Theoretische Informatik 1

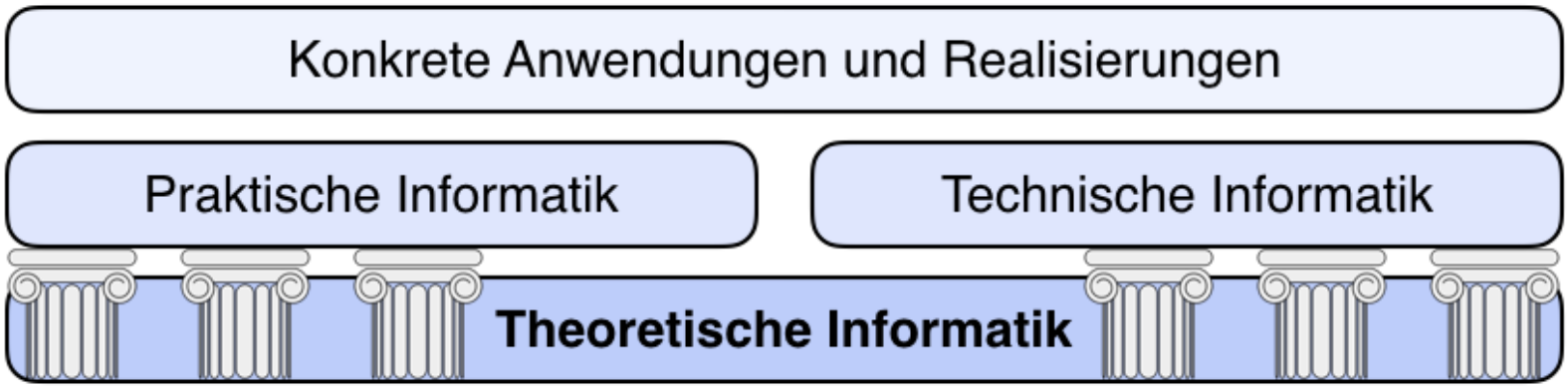
WiSe 23/24

Prof. Dr. Sebastian Siebertz
AG Theoretische Informatik
MZH, Raum 3160
siebertz@uni-bremen.de

Universität
Bremen

Theoretische Informatik: eine kurze Einführung

- Theoretische Informatik schafft formale und kulturelle Grundlage für die Informatik.
- 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: eine kurze Einführung

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

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

- **Schaffen von mathematischen Modellen und Abstraktionen**

  Unwichtiges ausblenden, Wesentliches klar herausstellen. Grundlage für exakte mathematische Behandlung informatischer Fragestellungen.

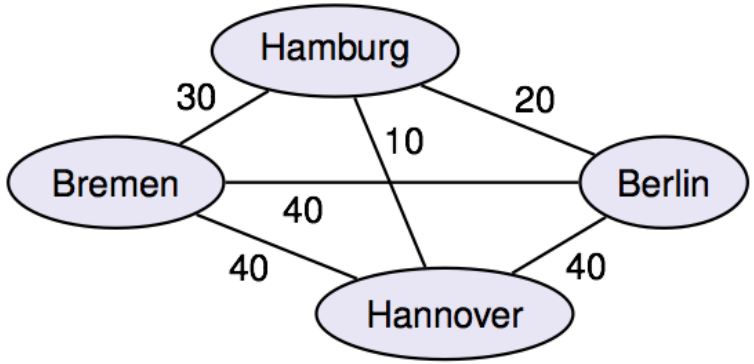
- **Bereitstellen von Berechnungsmodellen und algorithmischen Techniken**

  Abstrakte Modelle von Computern, Programmiersprachen etc. Wie unterscheiden sich PC, Tablet, das Web als „großer Computer", DNA-Computer und Quantencomputer?

- **Verständnis der Grenzen der (effizienten) Berechenbarkeit**

Theoretische Informatik: eine kurze Einführung

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

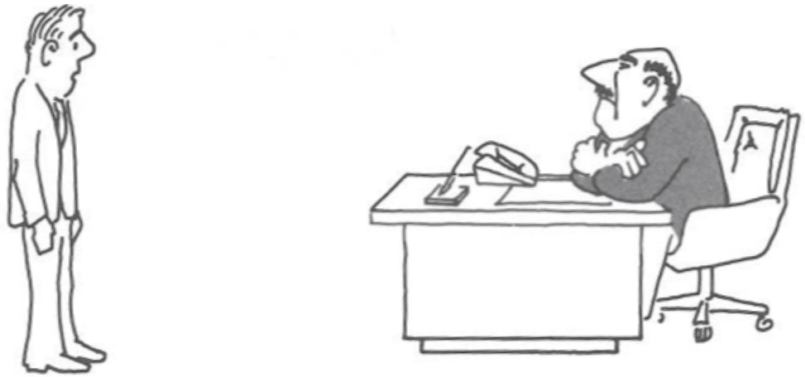


- Das Programm ist nicht sehr effizient?

  → Das liegt nicht an der Programmierer*in!

- Denn:
  - Es ist unbekannt, ob dieses Problem (Traveling Salesperson) effizient lösbar ist.
  - Frage ist äquivalent zu einem der wichtigsten offenen Probleme in der Informatik/Mathematik: Gilt P = NP?

Theoretische Informatik: eine kurze Einführung



"Ich kann keinen effizienten Algorithmus finden, ich bin wohl nicht gut genug"

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

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

Theoretische Informatik: eine kurze Einführung

- **Aufgabe:** Entwirf eine Raketensteuerung für die Ariane 5

© ESA (European Space Agency)

Theoretische Informatik: eine kurze Einführung

- **Aufgabe:** Entwirf eine Raketensteuerung für die Ariane 5

- Problem: Softwarefehler

Theoretische Informatik: eine kurze Einführung

- Schätzung: 10–20 Bugs pro 1000 Zeilen Code.
- 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: eine kurze Einführung

- Theoretische Informatik besteht aus vielen Teilgebieten:
  - Automaten und formale Sprachen
  - Berechenbarkeits- und Komplexitätstheorie
  - Verifikation und mathematische Logik
  - Kryptographie
  - Algorithmentheorie
  - Datenbanktheorie
  - etc.

Zur Rolle der Mathematik

- Exakte und formale Beschreibungen sind essentiell in der theoretischen Informatik.
- Lasst Euch davon nicht abschrecken:
  - Wir werden hier oft andere mathematische Methoden benötigen, als ihr aus der Schule kennt.
  - Wir wollen nicht rechnen,
    sondern Dinge beweisen (= verstehen).
  - Alles was ihr braucht lernt ihr hier!

Formale Sprachen – Grundlegende Definitionen

- Alle (sinnvollen) Ein- und Ausgaben können als Wörter über Alphabeten kodiert werden.
  - Z.B. kann jede Zahl $n \in \mathbb{N}$ in Binärdarstellung als Wort kodiert werden.
  - Dagegen kann nicht jede Zahl $x \in \mathbb{R}$ kodiert werden, sondern muss möglicherweise angenähert werden.

Alphabete

- Ein Alphabet ist eine endliche, nichtleere Menge von Symbolen.
- Wir benutzen griechische Großbuchstaben für Alphabete: $\Sigma, \Gamma, \ldots$
- und meistens römische Kleinbuchstaben für Symbole: $a, b, c, \ldots$
- Beispiele:
  - $\Sigma = \{a, b, c, \ldots, z\}$
  - $\Sigma_2 = \{0, 1\}$
  - $\Sigma_3 = \{\text{program, const, var, label, procedure, function, type, begin, end, if, then, else, case, of, repeat, until, while, do, for,to}\}$ $\cup \{VAR, VALUE, FUNCTION\}$.

Wörter

$\mathcal{A}\,\mathcal{B}\,\mathcal{C}\,\mathcal{D}\,\mathcal{E}\,\mathcal{F}$

- $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$ und $\mathbb{N} = \mathbb{N}_0 \setminus \{0\}$.
- $[n] = \{1, \ldots, n\}$ für $n \in \mathbb{N}$.
- $[0] = \varnothing$.
- Ein Wort über einem Alphabet Σ ist eine endliche Folge $w = w_1 \cdots w_n$ von Symbolen aus Σ.
  - Formal: eine endliche Folge ist eine Abbildung $w : [n] \to \Sigma : i \mapsto w_i$ für ein $n \in \mathbb{N}_0$. Die Zahl n bezeichnet die Länge der Folge.
  - Die Folge mit Definitionsbereich $[0] = \varnothing$ heißt leere Folge oder leeres Wort. Es wird mit ε bezeichnet.

Wörter

- Beispiele:
  - $w = abc$ ist ein Wort der Länge 3 über $\Sigma = \{a, b, \ldots, z\}$.
  - $w = ...140256.42.$ ist ein Wort der Länge 13 über dem Alphabet $\{0, \ldots, 9\} \cup \{.\}$.
  - Jedes Pascal-Programm kann als Wort über

    $\Sigma = \{$program, const, var, label, procedure, function, type, begin, end, if, then, else, case, of, repeat, until, while, do, for,to$\}$
    $\cup \{VAR, VALUE, FUNCTION\}$

    betrachtet werden (wenn wir Variablennamen, Werten und Funktionsnamen abstrahieren).
  - Nicht jedes Wort über dem obigen Alphabet ist ein wohlgeformtes Pascal-Programm.

Formale Sprachen

- Σ^* ist die Menge aller Wörter über Σ.
- $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$.
- Eine Sprache über Σ ist eine Menge $L \subseteq \Sigma^*$.
- Beispiele:
  - $L = \{abc\}$ ist eine Sprache über $\{a, \ldots, z\}$, die nur ein Wort enthält.
  - $L = \{w \in \{a, \ldots, z\}^+ : w \text{ ist ein Wort der deutschen Sprache}\}$.
  - Die Menge aller Wörter über

    $\Sigma = \{$program, const, var, label, procedure, function, type, begin, end, if, then, else, case, of, repeat, until, while, do, for,to$\}$
    $\cup \{VAR, VALUE, FUNCTION\}$,

    die wohlgeformte Pascal-Programme beschreiben.

Formale Sprachen

Formale Sprachen sind in der Informatik ein wichtiger Abstraktionsmechanismus:

- Menge aller wohlgeformten Pascal-/C++-/Java-/...-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. grep-Befehl in Linux: finde alle Dokumente, die die Wörter „Universität“ und „Bremen“ enthalten.
- Kommunikationsprotokolle
  z.B. Menge aller wohlgeformten TCP-Pakete.
- „Erlaubtes Verhalten“ von Software- und Hardwaresystemen
  z.B. die erlaubten Verhaltensweisen eines Moduls zur Raketensteuerung.

Konkatenation

- Konkatenation

  ... kann auf Wörter und Sprachen angewendet werden:

  - Auf Wörtern: $u \cdot v := uv$

    $u : [n] \to \Sigma,\ v : [m] \to \Sigma$

    $$\to uv : [n+m] \to \Sigma : i \mapsto \begin{cases} u_i & \text{falls } 1 \leq i \leq n \\ v_j & \text{falls } n+1 \leq i \leq n+m, j = n+i \end{cases}$$

  - Auf Sprachen: $L_1 \cdot L_2 := \{uv : u \in L_1, v \in L_2\}$.
  - Konkatenationspunkt wird häufig weggelassen.

- Beispiele:
  - $ab \cdot ba = abba$
  - $\varepsilon \cdot abba = abba$
  - Wenn $L_1 = \{ab, ac\}$ und $L_2 = \{ba, ca\}$,
    dann $L_1 \cdot L_2 = \{abba, abca, acba, acca\}$.

Wörter induktive Definition

- Alternative induktive Definition von Wörtern über Alphabet Σ:
- ε ist ein Wort der Länge 0.
- Wenn w ein Wort der Länge n ist, so ist für jedes $a \in \Sigma$ auch wa ein Wort der Länge $n+1$.

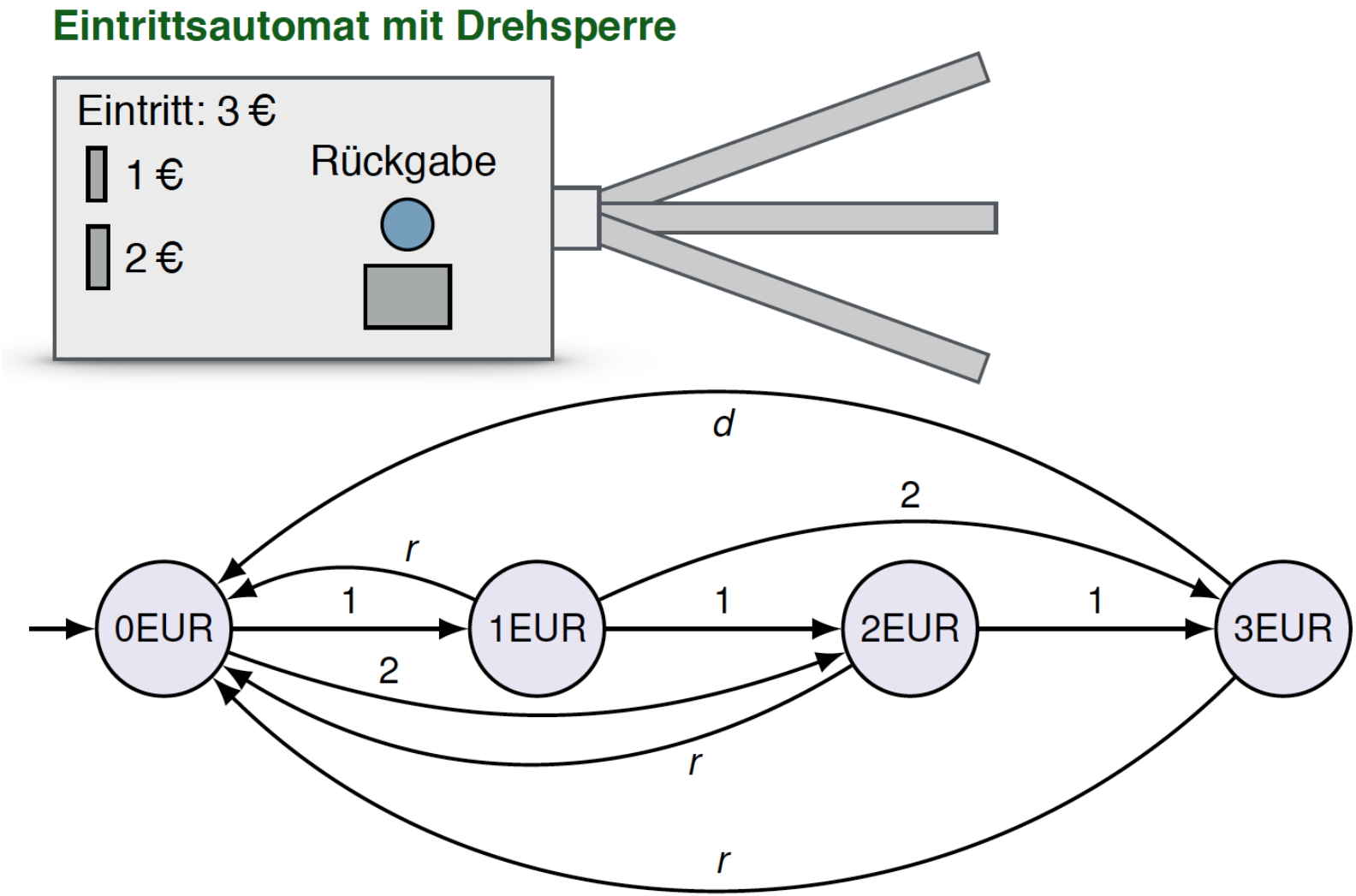
Formale Sprachen – Grundlegende Definitionen

- Zusammenfassung
  - Ein Alphabet ist eine endliche, nichtleere Menge von Symbolen.
  - Ein Wort über einem Alphabet Σ ist eine endliche Folge $w = w_1 \cdots w_n$ von Symbolen aus Σ.
  - Σ^* ist die Menge aller Wörter über Σ.
  - $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$.
  - Eine Sprache über Σ ist eine Menge $L \subseteq \Sigma^*$.

Endliche Automaten

- Endliches Mittel zur Beschreibung von (potentiall unendlichen) Sprachen.
- Abstraktion eines Systems (z.B. Software- oder Hardwaresystem) basierend auf Zuständen und Zustandsübergangen.
- Zustand:
  - Abstrakte Beschreibung des momentanen Systemzustands, Zustand als Name (q_0, q_1, usw.) ohne Beschreibung der Systemdetails.
- Zustandsübergänge:
  - Wechsel zwischen Zuständen. Ein Lauf des Systems ist eine Folge von Zuständen.

Beispiel


Eintrittsautomat mit Drehsperre
Eintritt: 3 €
1 €
2 €
Rückgabe
d
2
r
1
1
1
2
r
r
0EUR
1EUR
2EUR
3EUR

Automaten und formale Sprachen

- Automaten beschreiben formale Sprachen:
  - erhalten Wörter als Eingabe, lesen diese von links nach rechts,
  - befinden sich nach jedem Leseschritt in genau einem von endlich vielen möglichen Zuständen,
  - arbeiten ein „festes Programm“ ab,
  - verwerfen oder akzeptieren Eingabe über den am Ende erreichten Zustand.
- Wir werden mehrere Versionen von endlichen Automaten kennen lernen, die alle äquivalent sind.

Deterministische endliche Automaten

- Ein deterministischer endlicher Automat (DEA) ist ein Tupel

  $\mathcal{A} = (Q, \Sigma, q_s, \delta, F)$, wobei

  - Q eine endliche Menge von Zuständen ist,
  - Σ ein Eingabealphabet ist,
  - $q_s \in Q$ der Anfangszustand ist,
  - $\delta : Q \times \Sigma \to Q$ die Übergangsfunktion ist,
  - $F \subseteq Q$ eine Menge von akzeptierenden Zuständen ist.

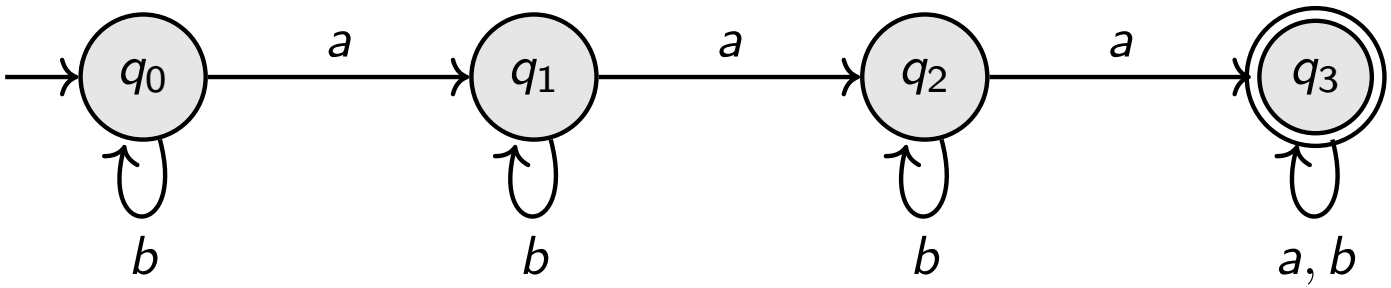
- Erinnerung:
  - $Q \times \Sigma = \{(q, a) : q \in Q, a \in \Sigma\}$.
  - δ bildet also Paare (q, a) auf Zustände q' ab.

Beispiel

Der DEA $\mathcal{A} = (Q, \Sigma, q_s, \delta, F)$ mit den Komponenten

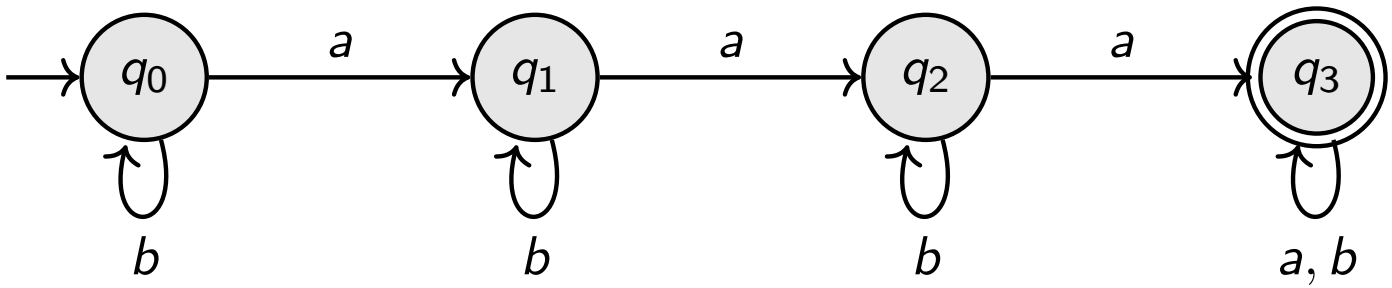
- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{a, b\}$
- $q_s = q_0$
- $\delta(q_0, a) = q_1 \qquad \delta(q_1, a) = q_2 \qquad \delta(q_2, a) = \delta(q_3, a) = q_3$
  $\delta(q_i, b) = q_i \quad$ für $i \in \{0, 1, 2, 3\}$
- $F = \{q_3\}$

wird graphisch dargestellt als:



Übergangsfunktion

- Die kanonische Fortsetzung von $\delta\colon Q \times \Sigma \to Q$ von einzelnen Symbolen auf beliebige Wörter ist die Funktion $\hat{\delta} : Q \times \Sigma^* \to Q$, die induktiv (über die Wortlänge) definiert ist als:
  - $\hat{\delta}(q, \varepsilon) := q$
  - $\hat{\delta}(q, wa) := \delta(\hat{\delta}(q, w), a)$
- Es gilt $\hat{\delta}(q, a) = \delta(q, a)$ für alle Symbole $a \in \Sigma$ und Zustände $q \in Q$.
- Beispiel



$\hat{\delta}(q_0, bbbabbbb) = q_1$ und $\hat{\delta}(q_0, baaab) = q_3$.

Akzeptierte Sprache

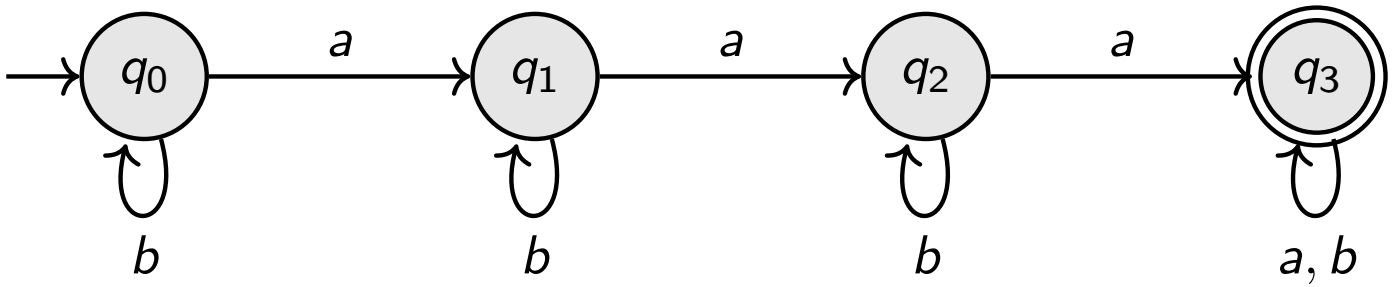
- Ein DEA $\mathcal{A} = (Q, \Sigma, q_s, \delta, F)$ akzeptiert ein Wort $w \in \Sigma^*$, wenn

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

Die von $\mathcal{A}$ akzeptierte Sprache ist

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

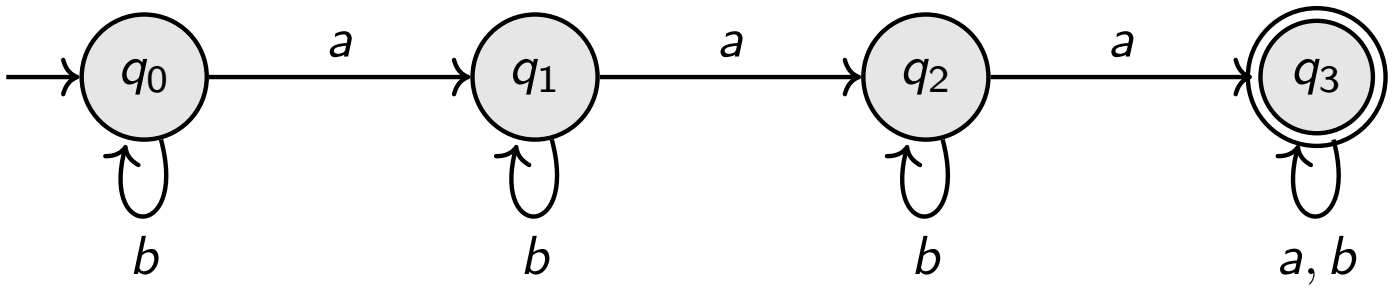
- Beispiel



$L(\mathcal{A}) = \{w \in \{a, b\}^* \mid \ w \text{ enhält mindestens drei } a\}.$

Reguläre Sprachen

- Eine Sprache $L \subseteq \Sigma^*$ heißt regulär, wenn es einen DEA $\mathcal{A}$ gibt mit $L = L(\mathcal{A})$.
- Beispiel:
  - Die Sprache $L = \{w \in \{a, b\}^* \mid \ w \text{ enhält mindestens drei } a\}$ regulär.

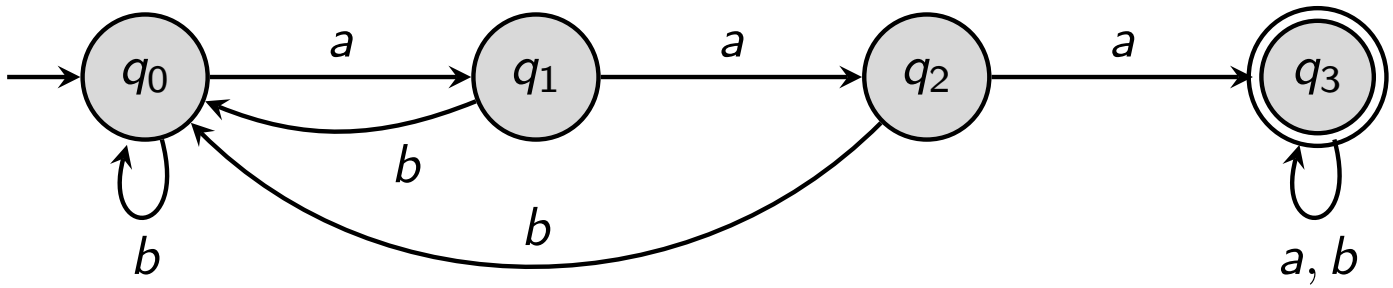


Beispiel

- Der folgende DEA erkennt die Sprache

$$L = \{w = uaaav : u, v \in \Sigma^*\}$$

mit $\Sigma = \{a, b\}$. Auch diese Sprache ist also regulär.



DEA vs Computer

- Im Prinzip sind Computer ebenfalls endliche Automaten: endlich viel Speicherplatz und daher nur endliche Menge an Systemzuständen
- Aber: Automaten sind keine geeignete Abstraktion für Rechner:
  - So ein Automat hätte extrem viele Zustände:

    16 GB Ram $\rightsquigarrow$ Anzahl Zustände: $2^{128 \mathit{Milliarden}}$
  - Bei der Programmierung können wir uns nicht darauf verlassen, dass der Speicher z. B. exakt 16 GB groß ist $\rightarrow$ wir wissen nicht welcher Automat den Rechner darstellt.
- Bessere Modellierung von Rechnern:
  - abstrahiert von Speicherbeschränkung $\rightarrow$ nimmt unendlichen Speicher an
  - verwendet Turing-Maschinen