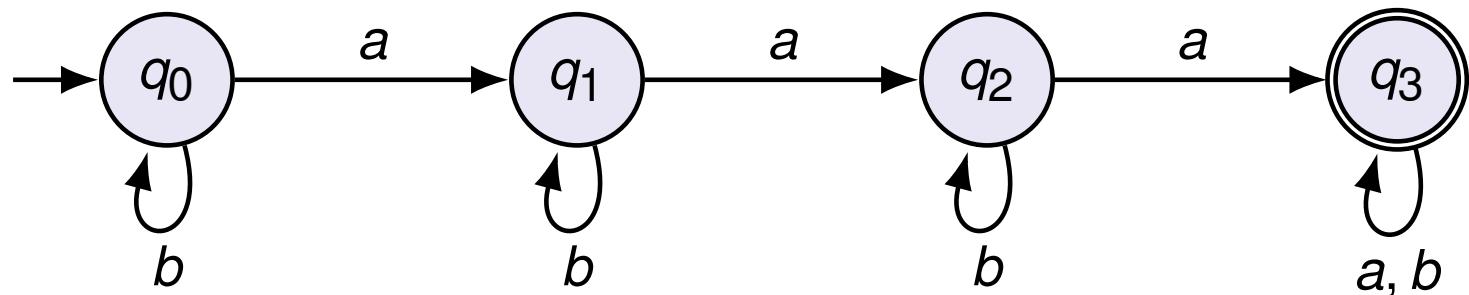
die per Induktion über die Wortlänge definiert ist:

$$\hat{\delta}(q, \varepsilon) := q \quad \hat{\delta}(q, wa) := \delta(\hat{\delta}(q, w), a)$$

**Beachte:**

Daraus folgt  $\hat{\delta}(q, a) = \delta(q, a)$  für alle Zustände  $q \in Q$  und Symbole  $a \in \Sigma$

## Beispiel



$$\hat{\delta}(q_0, babbabb) = \delta(\delta(\delta(\delta(\delta(\delta(\delta(q_0, \varepsilon), b), a), b), b), a), b), b) = q_2$$

# Akzeptanz

## Definition 1.4

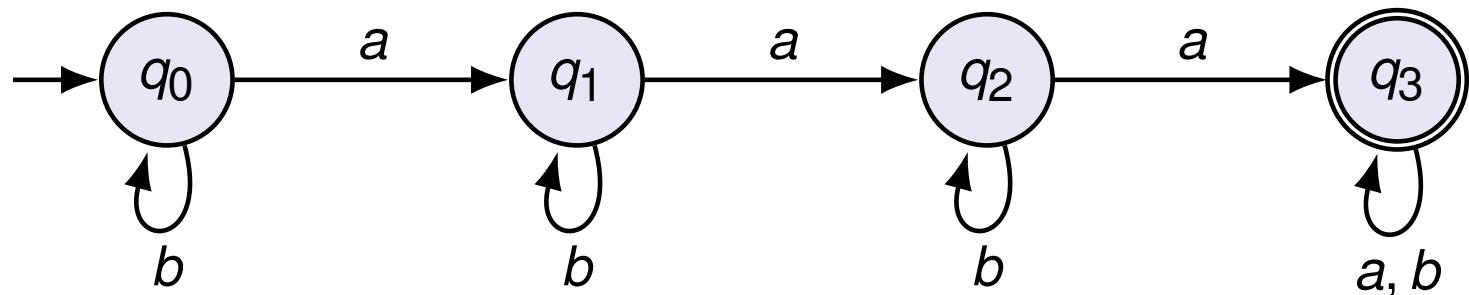
Ein DEA  $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$  akzeptiert das Wort  $w \in \Sigma^*$ , wenn gilt:

$$\hat{\delta}(q_0, w) \in F$$

Die von  $\mathcal{A}$  erkannte Sprache ist:

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

## Beispiel



$$L(\mathcal{A}) = \{w \in \{a, b\}^* \mid |w|_a \geq 3\}$$



## Definition 1.5

Eine Sprache  $L \subseteq \Sigma^*$  heißt **erkennbar**,  
wenn es einen **DEA  $\mathcal{A}$  gibt** mit  $L = L(\mathcal{A})$ .

Also ist z. B.

$$L = \{w \in \{a, b\}^* \mid |w|_a \geq 3\}$$

eine erkennbare Sprache.



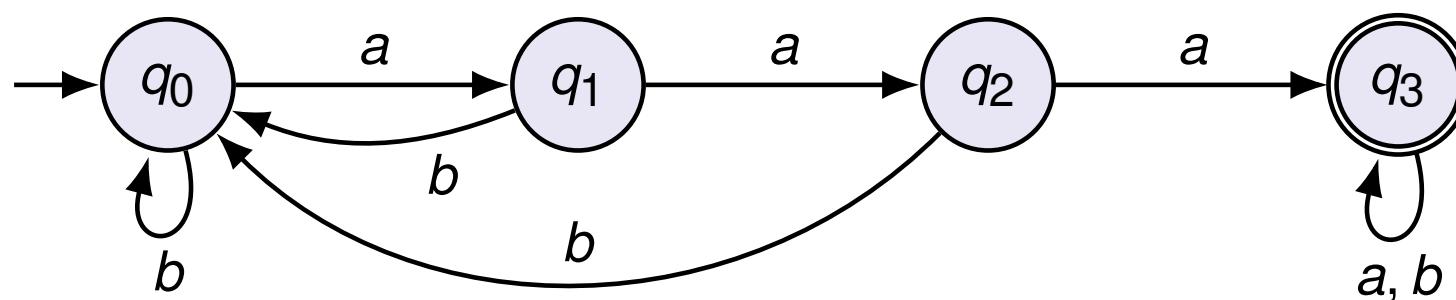
# Weitere Beispiele

## Beispiel 1.6

Die folgende Sprache ist ebenfalls erkennbar:

$$L = \{w = uaaav \mid u, v \in \Sigma^*\} \text{ mit } \Sigma = \{a, b\}$$

DEA  $\mathcal{A}$  mit  $L(\mathcal{A}) = L$ :



Beispieleingaben:      *bbaaabb*      *baaba*



# Weitere Beispiele

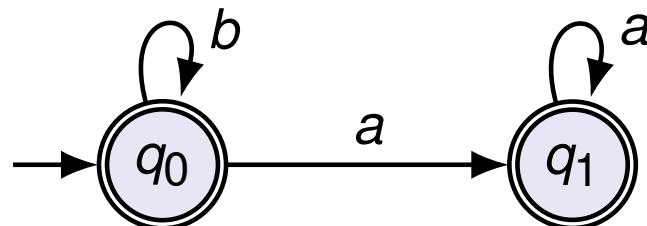
**Beispiel 1.7** Betrachte

$$L = \{w \in \{a, b\}^* \mid ab \text{ ist nicht Infix von } w\}$$

Es hilft, die Beschreibung der Sprache zunächst **umzuformulieren**:

$$L = \{w \in \{a, b\}^* \mid \exists n, k \geq 0 : w = b^n a^k\}$$

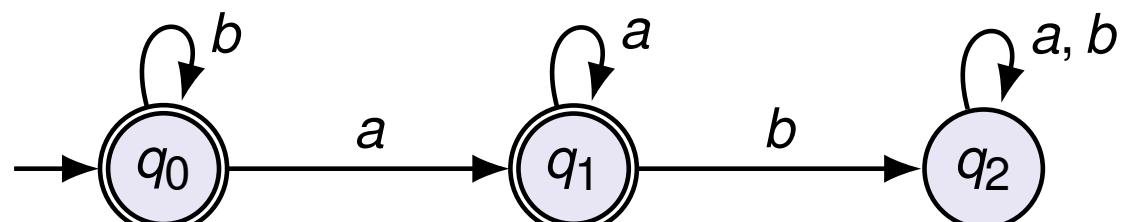
Erster Versuch eines DEAs:



Dies ist **kein** DEA:

Die Übergangsfunktion ist **total**; also muss für **jede mögliche Kombination** von Zustand und Symbol ein **Folgezustand** definiert sein – **fehlt** für  $q_1$  und  $b$ !

Man erhält aber leicht einen DEA durch zusätzlichen **Papierkorbzustand**:



# Weitere Beispiele

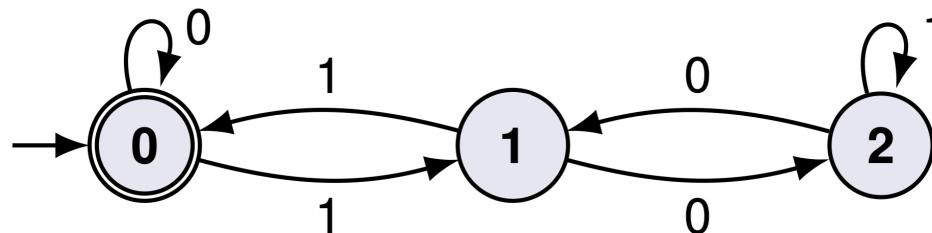
Wörter über  $\Sigma = \{0, 1\}$  können als **Binärzahlen** gelesen werden

Wir wollen zeigen, dass folgende Sprache erkennbar ist:

$$L = \{w \in \{0, 1\}^* \mid w \text{ ist ein Vielfaches von } 3\}$$

(niederwertigstes Bit rechts)

Betrachte folgenden DEA  $\mathcal{A}$ :



Wir behaupten, dass  $L(\mathcal{A}) = L$ .

Das ist natürlich keineswegs offensichtlich

Noch nicht einmal die erkannte Sprache  $L \subseteq \{0, 1\}^*$  ist so ganz einfach zu ermitteln.



# Weitere Beispiele

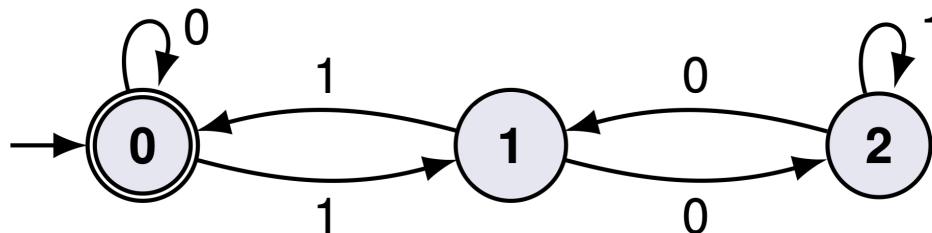
Wörter über  $\Sigma = \{0, 1\}$  können als **Binärzahlen** gelesen werden

Wir wollen zeigen, dass folgende Sprache erkennbar ist:

$$L = \{w \in \{0, 1\}^* \mid w \text{ ist ein Vielfaches von } 3\}$$

(niederwertigstes Bit rechts)

Betrachte folgenden DEA  $\mathcal{A}$ :



Wir zeigen:  $\hat{\delta}(\mathbf{0}, w) = w \bmod 3$  für alle  $w \in \{0, 1\}^*$

Also:  $\hat{\delta}(\mathbf{0}, w) = \mathbf{0}$  wenn  $w$  ohne Rest durch 3 teilbar

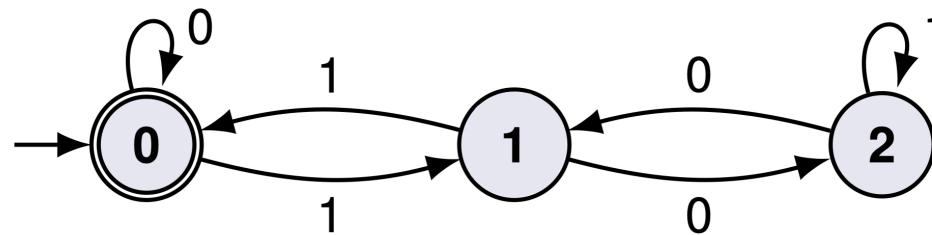
$\hat{\delta}(\mathbf{0}, w) = \mathbf{1}$  wenn  $w$  mit Rest 1 durch 3 teilbar

$\hat{\delta}(\mathbf{0}, w) = \mathbf{2}$  wenn  $w$  mit Rest 2 durch 3 teilbar

Darauf folgt offensichtlich, dass  $L(\mathcal{A}) = L$ .



# Weitere Beispiele



Wir zeigen:  $\hat{\delta}(0, w) = w \bmod 3$  für alle  $w \in \{0, 1\}^*$

Zunächst zwei Beobachtungen:

- Wenn  $w$  binäre Repräsentation von  $n$ , dann  $w0 = 2n$

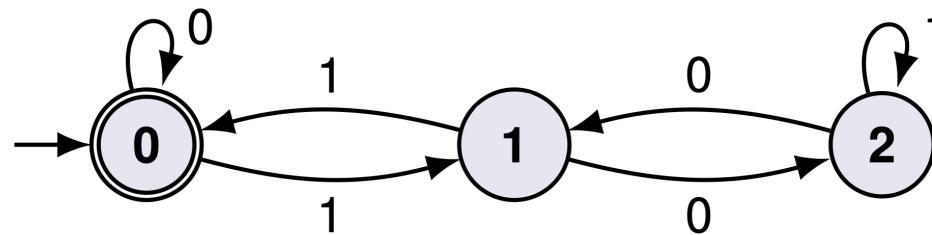
$$w1 = 2n + 1$$

- Im obigen Automaten gilt:

$$(*) \quad \delta(q, a) = (2q + a) \bmod 3 \text{ für alle } q \in \{0, 1, 2\} \text{ und alle } a \in \{0, 1\}$$

Beweis: alle 6 Fälle überprüfen

# Weitere Beispiele



$$(*) \quad \delta(q, a) = (2q + a) \bmod 3 \text{ für alle } q \in \{0, 1, 2\} \text{ und } a \in 0, 1$$

Wir zeigen:  $\hat{\delta}(0, w) = w \bmod 3$  für alle  $w \in \{0, 1\}^*$

Per vollständiger Induktion über die Länge  $|w|$  von  $w$

Induktionsanfang:  $|w| = 0$ , also  $w = \varepsilon \quad \hat{\delta}(0, \varepsilon) = 0$  nach Def.  $\hat{\delta}$

Induktionsschritt: Betrachte Wort  $w = va$  mit  $v \in \Sigma^*$  und  $a \in \{0, 1\}$ .

$$\begin{aligned} \hat{\delta}(0, va) &= \delta(\hat{\delta}(0, v), a) && (\text{Def } \hat{\delta}) \\ &= \delta(v \bmod 3, a) && (\text{Ind Vor.}) \\ &= (2(v \bmod 3) + a) \bmod 3 && (*) \\ &= (2v + a) \bmod 3 && (\text{Rechnen mit mod}) \\ &= va \bmod 3 && (1. \text{ Beobachtung}) \end{aligned}$$



# Reflektion

Wir haben also eine **recht komplexe Sprache** mittels endlichen Automaten beschrieben.

Oder scheint das nur so?

Wir konnten sie schließlich **mit nur drei Zuständen** beschreiben.

Wir werden später in der Vorlesung sehen:

Der Komplexität der mittels endlichen Automaten erkennbaren Sprachen ist **ziemlich beschränkt**.



# DEAs versus Computer

Im Prinzip sind **Rechner** ebenfalls **endliche Automaten**:  
endlich viel Speicherplatz und daher nur endliche Menge an Zuständen

**Aber:** Automaten sind keine geeignete Abstraktion für Rechner:

1. So ein Automat hätte extrem viele Zustände  
(16 GB RAM  $\leadsto$  Anzahl Zustände:  $2^{128}$  Milliarden  $\approx 10^{39}$  Milliarden )
2. Wenn man programmiert, dann verlässt man sich auch nicht darauf,  
dass der Speicher z. B. exakt 16 GB groß ist.

**Richtige Modellierung von Rechnern:**

- abstrahiert von konkreter Speicherbeschränkung,  
nimmt unendlichen Speicher an
- verwendet **Turing-Maschinen** (VL Berechenbarkeit)



## Teil I: Endliche Automaten und reguläre Sprachen

- 
- 0. Grundbegriffe
  - 1. Endliche Automaten
  - 2. Nachweis der Nichterkennbarkeit
  - 3. Abschlusseigenschaften
  - 4. Entscheidungsprobleme
  - 5. Reguläre Ausdrücke und Sprachen
  - 6. Minimale DEAs und die Nerode-Rechtskongruenz

## Teil II: Grammatiken, kontextfreie Sprachen und Kellerautomaten

- 7. Die Chomsky-Hierarchie
- 8. Rechtslineare Grammatiken und reguläre Sprachen
- 9. Normalformen und Entscheidungsprobleme
- 10. Abschlusseigenschaften und Pumping-Lemma
- 11. Kellerautomaten
- 12. Die Struktur kontextfreier Sprachen





Foto: pixabay.com (Public domain)

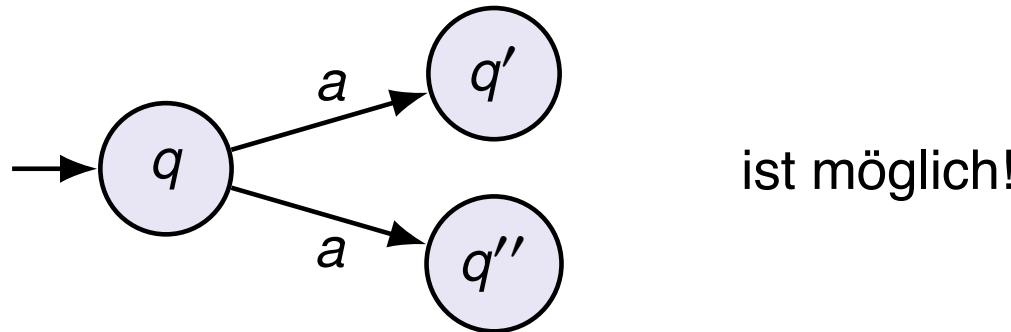
## § 1.2 Nichtdeterministische endliche Automaten

---



# Nichtdeterminismus

Wir generalisieren nun unser Automatenmodell, indem wir auch **Nichtdeterminismus** erlauben:



Ein Automat hat nun **mehrere Möglichkeiten**, ein Wort zu verarbeiten.

Er akzeptiert eine Eingabe, wenn

**eine Möglichkeit existiert**, einen Endzustand zu erreichen.

Nichtdeterminismus ist ein **fundamentales Konzept der Informatik**; wir werden es noch häufig verwenden!

Natürlich gibt es **in der Realität** keine nichtdeterministischen Maschinen.

**Dennoch ist Nichtdeterminismus von großer Bedeutung:**

- als **Abstraktionsmittel bei unvollständiger Information**:  
z. B. Benutzereingaben oder Ereignisse, deren Inhalte  
nicht im Detail modelliert werden
- ermöglicht **kleinere** und **elegantere** Automaten
- spielt sehr wichtige Rolle in der **Komplexitätstheorie**,  
z. B. „**P versus NP**“ (wichtigstes ungelöstes Problem der Informatik!)

➡ VL Berechenbarkeit



Natürlich

Dennoch

- als A

z. B.

nicht

- ermö

- spielen  
z. B.



UNIVERSITÄT  
LEIPZIG

# (Turing award 1976)

M. O. Rabin\*

D. Scott†

## Finite Automata and Their Decision Problems‡

**Abstract:** Finite automata are considered in this paper as instruments for classifying finite tapes. Each one-tape automaton defines a set of tapes, a two-tape automaton defines a set of pairs of tapes, et cetera. The structure of the defined sets is studied. Various generalizations of the notion of an automaton are introduced and their relation to the classical automata is determined. Some decision problems concerning automata are shown to be solvable by effective algorithms; others turn out to be unsolvable by algorithms.

### Introduction

Turing machines are widely considered to be the abstract prototype of digital computers; workers in the field, however, have felt more and more that the notion of a Turing machine is too general to serve as an accurate model of actual computers. It is well known that even for simple calculations it is impossible to give an *a priori* upper bound on the amount of tape a Turing machine will need for any given computation. It is precisely this feature that renders Turing's concept unrealistic.

In the last few years the idea of a *finite automaton* has appeared in the literature. These are machines having only a finite number of internal states that can be used for memory and computation. The restriction of finiteness appears to give a better approximation to the idea of a physical machine. Of course, such machines cannot do as much as Turing machines, but the advantage of being able to compute an arbitrary general recursive function is questionable, since very few of these functions come up in practical applications.

Many equivalent forms of the idea of finite automata have been published. One of the first of these was the definition of "nerve-nets" given by McCulloch and Pitts.<sup>3</sup> The theory of nerve-nets has been developed by authors too numerous to mention. We have been particularly influenced, however, by the work of S. C. Kleene<sup>2</sup> who proved an important theorem characterizing the possible action of such devices (this is the notion of "regular event" in Kleene's terminology). J. R. Myhill, in some unpublished work, has given a new treatment of Kleene's results and this has been the actual point of departure for the investigations presented in this report. We have not, however, adopted Myhill's use of directed graphs as

a method of viewing automata but have retained throughout a machine-like formalism that permits direct comparison with Turing machines. A neat form of the definition of automata has been used by Burks and Wang<sup>1</sup> and by E. F. Moore,<sup>4</sup> and our point of view is closer to theirs than it is to the formalism of nerve-nets. However, we have adopted an even simpler form of the definition by doing away with a complicated output function and having our machines simply give "yes" or "no" answers. This was also used by Myhill, but our generalizations to the "nondeterministic," "two-way," and "many-tape" machines seem to be new.

In Sections 1-6 the definition of the one-tape, one-way automaton is given and its theory fully developed. These machines are considered as "black boxes" having only a finite number of internal states and reacting to their environment in a deterministic fashion.

We center our discussions around the application of automata as devices for defining sets of tapes by giving "yes" or "no" answers to individual tapes fed into them. To each automaton there corresponds the set of those tapes "accepted" by the automaton; such sets will be referred to as *definable sets*. The structure of these sets of tapes, the various operations which we can perform on these sets, and the relationships between automata and definable sets are the broad topics of this paper.

After defining and explaining the basic notions we give, continuing work by Nerode,<sup>5</sup> Myhill, and Shepherdson,<sup>7</sup> an intrinsic mathematical characterization of definable sets. This characterization turns out to be a useful tool for both proving that certain sets are definable by an automaton and for proving that certain other sets are not.

In Section 4 we discuss decision problems concerning automata. We consider the three problems of deciding whether an automaton accepts any tapes, whether it ac-

Maschinen.

: :

er Informatik!)

rechenbarkeit

\*Now at the Department of Mathematics, Hebrew University in Jerusalem.

†Now at the Department of Mathematics, University of Chicago.

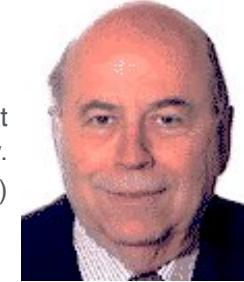
‡The bulk of this work was done while the authors were associated with the IBM Research Center during the summer of 1957.

# Nichtdeterministische endliche Automaten

Michael O. Rabin, Dana Scott

© University of Pittsburgh bzw.

CC BY-SA 2.5 SI (Wikipedia, Andrej Bauer)



## Definition 1.8 (NEA)

Ein **nichtdeterministischer endlicher Automat (NEA)** ist von der Form  
 $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ , wobei

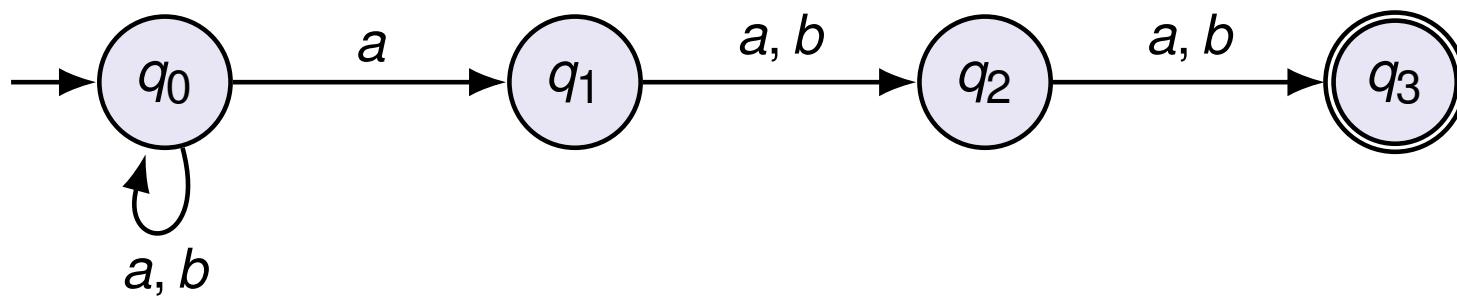
- $Q$  eine endliche Menge von Zuständen ist,
- $\Sigma$  das Eingabealphabet ist,
- $q_0 \in Q$  der Anfangszustand ist,
- $\Delta \subseteq Q \times \Sigma \times Q$  die **Übergangsrelation** ist,
- $F \subseteq Q$  eine Menge von akzeptierenden Zuständen ist.

Zur Erinnerung:  $Q \times \Sigma \times Q = \{(q, a, q') \mid q, q' \in Q, a \in \Sigma\}$



# Ein Beispiel

## Beispiel 1.9



Dieser NEA  $\mathcal{A}$  ist **kein DEA**, denn in  $q_0$  gibt es **zwei Übergänge** für  $a$  und in  $q_3$  **fehlen** Übergänge für  $a$  und  $b$ .

# Akzeptanz

## Definition 1.10 (Pfad in NEA)

Ein **Pfad** im NEA  $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$  von  $p_0 \in Q$  nach  $p_n \in Q$  ist eine Folge

$$p_0 \xrightarrow{\mathcal{A}}^{a_1} p_1 \xrightarrow{\mathcal{A}}^{a_2} \cdots \xrightarrow{\mathcal{A}}^{a_n} p_n$$

mit  $(p_i, a_{i+1}, p_{i+1}) \in \Delta$  für  $i = 0, \dots, n - 1$ .

Dieser Pfad hat die **Beschriftung**  $w = a_1 \cdots a_n$ .

## Definition 1.11 (NEA akzeptiert Wort)

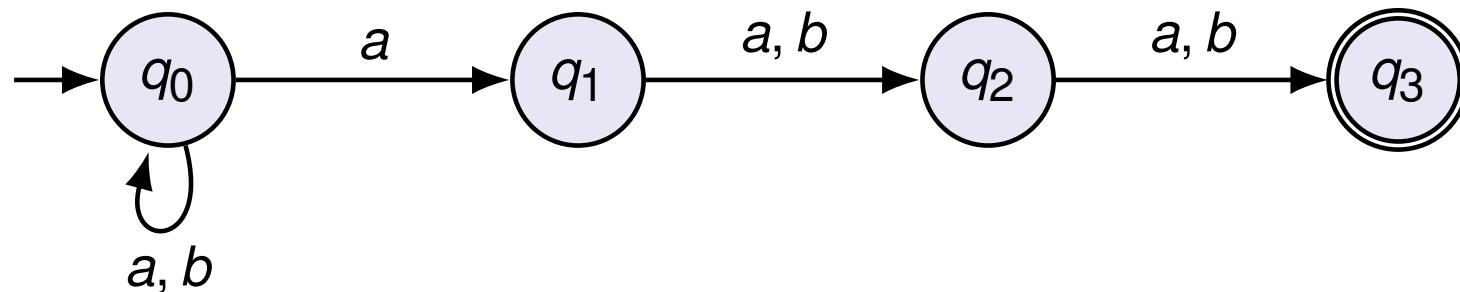
Der NEA  $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$  **akzeptiert** das Wort  $w = a_1 \cdots a_n \in \Sigma^*$ , wenn es einen Pfad  $p_0 \xrightarrow{\mathcal{A}}^{a_1} \cdots \xrightarrow{\mathcal{A}}^{a_n} p_n$  gibt mit  $p_0 = q_0$  und  $p_n \in F$ .

Die von  $\mathcal{A}$  **erkannte Sprache** ist wieder  $L(\mathcal{A}) := \{w \in \Sigma^* \mid \mathcal{A} \text{ akzeptiert } w\}$ .



[Zurück zum Beispiel](#)

## **Beispiel 1.9,** fortgesetzt



## Pfade sind z. B.:

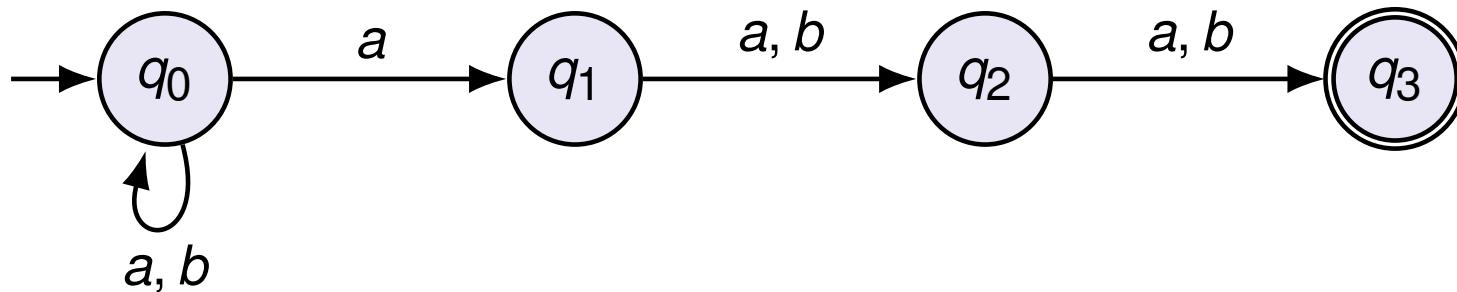
$q_0 \xrightarrow{\text{a}}_{\mathcal{A}} q_1 \xrightarrow{\text{b}}_{\mathcal{A}} q_2 \xrightarrow{\text{a}}_{\mathcal{A}} q_3$	mit Beschriftung $\text{aba}$	}	<b><math>\text{aba}</math> wird akzeptiert</b>
$q_0 \xrightarrow{\text{a}}_{\mathcal{A}} q_0 \xrightarrow{\text{b}}_{\mathcal{A}} q_0 \xrightarrow{\text{a}}_{\mathcal{A}} q_1$	mit Beschriftung $\text{aba}$		
$q_0$	mit Beschriftung $\varepsilon$	}	<b><math>\varepsilon</math> wird nicht akzeptiert</b>

Wenn Pfad von  $p$  nach  $q$  mit Beschriftung  $w$  existiert, so schreiben wir:

$$p \xrightarrow{w} \mathcal{A} q$$

# Zurück zum Beispiel

## Beispiel 1.9, fortgesetzt



**Beobachtung:** Dieser NEA erkennt die Sprache

$$L = \{w \in \{a, b\}^* \mid \text{das drittletzte Symbol in } w \text{ ist } a\}.$$

## Intuitionen:

- Wenn  $\mathcal{A}$  in  $q_0$  ein  $a$  liest, „räät“ er, ob er an der **drittletzten** Stelle ist.
- Hat er „ja“ geraten, so **verifiziert** er mit der  **$q_0$ - $q_3$ -Kette** diese Wahl.

## Wichtig:

- Für Wörter  $w \in L(\mathcal{A})$  gibt es die Möglichkeit, **richtig** zu raten.
- Für Wörter  $w \notin L(\mathcal{A})$  führt **falsches** Raten **niemals** zur Akzeptanz.

Es ist gar nicht so einfach, einen **DEA** für diese Sprache zu finden!

# DEAs versus NEAs



„David vs. Goliath“, CC BY-NC-ND 2.0 (Greg Foster, flickr)

Offensichtlich ist **jeder DEA auch ein NEA:**

DEA-Übergang  $\delta(q, a) = q'$  entspricht NEA-Übergang  $(q, a, q')$   
(jede Funktion ist auch eine Relation)

Damit ist (trivialerweise) jede durch einen **DEA** erkennbare Sprache  
**auch durch einen NEA** erkennbar.

**Gilt die Umkehrung auch?**

Interessanterweise **ja**  
d. h. DEAs und NEAs **erkennen dieselben Sprachen!**

Man beweist dies mittels der **Potenzmengenkonstruktion**

Zwei NEAs heißen **äquivalent**, wenn sie dieselbe Sprache erkennen.

# Der Satz von Rabin und Scott

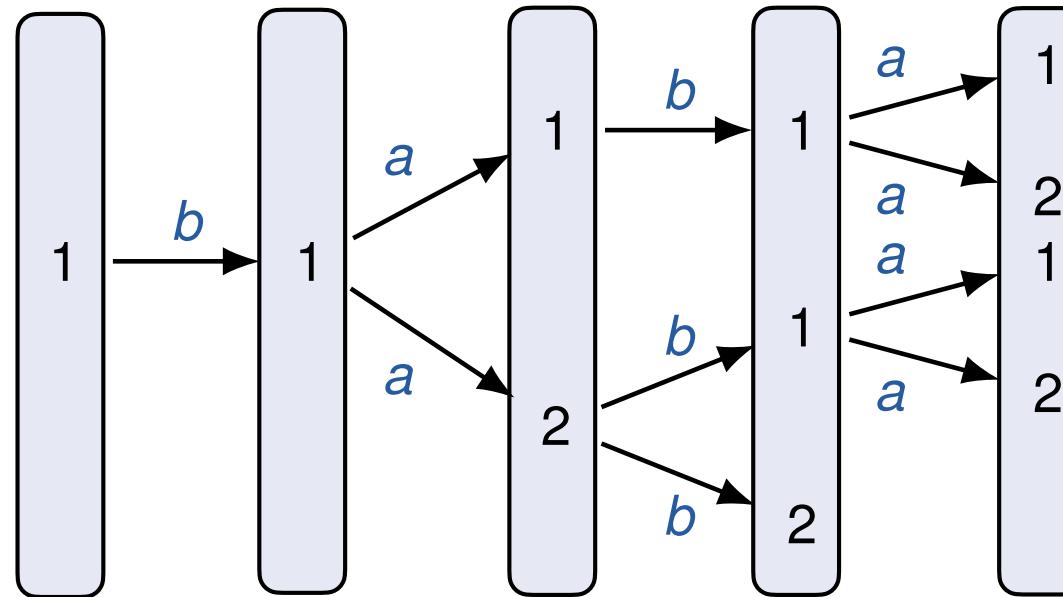
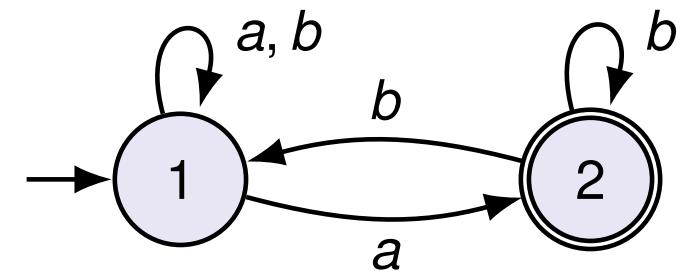
## Satz 1.12 (Rabin/Scott)

Zu jedem NEA kann man einen äquivalenten DEA konstruieren.

## Intuition für den Beweis:

## Finde äquivalente DEA zu

Eingabewort *baba*, Pfade ab  $q_0$ :



DEA-Zustände  
sind **Mengen** von  
NEA-Zuständen

Akzeptanz: mind. ein Zustand aus **Menge ganz rechts** ist akzeptierend

# Die Potenzmengenkonstruktion

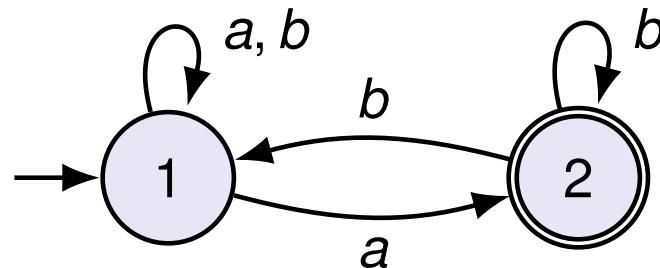
## Beweisidee für Satz 1.12

Sei  $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$  ein NEA.

Wir definieren einen DEA mit Zustandsmenge

$$2^Q := \{P \mid P \subseteq Q\} \quad (\text{Potenzmenge von } Q)$$

Um z. B. DEA zu konstruieren, der äquivalent ist zum NEA



verwenden wir also die Zustandsmenge

$$2^Q = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}$$

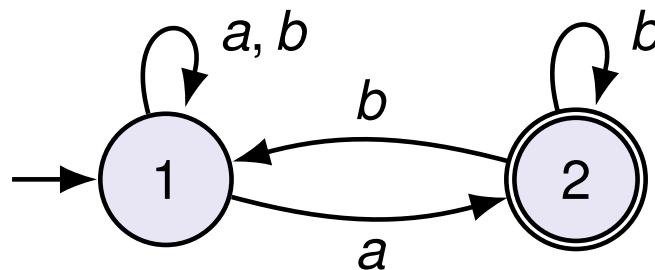
Der DEA-Startzustand ist dann  $\{q_0\}$ . (im Beispiel:  $\{1\}$ )

# Die Potenzmengenkonstruktion

## Beweisidee für Satz 1.12

Sei  $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$  ein NEA.

Die **Übergangsfunktion** definieren wir so, dass der DEA **alle möglichen Pfade des NEA** (für die jeweilige Eingabe) **gleichzeitig** verfolgt.



Für Zustand  $\{1, 2\}$  und Eingabesymbol  $b$  gilt also z.B.:

$$\delta(\{1, 2\}, b) = \{1, 2\}$$

denn  $(1, b, 1) \in \Delta$   
(und auch  $(2, b, 1)$ )

denn  $(2, b, 2) \in \Delta$



# Beweis Satz von Rabin/Scott

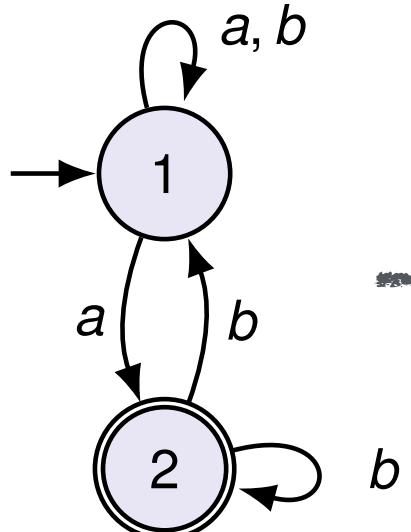
## Beweis des Satzes 1.12

Sei  $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$  ein NEA.

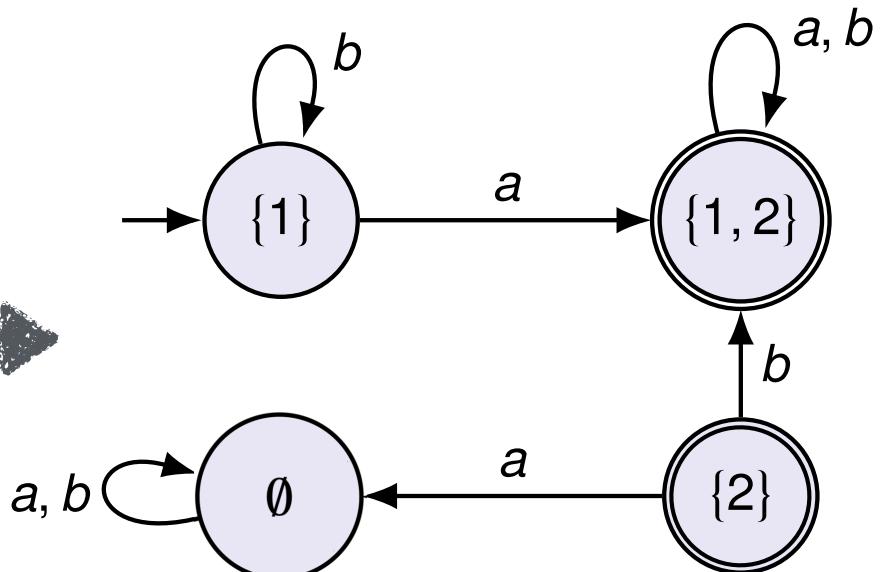
Der DEA  $\mathcal{A}' = (2^Q, \Sigma, \{q_0\}, \delta, F')$  ist definiert durch:

- $\delta(P, a) = \bigcup_{p \in P} \{p' \mid (p, a, p') \in \Delta\}$  für alle  $P \in 2^Q$  und  $a \in \Sigma$
- $F' = \{P \in 2^Q \mid P \cap F \neq \emptyset\}$

## Beispiel 1.13



Potenzmengen-  
konstruktion



# Beweis Satz von Rabin/Scott

## Beweis des Satzes 1.12

Sei  $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$  ein NEA.

Der DEA  $\mathcal{A}' = (2^Q, \Sigma, \{q_0\}, \delta, F')$  ist definiert durch:

- $\delta(P, a) = \bigcup_{p \in P} \{p' \mid (p, a, p') \in \Delta\}$  für alle  $P \in 2^Q$  und  $a \in \Sigma$
- $F' = \{P \in 2^Q \mid P \cap F \neq \emptyset\}$

**Hilfsaussage:** für alle  $w \in \Sigma^*$ :  $\hat{\delta}(\{q_0\}, w) = \{q \mid q_0 \xrightarrow[w]{\mathcal{A}} q\}$

Daraus folgt  $L(\mathcal{A}) = L(\mathcal{A}')$ , da:

$$w \in L(\mathcal{A}) \quad \text{gdw. } \exists q \in F : q_0 \xrightarrow[w]{\mathcal{A}} q \quad (\text{Def. } L(\mathcal{A}))$$

$$\quad \text{gdw. } \exists q \in F : q \in \hat{\delta}(\{q_0\}, w) \quad (\text{Hilfsaussage})$$

$$\quad \text{gdw. } \hat{\delta}(\{q_0\}, w) \cap F \neq \emptyset$$

$$\quad \text{gdw. } \hat{\delta}(\{q_0\}, w) \in F' \quad (\text{Def. } F')$$

$$\quad \text{gdw. } w \in L(\mathcal{A}') \quad (\text{Def. } L(\mathcal{A}'))$$



# Nachteil der Potenzmengenkonstruktion

## Zustandszahl wird **exponentiell vergrößert!**

- Im Allgemeinen ist das nicht vermeidbar (Beweis später).
- In vielen Fällen kommt man aber mit weniger Zuständen aus.

Später: Konstruktion von äquivalenten DEAs mit **minimaler Zustandszahl**

In der Praxis ist Konstruktion bequem mittels **Tabelle** anwendbar:

- eine Spalte für jedes Symbol
- eine Zeile für jeden Zustand

Dabei werden die vom Startzustand aus **nicht erreichbaren** Zustände (die offensichtlich unnütz sind) automatisch weggelassen.

## Beispiel: Tafel





Bild: pixabay.com (Public domain)

## § 1.3 Endliche Automaten mit Wortübergängen

---



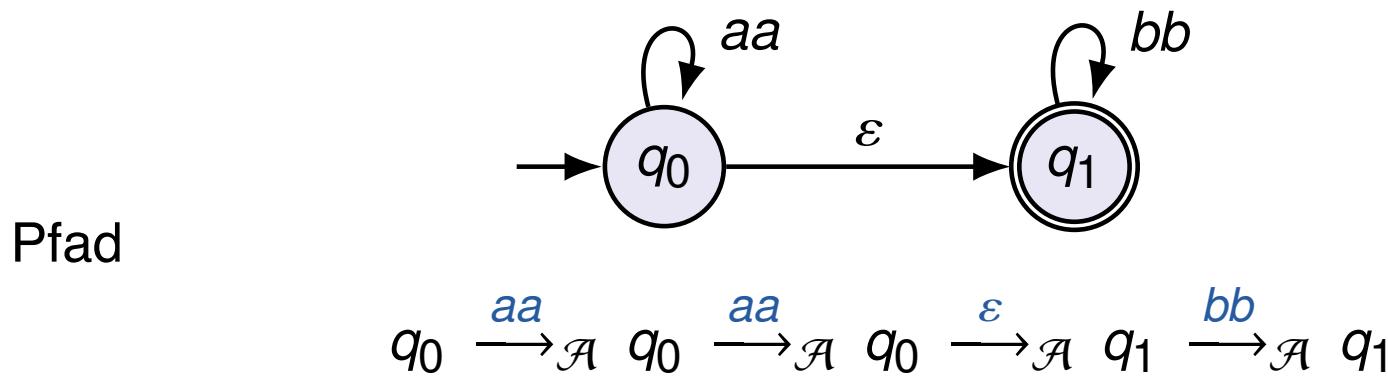
UNIVERSITÄT  
LEIPZIG

# NEAs mit Wortübergängen

Definition 1.14 (NEA mit Wort- und  $\varepsilon$ -Übergängen)

1. Ein NEA mit Wortübergängen hat die Form  $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ , wobei  $Q, \Sigma, q_0, F$  wie beim NEA definiert sind und  $\Delta \subseteq Q \times \Sigma^* \times Q$  eine endliche Menge von Wortübergängen ist.
2. Ein  $\varepsilon$ -NEA ist ein NEA mit Wortübergängen der Form  $(p, \varepsilon, q)$  und  $(p, a, q)$  mit  $a \in \Sigma$ .

Beispiel für NEA mit Wortübergängen:



hat Beschriftung  $aa \cdot aa \cdot \varepsilon \cdot bb = aaaabb$ .



# ... sind genauso mächtig wie NEAs

## Satz 1.15

Zu jedem NEA mit Wortübergängen kann man einen äquivalenten NEA (ohne Wortübergänge) konstruieren.

### Beweis:

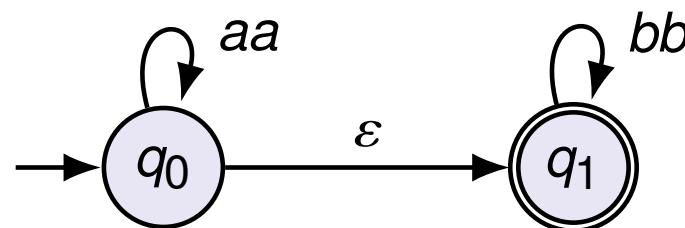


# Von Wort- zu epsilon-Übergängen

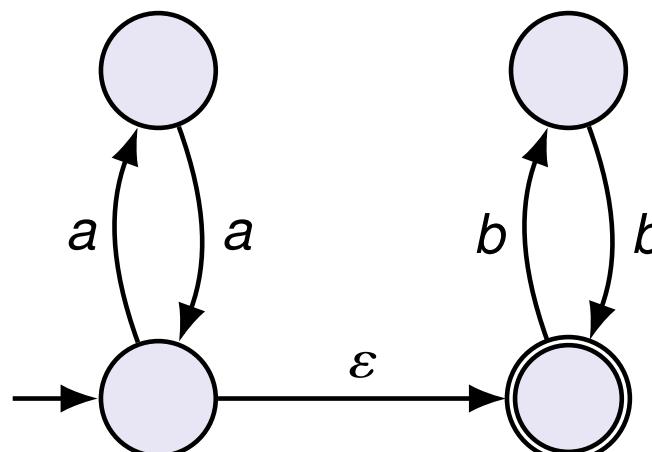
Lemma 1.16

Zu jedem NEA mit Wortübergängen kann man einen äquivalenten  $\varepsilon$ -NEA konstruieren.

Beispiel 1.17



wird zu

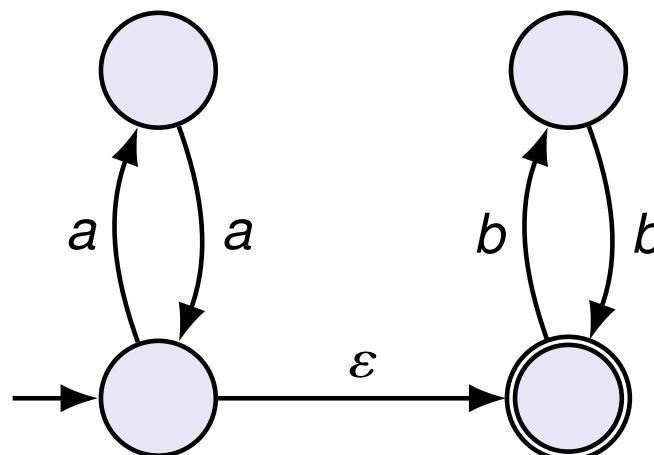


# Elimination von epsilon-Übergängen

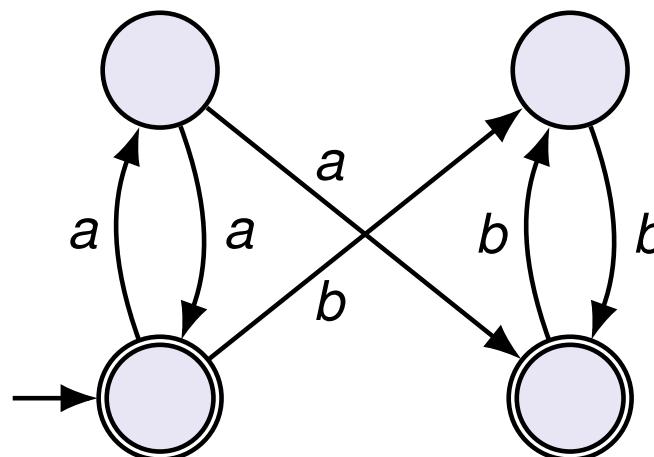
Lemma 1.18

Zu jedem  $\varepsilon$ -NEA kann man einen äquivalenten NEA konstruieren.

Beispiel 1.19



wird zu



# Elimination von epsilon-Übergängen

Lemma 1.18

Zu jedem  $\varepsilon$ -NEA kann man einen äquivalenten NEA konstruieren.

**Beweis:** Sei  $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$  ein  $\varepsilon$ -NEA.

Wir konstruieren daraus einen NEA ohne  $\varepsilon$ -Übergänge wie folgt:

$$\mathcal{A}' = (Q, \Sigma, q_0, \Delta', F')$$

wobei

$$\Delta' := \{(p, a, q) \in Q \times \Sigma \times Q \mid p \xrightarrow[\mathcal{A}]{}^a q\}$$

beliebig viele  $\varepsilon$ -Übergänge,  
dann ein  $a$ -Übergang,  
dann beliebig viele  $\varepsilon$ -Überg.

$$F' := \begin{cases} F \cup \{q_0\} & \text{falls } q_0 \xrightarrow[\mathcal{A}]{}^\varepsilon q_f \text{ für ein } q_f \in F \\ F & \text{sonst} \end{cases}$$

Dann gilt:  $L(\mathcal{A}) = L(\mathcal{A}')$

□



## Teil I: Endliche Automaten und reguläre Sprachen

- 0. Grundbegriffe
- 1. Endliche Automaten
- 2. Nachweis der Nichterkennbarkeit
- 3. Abschlusseigenschaften
- 4. Entscheidungsprobleme
- 5. Reguläre Ausdrücke und Sprachen
- 6. Minimale DEAs und die Nerode-Rechtskongruenz



## Teil II: Grammatiken, kontextfreie Sprachen und Kellerautomaten

- 7. Die Chomsky-Hierarchie
- 8. Rechtslineare Grammatiken und reguläre Sprachen
- 9. Normalformen und Entscheidungsprobleme
- 10. Abschlusseigenschaften und Pumping-Lemma
- 11. Kellerautomaten
- 12. Die Struktur kontextfreier Sprachen