Endliche Automaten

Endliche Automaten

Endliche Automaten erlauben eine **Beschreibung von Handlungsabläufen**:

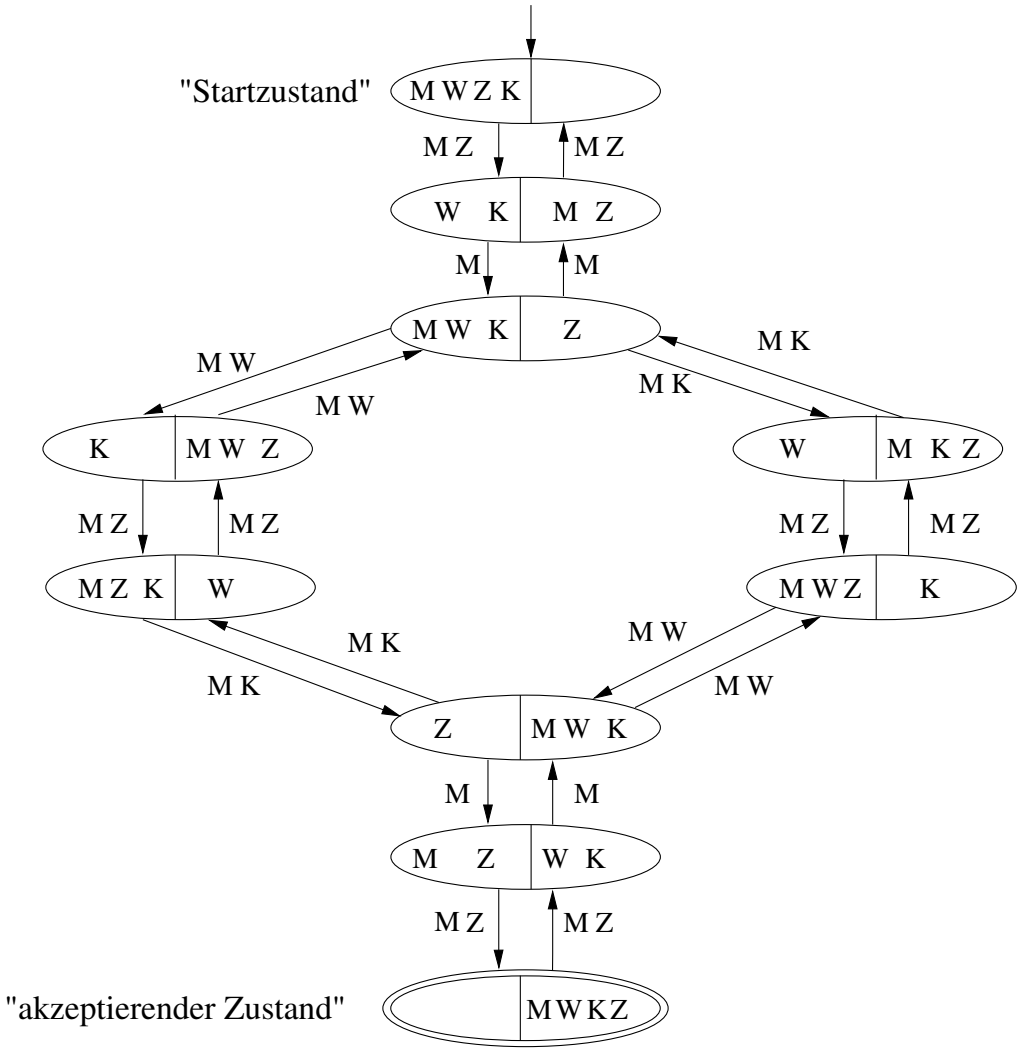
Wie ändert sich ein Systemzustand in Abhängigkeit von veränderten Umgebungsbedingungen?

Vielfältiges Einsatzgebiet, nämlich:

- in der Definition der **regulären Sprachen**

  also der Menge aller Folgen von Ereignissen,
  die von einem Startzustand in einen gewünschten Zustand führen,
- in der **Entwicklung digitaler Schaltungen**
- in der **Softwaretechnik** (z. B. in der Modellierung des Applikationsverhaltens)
- in der **Compilierung**: Lexikalische Analyse
- im **Algorithmenentwurf** für String Probleme
- in der **Abstraktion tatsächlicher Automaten** (wie Bank- und Getränkeautomaten, Fahrstühle etc.)

Das Problem der Flußüberquerung

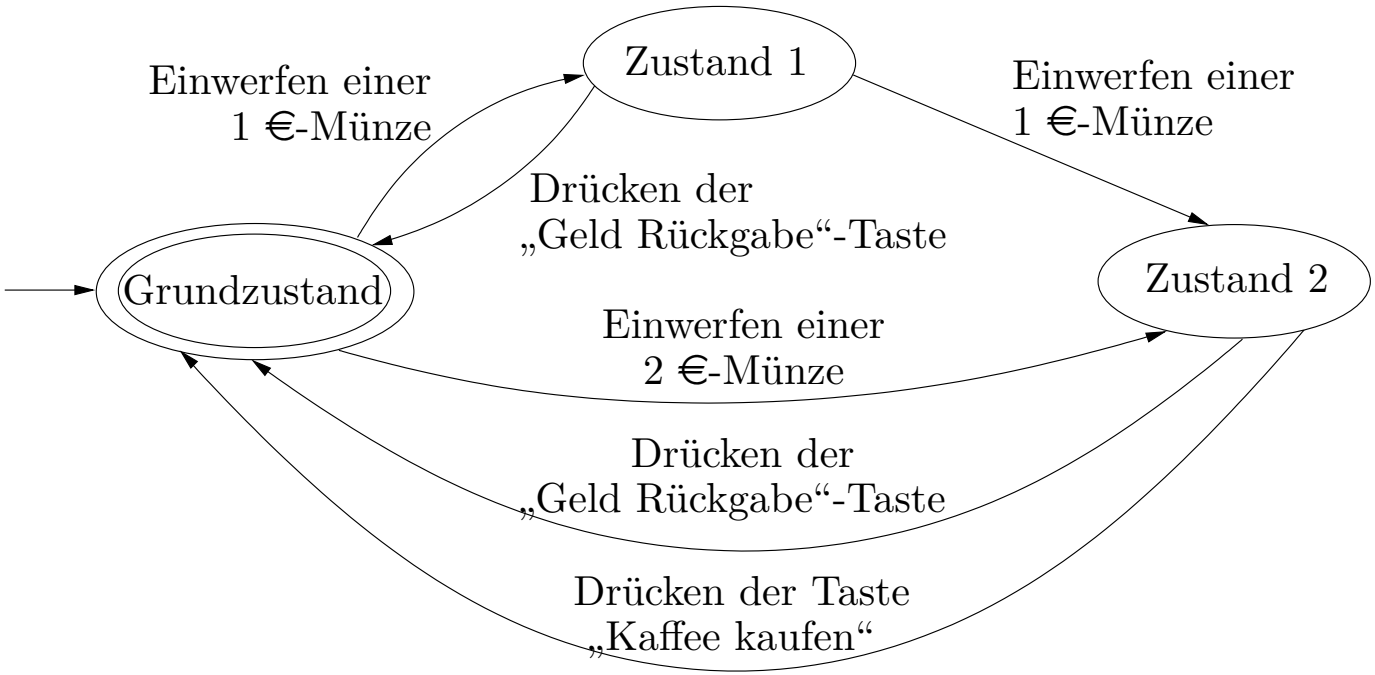


Dieser endliche Automat „akzeptiert“ genau diejenigen Folgen von einzelnen Flussüberquerungen, die vom Startzustand in den akzeptierenden Zustand führen.

Ein Kaffeeautomat

Der Automat erlaubt das Einwerfen von 1 oder 2 €-Münzen, sowie das Drücken der Tasten „Geldrückgabe“ und „Kaffee kaufen“. (Der Kaffee kostet 2 €.)

Und hier ist die Spezifikation des Automaten:



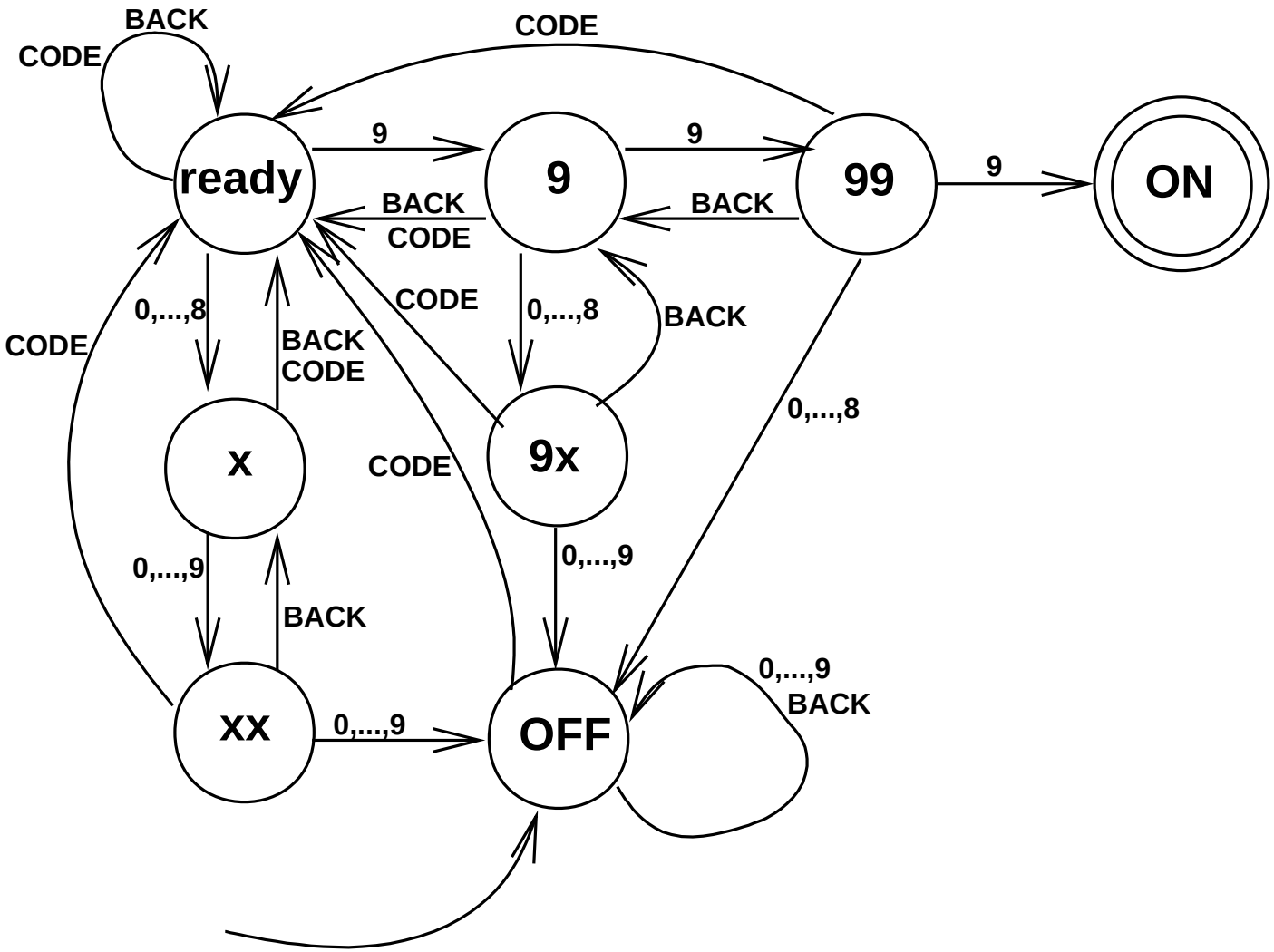
Dieser Automat „akzeptiert“ genau diejenigen Folgen von Bedienoperationen, die vom Grundzustand aus wieder in den Grundzustand führen.

Freischaltung eines Fernsehers (1/2)

- Um die Kindersicherung des Fernsehers über die Fernbedienung freizuschalten, muss ein dreistelliger Code korrekt eingegeben werden. Dabei sind die folgenden Tasten relevant:
  - Die Tasten $0, \ldots, 9$,
  - die Taste `CODE` sowie
  - die Taste `BACK`.
- Die Taste `CODE` muss vor Eingabe des Codes gedrückt werden.
- Wird `CODE` während der Codeeingabe nochmals gedrückt, so wird die Eingabe neu begonnen.
- Wird `BACK` gedrückt, so wird die zuletzt eingegebene Zahl zurückgenommen.

Der Code zum Entsperren ist 999.

Freischaltung eines Fernsehers (2/2)



Der Automat „akzeptiert“ alle Folgen von Bedienoperationen, die vom Zustand „**ready**“ in den Zustand „**ON**“ führen.

Was ist ein endlicher Automat?

Ein **deterministischer endlicher Automat** (engl. deterministic finite automaton, kurz **DFA**)

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

besteht aus:

- einer endlichen Menge Σ, dem **Eingabealphabet**,
- einer endlichen Menge Q, der **Zustandsmenge** (die Elemente aus Q werden Zustände genannt),
- einer **partiellen** Funktion δ von $Q \times \Sigma$ nach Q, der **Übergangsfunktion** (oder Überführungsfunktion),
- einem Zustand $q_0 \in Q$, dem **Startzustand**,
- sowie einer Menge $F \subseteq Q$ von **Endzuständen** bzw. akzeptierenden Zuständen. (Der Buchstabe F steht für „final states“, also „Endzustände“).

Der Kaffeautomat als DFA $A = (\Sigma, Q, \delta, q_0, F)$

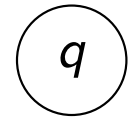
- Σ = {Einwerfen einer 1 €-Münze, Einwerfen einer 2 €-Münze, Drücken der „Geld Rückgabe“-Taste, Drücken der Taste „Kaffee kaufen“}
- Q = {Grundzustand, Zustand 1, Zustand 2}
- q_0 = Grundzustand
- F = {Grundzustand}
- die Funktion $\delta : Q \times \Sigma \to Q$ ist partiell und besitzt die „**Automatentabelle**“

$$
\begin{aligned}
\delta(\text{Grundzustand}, \text{Einwerfen einer 1 €-Münze}) &= \text{Zustand 1},\\
\delta(\text{Grundzustand}, \text{Einwerfen einer 2 €-Münze}) &= \text{Zustand 2},\\
\delta(\text{Zustand 1}, \text{Einwerfen einer 1 €-Münze}) &= \text{Zustand 2},\\
\delta(\text{Zustand 1}, \text{Drücken der „Geld Rückgabe“-Taste}) &= \text{Grundzustand},\\
\delta(\text{Zustand 2}, \text{Drücken der „Geld Rückgabe“-Taste}) &= \text{Grundzustand},\\
\delta(\text{Zustand 2}, \text{Drücken der Taste „Kaffee kaufen“}) &= \text{Grundzustand}.
\end{aligned}
$$

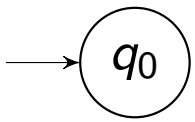
Der Automatengraph, die grafische Darstellung eines DFA

Grafische Darstellung: Der Automatengraph

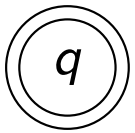
$A = (\Sigma, Q, \delta, q_0, F)$ sei ein DFA.

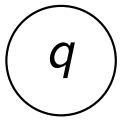
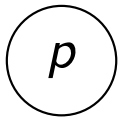
- Für jeden Zustand $q \in Q$ gibt es einen durch  dargestellten Knoten.

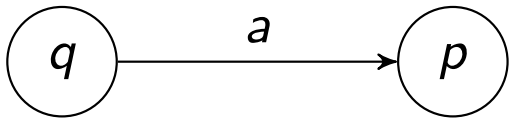
- Der Startzustand q_0 wird durch einen in ihn hinein führenden Pfeil markiert, d.h.:



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



- Seien $p, q \in Q$ Zustände und $a \in \Sigma$ ein Symbol des Alphabets mit $\delta(q, a) = p$. Dann füge einen mit dem Symbol a beschrifteten Pfeil von Knoten  zu Knoten  ein, d.h.:



Wie arbeitet ein DFA?
Die erweiterte Übergangsfunktion

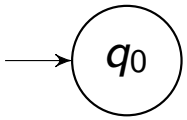
Wie arbeitet ein DFA $A = (\Sigma, Q, \delta, q_0, F)$? (1/2)

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

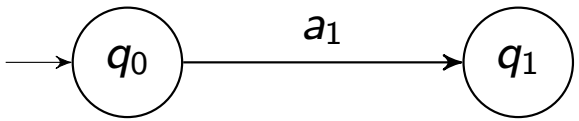
A wird im Startzustand q_0 gestartet. Falls w das leere Wort ist, d.h. $w = \varepsilon$, dann verbleibt A im Zustand q_0.

Also gelte $w = a_1 \cdots a_n$ mit $n \in \mathbb{N}_{>0}$ und $a_1, \ldots, a_n \in \Sigma$.

1. Der Automat liest den ersten Buchstaben a_1 von w.
   - Wenn δ nicht für das Paar (q_0, a_1) definiert ist, dann stürzt A ab ↯
   - Ansonsten wechselt A in den Zustand $q_1 := \delta(q_0, a_1)$.
     In der grafischen Darstellung von A wird der Zustand

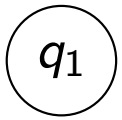
     

     durch die mit a_1 beschriftete Kante verlassen, und q_1 ist der Endknoten dieser Kante, d.h.

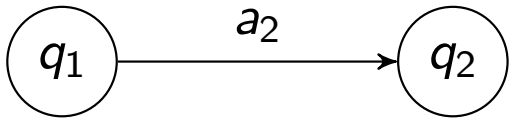
     

Wie arbeitet ein DFA $A = (\Sigma, Q, \delta, q_0, F)$? (2/2)

2. Durch Lesen von a_2, dem zweiten Symbol von w, wechselt A in den Zustand $q_2 := \delta(q_1, a_2)$, bzw. „stürzt ab“, falls δ nicht für (q_1, a_2) definiert ist.
   In der grafischen Darstellung von A wird



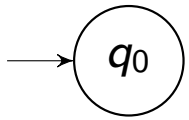
   durch die mit a_2 beschriftete Kante verlassen (falls diese Kante existiert)



   und der Automat landet in Zustand $q_2 = \delta(q_1, a_2)$.

n. Auf diese Weise wird das gesamte Eingabewort $w = a_1 \cdots a_n$ abgearbeitet.

- Ausgehend vom Startzustand q_0 erreicht A nacheinander Zustände $q_1, \ldots, q_n$.
- In der grafischen Darstellung von A entspricht diese Zustandsfolge einem Weg der Länge n, der im Knoten



  startet und dessen Kanten mit den Buchstaben $a_1, \ldots, a_n$ beschriftet sind.
- Schreibe $\widehat{\delta}(q_0, w) := q_n$, bzw. $\widehat{\delta}(q_0, w) := \perp$, wenn A zwischenzeitlich abstürzt.

Was genau ist $\widehat{\delta}(q, w)$? (1/2)

Sei $A := (\Sigma, Q, \delta, q_0, F)$ ein DFA. Sei $\perp$ ein Symbol, das **nicht** in der Zustandsmenge Q liegt, und sei $Q_\perp := Q \cup \{\perp\}$. Die Funktion

$$\widehat{\delta} \ : \ Q_\perp \times \Sigma^* \ \rightarrow \ Q_\perp$$

ist rekursiv wie folgt definiert:

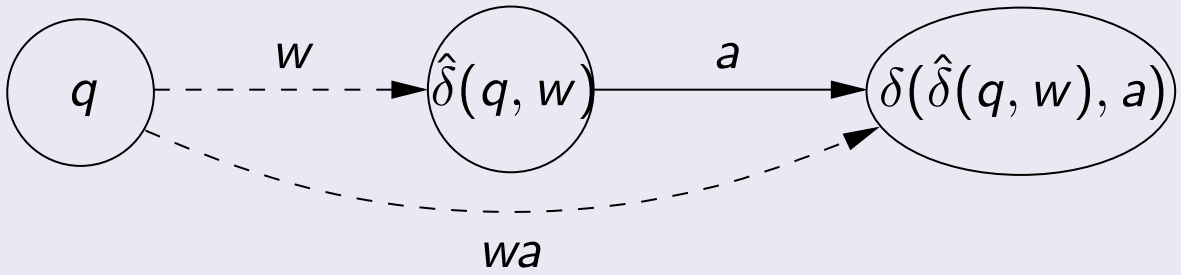
- F.a. $w \in \Sigma^*$ ist $\widehat{\delta}(\perp, w) := \perp$. (Einmal abgestürzt, immer abgestürzt.)
- Der **Rekursionsanfang**: f.a. $q \in Q$ ist $\widehat{\delta}(q, \varepsilon) := q$.
- Und der **Rekursionsschritt**?

Was genau ist $\widehat{\delta}(q, w)$? (2/2)

- Der **Rekursionsschritt**: F.a. $q \in Q$, $w \in \Sigma^*$ und $a \in \Sigma$ gilt für $q' := \widehat{\delta}(q, w)$:

$$\widehat{\delta}(q, wa) := \begin{cases} \bot & \text{falls } \delta \text{ nicht für } (q', a) \text{ definiert ist,} \\ \delta(q', a) & \text{falls } \delta \text{ für } (q', a) \text{ definiert ist.} \end{cases}$$

Grafische Darstellung:

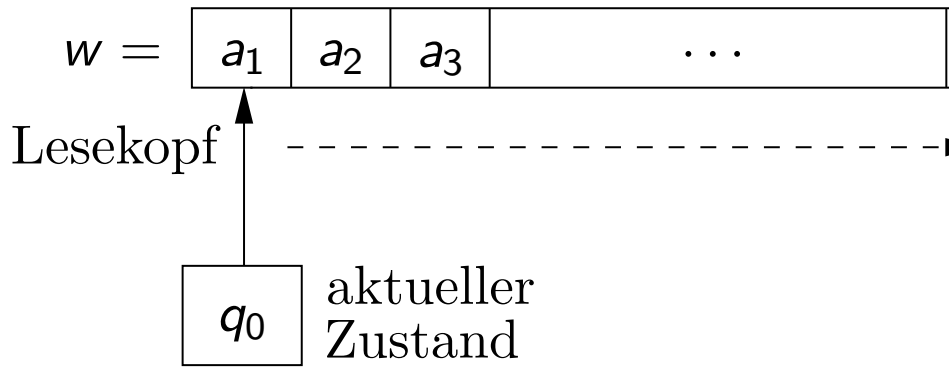


Insgesamt gilt:

- Falls $\widehat{\delta}(q_0, w) \neq \bot$, so ist $\hat{\delta}(q_0, w)$ der Zustand, der durch Verarbeiten des Worts w erreicht wird.
- Falls $\widehat{\delta}(q_0, w) = \bot$, so stürzt der Automat beim Verarbeiten des Worts w ab.

DFAs: Die Maschinensichtweise

Verarbeitung eines Eingabeworts durch einen DFA A:



Wir können uns einen DFA als eine Maschine vorstellen, die

- ihre Eingabe $w = a_1 a_2 a_3 \cdots a_n$ von links-nach-rechts mit Hilfe eines Lesekopfes durchläuft
- und dabei Zustandsübergänge durchführt.

Können wir heutige Rechner durch DFAs modellieren?

Die akzeptierte Sprache eines DFA

Wann akzeptiert $A = (\Sigma, Q, \delta, q_0, F)$ ein Wort w?

Das Eingabewort w wird vom DFA A **akzeptiert**, falls

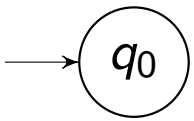
$$\widehat{\delta}(q_0, w) \in F,$$

d.h., A stürzt bei Eingabe von w nicht ab, und der nach Verarbeiten von w erreichte Zustand gehört zur Menge F der akzeptierenden Zustände.

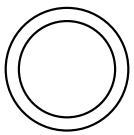
Wie „übersetzt“ sich

$$\text{Akzeptanz von } w = a_1 \cdots a_n$$

in der grafischen Darstellung von A? Es gibt einen in



startenden Weg der Länge n,

- dessen Kanten mit den Symbolen $a_1, \ldots, a_n$ beschriftet sind,
- und der in einem akzeptierenden Zustand  endet.

Die akzeptierte Sprache $L(A)$

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

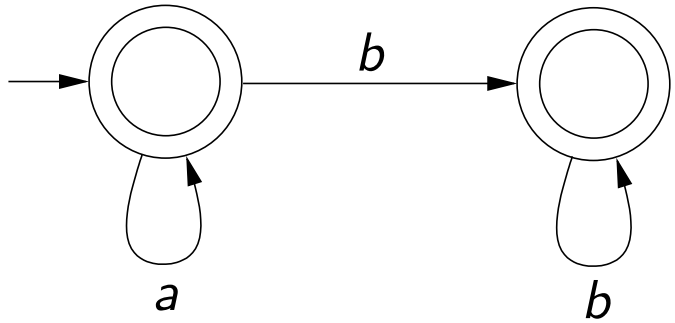
$$L(A) := \{ w \in \Sigma^* \quad : \quad \widehat{\delta}(q_0, w) \in F \}.$$

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

Der Automat A_1

Wir betrachten das Eingabealphabet $\Sigma := \{a, b\}$.

Sei A_1 ein DFA mit folgender graphischer Darstellung:



- A_1 akzeptiert z.B. folgende Worte: ε, a, b, aaa, $aaab$, $aaaabbbb$, bbb, ...
- A_1 „stürzt ab“ z.B. bei Eingabe von ba, $aabbba$.

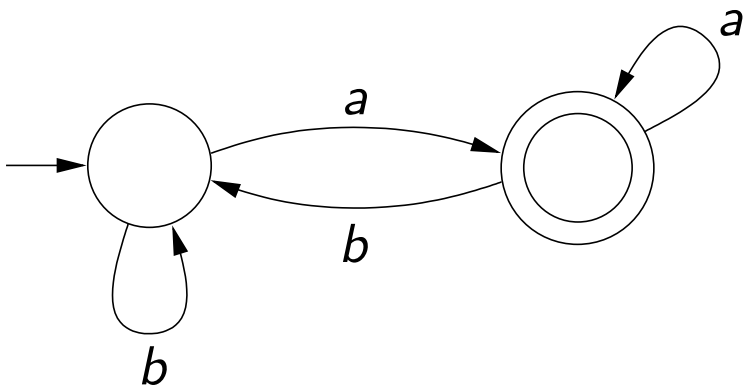
Insgesamt gilt:

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

$a^n b^m$ bezeichnet das Wort $a \cdots ab \cdots b$ der Länge $n + m$,
das aus n a's gefolgt von m b's besteht, z.B. ist $a^3 b^4$ das Wort $aaabbbb$.

Der Automat A_2

Der DFA A_2 hat den Automatengraphen



A_2 akzeptiert die Sprache

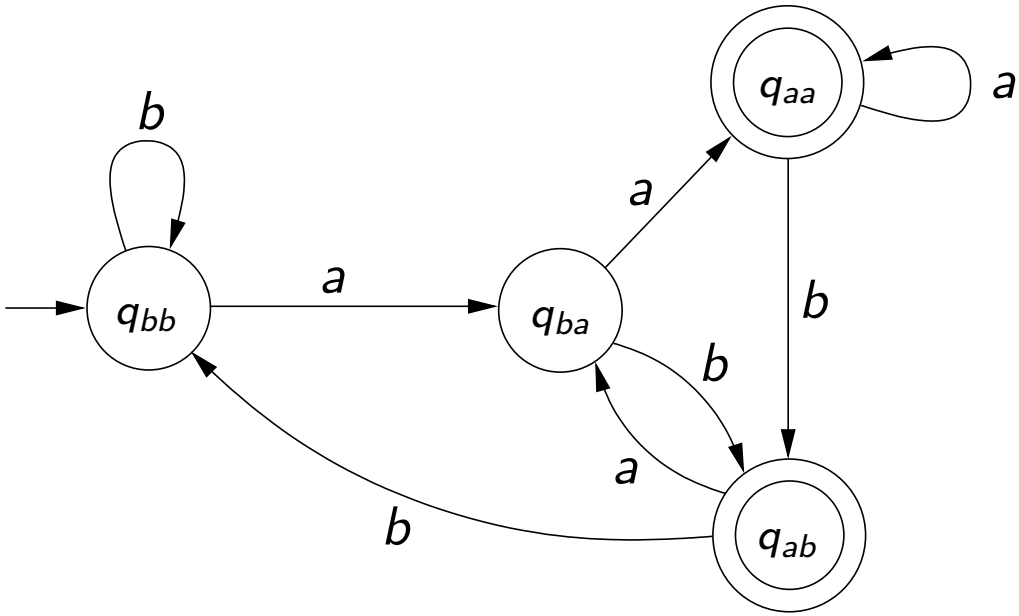
$$L(A_2) = \{w \in \{a,b\}^* : \text{der letzte Buchstabe von } w \text{ ist ein } a\}$$

Der Automat A_3

Die graphische Darstellung eines DFA A_3 mit

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

ist



Ein Paritätscheck

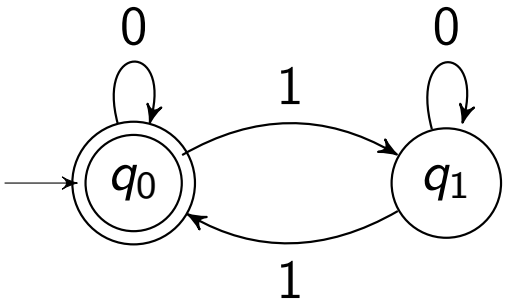
Bei der Speicherung von Daten auf einem Speichermedium eines Computers werden Informationen durch binäre Worte über dem Alphabet $\{0, 1\}$ **kodiert**.

Um Fehler bei der Datenübertragung zu erkennen, wird oft ein „**Paritätsbit** angehängt, so dass die Summe der Einsen im resultierenden Wort w gerade ist.

Für ein beliebiges Wort $w \in \{0, 1\}^*$ sagen wir

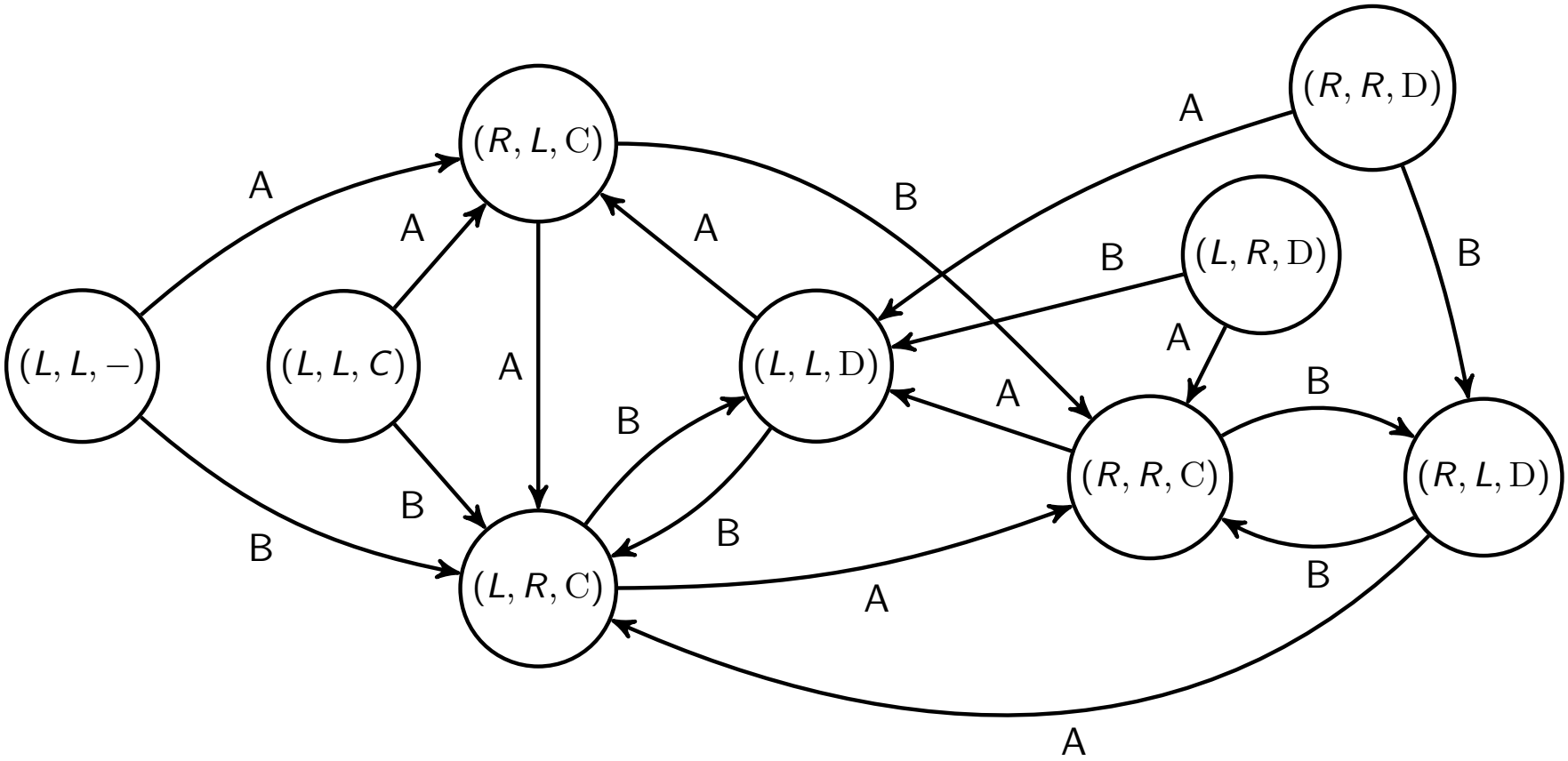
w „besteht den Paritätscheck“,

falls die Anzahl der Einsen in w **gerade** ist.
Der folgende DFA A führt einen Paritätscheck durch:



Für A gilt: $L(A) = \{w \in \{0, 1\}^* \ : \ w \text{ besteht den Paritätscheck } \}$.

Das Murmelspiel



Wir interessieren uns für alle Folgen von Einwürfen von Murmeln in die Eingänge A oder B, so dass die letzte Murmel am Ausgang D herausrollt:

> *Uns interessiert also die Sprache aller Worte $w \in \{A, B\}^*$, so dass der Automat vom Startzustand $(L, L, -)$ aus einen Zustand $(*, *, D)$ erreicht!*

Endliche Automaten in der technischen Informatik

Schaltnetze

Mit einem **Schaltnetz** wird ein Tupel boolescher Funktionen berechnet, also eine Funktion der Form

$$f : \{0,1\}^n \to \{0,1\}^m.$$

Schaltnetze können z.B. arithmetische Funktionen berechnen wie etwa die Addition oder Multiplikation von Zahlen, deren Binärdarstellung $n/2$ Bits lang ist.

Boolesche Funktionen und endliche Automaten

Sei $f : \{0,1\}^n \to \{0,1\}$ eine boolesche Funktion.

Wie berechnet man f mit einem endlichen Automaten $A_f = (\Sigma, Q, \delta, q_0, F)$?

1. A_f besitzt das Eingabealphabet $\Sigma = \{0,1\}^n$,
2. die Zustandsmenge $Q = \{\varepsilon, 0, 1\}$ mit Startzustand $q_0 = \varepsilon$,
3. die **partielle** Übergangsfunktion $\delta : Q \times \Sigma \to Q$ mit

$$\delta(\varepsilon, x) = f(x)$$

4. und die Menge $F = \{1\}$ akzeptierender Zustände.

Aber endliche Automaten können kein Tupel g boolescher Funktionen mit

$$g : \{0,1\}^n \to \{0,1\}^m$$

für $m > 1$ berechnen!

Moore Automaten und Mealy Automaten: DFAs mit Ausgaben

Moore Automaten

Endliche Automaten können „nur“ akzeptieren oder verwerfen,
Moore Automaten können beliebige Ausgaben ausgeben.

Ein **Moore Automat** $(\Sigma, Q, \delta, q_0, \lambda, \Omega, F)$ ist wie ein DFA aufgebaut, besitzt aber zusätzlich

- ein Ausgabealphabet Ω und
- eine Ausgabefunktion $\lambda : Q \to \Omega$.

Ein Moore Automat produziert für jeden Zustand eine Ausgabe.

Moore Automaten und Tupel boolescher Funktionen

Das Tupel

$$f : \{0,1\}^n \to \{0,1\}^m$$

boolescher Funktionen sei vorgegeben.

f wird von dem folgenden Moore Automaten $(\Sigma, Q, \delta, q_0, \lambda, \Omega, F)$ berechnet:

- $\Sigma = \{0,1\}^n$ ist das Eingabealphabet,
- $Q = \{\varepsilon\} \cup \{0,1\}^m$ ist die Zustandsmenge und $q_0 = \varepsilon$ der Startzustand,
- $\delta : Q \times \Sigma \to Q$ mit

$$\delta(\varepsilon, x) = f(x)$$

  ist die **partielle** Übergangsfunktion (δ ist nur für von ε ausgehende Übergänge definiert),
- $\Omega = \{0,1\}^m$ ist das Ausgabealphabet und
- $\lambda : Q \to \Omega$ mit $\lambda(q) = q$ ist die Ausgabefunktion.

Schaltwerke

In einem **Schaltwerk** wird Rückkopplung erlaubt: In „einem Schritt“ wird eine **Ausgabefunktion** g_A und eine **Rückkopplungsfunktion** g_R berechnet mit

$$g_A : \{0,1\}^n \times \{0,1\}^r \to \{0,1\}^m \text{ und } g_R : \{0,1\}^n \times \{0,1\}^r \to \{0,1\}^r$$

Erscheinen nacheinander die Eingaben $x_1, \ldots, x_k \in \{0,1\}^n$, dann werden

Ausgaben $y_1, \ldots, y_k \in \{0,1\}^m$ und **Rückkopplungen** $r_0 = 0^r, r_1, \ldots, r_{k-1}$

berechnet, wobei

$$y_i = g_A(x_i, r_{i-1}) \text{ und } r_i = g_R(x_i, r_{i-1}).$$

Beispielsweise kann ein Steuerwerk durch ein Schaltwerk berechnet werden.

1. Ein Steuerwerk lädt den nächsten Befehl und interpretiert ihn.
2. Operanden, auf die sich der Befehl bezieht, werden geladen und Steuersignale an andere Funktionseinheiten (wie etwa das Rechenwerk) erstellt.

Mealy Automaten

Auch **Mealy Automaten** sind wie DFAs aufgebaut, besitzen aber zusätzlich

- ein Ausgabealphabet Ω und
- eine Ausgabefunktion $\lambda : Q \times \Sigma \to \Omega$.

*Ein Mealy Automat produziert eine vom Zustand **und** vom Eingabesymbol abhängige Ausgabe.*

Warum können wir einen Moore Automaten mit Ausgabefunktion

$$\lambda : Q \to \Omega$$

auch als Mealy Automaten mit Ausgabefunktion

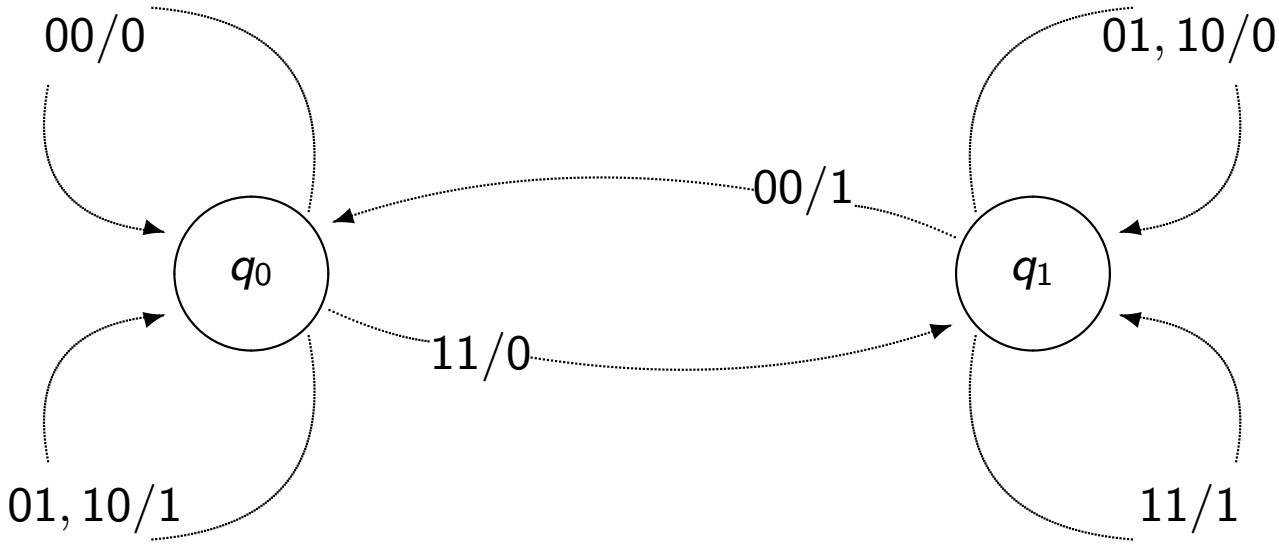
$$\lambda' : Q \times \Sigma \to \Omega$$

auffassen? Wie sollten wir $\lambda'(q, a)$ definieren?

Ein Mealy Automat für die Addition

Die Binärzahlen $x = 0x_n \cdots x_0$ und $y = 0y_n \cdots y_0$ sind zu addieren.
Die Eingabe ist $[x_0 y_0][x_1 y_1] \cdots [x_n y_n][00]$.

Das i-te Ausgabe-Bit hängt nur ab von x_{i-1}, y_{i-1} und dem im vorherigen Schritt evtl. erzeugten Übertrag.



Mealy Automaten und Schaltwerke

Schaltwerke lassen sich durch (Moore Automaten oder) Mealy Automaten modellieren.

1. Angenommen, das Schaltwerk berechnet

$$\begin{aligned} \text{die Ausgabefunktion} \quad & g_A : \{0,1\}^r \times \{0,1\}^n \to \{0,1\}^m \\ \text{mit der Rückkopplung} \quad & g_R : \{0,1\}^r \times \{0,1\}^n \to \{0,1\}^r. \end{aligned}$$

2. Wir definieren einen Mealy Automaten $M = (\Sigma, Q, \delta, q_0, \lambda, \Omega)$ wie folgt:
   - Eingabealphabet $\Sigma = \{0,1\}^n$ und Ausgabealphabet $\Omega = \{0,1\}^m$,
   - Zustandsmenge $Q = \{0,1\}^r$ und Anfangszustand $q_0 = 0^r$,
   - Ausgabefunktion $\lambda : Q \times \Sigma \to \Omega$ mit $\lambda = g_A$ und
   - Übergangsfunktion $\delta : Q \times \Sigma \to Q$ mit $\delta = g_R$.

   M verhält sich wie das Schaltwerk, berechnet also dieselbe Ausgabenfolge.

Automaten können Rückkopplung mit Hilfe ihrer Zustände simulieren und bieten sich deshalb für die (partielle) Beschreibung von Schaltwerken an!

Technische Informatik: Der Entwurf von Schaltwerken

Moore oder Mealy Automaten spielen eine zentrale Rolle.

Der **Entwurfsprozess**

1. Ein Mealy Automat wird zur Lösung eines Teils der Problemstellung entwickelt.
2. Der Automat wird **minimiert**, d.h. ein äquivalenter Automat mit kleinster Zustandsmenge wird berechnet und seine Automatentabelle bestimmt.
3. Mit Verfahren der **Logik-Optimierung** (wie etwa Quine/McCluskey) bestimmt man die minimale Gatterzahl für den kombinatorischen Teil der Schaltung.

Alle Details und viel, viel mehr in der Vorlesung

Hardwarearchitekturen und Rechensysteme.

Bitte, bitte, bitte

keinen **partiell definierten** Moore oder Mealy Automaten

„in Silikon gießen“.

Abstürzen ist **nicht** erlaubt!

Wir haben partiell definierte Automaten nur benutzt, um kompakte Beschreibungen zu erhalten.

Können wir heutige Rechner durch Mealy Automaten simulieren?

1. „Im Prinzip“ ja: Heutige Rechner besitzen zwar einen modifizierbaren Speicher, aber dieser Speicher ist beschränkt.
   - Was immer ein moderner Rechner berechnen kann, lässt sich auch durch Moore- oder Mealy-Automaten berechnen.
2. Aber um k Flip-Flops zu simulieren, brauchen wir Mealy Automaten mit bis zu 2^k Zuständen.
   - Eine Simulierung moderner Rechner durch Mealy Automaten ist völlig unsinning, weil der notwendige Speicher im schlimmsten Fall **exponentiell** anwächst.
3. Aber Mealy Automaten bieten sich zum Beispiel an, um Steuerwerke zu simulieren.
   - Und dann sollten wir mit Ihnen arbeiten, denn wir können Mealy Automaten **minimieren** wie wir gleich sehen werden.

Minimierung von DFAs

Minimiere die Zustandszahl

$A = (\Sigma, Q^A, \delta^A, q_0^A, F^A)$ und $B = (\Sigma, Q^B, \delta^B, q_0^B, F^B)$ seien vollständige DFAs.

(a) Wir nennen A und B **äquivalent**, wenn gilt

$$\mathbf{L(A) = L(B)}.$$

(b) A heißt

minimal,

wenn kein mit A äquivalenter vollständiger DFA eine kleinere Zustandszahl besitzt.

DAS ZIEL:

- Gegeben ist ein vollständiger DFA A.
- Bestimme einen minimalen, mit A äquivalenten vollständigen DFA.

Zustandsminimierung: Die Idee

Der DFA $A = (\Sigma, Q, \delta, q_0, F)$ sei gegeben.

Die Idee: Wir sollten doch zwei Zustände $p, q \in Q$ zu einem einzigen Zustand verschmelzen dürfen, wenn p und q „äquivalent“ sind,

also dasselbe Ausgabeverhalten besitzen.

Was bedeutet das?

Die **Verschmelzungsrelation** $\equiv_A$ ist eine 2-stellige Relation über der Zustandsmenge Q. Wir sagen, dass Zustände $p, q \in Q$ äquivalent bzgl. A sind, (kurz $p \equiv_A q$), wenn

$$\text{f.a. Worte } w \in \Sigma^* : \widehat{\delta}(p, w) \in F \Leftrightarrow \widehat{\delta}(q, w) \in F.$$

Wir nennen $\equiv_A$ **Verschmelzungsrelation**, da wir bzgl. $\equiv_A$ äquivalente Zustände in einen Zustand verschmelzen möchten.

Einschub: Äquivalenzrelationen

2-stellige Relationen und gerichtete Graphen

Zur Erinnerung: Für eine 2-stellige Relation R über der Knotenmenge V gilt

$$R \subseteq V \times V.$$

Wir können somit

- die Relation R als Kantenmenge und
- umgekehrt eine Kantenmenge als eine 2-stellige Relation auffassen.

Gerichtete Graphen mit Knotenmenge V

und

2-stellige Relationen über V

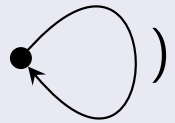
sind äquivalente Konzepte!

Wichtige Eigenschaften 2-stelliger Relationen

Sei E eine 2-stellige Relation über einer Menge V, d.h. $G = (V, E)$ ist ein gerichteter Graph.

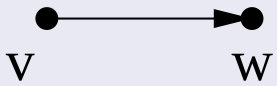
(a) E heißt **reflexiv**, falls für alle $v \in V$ gilt:

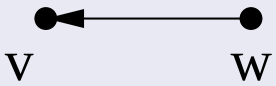
$$(v, v) \in E. \qquad \text{(Skizze: } v \text{)}$$



(b) E heißt **symmetrisch**, falls f.a. $v, w \in V$ gilt:

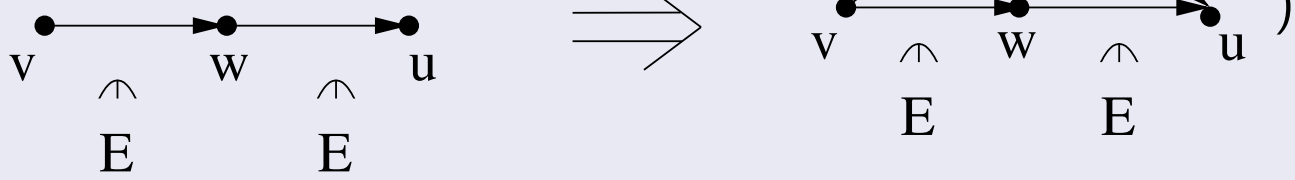
$$\text{Wenn } (v, w) \in E, \text{ dann auch } (w, v) \in E.$$

zur Kante 

gibt es auch die "Rückwärtskante" 

(c) E heißt **transitiv**, falls f.a. $v, w, u \in V$ gilt:

$$\text{Ist } (v, w) \in E \text{ und } (w, u) \in E, \text{ so auch } (v, u) \in E.$$

(Skizze: 

)

Äquivalenzrelationen

(a) Eine **Äquivalenzrelation** ist eine 2-stellige Relation, die

reflexiv, **transitiv** und **symmetrisch** ist.

(b) Sei E eine Äquivalenzrelation über einer Menge V.

- Für jedes $v \in V$ bezeichnet

$$[v]_E := \{ v' \in V \mid (v, v') \in E \}$$

die **Äquivalenzklasse von v** bezüglich E.

  - ⋆ $[v]_E$ besteht aus allen Elementen von V, die gemäß E "äquivalent" zu v sind.

- Eine Menge $W \subseteq V$ heißt **Äquivalenzklasse** (bzgl. E), falls $W = [v]_E$ für ein Element $v \in V$ gilt.

  - ⋆ Das Element v heißt **Vertreter** seiner Äquivalenzklasse W.

Die wichtigste Eigenschaft von Äquivalenzrelationen

Sei E eine Äquivalenzrelation über der Menge V.

(a) Dann gilt für alle Elemente $v, w \in V$

$$[v]_E = [w]_E \text{ oder } [v]_E \cap [w]_E = \emptyset.$$

(b) V ist eine disjunkte Vereinigung von Äquivalenzklassen.

Beweis: Siehe Tafel.

Äquivalenzrelationen: Beispiele (1/2)

(a) **Gleichheit**: Für jede Menge M ist

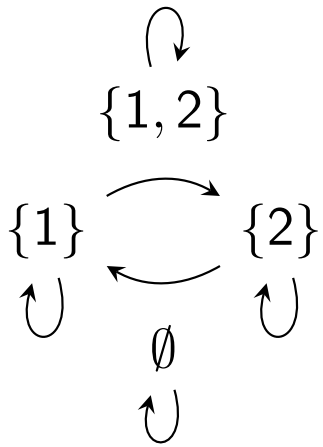
$$E := \{ (m, m) : m \in M \}$$

eine Äquivalenzrelation: „$(x, y) \in E$“ gilt genau dann, wenn „$x = y$“.

(b) **Gleichmächtigkeit**: Für jede endliche Menge M ist

$$E := \{ (A, B) : A \subseteq M, B \subseteq M, |A| = |B| \}$$

eine Äquivalenzrelation über der Potenzmenge $\mathcal{M}$. *Skizze für* $M = \{1, 2\}$*:*



Äquivalenzrelationen: Beispiele (2/2)

(c) **Logische Äquivalenz**: Die Relation

$$E := \{ (\phi, \psi) : \phi, \psi \in \mathsf{AL}, \phi \equiv \psi \}$$

ist eine Äquivalenzrelation über der Menge AL aller aussagenlogischen Formeln.

(d) **Graph-Isomorphie**: Für jede Menge V ist

$$E := \{(G_1, G_2) \mid G_1 = (V, E_1),\ G_2 = (V, E_2) \text{ sind isomorphe ungerichtete Graphen } \}$$

eine Äquivalenzrelation über der Menge aller ungerichteten Graphen mit Knotenmenge V.

Der Index

Sei E eine Äquivalenzrelation. Dann ist der **Index** von E die Anzahl der verschiedenen Äquivalenzklassen.

Der Index einer Äquivalenzrelation gibt an,
wie viele verschiedene Äquivalenzklassen es gibt.

(a) Betrachte wieder die Äquivalenzrelation der Gleichmächtigkeit für Teilmengen $A, B \subseteq M$, also

$$A \cong B \Leftrightarrow |A| = |B|.$$

Dann stimmt der Index überein mit $|M| + 1$.

(b) Der Index der Verschmelzungsrelation $\equiv_A$

stimmt mit der minimalen Zustandszahl überein,

WENN der Automat nach Verschmelzung minimal ist.

Zurück zur Verschmelzungsrelation $\equiv_A$

Die Verschmelzungsrelation ist eine Äquivalenzrelation

Der DFA $A = (\Sigma, Q, \delta, q_0, F)$ sei gegeben. Zur Erinnerung:

$$p \equiv_A q \text{ wenn f.a. Worte } w \in \Sigma^* \text{ gilt: } \left(\widehat{\delta}(p, w) \in F \Leftrightarrow \widehat{\delta}(q, w) \in F\right).$$

(a) Die Verschmelzungsrelation ist

- **reflexiv**, f.a. $p \in Q$: $p \equiv_A p$,
- **symmetrisch**, f.a. $p, q \in Q$: wenn $p \equiv_A q$, dann $q \equiv_A p$ und
- **transitiv**, f.a. $p, q, r \in Q$: wenn $p \equiv_A q$ und $q \equiv_A r$, dann $p \equiv_A r$.

Warum? Siehe Tafel.

(b) Die Verschmelzungsrelation ist eine Äquivalenzrelation!

Die Zustandsmenge Q ist eine disjunkte Vereinigung der Klassen von $\equiv_A$.

Was ist zu tun?

Sei der DFA $A = (\Sigma, Q, \delta, q_0, F)$ gegeben.

Wir führen die folgenden Schritte durch:

1. Wir bestimmen die Äquivalenzklassen der Verschmelzungsrelation $\equiv_A$.
2. Für jede Äquivalenzklasse von $\equiv_A$ verschmelzen wir alle Zustände der Klasse zu einem einzigen Zustand und fügen „entsprechende“ Übergänge ein. Den neuen Automaten nennen wir $\mathbf{A'}$ und bezeichnen ihn als

   den **Äquivalenzklassenautomaten** von A.

Die wichtigen Fragen

(a) Wie sollen wir die Zustandsübergänge von A' definieren, so dass A und A' dieselbe Sprache berechnen? Können wir
- die Verschmelzungsrelation $\equiv_A$ wie auch den
- Äquivalenzklassenautomaten A' effizient berechnen?

(b) Die Anzahl der Zustände von A' stimmt mit dem Index von $\equiv_A$ überein.
- Wenn der Index mit der minimalen Zustandszahl übereinstimmt, dann ist A' **minimal**!

Wir bestimmen die Verschmelzungsrelation $\equiv_A$

Zeugen (1/2)

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

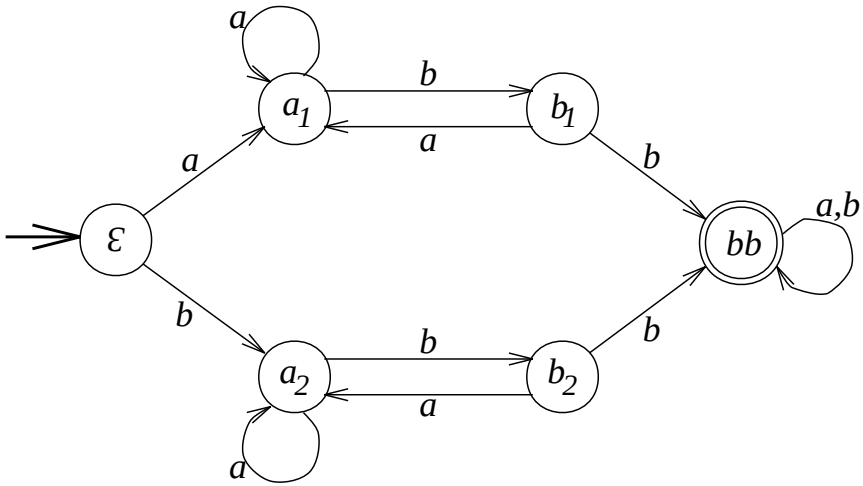
(a) Das Wort $w \in \Sigma^*$ ist **Zeuge** für die Inäquivalenz von p und q, wenn

$$\Big(\widehat{\delta}(p,w) \in F \wedge \widehat{\delta}(q,w) \notin F\Big) \vee \Big(\widehat{\delta}(p,w) \notin F \wedge \widehat{\delta}(q,w) \in F\Big).$$

Wir sagen auch, dass w die Zustände p und q **trennt**.

(b) Es ist $p \not\equiv_A q$ genau dann, wenn es einen Zeugen für die Inäquivalenz von p und q gibt, bzw wenn es ein Wort gibt, das p und q trennt.

Zeugen (2/2)



- Finde einen Zeugen für die Inäquivalenz von b_1 und bb.
- Welcher Zeuge trennt ε und a_1?
- Gibt es Zeugen, die die Zustände a_1 und a_2 trennen?

Wir bestimmen alle Paare **nicht-äquivalenter** Zustände

1. **Markiere** alle Paarmengen $\{p, q\}$ mit $p \in F$ und $q \notin F$ (als nicht-äquivalent).
   - Es ist $\delta(p, \varepsilon) \in F$ und $\delta(q, \varepsilon) \notin F$.
   - $w = \varepsilon$ ist **Zeuge** für die Nicht-Äquivalenz von p und q.
2. Wenn die Paarmenge $\{p, q\}$ bereits markiert wurde und

   $$\text{wenn } \delta(r, a) = p \text{ sowie } \delta(s, a) = q$$

   für ein $a \in \Sigma$, dann **markiere** $\{r, s\}$.
   - Da $p \not\equiv_A q$, gibt es einen **Zeugen** w mit

     $$\left(\widehat{\delta}(p, w) \in F \text{ und } \widehat{\delta}(q, w) \notin F\right) \text{ oder } \left(\widehat{\delta}(p, w) \notin F \text{ und } \widehat{\delta}(q, w) \in F\right).$$

   - Das Wort aw **trennt** r und s.
3. Halte, wenn keine neuen Paarmengen $\{r, s\}$ markiert werden können.
   - Unser Verfahren behauptet, dass $p \not\equiv_A q$ genau dann gilt, wenn die Paarmenge $\{p, q\}$ markiert wurde.

Stimmt die Behauptung: Finden wir alle Paare nicht-äquivalenter Zustände?

Unser Verfahren funktioniert!

Sei P die Menge aller Paare $\{r, s\}$ nicht-äquivalenter Zustände, die aber von unserem Verfahren **nicht** gefunden werden. **Zeige**, dass P leer ist!

Angenommen, P ist nicht-leer.

0. $\{p, q\} \in P$ habe unter allen Paaren in P einen **kürzesten** Zeugen w.
1. Wenn $w = \varepsilon$, dann ist
   - $\big(\delta(p, \varepsilon) \in F$ und $\delta(q, \varepsilon) \notin F\big)$ oder $\big(\delta(p, \varepsilon) \notin F$ und $\delta(q, \varepsilon) \in F\big)$,
   - bzw. $\big(p \in F$ und $q \notin F\big)$ oder $\big(p \notin F$ und $q \in F\big)$.

   Aber dann haben wir $\{p, q\}$ im Schritt 1. markiert. ↯
2. Wenn $w = au$ für den Buchstaben $a \in \Sigma$, dann ist
   - $\big(\widehat{\delta}(p, au) \in F$ und $\widehat{\delta}(q, au) \notin F\big)$ oder $\big(\widehat{\delta}(p, au) \notin F$ und $\widehat{\delta}(q, au) \in F\big)$,
   - bzw. $\big(\widehat{\delta}(\delta(p, a), u) \in F$ und $\widehat{\delta}(\delta(q, a), u) \notin F\big)$ oder $\big(\widehat{\delta}(\delta(p, a), u) \notin F$ und $\widehat{\delta}(\delta(q, a), u) \in F\big)$.

   Aber dann ist $\delta(p, a) \not\equiv_A \delta(q, a)$ mit dem **kürzeren** Zeugen u:
   - Nach Annahme haben wir $\{\delta(p, a), \delta(q, a)\}$ z.B. in einer Menge M_i markiert
   - und werden darauf folgend $\{p, q\}$ in M_{i+1} markieren. ↯

Der Automat nach allen Verschmelzungen

Wie sieht der Automat nach dem Verschmelzen aller äquivalenten Zustände aus?

- Für Zustand $p \in Q$ bezeichnet

$$[p]_A := \{ q \in Q \mid p \equiv_A q \}$$

die Äquivalenzklasse von p.

- Der **Äquivalenzklassenautomat** A' für A besitzt
  - die Zustandsmenge

$$Q' := \{ [p]_A \mid p \in Q \},$$

  - den Anfangszustand $q_0' := [q_0]_A$,
  - die Menge $F' := \{ [p]_A \mid p \in F \}$ der akzeptierenden Zustände und
  - das Programm δ' mit

$$\delta'([p]_A, a) := [\ \delta(p, a)\ ]_A$$

für alle $q \in Q$, $a \in \Sigma$.

Der Minimierungsalgorithmus

Berechnung von $\equiv_A$ und A'

Eingabe: Ein DFA $A = (Q, \Sigma, \delta, q_0, F)$.

Schritt 1: Entferne aus A alle **überflüssigen** Zustände, d.h. alle Zustände, die nicht von q_0 aus erreichbar sind.

Schritt 2: Bestimme alle Paarmengen $\{p, q\}$ mit $p, q \in Q$ und $p \not\equiv_A q$:

1. Markiere alle Paarmengen in $M_0 := \Big\{ \{p, q\} \ : \ p \in F, \ q \in Q \setminus F \Big\}$; Setze $i := 0$
2. Wiederhole
3. Für alle Paarmengen $\{p, q\}$ mit $p \neq q$ und für alle $x \in \Sigma$ tue folgendes:
4. Markiere $\{p, q\}$, falls $\{\delta(p, x), \delta(q, x)\} \in M_i$.
5. Sei M_{i+1} die Menge aller hierbei **neu** markierten Paarmengen.
6. $i := i + 1$
7. bis $M_i = \emptyset$
8. **Ausgabe:** $M := M_0 \cup \cdots \cup M_{i-1}$.

Schritt 3: Konstruiere $A' := (Q', \Sigma, \delta', q_0', F')$:

$Q' := \Big\{ [q]_A \ : \ q \in Q \Big\}$, wobei $[q]_A = \Big\{ p \in Q \ : \ \{p, q\} \notin M \Big\}$

$q_0' := [q_0]_A, \quad F' := \Big\{ [q]_A \ : \ q \in F \Big\}$

$\delta' : Q' \times \Sigma \to Q' \quad \text{mit} \quad \delta'\big([q]_A, x\big) := [\delta(q, x)]_A \quad \text{für alle } q \in Q \text{ und } x \in \Sigma.$

Ist der Algorithmus korrekt und effizient?

In jeder Iteration in Schritt 2 wird mindestens eine neue Paarmenge markiert:
Es gibt also höchstens $|Q|^2$ Iterationen und der Algorithmus ist effizient.

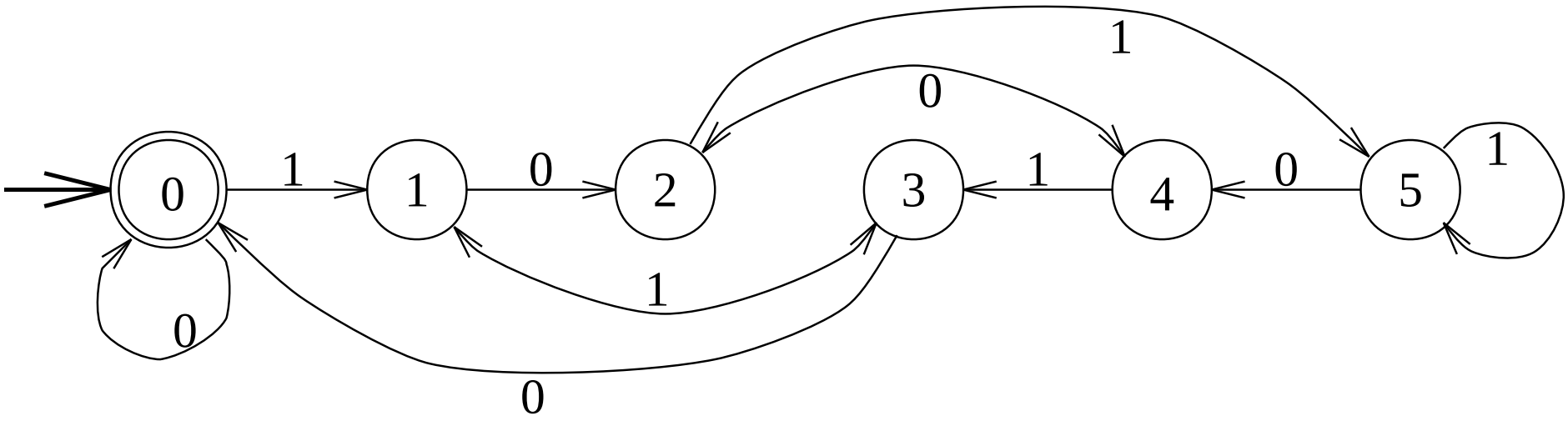
Die wichtigen Fragen:

1. Sind A und der Äquivalenzklassenautomat A' äquivalent, d.h. berechnet A' dieselbe Sprache wie A? ✓
2. Ist A' minimal?

Aber zuerst rechnen wir einige Beispiele durch.

Ein erstes Beispiel

Der DFA A mit Zustandsdiagramm



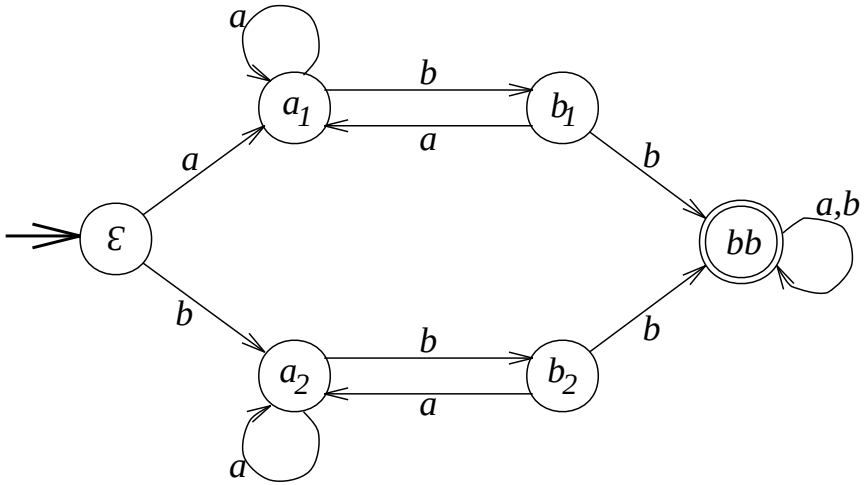
akzeptiert die Binärdarstellungen aller durch 6 teilbaren Zahlen. D.h.:

$$L(A) \quad = \quad \left\{ w \in \{0,1\}^* \ : \ \sum_{i=1}^{|w|} w_i \, 2^{|w|-i} \equiv 0 \ (\text{mod } 6) \right\}.$$

Aufgabe: Bestimme den Äquivalenzklassenautomaten A'.

– siehe Tafel –

Zustandsminimierung: Die Mengen M_0, M_1, M_2

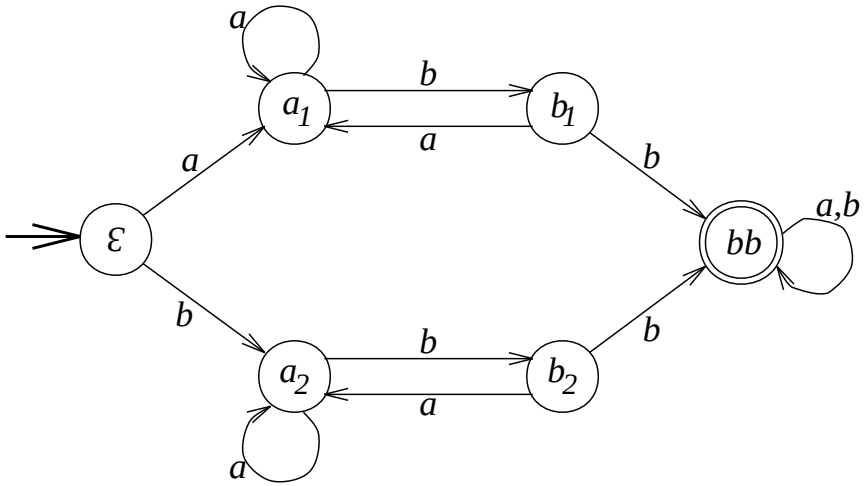


a_1	M_2				
a_2	M_2				
b_1	M_1	M_1	M_1		
b_2	M_1	M_1	M_1		
bb	M_0	M_0	M_0	M_0	M_0
	ε	a_1	a_2	b_1	b_2

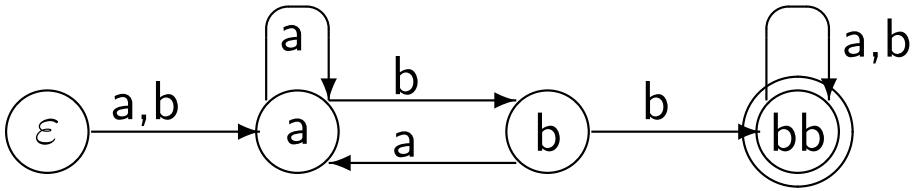
$M_3 = \emptyset$.

Zustandsminimierung: Der Äquivalenzklassenautomat A'

$M_3 = \emptyset \Rightarrow \{bb\}$ ist die Klasse des einzigen akzeptierenden Zustands, $\{\varepsilon\}$ ist die Klasse des Startzustands, $\{a_1, a_2\}$ und $\{b_1, b_2, \}$ sind die restlichen Klassen.



Der **Äquivalenzklassenautomat**:



Genügt es, wenn wir nur Zustände mit identischen Nachfolgezuständen verschmelzen? **NEIN!**

Zustandsminimierung: Die Tabelle

Wie haben wir die Tabelle gefüllt?

Angenommen, wir haben die Mengen $M_0, \ldots, M_i$ bestimmt und die „entsprechenden“ Positionen in der Tabelle markiert.

1. Wir haben nacheinander alle unmarkierten Einträge der Tabelle inspiziert.
2. Angenommen, die Position in Zeile p und Spalte q ist unmarkiert. Markiere diese Position mit M_{i+1}, wenn es einen Buchstaben $x \in \Sigma$ gibt, so dass
   - die Position in Zeile $\delta(p, x)$ und Spalte $\delta(q, x)$
   - bzw. in Spalte $\delta(p, x)$ und Zeile $\delta(q, x)$

   markiert ist.

Was ist Sache?

1. Der ursprüngliche Automat $A = (Q, \Sigma, \delta, q_0, F)$ und sein Äquivalenzklassenautomat A' sind äquivalent.
2. Wir können A' effizient berechnen.

? Aber hat A' unter allen mit A äquivalenten DFAs die kleinste Zustandszahl?

Die Nerode-Relation für die Sprache $L = L(A)$ hat die Antwort.

Die Nerode-Relation

Sei $A = (\Sigma, Q, \delta, q_0, F)$ ein DFA. Wenn $\widehat{\delta}(q_0, u) = \widehat{\delta}(q_0, v)$„ dann

$$\text{f.a. } w \in \Sigma^* : (uw \in L(A) \Leftrightarrow vw \in L(A)).$$

L sei eine Sprache über der endlichen Menge Σ, d.h. es gilt $L \subseteq \Sigma^*$.

(a) Die **Nerode-Relation** $\equiv_L$ für L ist eine 2-stellige Relation über Σ^*.
Für alle Worte $u, v \in \Sigma^*$ definiere

$$u \equiv_L v \ :\Leftrightarrow \ \text{f.a. } w \in \Sigma^* \text{ gilt: } \big(uw \in L \Leftrightarrow vw \in L\big).$$

(b) Wir sagen, dass das Wort $w \in \Sigma^*$ die Worte $u, v \in \Sigma^*$ **trennt**,
bzw. dass w ein **Zeuge** für die Inäquivalenz von u und v ist, wenn

$$\Big(uw \in L \wedge vw \notin L\Big) \vee \Big(uw \notin L \wedge vw \in L\Big).$$

(c) **Index(L)** ist die Anzahl der Äquivalenzklassen von $\equiv_L$.

Die Nerode-Relation ist eine Äquivalenzrelation

Warum? Die Nerode-Relation $\equiv_L$ ist

1. reflexiv, denn $u \equiv_L u$ gilt f.a. $u \in \Sigma^*$,
2. symmetrisch, denn aus $u \equiv_L v$ folgt $v \equiv_L u$ f.a. $u, v \in \Sigma^*$,
3. transitiv, denn aus $u \equiv_L v$, $v \equiv_L w$ folgt $u \equiv_L w$ f.a. $u, v, w \in \Sigma^*$.

Beweis: Siehe Tafel.

Der absolute Hammer:

Für eine reguläre Sprache L stimmt **Index(L)**,

die Anzahl der Äquivalenzklassen von $\equiv_L$,

mit der **minimalen Zustandszahl** für DFAs A mit $L(A) = L$ überein!

Die minimale Zustandszahl $\geqslant$ Index(L)

Der DFA $A = (Q, \Sigma, \delta, q_0, F)$ akzeptiere die Sprache L, es gelte also $L(A) = L$.

Angenommen, es ist

$$\widehat{\delta}(q_0, u) = \widehat{\delta}(q_0, v).$$

1. Dann ist $\widehat{\delta}(q_0, uw) = \widehat{\delta}(q_0, vw)$ für alle Worte $w \in \Sigma^*$ und A akzeptiert das Wort uw genau dann, wenn A das Wort vw akzeptiert.
2. Aber $L(A) = L$ und $uw \in L \Leftrightarrow vw \in L$ folgt für alle Worte $w \in \Sigma^* \quad \Rightarrow$

$$u \equiv_L v.$$

(a) Alle Worte, die auf denselben Zustand von A führen, sind äquivalent bzgl. $\equiv_L$.
(b) Jeder DFA A mit $L = L(A)$ hat mindestens **Index(L)** Zustände.

Die minimale Zustandszahl $\leqslant$ Index(L)

Der DFA $A = (Q, \Sigma, \delta, q_0, F)$ akzeptiere die Sprache L, es gelte also $L(A) = L$.
$A' = (Q', \Sigma, \delta', q_0', F')$ sei sein Äquivalenzklassenautomat.

Angenommen, Worte $u, v \in \Sigma^*$ führen in **A′** zu verschiedenen Zuständen.

1. Für A sei $p = \widehat{\delta}(q_0, u)$ und $q = \widehat{\delta}(q_0, v)$.
2. Es folgt $p \not\equiv_A q$ und es gibt einen Zeugen $w \in \Sigma^*$ für die Nicht-Äquivalenz.
   - Also: $\left(\widehat{\delta}(p, w) \in F \wedge \widehat{\delta}(q, w) \notin F\right) \vee \left(\widehat{\delta}(p, w) \notin F \wedge \widehat{\delta}(q, w) \in F\right)$, bzw.
   - $\left(\widehat{\delta}(q_0, uw) \in F \wedge \widehat{\delta}(q_0, vw) \notin F\right) \vee \left(\widehat{\delta}(q_0, uw) \notin F \wedge \widehat{\delta}(q_0, vw) \in F\right)$, bzw.
   - $\Big(uw \in L \wedge vw \notin L\Big) \vee \Big(uw \notin L \wedge vw \in L\Big)$.
3. Wenn u und v in **A′** zu verschiedenen Zuständen führen, dann folgt $u \not\equiv_L v$.
4. Wenn $u \equiv_L v$, dann führen u und v in **A′** zum selben Zustand.

Die **Zustandszahl von A′** stimmt überein mit **Index(L(A))**:

Der Äquivalenzklassenautomat A′ ist ein minimaler DFA für L(A).

Die Nerode-Relation: Beispiele

Zeige in jedem Fall, dass Index(L) möglichst groß ist:

1. $L = \{w \in \{0,1\}^* : w \text{ hat gerade viele Einsen}\}$,
2. $L = \{a,b\}^*\{ab\}$,
3. L die Menge aller Binärdarstellungen für durch 6 teilbare Zahlen,
4. $L_u = \{w \in \Sigma^* : u \text{ ist ein Teilwort von } w\}$ für ein Wort $u \in \Sigma^*$.

Reguläre Sprachen

Reguläre Sprachen

Eine Teilmenge $L \subseteq \Sigma^*$ heißt eine **reguläre Sprache**, wenn es einen DFA A gibt mit

$$L = L(A).$$

(a) Gibt es Teilmengen von Σ^*, die **keine** regulären Sprachen sind?

(b) Ist $L = \{ a^n b^n \ : \ n \in \mathbb{N} \}$ eine reguläre Sprache?

(c) Sei Σ ein beliebiges Alphabet und sei $w \in \Sigma^*$ ein Wort über Σ. Ist

$$L = \{u \in \Sigma^* \ : \ w \text{ ist ein Teilwort von } u \ \}$$

eine reguläre Sprache?

Der Satz von Nerode

Wann ist eine Sprache regulär?

Satz von Nerode

Eine Teilmenge $L \subseteq \Sigma^*$ ist **regulär** $\Leftrightarrow$ **Index(L) ist endlich.**

$\Rightarrow$ $L \subseteq \Sigma^*$ sei regulär. Dann gibt es einen DFA A mit $L = L(A)$.

- A hat mindestens Index(L) Zustände.
- Also ist Index(L) endlich.

$\Leftarrow$ Index(L) sei endlich.

- Es gibt stets einen (möglicherweise **unendlichen**) Automaten A mit $L = L(A)$.
- Der Äquivalenzklassenautomat A' ist minimal
  und hat nach Annahme genau Index(L) viele Zustände.
- Es ist $L = L(A')$ für den **DFA** $A' \Rightarrow L$ ist regulär.

$L = \{a^n b^n \ : \ n \in \mathbb{N}\}$ ist nicht regulär

Bestimme unendlich viele Worte $u_k \in \{a, b\}^*$, so dass

$$u_k \not\equiv_L u_\ell$$

für alle $k \neq \ell$ gilt.

Setze $u_i := a^i$. Für $k \neq \ell$ gilt $u_k \not\equiv_L u_\ell$, denn

$$u_k b^k \in L, \quad \text{aber} \quad u_\ell b^k \notin L.$$

Index$(L) = \infty$ und L ist nicht regulär.

DFAs können nicht (unbeschränkt) zählen.

$L = \{ww \ : \ w \in \{a, b\}^*\}$ ist nicht regulär

Wir bestimmen unendlich viele Worte $u_k \in \{a, b\}^*$, so dass

$$u_k \not\equiv_L u_\ell$$

für alle $k \neq \ell$ gilt.

Setze $u_i := a^i b$. Für $k \neq \ell$ gilt $u_k \not\equiv_L u_\ell$, denn

$$u_k a^k b \in L, \quad \text{aber} \quad u_\ell a^k b \notin L.$$

Index$(L) = \infty$ und L ist nicht regulär.

DFAs können sich nur beschränkt viele Dinge merken.

Weitere nicht-reguläre Sprachen

Keine der folgenden Sprachen ist regulär.

- $L_1 = \{ a^n b^m \ : \ n, m \in \mathbb{N},\ n \leqslant m \}$:
  - DFAs können nicht vergleichen,
- $L_2 = \{ a^n b^m c^{n+m} \ : \ n, m \in \mathbb{N} \}$:
  - DFAs können nicht addieren,
- $L_3 = \{ a^{n^2} \ : \ n \in \mathbb{N} \}$:
  - DFAs können nicht quadrieren,
- $L_4 = \{ w \in \{a, b\}^* \ : \ w \text{ ist ein Palindrom} \}$:
  - Endliche Automaten haben ein nur beschränkt großes Gedächtnis.

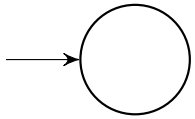
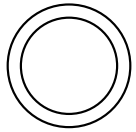
Zeige, dass der Index unendlich ist. Das gelingt häufig sogar mit unendlich vielen Worten $u_1, u_2, \ldots$ und $v_1, v_2, \ldots$ mit den Eigenschaften

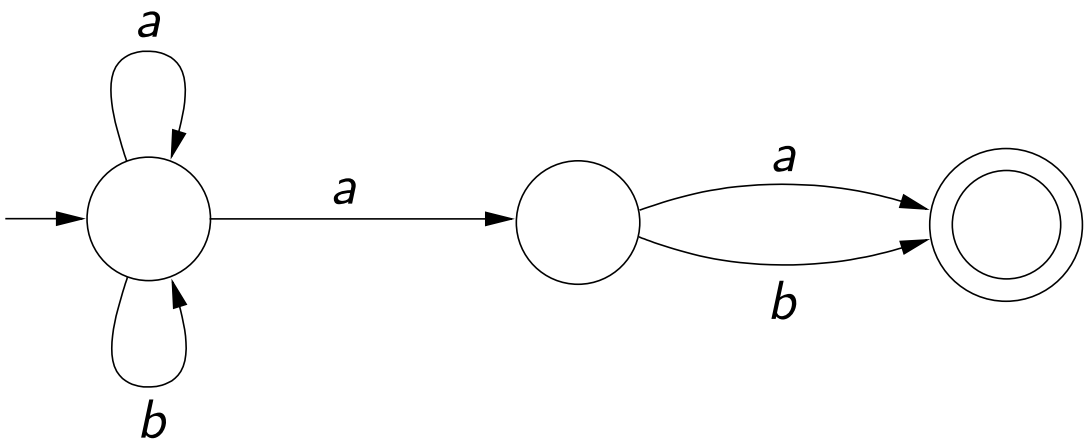
- $u_i v_i \in L$ und
- $u_j v_i \notin L$ für $i \neq j$.

NFAs

NFAs: DFAs, die raten dürfen

Ein „NFA“ akzeptiert ein Eingabewort $w \in \{a, b\}^*$ **genau dann**, wenn es im Automatengraphen **mindestens einen Weg gibt**,

- der im Startzustand  beginnt,
- dessen Kanten mit w beschriftet sind,
- und der in einem akzeptierenden Zustand  endet.



Der „NFA“ akzeptiert

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

NFAs: Die formale Definition

Ein nichtdeterministischer endlicher Automat (kurz: **NFA**)

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

besteht aus:

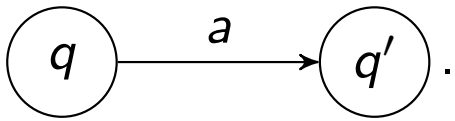
- einer endlichen Menge Σ, dem Eingabealphabet,
- einer endlichen Menge Q, der Zustandsmenge,
- einer Funktion $\delta\colon Q \times \Sigma \to \mathcal{P}(Q)$, der Übergangsfunktion,
  - *die jedem Zustand $q \in Q$ und jedem Symbol $a \in \Sigma$ eine **Menge** $\delta(q, a)$ von möglichen Nachfolgezuständen zuordnet.*
  - *Möglicherweise ist $\delta(q, a) = \emptyset$: Dann „stürzt“ der Automat ab, wenn er im Zustand q ist und das Symbol a liest.*
- dem Startzustand $q_0 \in Q$ und
- einer Menge $F \subseteq Q$ von Endzuständen bzw. akzeptierenden Zuständen.

Der Automatengraph von NFAs

Es sei

- $q \in Q$ ein Zustand,
- $a \in \Sigma$ ein Eingabesymbol und
- $q' \in \delta(q, a)$ ein „möglicher Nachfolgezustand“.

Dann gibt es im Automatengraphen einen mit dem Symbol a beschrifteten Pfeil von Knoten q zu Knoten q', d.h.



.

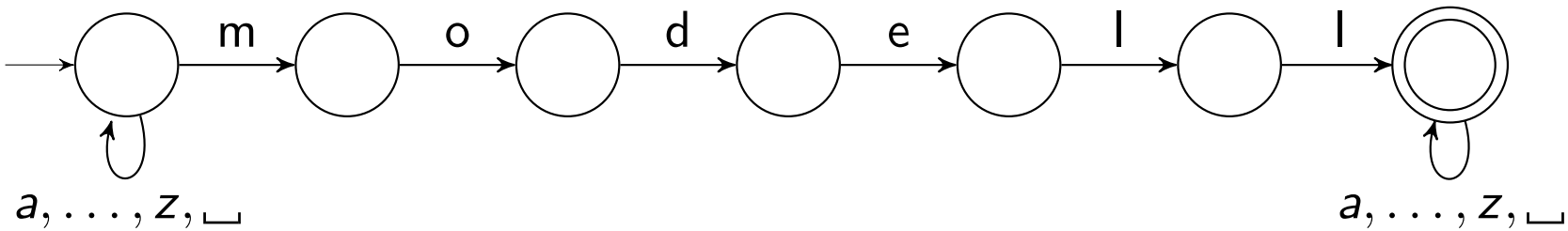
Stichwortsuche mit NFAs (1/2)

Gegeben: Ein Stichwort, z.B. „modell“

Eingabe: Ein Text, der aus den Buchstaben „a“ bis „z“ sowie dem Leerzeichen „$\sqcup$“ besteht

Frage: Kommt das Stichwort „modell“ irgendwo im Eingabetext vor?

Der Automatengraph eines NFAs, der dies bewerkstelligt:

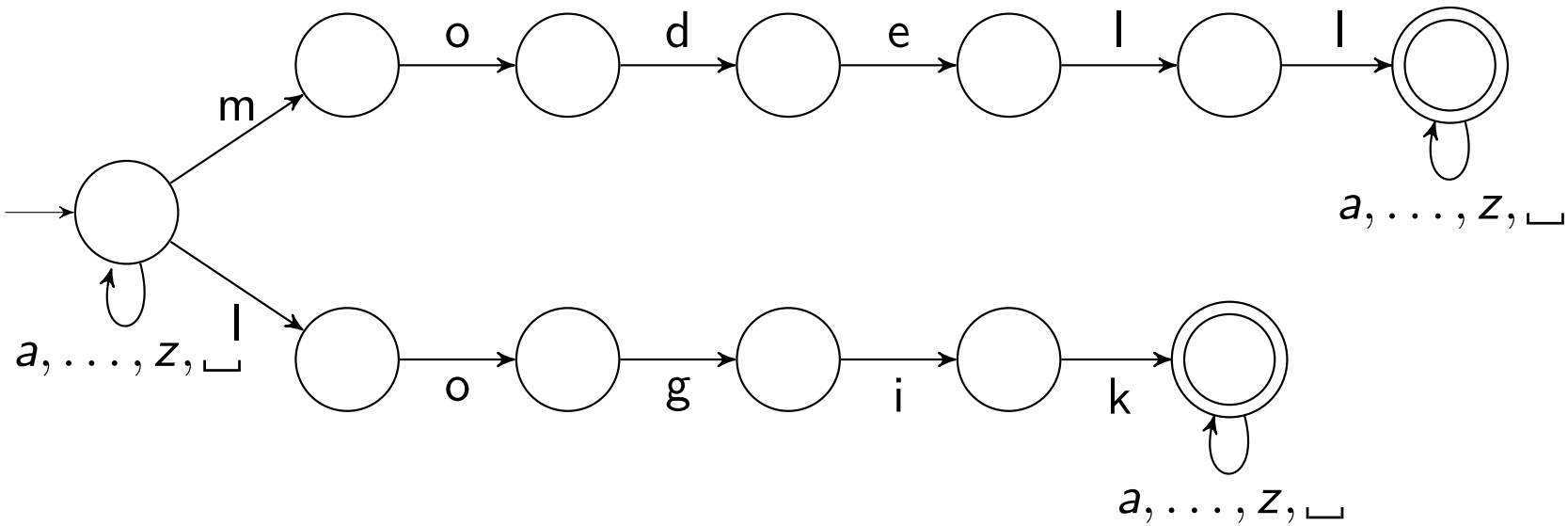


Stichwortsuche mit NFAs (2/2)

Auf ähnliche Art können auch Varianten dieser Stichwortsuche behandelt werden, zum Beispiel die Frage

> *Kommt mindestens eins der Stichworte „modell“ bzw. „logik“ im Eingabetext vor?*

Graphische Darstellung eines NFAs, der dies bewerkstelligt:

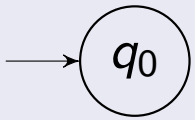


Die von einem NFA akzeptierte Sprache

Wann akzeptiert ein NFA?

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

(a) Sei $n \in \mathbb{N}$ und sei $w = a_1 \cdots a_n$ ein Eingabewort der Länge n.
Das Wort w wird ganau dann vom NFA A **akzeptiert**, wenn es im Automatengraphen von A einen im Startzustand



beginnenden Weg der Länge n gibt, dessen Kanten mit den Symbolen $a_1 \ldots a_n$ beschriftet sind und der in einem akzeptierenden Zustand endet.

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

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

Das ist keine „wirklich“ präzise Definition,
denn der Automatengraph soll doch nur unsere Intuition unterstützen.

Die erweiterte Übergangsfunktion

$\widehat{\delta}(q, w) :=$ die MENGE aller möglichen Zustände nach Lesen von w im „Startzustand“ q

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

$$\widehat{\delta} \ : \ Q \times \Sigma^* \ \to \ \mathcal{P}(Q)$$

ist rekursiv wie folgt definiert:

- **Rekursionsanfang**: F.a. $q \in Q$ ist $\widehat{\delta}(q, \varepsilon) \ := \{q\}$.
- **Rekursionsschritt**: F.a. $q \in Q$, $w \in \Sigma^*$ und $a \in \Sigma$ ist

$$\widehat{\delta}(q, wa) \ := \bigcup_{q' \in \widehat{\delta}(q,w)} \delta(q', a).$$

Ein möglicher Zustand q'' wird nach Lesen von wa genau dann erreicht, wenn nach Lesen von w (im Zustand q) ein Zustand q' erreicht wird und $q'' \in \delta(q', a)$ gilt:

$$q \xrightarrow{w} q' \xrightarrow{a} q''.$$

Wann genau akzeptiert denn nun ein NFA?

Der NFA $A = (\Sigma, Q, \delta, q_0, F)$ akzeptiert ein Wort w genau dann, wenn:

$$\widehat{\delta}(q_0, w) \cap F \neq \emptyset.$$

Somit ist

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

Äquivalenz von NFAs und DFAs

Können NFAs Sprachen akzeptieren, die DFAs nicht akzeptieren können?

NEIN!

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

$$L(A') = L(A).$$

D.h.: NFAs und DFAs akzeptieren genau dieselben Sprachen.

D.h. wir können Stichwortsuche auch mit DFAs durchführen?

- Natürlich und sogar mit wenigen Zuständen.

Aber im Allgemeinen sind die DFAs doch bestimmt sehr viel größer?!?

Die Potenzmengenkonstruktion (1/2)

Sei $A = (\Sigma, Q, \delta, q_0, F)$ der gegebene NFA.

Idee: Wir konstruieren einen DFA $A' = (\Sigma, Q', \delta', q_0', F')$, der in seinem aktuellen Zustand $q' \in Q'$

*die **Menge** aller Zustände abspeichert, in denen der Automat A in der aktuellen Situation sein **könnte**.*

Wir definieren die Komponenten von A' daher wie folgt:

- Eingabealphabet Σ,
- Zustandsmenge $Q' := \mathcal{P}(Q)$,
- Startzustand $q_0' := \{q_0\}$,
- Endzustandsmenge $F' := \{X \in Q' \ : \ X \cap F \neq \emptyset\}$,
- Übergangsfunktion $\delta' : Q' \times \Sigma \to Q'$, wobei für alle $X \in Q' = \mathcal{P}(Q)$ und alle $a \in \Sigma$ gilt:

$$\delta'(X, a) := \bigcup_{q \in X} \delta(q, a).$$

Die Potenzmengenkonstruktion (2/2)

Und wie zeigt man, dass A und A' dieselbe Sprache akzeptieren?

Zeige, dass

$$\widehat{\delta'}(\{q_0\}, w) \;=\; \widehat{\delta}(q_0, w)$$

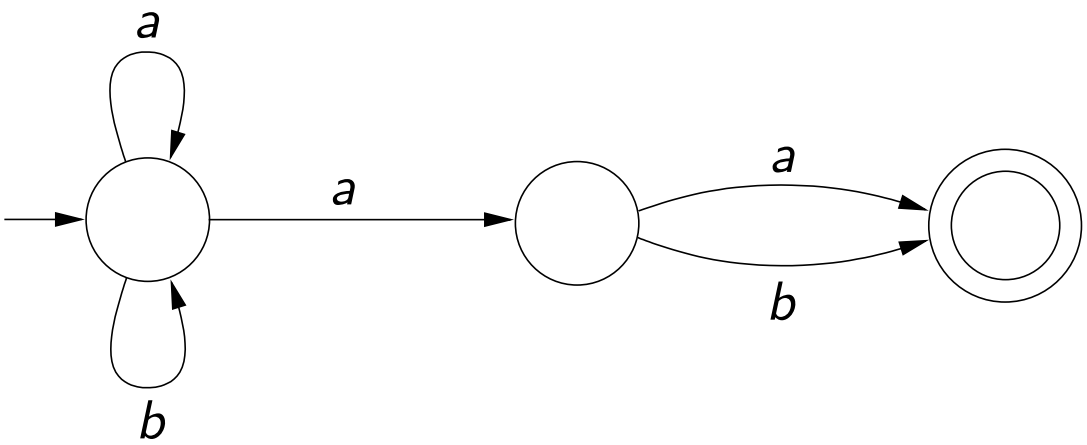
für jedes Wort $w \in \Sigma^*$ gilt.

Und wie, bitte schön, sollen wir das zeigen?

- Wir haben die erweiterten Übergangsfunktionen $\widehat{\delta}$ und $\widehat{\delta'}$ rekursiv definiert.
- Dann werden wir wohl eine vollständige Induktion nach $n = |w|$ ausführen!

Die Potenzmengenkonstruktion: Ein Beispiel (1/2)

Wie führt man die Potenzmengenkonstruktion für den NFA

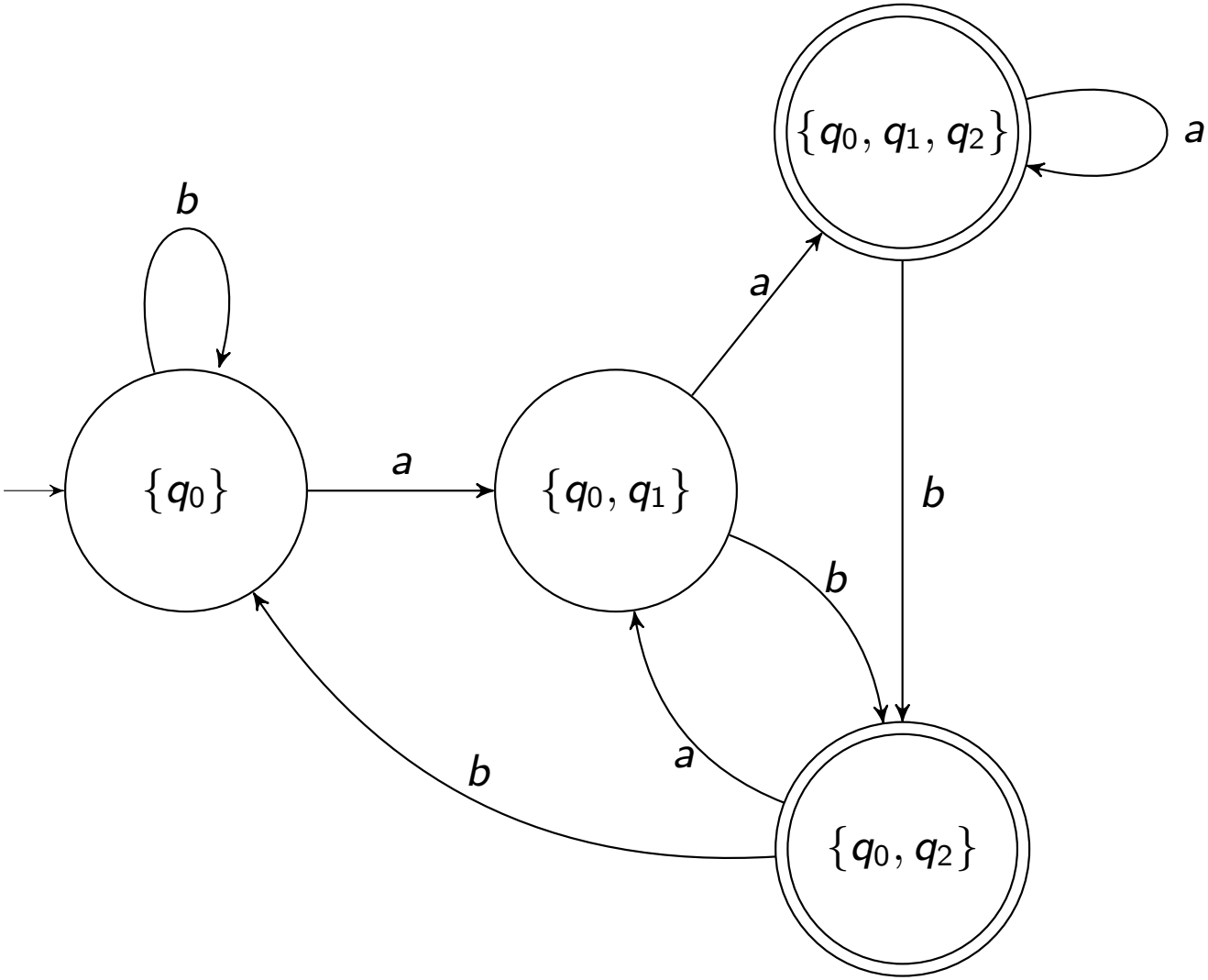


aus? Bezeichne die Zustände, von links nach rechts gelesen, mit q_0, q_1 und q_2.

Wichtig:

1. Beginne mit allen möglichen Nachfolgezuständen des Startzustands $q_0' := \{q_0\}$.
2. Definiere die Übergangsfunktion von A' nur für solche Zustände aus $\mathcal{P}(\{q_0, q_1, q_2\})$, die vom Startzustand q_0' aus erreicht werden können.

Die Potenzmengenkonstruktion: Ein Beispiel (2/2)



Reguläre Ausdrücke

Reguläre „Muster“

Sei Σ ein Alphabet.

1. Die Menge aller Worte über Σ haben wir mit

$$\Sigma^*$$

   beschrieben.
2. Sei u ein Wort über Σ. Dann wird die Menge aller Worte über Σ, die u als Teilwort besitzen, beschrieben durch

$$\Sigma^* \cdot u \cdot \Sigma^*.$$

3. Die Menge aller Worte über Σ, deren vorletzter Buchstabe ein a ist, wird beschrieben durch:

$$\Sigma^* \cdot a \cdot \Sigma$$

Reguläre Ausdrücke: Eine rekursive Definition der Syntax

Sei Σ ein endliches Alphabet.

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

Basisregeln:

- $\emptyset$ ist ein regulärer Ausdruck über Σ („leere Menge“).
- ε ist ein regulärer Ausdruck über Σ („leeres Wort“).
- Für jedes $a \in \Sigma$ gilt: a ist ein regulärer Ausdruck über Σ.

Rekursive Regeln:

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

Reguläre Ausdrücke: Syntax und Semantik

Wir haben gerade **formal definiert**,

„was ein regulärer Ausdruck R ist“

Aber was

„**bedeutet**“ R?

R sollte für eine Sprache stehen!

Reguläre Ausdrücke: Eine rekursive Definition der Semantik

Sei Σ ein endliches Alphabet.

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

Basisregeln:

- $L(\emptyset) := \emptyset$.
- $L(\varepsilon) := \{\varepsilon\}$.
- Für jedes $a \in \Sigma$ gilt: $L(a) := \{a\}$.

Rekursive Regeln:

- Ist R ein regulärer Ausdruck über Σ, so ist

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

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

Reguläre Ausdrücke: Vereinfachte Schreibweise

Zur vereinfachten Schreibweise und besseren Lesbarkeit regulärer Ausdrücke:

- Den „Punkt“ bei der Konkatenation

$$(R \cdot S)$$

  darf man weglassen.
- Bei Ketten gleichartiger Operatoren verzichten wir auf Klammern:

$$(R_1 \mid R_2 \mid R_3 \mid R_4) \quad \text{statt} \quad \Big(((R_1 \mid R_2) \mid R_3) \mid R_4\Big) \text{ und}$$

$$(R_1 R_2 R_3 R_4) \quad \text{statt} \quad \Big(((R_1 R_2) R_3) R_4\Big).$$

- „Präzedenzregeln“: * bindet stärker als $\cdot$
  $\cdot$ bindet stärker als $\mid$
- Äußere Klammern, die einen regulären Ausdruck umschließen, dürfen weggelassen werden.
- Zur besseren Lesbarkeit dürfen zusätzliche Klammern benutzt werden.

Reguläre Ausdrücke: Beispiele (1/3)

(a) $a \mid bc^*$ ist eine verkürzte Schreibweise für $(a \mid (b \cdot c^*))$.
Die von diesem regulären Ausdruck beschriebene Sprache $L = L(a \mid bc^*)$ ist

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

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

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

$$(a \mid b \mid c)^* abb\, (a \mid b \mid c)^*.$$

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

$$(a \mid b \mid c)^* a\, (\varepsilon \mid a \mid b \mid c).$$

Reguläre Ausdrücke: Beispiele (2/3)

Wir wollen einen regulären Ausdruck angeben, der alle Telefonnummern der Form

$$\textit{Vorwahl}/\textit{Nummer}$$

beschreibt, wobei

1. *Vorwahl* und *Nummer* nicht-leere Ziffernfolgen sind,
2. *Vorwahl* mit genau einer Null beginnt und *Nummer nicht* mit einer Null beginnt.

Der Ausdruck

$$0\ (1|2|\cdots|9)\ (0|1|\cdots|9)^*\ /\ (1|2|\cdots 9)\ (0|1|\cdots 9)^*$$

definiert die gewünschte Sprache.

Reguläre Ausdrücke: Beispiele (3/3)

Der Ausdruck:

$$R := \left(\varepsilon \mid 069/\right)\ \ 798\ \ \left(\varepsilon \mid \text{-}\right)\ \ \left(\ 0\ \ \mid (1|2|\cdots|9)\,(0|1|\cdots|9)^*\ \right)$$

definiert eine Sprache. Welche der folgenden Worte gehören zu R?

- ? 069/798-0
- ? 7980
- ? 069/798-028551.

Reguläre Sprachen

Reguläre Sprachen und reguläre Ausdrcke

Jeder reguläre Ausdruck R definiert eine reguläre Sprache.

Beweis durch Induktion über den Aufbau von R: Siehe Tafel.

Die Klasse der regulären Sprache ist ein fundamentales Konzept mit verschiedensten Sichtweisen, denn

DFAs, NFAs oder reguläre Ausdrücke

definieren dieselben Sprachen!

Dieses Ergebnis wird in der Veranstaltung „**Theoretische Informatik 2**“ gezeigt. Insbesondere besitzt also jede reguläre Sprache einen regulären Ausdruck!

Reguläre Sprachen: Ein Ausblick

In der Veranstaltung „**Theoretische Informatik 2**“ wird unter Anderem gezeigt:

(a) dass auch „**Zweiweg-Automaten**“ und „**reguläre Grammatiken**“ die Klasse der regulären Sprachen definieren,

(b) und dass – unter bestimmten Umständen – auch **würfelnde Automaten** genau die Klasse der regulären Sprachen beschreiben,

(c) dass viele Entscheidungsfragen wie

- akzeptieren zwei DFAs dieselbe Sprache?
- akzeptiert ein NFA mindestens ein Wort?

effizient beantwortet werden können, andere hingegen, wie etwa

- akzeptieren zwei NFAs dieselbe Sprache?
- akzeptiert ein NFA alle Worte eines Alphabets?

viel zu schwierig sind!