Kapitel:

Endliche Automaten & reguläre Sprachen

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 bzw. Repräsentation von Sprachen (sog. reguläre Sprachen)
- in der Entwicklung digitaler Schaltungen
- in der Softwaretechnik (zum Beispiel in der Modellierung des Applikationsverhaltens)
- in der Compilierung: Lexikalische Analyse
- im Algorithmenentwurf für String Probleme
- in der Abstraktion tatsächlicher Automaten (wie Bank- und Getränkeautomaten)

Die Komponenten eines endlichen Automaten (DFA)

- das Eingabealphabet Σ:
  Beschreibung der atomaren Umgebungseinflüsse,
- die endliche Menge Q der Zustände,
- der Anfangszustand $q_0 \in Q$,
- die Menge $F \subseteq Q$ der akzeptierenden Zustände,
- das Programm bzw. die Zustandsüberführungsfunktion δ, die eine partielle Funktion von $Q \times \Sigma$ nach Q ist. Der Automat heißt vollständig, falls δ eine totale Funktion $\delta : Q \times \Sigma \to Q$ ist.

Wie "rechnet" ein endlicher Automat A?

1. A beginnt im Zustand q_0 und
2. liest die Eingabe Buchstabe für Buchstabe und bewirkt Zustandsänderungen (oder stürzt ab).
3. Die Eingabe wird akzeptiert, wenn der Automat nicht abstürzt und der zuletzt erhaltene Zustand akzeptierend ist.

Endliche Automaten: Die Definition I

Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein endlicher Automat (DFA).

Das Programm δ wird von Buchstaben auf Worte in Σ^* fortgesetzt:

$\delta(q, w_1 \cdots w_n)$ ist der Zustand, den A nach Lesen der Eingabe $w_1 \cdots w_n$ erreicht, wenn im Zustand q begonnen wird — oder das Symbol $\perp$, falls A beim Verarbeiten von $w_1 \cdots w_n$ abstürzt.

Präzise: Sei $Q_\perp := Q \dot\cup \{\perp\}$. Definiere $\hat\delta : Q_\perp \times \Sigma^* \to Q_\perp$ wie folgt:

- Für alle $w \in \Sigma^*$ ist $\hat\delta(\perp, w) := \perp$.
- Für alle $q \in Q$ ist $\hat\delta(q, \varepsilon) := q$.
- Für alle $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$ und für $q' := \hat\delta(q, w)$ ist

$$\hat\delta(q, wa) := \begin{cases} \perp & \text{falls } q' = \perp \\ \delta(q', a) & \text{sonst.} \end{cases}$$

An Stelle von $\hat\delta$ schreiben wir im Folgenden auch kurz δ.

Endliche Automaten: Die Definition II

Was ist die Sprache eines DFA A?

- A akzeptiert $w \in \Sigma^*$ genau dann, wenn $\delta(q_0, w) \in F$.
- $$L(A) = \{w \in \Sigma^* \mid A \text{ akzeptiert } w\}$$
  ist die von A akzeptierte (oder erkannte) Sprache.

Eine Sprache $L \subseteq \Sigma^*$ heißt regulär, wenn es einen DFA A mit $L = L(A)$ gibt.

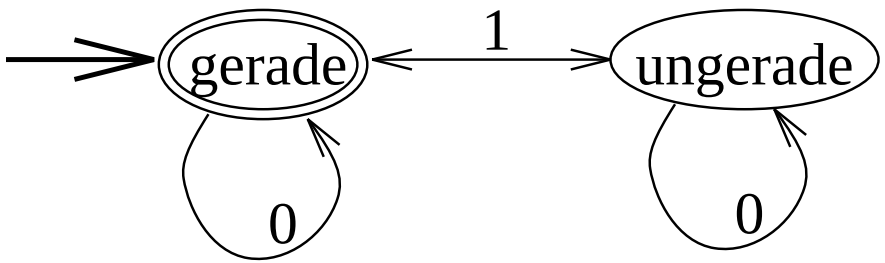
Graphische Darstellung: Zustandsdiagramme

- Die Knoten des Diagramms entsprechen den Zuständen.
- Für jeden Befehl $\delta(q, a) = q'$ wird eine mit a beschriftete Kante eingesetzt, die *vom* Knoten q *zum* Knoten q' verläuft.
- Der Anfangszustand wird durch einen Pfeil hervorgehoben, akzeptierende Zustände werden doppelt umrandet.

Die Sprache

$$\text{Parität} = \left\{ w \in \{0,1\}^* \mid w \text{ hat gerade viele Einsen} \right\}$$

ist regulär. Warum?

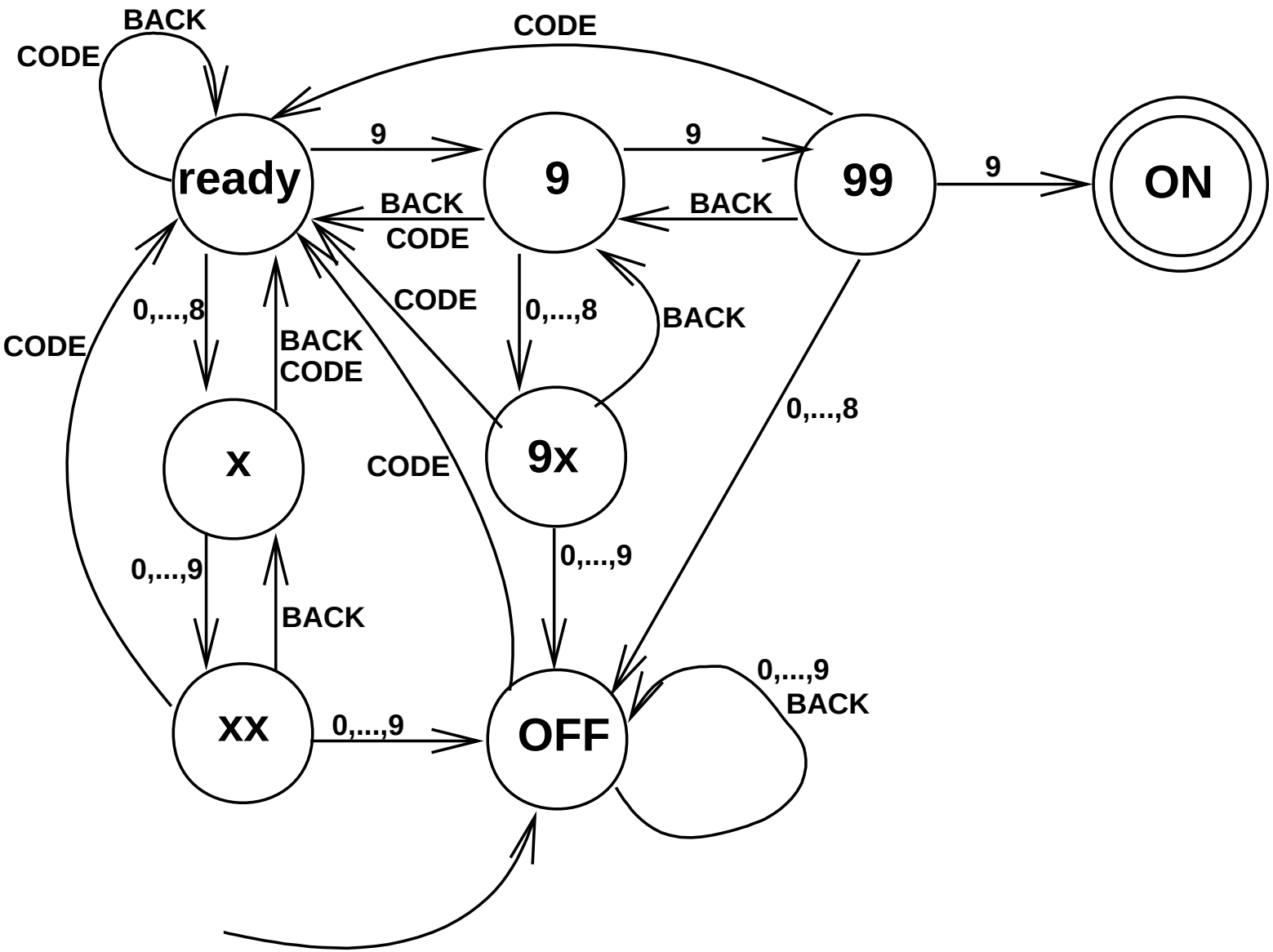


Beispiel: Freischaltung eines Fernsehers

- 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: Der Automat



Mealy Automaten

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

Mealy Automaten sind wie vollständige DFAs aufgebaut, besitzen aber zusätzlich

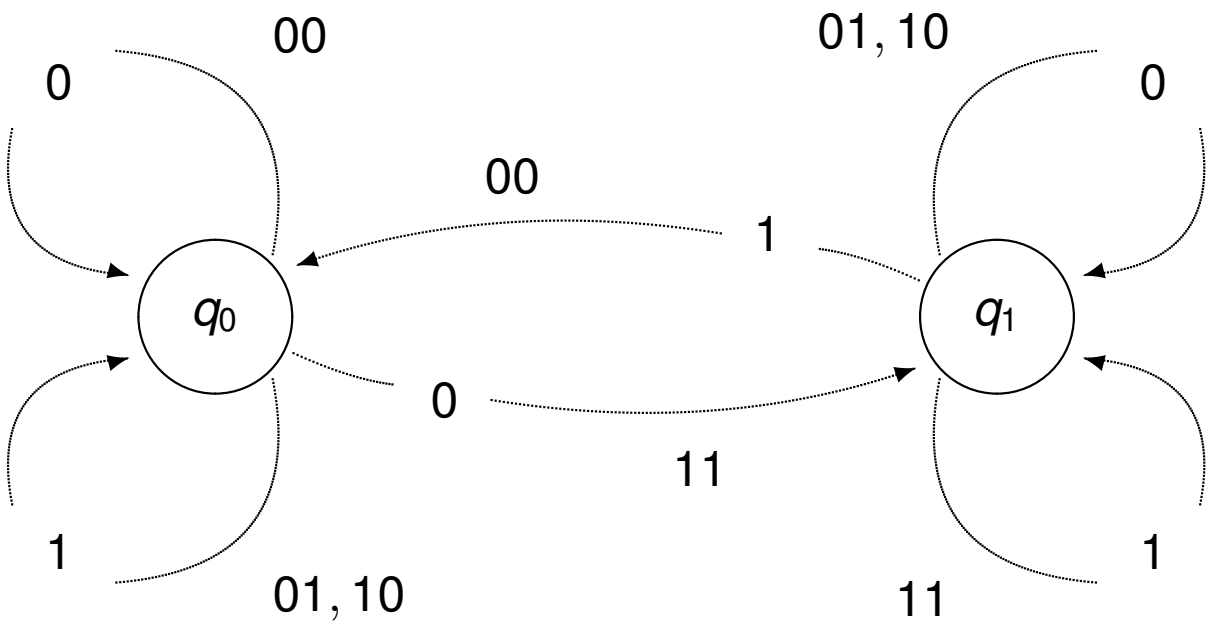
- ein Ausgabealphabet Ω und
- eine Ausgabefunktion $\lambda : Q \times \Sigma \to \Omega$.
- Ein Mealy Automat produziert für jeden Zustand und für jeden gelesenen Buchstaben eine Ausgabe.

Was kann man mit Mealy Automaten anstellen?

Beispiel: 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 $[x_0y_0][x_1y_1] \cdots [x_ny_n][00]$.

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



Lexikalische Analyse

Zu Beginn liest der Compiler Anweisung nach Anweisung und bricht Anweisungen in Tokenklassen auf.

Betrachte zum Beispiel die Anweisung

```
if distance >= rate * (end - start)
then distance = maxdistance;
```

mit den Tokenklassen:

- Keywords (für `if` und `then`),
- Variablen (für `distance`, `rate`, `end`, `start`, `maxdistance`),
- Operatoren (für `*`, `-`, `=`) und Vergleichsoperatoren (für `>=`),
- Klammern (für `(` und `)`) und Semikolon (für `;`).

Die lexikalische Analyse benutzt reguläre Grammatiken, bzw nichtdeterminische endliche Automaten.

Nichtdeterministische Automaten (NFAs)

Nichtdeterministische Automaten (NFAs)

Nichtdeterminismus kann die Modellierung vereinfachen!

Angenommen, wir befinden uns im Zustand q und lesen den Buchstaben $a \in \Sigma$.

- Ein deterministischer Automat legt den Nachfolgezustand $\delta(q, a)$ eindeutig fest.

  $\delta : Q \times \Sigma \to Q$ ist eine (partielle) Funktion.

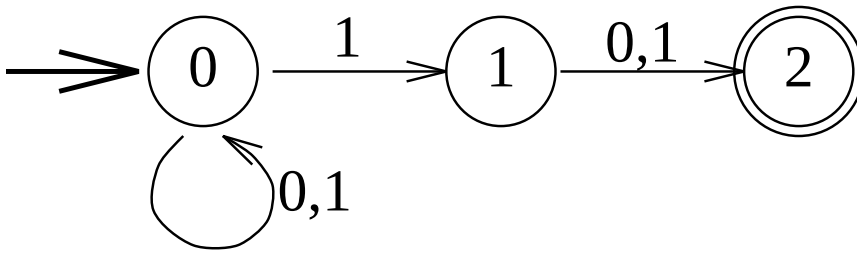
- Ein nichtdeterministischer Automat hingegen hat möglicherweise viele Optionen.

  $\delta(q, a)$ ist eine Teilmenge von Q: Der Automat darf mit jedem beliebigen Zustand der Menge $\delta(q, a)$ weiterrechnen!

Ein nichtdeterministischer Automat besitzt für jede Eingabe also unter Umständen viele Berechnungen, einige sind „erfolgreich“, andere nicht.

Ein Beispiel für einen NFA

Der nichtdeterministische Automat N



akzeptiert die Sprache

$$L(N) = \{0,1\}^* \cdot \{1\} \cdot \{0,1\}$$

aller Worte über dem Alphabet $\{0,1\}$, deren vorletzter Buchstabe eine 1 ist.

Die Komponenten eines NFAs

- die Zustandsmenge Q,
- das Eingabealphabet Σ,
- der Anfangszustand $q_0 \in Q$,
- die Menge $F \subseteq Q$ der akzeptierenden Zustände,
- das Programm δ, das wir als Funktion

$$\delta \colon Q \times \Sigma \to \mathcal{P}(Q)$$

auffassen, wobei $\mathcal{P}(Q)$ die Potenzmenge von Q bezeichnet.

Nur die Form des Programms ist neu: Jedem Zustand und gelesenen Buchstaben ist eine Menge von Nachfolgezuständen zugeordnet.

Wie rechnet ein nichtdeterministischer Automat N?

- N hat möglicherweise viele Berechnungen auf einer Eingabe w,
  - einige sind akzeptierend,
  - andere nicht.
- Die Philosophie:
  - Ein NFA rät und das kann schon mal daneben gehen.
  - Wir wollen erfolgreiche Berechnungen belohnen und wir übersetzen „erfolgreich“ als „akzeptierend“

Akzeptiere w, wenn mindestens eine Berechnung in einen akzeptierenden Zustand führt.

Wie rechnet ein nichtdeterministischer Automat N?

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

Das Programm δ wird von Buchstaben auf Worte in Σ^* fortgesetzt:

$\delta(q, w_1 \cdots w_n)$ ist die Menge der Zustände, in denen N nach Lesen der Eingabe $w_1 \cdots w_n$ sein kann, wenn im Zustand q begonnen wird.

Präzise ist die Fortsetzung $\delta : Q \times \Sigma^* \to \mathcal{P}(Q)$ von Buchstaben auf Worte rekursiv wie folgt definiert:

$$\delta(q, \epsilon) = \{q\} \qquad \text{und} \qquad \delta(q, wa) = \bigcup_{p \in \delta(q,w)} \delta(p, a).$$

$\delta(q, wa)$ ist die Menge der von Zustand q aus erreichbaren Zustände, wenn wa die Eingabe ist.

Definition

N akzeptiert w genau dann, wenn $\delta(q_0, w) \cap F \neq \emptyset$.

$L(N) = \{w \in \Sigma^* \mid N \text{ akzeptiert } w\}$ ist die von N akzeptierte (oder erkannte) Sprache.

Die Potenzmengenkonstruktion I

Akzeptieren nichtdeterministische Automaten aber möglicherweise nicht-reguläre Sprachen? — Nein!

Für jeden nichtdeterministischen Automaten N bauen wir einen äquivalenten deterministischen Automaten D:

- Sei $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$.
- Idee: Wir lassen D die Menge $\delta_N(q_0, w)$ der von q_0 aus erreichbaren Zustände berechnen!
- Und wie soll das gehen?
  - Die Teilmengen von Q_N sind die Zustände von D.
  - Wir müssen das deterministische Programm δ_D so definieren, dass der Zustand von D auf Eingabe w mit der Menge $\delta_N(q_0, w)$ übereinstimmt.

Die Potenzmengenkonstruktion II

Sei N ein NFA mit Komponenten Q_N, Σ, δ_N, q_0 und F_N.

Wir definieren einen äquivalenten deterministischen Automaten D mit:

- $Q_D = \{t \mid t \subseteq Q_N\} = \mathcal{P}(Q_N)$,
- Anfangszustand $q_0' = \{q_0\}$,
- akzeptierende Zustände $F_D = \{t \in Q_D \mid t \text{ enthält einen Zustand aus } F_N\}$
- $\delta_D(t, a) = \left\{p \in Q_N \mid \text{Es gibt } q \in t \text{ mit } p \in \delta_N(q, a)\right\} = \bigcup_{q \in t} \delta_N(q, a)$.

Dann sind D und N äquivalent, das heißt es gilt $L(D) = L(N)$.

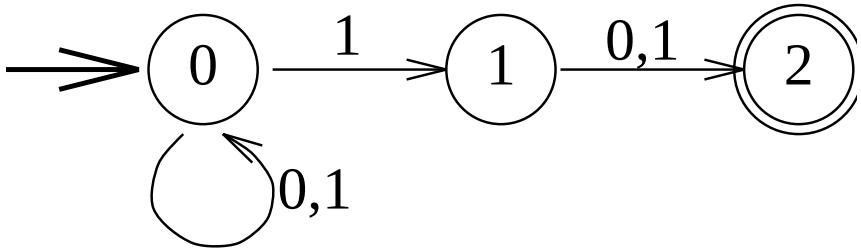
Beweis: Per Induktion nach n zeigen wir, dass für alle $n \in \mathbb{N}$ und alle $w \in \Sigma^n$ gilt: $\delta_D(q_0', w) = \delta_N(q_0, w)$.

– siehe Tafel –

Die Potenzmengenkonstruktion: Ein Beispiel

Beginne die Potenzmengenkonstruktion stets mit dem Startzustand $\{q_0\}$ und betrachte im Folgenden nur die von $\{q_0\}$ aus erreichbaren Zustände.

Der NFA N



akzeptiert die Sprache $L(N) = \{0,1\}^* \cdot \{1\} \cdot \{0,1\}$.

Führe die Potenzmengenkonstruktion aus.
Wie sieht der resultierende DFA aus?

– siehe Tafel –

Das Pumping Lemma

Methoden zum Nachweis von Nicht-Regularität

Wie zeigt man, dass eine Sprache L nicht regulär ist?

(1) Man zeigt, dass Index(L) unendlich ist.

(2) Oder man wendet das Pumping Lemma an. Idee:

- Angenommen, A ist ein Automat mit $|Q|$ Zuständen.
- Die Zustandsübergänge von A auf einer Eingabe $x = x_1 \cdots x_n$:

$$q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} q_2 \xrightarrow{x_3} \cdots \xrightarrow{x_{n-1}} q_{n-1} \xrightarrow{x_n} q_n.$$

- Wenn $n \geqslant |Q|$, dann werden dabei $n+1 > |Q|$ Zustände durchlaufen. Es muss daher (mind.) einen Zustand q geben, der (mind.) zweimal durchlaufen wird.

  Skizze: Es gibt Zahlen $0 \leqslant j < k \leqslant |Q|$, so dass

$$q_0 \xrightarrow{x_1 \cdots x_j} q \xrightarrow{x_{j+1} \cdots x_k} q \xrightarrow{x_{k+1} \cdots x_n} q_n.$$

Vorarbeiten zum Pumping Lemma

Verarbeitung eines Worts $z = z_1 \cdots z_n$ durch Automat A:

Wenn $n \geqslant |Q|$, dann $\quad q_0 \overset{x_1 \cdots x_j}{\longrightarrow} q \overset{x_{j+1} \cdots x_k}{\longrightarrow} q \overset{x_{k+1} \cdots x_n}{\longrightarrow} q_n.$

Und wenn $q_n \in F$, dann $x \in L(A)$ und

$$\begin{aligned}
(x_1 \cdots x_j) \cdot (x_{k+1} \cdots x_n) &\in L(A) \quad \text{und} \\
(x_1 \cdots x_j) \cdot (x_{j+1} \cdots x_k) \cdot (x_{k+1} \cdots x_n) &\in L(A) \quad \text{und} \\
(x_1 \cdots x_j) \cdot (x_{j+1} \cdots x_k)^2 \cdot (x_{k+1} \cdots x_n) &\in L(A) \quad \text{und} \\
(x_1 \cdots x_j) \cdot (x_{j+1} \cdots x_k)^3 \cdot (x_{k+1} \cdots x_n) &\in L(A) \quad \ldots
\end{aligned}$$

Damit haben wir Folgendes bewiesen:

Das Pumping Lemma

Sei L eine reguläre Sprache.

- Dann gibt es eine **Pumpingkonstante** $z \geqslant 1$, so dass
- jedes Wort $x \in L$ mit $|x| \geqslant z$
- eine Zerlegung mit den folgenden Eigenschaften besitzt:
  - $x = uvw, \quad |uv| \leqslant z, \quad |v| \geqslant 1, \quad$ und
  - $uv^i w \in L$ für jedes $i \geqslant 0$.

Fazit: Wenn Worte der Sprache lang genug sind (also $|x| \geqslant z$), dann gibt es ein nicht-leeres Teilwort v, (also $v = x_{j+1} \cdots x_k$) das

- "aufgepumpt" ($i \geqslant 1$)
- und "abgepumpt" ($i = 0$) werden kann.

Kennen wir die Pumpingkonstante z? — Ist A ein NFA für L, so können wir $z = |Q|$ wählen. Aber zu gegebenem L ist z zunächst nicht bekannt!

Kennen wir die Zerlegung von x in uvw? — NEIN! Wir wissen nur, dass $|uv| \leqslant z$ und $|v| \geqslant 1$ ist.

Anwendung des Pumping Lemmas: Die Spielregeln

Wie kann man das Pumping Lemma nutzen, um zu zeigen, dass eine Sprache L nicht regulär ist?

(1) Für jede mögliche Pumpingkonstante $z \geqslant 1$

(2) müssen wir ein Wort $x \in L$ mit $|x| \geqslant z$ konstruieren,

(3) so dass für jede mögliche Zerlegung $x = uvw$ mit $|uv| \leqslant z$ und $|v| \geqslant 1$

$$uv^i w \notin L$$

für mindestens ein $i \geqslant 0$ gilt.

Der "Gegner" kontrolliert die **Pumpingkonstante** z vollständig und die **Zerlegung** $x = uvw$ teilweise, denn er muss $|uv| \leqslant z$ und $|v| \geqslant 1$ garantieren.

Das Pumping Lemma: Ein Anwendungsbeispiel

Beispiel:

Die Palindromsprache

$$L = \left\{ w \in \{0,1\}^* \mid w \text{ ist ein Palindrom} \right\}$$

ist nicht regulär.

Beweis: Angenommen, L ist doch regulär.
Gemäß Pumping Lemma gibt es dann eine Pumpingkonstante $z \geqslant 1$, so dass jedes Wort $x \in L$ mit $|x| \geqslant z$ eine Zerlegung in $x = uvw$ besitzt, so dass gilt:

$$(*): \qquad |uv| \leqslant z \quad \text{und} \quad |v| \geqslant 1 \quad \text{und} \quad uv^i w \in L \text{ für alle } i \geqslant 0.$$

Betrachte insbesondere das Wort $x := 0^z 1 0^z$. Es gilt: $x \in L$ und $|x| \geqslant z$. Gemäß Pumping Lemma gibt es also eine Zerlegung $x = uvw$, so dass $(*)$ gilt.
Wegen $uvw = x = 0^z 1 0^z$ und $|uv| \leqslant z$ und $|v| \geqslant 1$ gibt es eine Zahl $k \geqslant 1$, so dass $v = 0^k$.
Aber dann ist $uv^2 w = 0^{z+k} 1 0^z$ kein Palindrom, obwohl gemäß Pumping Lemma dieses Wort in L liegen müsste. WIDERSPRUCH! □

Grenzen des Pumping Lemmas

Es gibt Sprachen, die nicht regulär sind, deren Nicht-Regularität mit dem Pumping Lemma aber nicht nachgewiesen werden kann.

Beispiel: Die Sprache $L = \{a,b\}^* \cup \{c^k\, a^n\, b^n : k,n \geqslant 0\}$

ist nicht regulär, erfüllt aber die Aussage des Pumping Lemmas. D.h:

Es gibt eine Pumpingkonstante $z \geqslant 1$ (nämlich $z = 1$), so dass

- jedes Wort $x \in L$ mit $|x| \geqslant z$
- eine Zerlegung mit den folgenden Eigenschaften besitzt:
  - $x = uvw, \quad |uv| \leqslant z, \quad |v| \geqslant 1, \quad$ und
  - $uv^i w \in L$ für jedes $i \geqslant 0$.

Beweis: Siehe Tafel!

Der Fahrplan für den Rest dieses Kapitels

- Wie minimiert man deterministische endliche Automaten (DFAs)?
- Wie stellt man fest, ob eine Sprache regulär ist?
- Wie vergleichen sich DFAs, NFAs, reguläre Ausdrücke und reguläre Grammatiken?
- Welche — endliche Automaten betreffende — Fragen lassen sich effizient beantworten?
  - Akzeptiert ein DFA **kein** Wort? Einfach!
  - Sind zwei endliche DFAs äquivalent? Machbar!
  - Akzeptiert ein NFA alle Worte? Schwer, aber machbar!

Wir beginnen mit dem Problem der Minimierung von DFAs.

Minimierung von DFAs

Ein Beispiel: Die reguläre Sprache $\{a, b\}^* \cdot \{ab\}$

Wie stellt man fest, ob ein Wort das Suffix *ab* besitzt?

Ein erster Ansatz:

> Speichere im aktuellen Zustand die beiden zuletzt gelesenen Buchstaben.

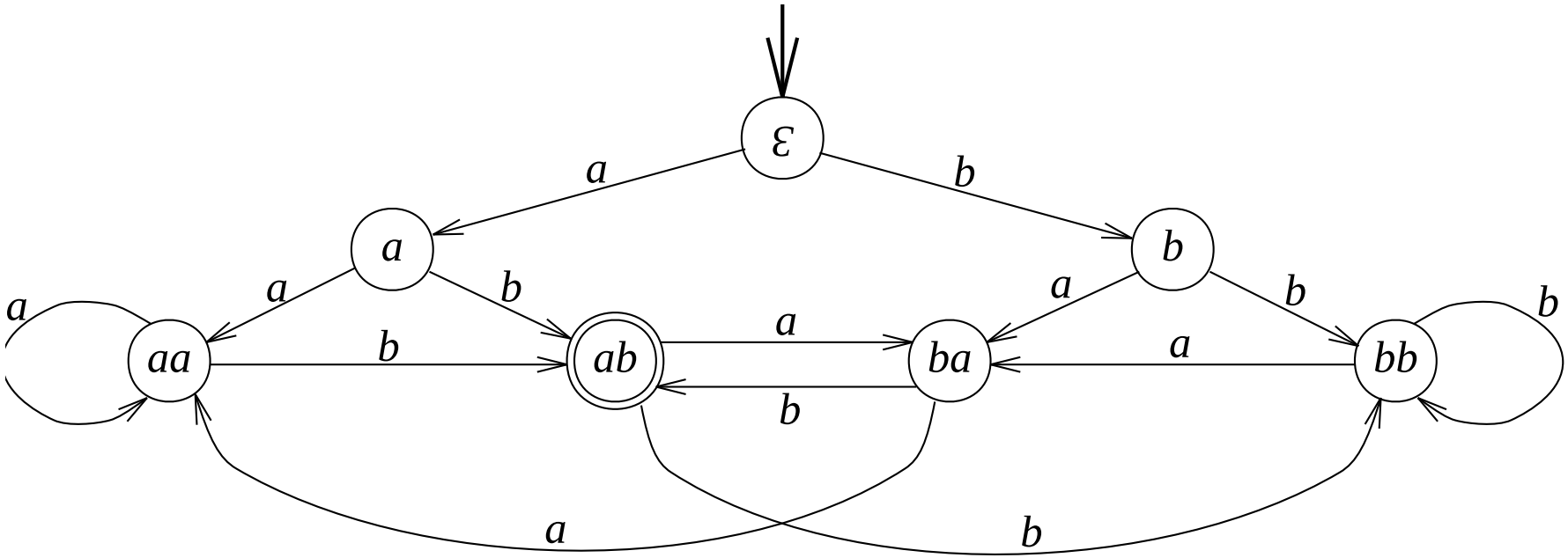
- Wir benutzen deshalb die Zustände

$$Q = \{\epsilon, a, b, aa, ab, ba, bb\}$$

- mit Startzustand $q_0 = \epsilon$ (“wir haben noch nichts gelesen”)
- und dem akzeptierenden Zustand *ab*
  (“die beiden letzten Buchstaben sind *ab*”).

Welche Zustandsübergänge?

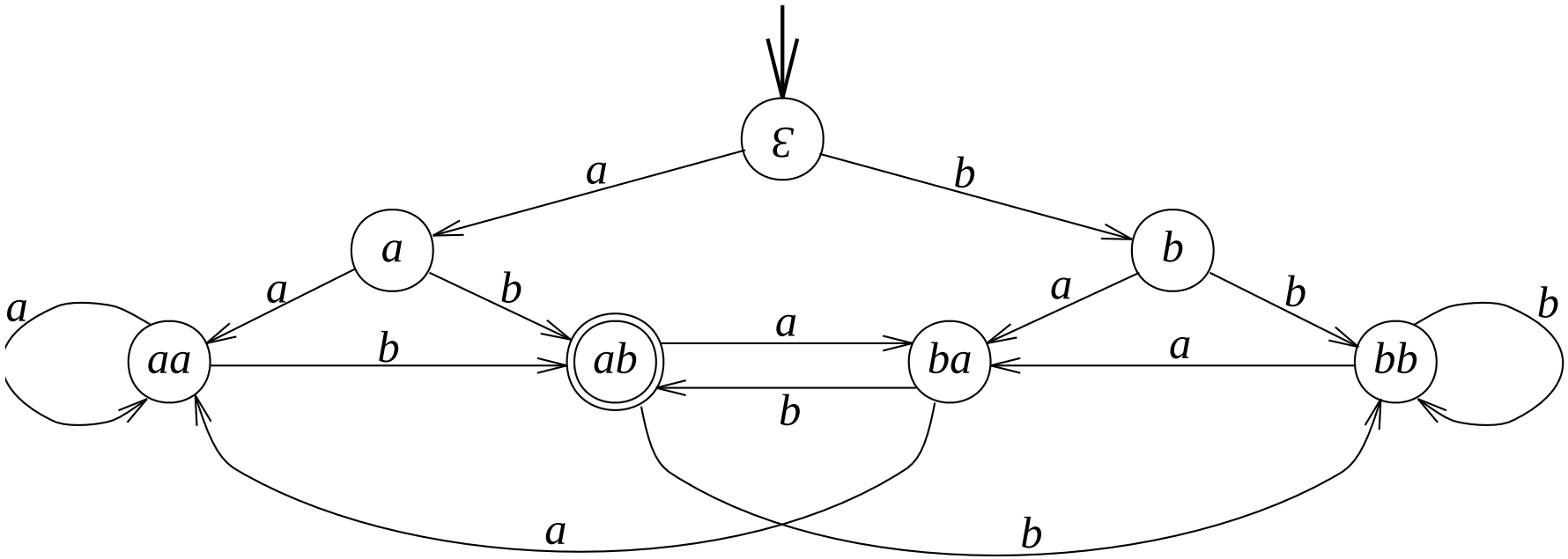
$\{a, b\}^* \cdot \{ab\}$: 1. Versuch



Der Automat merkt sich tatsächlich die beiden letzten Buchstaben, denn:

$$\delta(\epsilon, w) = z \iff (|w| \leqslant 1 \text{ und } w = z) \text{ oder } (|z| = 2 \text{ und } w = w'z)$$

$\{a, b\}^* \cdot \{ab\}$: 1. Versuch

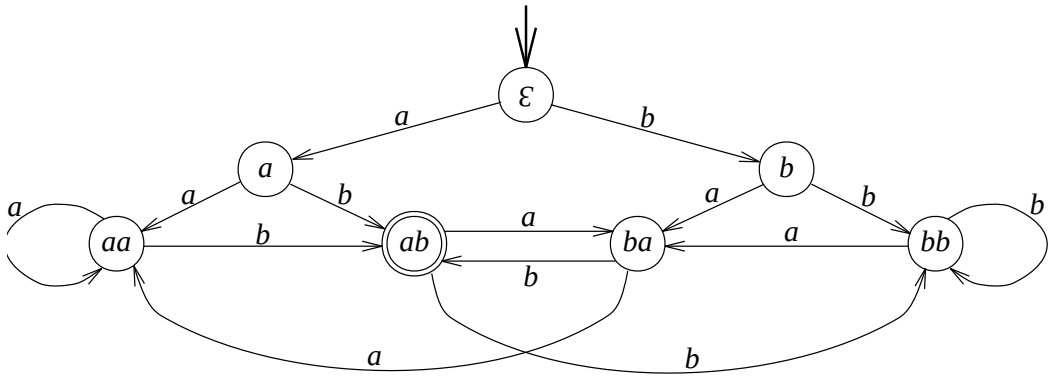


Ist der Automat minimal?

Sicherlich nicht: Wenn *a* der letzte Buchstabe ist,
dann ist der vorletzte Buchstabe uninteressant!

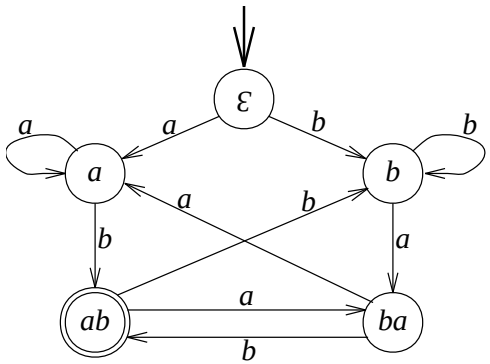
$\{a, b\}^* \cdot \{ab\}$: 2. Versuch

Vorher:

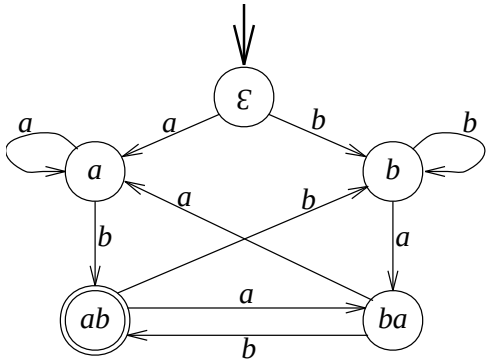


Verschmelze die Zustände *a* und *aa*: Beide Zustände erreichen unter *a* wie auch *b* identische Nachfolgezustände.
Verschmelze *b* und *bb* aus dem gleichen Grund.

Nachher:

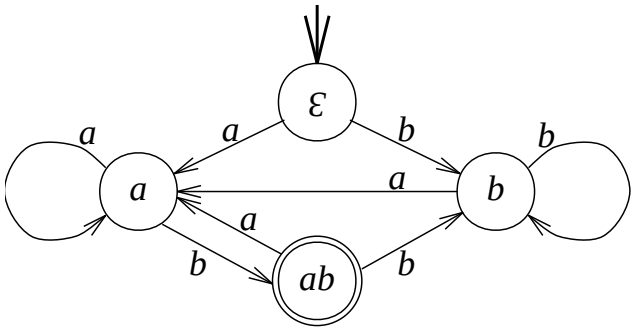


$\{a, b\}^* \cdot \{ab\}$: 3. Versuch

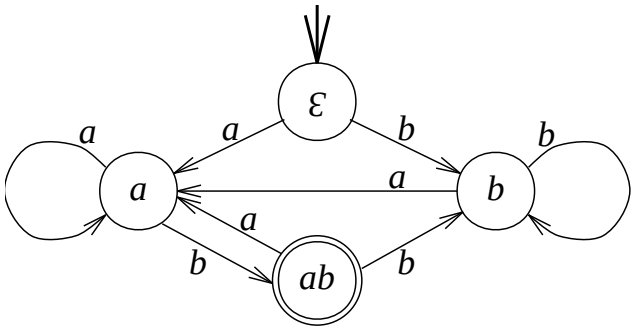


Der Automat ist immer noch nicht minimal, da die Zustände *a* und *ba* identische Nachfolgezustände haben.
(Wenn *a* der letzte Buchstabe ist, dann ist der vorletzte egal.)

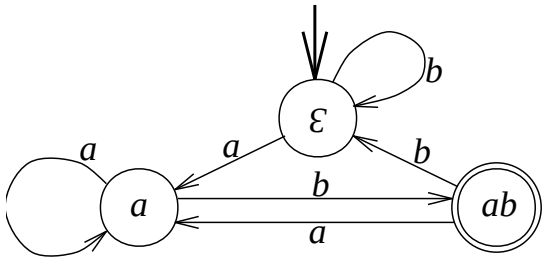
Nach der Verschmelzung von *a* und *ba*:



$\{a, b\}^* \cdot \{ab\}$: 4. Versuch



Der Automat ist immer noch nicht minimal, weil ϵ und b identische Nachfolgezustände besitzen. Verschmelze!



Frage:

Ist der Automat jetzt minimal?

Allgemeine Minimierung: Ein erster Versuch

Das Minimierungsproblem

Gegeben sei ein vollständiger DFA $A = (Q, \Sigma, \delta, q_0, F)$.
Ziel: Konstruiere einen äquivalenten vollständigen DFA mit minimaler Zustandszahl.

(1) Entferne alle überflüssigen Zustände q.

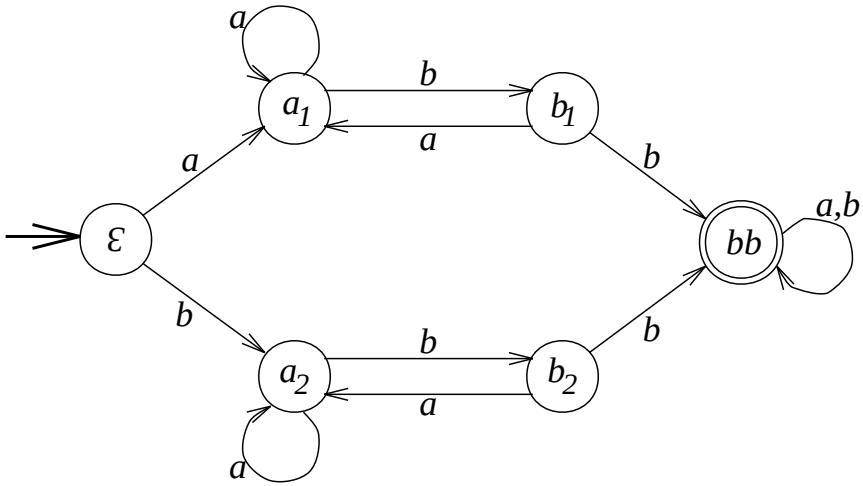
- q heißt "überflüssig", wenn q von q_0 aus nicht erreichbar ist.
- Wie erkennen wir überflüssige Zustände? Starte eine Breiten- oder Tiefensuche im Zustand q_0. Laufzeit: $O(|Q| \cdot |\Sigma|)$.

(2) Wir haben Zustände p und q verschmolzen, wenn sie identische Nachfolgezustände besitzen. Genauer:

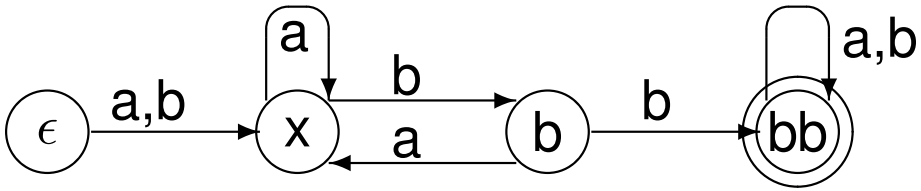
- Wir sagen, "p und q verhalten sich gleich" genau dann, wenn gilt:
  - $p, q \in F$ oder $p, q \notin F$, und
  - $\delta(p, a) = \delta(q, a)$ für alle $a \in \Sigma$.
- Verschmelze Zustände, die sich gleich verhalten.

Frage: Funktioniert das?

Leider, leider, . . .

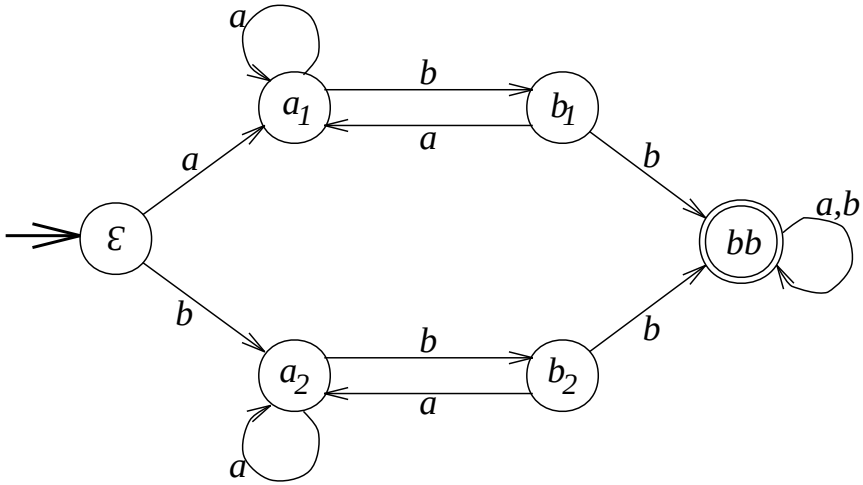


hat keine Zustände , die sich gleich verhalten, aber



akzeptiert dieselbe Sprache mit weniger Zuständen.

Woran liegt's?



- Wir erhalten den kleineren Automaten, wenn wir sowohl die Zustände a_1, a_2 als auch die Zustände b_1, b_2 verschmelzen.
- Warum dürfen wir das? Für a_1, bzw. a_2 als Anfangszustand wird die **gleiche** Sprache $\{a, b\}^*\{bb\}\{a, b\}^*$ akzeptiert!

a_1, a_2 wie auch b_1, b_2 unterscheiden sich nicht, wenn

wenn es um's Akzeptieren/Verwerfen geht!

Die Verschmelzungsrelation

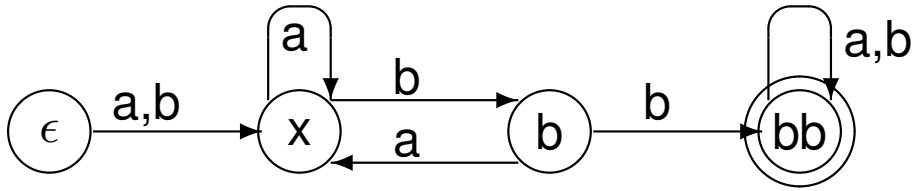
$A = (Q, \Sigma, \delta, q_0, F)$ sei ein DFA. Wir sagen, dass die Zustände $p, q \in Q$ **äquivalent** sind (kurz: $p \equiv_A q$) genau dann, wenn für alle $w \in \Sigma^*$ gilt:

$$\delta(p, w) \in F \quad \Longleftrightarrow \quad \delta(q, w) \in F.$$

$p, q \in Q$ sind also äquivalent, wenn p und q sich nicht unterscheiden, wenn es "ums Akzeptieren geht".

Ein Beispiel

Hat der folgende DFA A



äquivalente Zustände?

- $\epsilon \not\equiv_A \mathrm{x}$: Denn $\delta(\mathrm{x}, \mathrm{bb}) \in F$, aber $\delta(\epsilon, \mathrm{bb}) \notin F$.
- $\epsilon \not\equiv_A \mathrm{b}$: Denn $\delta(\epsilon, \mathrm{b}) \notin F$, aber $\delta(\mathrm{b}, \mathrm{b}) \in F$.
- $\mathrm{x} \not\equiv_A \mathrm{b}$: Denn $\delta(\mathrm{x}, \mathrm{b}) \notin F$, aber $\delta(\mathrm{b}, \mathrm{b}) \in F$.
- Der Zustand bb ist der einzige akzeptierende Zustand und ist deshalb zu keinem anderen Zustand äquivalent.
  Denn: $\delta(\mathrm{bb}, \varepsilon) \in F$, aber $\delta(\mathrm{q}, \varepsilon) \notin F$ für alle $\mathrm{q} \in \{\epsilon, \mathrm{x}, \mathrm{b}\}$.

Der DFA A besitzt keine äquivalenten Zustände.

Die Verschmelzungsrelation ist eine Äquivalenzrelation

Für jeden DFA A ist die Relation $\equiv_A$

- reflexiv, d.h. es ist $q \equiv_A q$ für alle Zustände q von A,
- symmetrisch, d.h. für alle Zustände p und q von A gilt $p \equiv_A q$ genau dann, wenn $q \equiv_A p$ gilt, und
- transitiv, d.h. für alle Zustände p, q und r folgt aus $p \equiv_A q$ und $q \equiv_A r$ stets $p \equiv_A r$.

Also ist $\equiv_A$ eine Äquivalenzrelation.

Diese zerlegt die Zustandsmenge in **disjunkte** Äquivalenzklassen.

Der Äquivalenzklassenautomat

Wie sieht der Automat nach dem Verschmelzen äquivalenter Zustände aus?

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

- 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 $q \in Q$, $a \in \Sigma$.

Die Wohldefiniertheit von δ'

Wir haben $\delta'([p]_A, a) := [\delta(p, a)]_A$ gesetzt.

Falls $p, q \in Q$ so dass $p \equiv_A q$ ist, so gilt: $[p]_A = [q]_A$.
Unsere Definition von δ' macht also nur dann Sinn (fachbegriff: "δ' ist wohldefiniert"), wenn gilt:

$$[\delta(p, a)]_A = [\delta(q, a)]_A \qquad \text{für alle } p, q \in Q \text{ mit } [p]_A = [q]_A.$$

Seien also $p, q \in Q$ mit $[p]_A = [q]_A$. Dann gilt:

$$\begin{aligned}
& p \equiv_A q \\
\Longrightarrow \quad & \text{für alle } w' \in \Sigma^* \text{ gilt: } \delta(p, w') \in F \iff \delta(q, w') \in F \\
\Longrightarrow \quad & \text{für alle } w \in \Sigma^* \text{ gilt: } \delta(p, aw) \in F \iff \delta(q, aw) \in F \\
\Longrightarrow \quad & \text{für alle } w \in \Sigma^* \text{ gilt: } \delta\big(\delta(p, a), w\big) \in F \iff \delta\big(\delta(q, a), w\big) \in F \\
\Longrightarrow \quad & \delta(p, a) \equiv_A \delta(q, a) \\
\Longrightarrow \quad & [\delta(p, a)]_A = [\delta(q, a)]_A.
\end{aligned}$$

Somit ist δ' tatsächlich wohldefiniert.

A' ist äquivalent zu A, d.h. es gilt: $L(A') = L(A)$ (1/3)

Wir zeigen nun, dass der Äquivalenzklassenautomat A' dieselbe Sprache akzeptiert wie A. Dazu gehen wir in drei Schritten vor:

- Schritt 1: Für alle $w \in \Sigma^*$ ist $\delta'([q_0]_A, w) = [\delta(q_0, w)]_A$.
- Schritt 2: $L(A) \subseteq L(A')$.
- Schritt 3: $L(A') \subseteq L(A)$.

Beweis von Schritt 1: Per Induktion über die Länge von w.

Induktionsanfang: Betrachte $w = \varepsilon$.
Gemäß Definition gilt: $\delta'([q_0]_A, \varepsilon) = [q_0]_A = [\delta(q_0, \varepsilon)]_A$.

Induktionsschritt: Betrachte $w = ua$ mit $u \in \Sigma^*$ und $a \in \Sigma$.

Ind.annahme: $\delta'([q_0]_A, u) = [\delta(q_0, u)]_A$. Beh.: $\delta'([q_0]_A, ua) = [\delta(q_0, ua)]_A$.

Beweis:

$$\begin{aligned} \delta'([q_0]_A, ua) &= \delta'\big(\delta'([q_0]_A, u), a\big) \overset{\text{Ind.ann.}}{=} \delta'\big([\delta(q_0, u)]_A, a\big) \\ &= \delta'([p]_A, a), \quad \text{für } p := \delta(q_0, u) \\ &\overset{\text{Def. } \delta'}{=} [\delta(p, a)]_A = [\delta\big(\delta(q_0, u), a\big)]_A = [\delta(q_0, ua)]_A. \end{aligned}$$

□ Schritt 1

A' ist äquivalent zu A, d.h. es gilt: $L(A') = L(A)$ (2/3)

Wir zeigen nun, dass der Äquivalenzklassenautomat A' dieselbe Sprache akzeptiert wie A. Dazu gehen wir in drei Schritten vor:

- Schritt 1: Für alle $w \in \Sigma^*$ ist $\delta'([q_0]_A, w) = [\delta(q_0, w)]_A$.
- Schritt 2: $L(A) \subseteq L(A')$.
- Schritt 3: $L(A') \subseteq L(A)$.

Beweis von Schritt 2: Behauptung: $L(A) \subseteq L(A')$.

Sei $w \in L(A)$.

Somit akzeptiert A die Eingabe w, d.h. es gilt: $p := \delta(q_0, w) \in F$.

Gemäß Schritt 1 gilt: $\delta'([q_0]_A, w) = [\delta(q_0, w)]_A = [p]_A$.

Gemäß Definition von F' gilt: $[p]_A \in F'$ (da $p \in F$).

Somit wird w von A' akzeptiert, d.h. es gilt $w \in L(A')$.

□Schritt 2

A' ist äquivalent zu A, d.h. es gilt: $L(A') = L(A)$ (3/3)

Wir zeigen nun, dass der Äquivalenzklassenautomat A' dieselbe Sprache akzeptiert wie A. Dazu gehen wir in drei Schritten vor:

- Schritt 1: Für alle $w \in \Sigma^*$ ist $\delta'([q_0]_A, w) = [\delta(q_0, w)]_A$.
- Schritt 2: $L(A) \subseteq L(A')$.
- Schritt 3: $L(A') \subseteq L(A)$.

Beweis von Schritt 3: Behauptung: $L(A') \subseteq L(A)$. Sei $w \in L(A')$.

Somit akzeptiert A' die Eingabe w, d.h. es gilt: $\delta'([q_0]_A, w) \in F'$.

Ziel: Zeige, dass $w \in L(A)$, d.h., dass $\delta(q_0, w) \in F$.

Gemäß Schritt 1 gilt: $\delta'([q_0]_A, w) = [\delta(q_0, w)]_A$.

Wegen $\delta'([q_0]_A, w) = [\delta(q_0, w)]_A \in F'$ muss es gemäß der Definition von F' ein $p \in F$ geben, so dass $p \equiv_A \delta(q_0, w)$.

Aus der Definition von $\equiv_A$ folgt: $\delta(p, \varepsilon) \in F \iff \delta\big(\delta(q_0, w), \varepsilon\big) \in F$.

Wegen $\delta(p, \varepsilon) = p \in F$ und $\delta\big(\delta(q_0, w), \varepsilon\big) = \delta(q_0, w)$ folgt daraus, dass $\delta(q_0, w) \in F$ ist, d.h. $w \in L(A)$.

□Schritt 3

Zwischenstand

+ Der Äquivalenzklassenautomat A' akzeptiert dieselbe Sprache wie der ursprüngliche Automat A.
? Die verbleibenden Fragen:
  (1) Ist A' tatsächlich minimal?
  (2) Wie kann A' effizient berechent werden?

Wie beginnen mit der effizienten Berechnung von A':

Hauptproblem:
Wie überprüft man effizient, ob zwei Zustände p, q äquivalent sind?

$$p \equiv_A q \overset{\text{Def}}{\iff} \text{für alle } w \in \Sigma^* \text{ gilt: } \delta(p, w) \in F \iff \delta(q, w) \in F.$$

Die Berechnung des Äquivalenzklassenautomaten (1/4)

Idee: Anstatt Äquivalenz nachzuweisen, finde Zeugen für die Nicht-Äquivalenz.
Ein Wort $w \in \Sigma^*$ heißt Zeuge für $p \not\equiv_A q$, wenn gilt:

(1) $\delta(p, w) \in F$ und $\delta(q, w) \notin F$ oder

(2) $\delta(p, w) \notin F$ und $\delta(q, w) \in F$.

Eine zentrale Beobachtung:

Wenn ein Wort w die Nicht-Äquivalenz von $p = \delta(p', a)$ und $q = \delta(q', a)$ bezeugt, dann sind auch p' und q' nicht-äquivalent, und das Wort aw ist ein Zeuge dafür.

Beweis: w bezeugt die Nicht-Äquivalenz von p und q. D.h.:

$$\delta(p, w) \in F \iff \delta(q, w) \notin F.$$

Wegen $p = \delta(p', a)$ und $q = \delta(q', a)$ gilt:

$$\delta(p, w) = \delta\big(\delta(p', a), w\big) = \delta(p', aw) \quad \text{und} \quad \delta(q, w) = \delta\big(\delta(q', a), w\big) = \delta(q', aw).$$

Somit: $\delta(p', aw) \in F \iff \delta(q', aw) \notin F$.

Also bezeugt aw, dass $p' \not\equiv_A q'$. □

Die Berechnung des Äquivalenzklassenautomaten (2/4)

Wie bestimmt man alle nicht-äquivalenten Paare?

Die Grundidee: Betrachte alle Paare $\{p, q\}$ von Zuständen mit $p \neq q$.

(1) Zuerst markiere $\{p, q\}$ genau dann als nicht-äquivalent, wenn gilt: $p \in F \iff q \notin F$.

(2) Wenn für $p = \delta(p', a)$ und $q = \delta(q', a)$ das Paar $\{p, q\}$ als nicht-äquivalent markiert ist, dann markiere auch $\{p', q'\}$ als nicht-äquivalent.
Iteriere so lange, bis sich nichts mehr ändert.

Gemäß der vorherigen "zentralen Beobachtung" gilt:

- Dadurch werden nur solche Paare $\{p, q\}$ markiert, die nicht-äquivalent sind.

Fragen:

- Werden wirklich alle nicht-äquivalenten Paare gefunden?
- Wie führt man die Grundidee effizient aus?

Die Berechnung des Äquivalenzklassenautomaten (3/4)

Warum finden wir alle nicht-äquivalenten Paare?

Angenommen, es gibt nicht-äquivalente Zustände p und q, so dass das Paar $\{p, q\}$ nicht markiert wurde.

Wir wählen unter allen **nicht-äquivalenten, nicht markierten** Zuständspaaren ein Paar $\{p', q'\}$ mit kürzestem Zeugen w.

- Es gelte o.B.d.A. $\delta(p', w) \in F$ und $\delta(q', w) \notin F$.
- Fall 1: $w = \varepsilon$.
  Dann ist $p' = \delta(p', w) \in F$ und $q' = \delta(q', w) \notin F$, und daher wurde $\{p', q'\}$ in Schritt (1) markiert. WIDERSPRUCH!
- Fall 2: $w = aw'$ für ein $a \in \Sigma$ und ein $w' \in \Sigma^*$.
  Dann sind auch die Zustände $p := \delta(p', a)$ und $q := \delta(q', a)$ nicht-äquivalent, denn: $\delta(p, w') = \delta(\delta(p', a), w') \in F$ und $\delta(q, w') = \delta(\delta(q', a), w') \notin F$.
- p und q besitzen also den Zeugen w', der kürzer ist als w.
  Daher wurde das Paar $\{p, q\}$ markiert.
- Schritt (2) unserer Markierungsstrategie markiert dann aber auch $\{p', q'\}$.
  WIDERSPRUCH!

Somit findet unser Verfahren alle nicht-äquivalenten Paare von Zuständen!

Die Berechnung des Äquivalenzklassenautomaten (4/4)

Effizientes Bestimmen aller nicht-äquivalenten Paare:

Wir betrachten folgenden gerichteten Graphen G:

- Die Knotenmenge besteht aus allen Paaren $\{p, q\}$ von Zuständen mit $p \neq q$.
- Es gibt eine Kante von $\{p, q\}$ zu $\{p', q'\}$, falls gilt:

  es gibt ein $a \in \Sigma$ mit $p = \delta(p', a)$ und $q = \delta(q', a)$.

Effiziente Implementierung unserer "Grundidee":

(1) **Initialisierung**:
- Bestimme die Adjazenzlisten-Darstellung von G.
- Markiere alle Paare $\{p, q\}$ für die gilt: $p \in F \iff q \notin F$.

$$M_0 := \big\{ \{p, q\} \mid p, q \in Q \text{ mit } p \in F \text{ und } q \notin F \big\}.$$

(2) **Berechnung**: Ausgehend von den in Schritt (1) markierten Knoten, führe eine Breiten- oder Tiefensuche durch: Markiere alle von einem Knoten in M_0 aus durch einen Weg erreichbaren Knoten.

Laufzeit:

- Der Graph G hat weniger als $|Q|^2$ Knoten und $|\Sigma| \cdot |Q|^2$ Kanten.
- Breiten-/Tiefensuche läuft in Zeit proportional zur Anzahl der Knoten und Kanten.
- Insgesamt berechnen wir den Äquivalenzklassenautomat A' in Zeit $O(|\Sigma| \cdot |Q|^2)$.

Die Berechnung des Äquivalenzklassenautomaten (4/4)

Effizientes Bestimmen aller nicht-äquivalenten Paare:

Wir betrachten folgenden gerichteten Graphen G:

- Die Knotenmenge besteht aus allen Paaren $\{p, q\}$ von Zuständen mit $p \neq q$.
- Es gibt eine Kante von $\{p, q\}$ zu $\{p', q'\}$, falls gilt:

  es gibt ein $a \in \Sigma$ mit $p = \delta(p', a)$ und $q = \delta(q', a)$.

Effiziente Implementierung unserer "Grundidee":

(1) **Initialisierung**:
- Bestimme die Adjazenzlisten-Darstellung von G.
- Markiere alle Paare $\{p, q\}$ für die gilt: $p \in F \iff q \notin F$.

$$M_0 := \big\{ \{p, q\} \mid p, q \in Q \text{ mit } p \in F \text{ und } q \notin F \big\}.$$

(2) **Berechnung**: Ausgehend von den in Schritt (1) markierten Knoten, führe eine Breiten- oder Tiefensuche durch: Markiere alle von einem Knoten in M_0 aus durch einen Weg erreichbaren Knoten.

Zur Effizienzsteigerung:

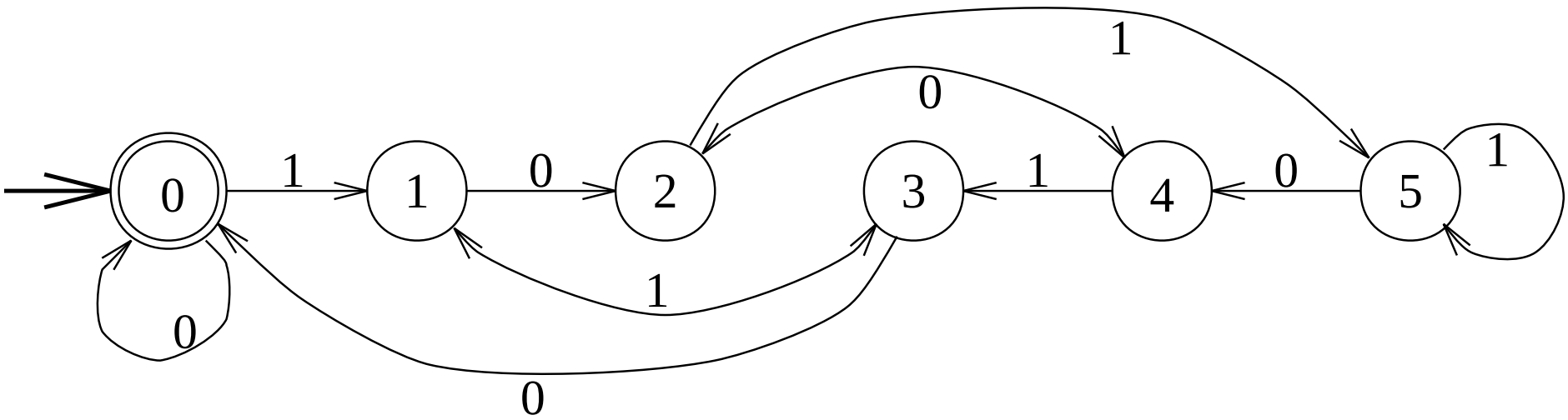
An Stelle der Schritte (1) und (2) tue folgendes:
Beginne mit der Menge M_0 und berechne von G nur die von M_0 aus erreichbaren Knoten.

Ein Beispiel

Ziel: Konstruiere einen DFA A, der die Binärdarstellungen aller durch 6 teilbaren Zahlen akzeptiert. D.h.:

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

Lösung:



Aufgabe: Bestimme den Äquivalenzklassenautomaten A'.

– siehe Tafel –

Zusammenfassung der Berechnung von A' (1/2)

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

Schritt 1: Falls A nicht vollständig ist: vervollständige A.

Schritt 2: Entferne aus A alle überflüssigen Zustände (d.h. alle Zustände, die nicht von q_0 aus erreichbar sind).

Schritt 3: Bestimme alle Paare $\{p, q\}$ mit $p, q \in Q$ und $p \not\equiv_A q$:

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

Zusammenfassung der Berechnung von A' (2/2)

Schritt 4: Konstruiere $A' := (Q', \Sigma, \delta', q_0', F')$:

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

$q_0' = [q_0]_A$

$F' = \{\ [q]_A\ :\ q \in F\ \}$

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

Zwischenstand

+ Der Äquivalenzklassenautomat A' akzeptiert dieselbe Sprache wie A.
+ Wir können A' effizient berechnen (in Zeit $O(|\Sigma| \cdot |Q|^2)$).
? Ist denn A' auch wirklich minimal?

Wir zeigen im Folgenden, dass A' tatsächlich minimal ist.

Die Nerode-Relation und der Index einer Sprache L

Eine zweite zentrale Idee:

Sei A ein vollständiger DFA für die Sprache L.
Repäsentiere einen beliebigen Zustand p von A durch die Worte in Σ^*, die zu p führen!

Beobachtung:
Wenn zwei Worte u und v beide zu p führen (d.h. $\delta(q_0, u) = \delta(q_0, v) = p$), dann gilt für alle Worte $w \in \Sigma^*$:

$$uw \in L \iff vw \in L.$$

Idee: Betrachte dies nun unabhängig von einem konkreten Automaten A.

Definition der Nerode-Relation $\equiv_L$ für eine Sprache $L \subseteq \Sigma^*$:

Für alle Worte $u, v \in \Sigma^*$ sei

$$u \equiv_L v \quad \overset{\text{Def}}{\iff} \quad \text{für alle } w \in \Sigma^* \text{ ist } uw \in L \iff vw \in L.$$

Klar: $\equiv_L$ ist eine Äquivalenzrelation auf Σ^*.

Für $u \in \Sigma^*$ sei $[u]_L := \{v \in \Sigma^* : u \equiv_L v\}$ die Äquivalenzklasse von u bezüglich $\equiv_L$.
Definiere Index(L) als die Anzahl der Äquivalenzklassen der Nerode-Relation $\equiv_L$.

Beispiel: $L_m := \{w \in \{0,1\}^* \mid |w|_1$ ist durch m teilbar$\}$.

Betrachte die Nerode-Relation $\equiv_{L_m}$:

- Für alle i, j mit $0 \leqslant i < j \leqslant m-1$ gilt:

  $$1^i \not\equiv_L 1^j, \qquad \text{denn: } 1^i 1^{m-i} \in L_m, \text{ aber } 1^j 1^{m-i} \notin L_m.$$

  Somit ist $\text{Index}(L_m) \geqslant m$.

- Die Nerode-Äquivalenzklasse des Worts 11 bzgl. L_m ist

  $$[11]_{L_m} = \{u \in \{0,1\}^* : |u|_1 \equiv 2 \pmod m\},$$

  denn für jedes solche u und für alle $w \in \{0,1\}^*$ gilt:

  $$uw \in L_m \iff |w|_1 \equiv m-2 \pmod m \iff 11w \in L_m.$$

- Für jedes i mit $0 \leqslant i \leqslant m-1$ gilt:

  $$[1^i]_{L_m} = \{ u \in \{0,1\}^* : |u|_1 \equiv i \pmod m \}.$$

- Klar: $\{0,1\}^* = \dot{\bigcup}_{i=0}^{m-1} [1^i]_{L_m}$

- Somit sind $[\varepsilon]_{L_m}, [1]_{L_m}, \ldots, [1^{m-1}]_{L_m}$ sämtliche Äquivalenzklassen der Nerode-Relation $\equiv_{L_m}$. Insbesondere ist $\text{Index}(L_m) = m$.

Ein Bezug zwischen der Nerode-Relation und DFAs

Zur Erinnerung: $u \equiv_L v \overset{\text{Def}}{\iff}$ für alle $w \in \Sigma^*$ ist $uw \in L \iff vw \in L$.

Sei L eine reguläre Sprache und sei A ein vollständiger DFA ohne überflüssige Zustände, der L akzeptiert.

Nenne zwei Worte $u, v \in \Sigma^*$ A-äquivalent, falls u und v auf denselben Zustand von A führen (d.h.: $\delta(q_0, u) = \delta(q_0, v)$). **Klar:**

- A-Äquivalenz ist eine Äquivalenzrelation auf Σ^*.
- Die Anzahl der A-Äquivalenzklassen stimmt überein mit der Zustandszahl.
- Wenn u und v A-äquivalent sind, dann sind u und v auch Nerode-äquivalent, denn wenn u und v beide zum selben Zustand p führen (d.h. $\delta(q_0, u) = \delta(q_0, v) = p$), dann gilt

$$\text{für alle } w \in \Sigma^* : \quad uw \in L \iff vw \in L.$$

Somit ist $u \equiv_L v$.

Folgerung: Sei A ein vollständiger DFA, der die Sprache L akzeptiert. Es gilt:

(1) Nerode-Äquivalenzklassen sind Vereinigungen von A-Äquivalenzklassen.

(2) A hat mindestens Index(L) Zustände, d.h. $|Q| \geqslant \text{Index}(L)$.

Zwischenstand

+ Wir haben den Äquivalenzklassenautomaten berechnet.
+ Wir haben die Nerode-Relation $\equiv_L$ eingeführt und Index(L) als die Anzahl der Äquivalenzklassen der Nerode-Relation definiert.
+ Wir wissen, dass jeder vollständige DFA für eine reguläre Sprache L mindestens Index(L) Zustände besitzt.
? Warum ist der Äquivalenzklassenautomat A' minimal?

Zeige, dass für alle Worte u und v gilt:

$$u \equiv_L v \iff u \text{ und } v \text{ sind } A'\text{-äquivalent, d.h. } \delta'(q_0', u) = \delta'(q_0', v)$$

Dann stimmt die Zustandszahl mit dem Index überein, d.h. $|Q'| = \text{Index}(L)$.

Die Richtung "$\Longleftarrow$" haben wir auf der vorherigen Folie bereits gezeigt.
Im Folgenden betrachten wir die Richtung "$\Longrightarrow$".

Zu zeigen: $u \equiv_L v \Longrightarrow \delta'(q_0', u) = \delta'(q_0', v)$

Seien u und v Worte mit $u \equiv_L v$.

Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein DFA, der die Sprache L erkennt, und
sei $A' = (Q', \Sigma, \delta', q_0', F')$ der zugehörige Äquivalenzklassenautomat.

Gemäß Definition von A' gilt:

$$\delta'(q_0', u) = [\delta(q_0, u)]_A \quad \text{und} \quad \delta'(q_0', v) = [\delta(q_0, v)]_A.$$

Sei $p := \delta(q_0, u)$ und $q := \delta(q_0, v)$, d.h. in A führen u und v zu den Zuständen p und q.

Angenommen, $\delta'(q_0', u) \neq \delta'(q_0', v)$. Dann ist $[p]_A \neq [q]_A$, d.h. $p \not\equiv_A q$.

Dann muss es einen "Zeugen" $w \in \Sigma^*$ geben, so dass

$$\delta(p, w) \in F \iff \delta(q, w) \notin F.$$

Aber dann gilt auch:

$$uw \in L \iff vw \notin L.$$

Somit ist $u \not\equiv_L v$. WIDERSPRUCH! □

Folgerung:

Der Äquivalenzklassenautomat ist minimal! Seine Zustandszahl ist $|Q'| = \text{Index}(L)$.

Wie sieht der minimale endliche Automat aus?

Wir wissen: Für jeden vollständigen DFA ohne überflüssige Zustände, der die Sprache L akzeptiert, gilt:

- Jeder Zustand q von A repräsentiert eine A-Äquivalenzklasse, nämlich die Menge aller Worte u mit $\delta(q_0, u) = q$.
- Nerode-Äquivalenzklassen sind Vereinigungen von A-Äquivalenzklassen.
- Somit ist $|Q| \geqslant \text{Index}(L)$.
- Und wenn $|Q| = \text{Index}(L)$ ist, dann muss jede Nerode-Äquivalenzklasse mit einer einzelnen A-Äquivalenzklasse übereinstimmen.
  D.h. für jedes Wort u ist $[u]_L = \{v \in \Sigma^* : \delta(q_0, v) = \delta(q_0, u)\}$.
  Somit gilt für alle Worte u, v:

$$u \equiv_L v \iff \delta(q_0, u) = \delta(q_0, v)$$

  D.h.: Genau die Worte v mit $u \equiv_L v$ erreichen den Zustand $\delta(q_0, u)$.

Folgerung: Eindeutigkeit des minimalen Automaten

Für jede reguläre Sprache L gibt es (bis auf Umbenennung der Zustände) genau einen vollständigen DFA mit minimaler Zustandszahl.
Seine Zustände entsprechen den Äquivalenzklassen der Nerode-Relation.

Der Nerode-Automat N_L für L

Folgerung: Eindeutigkeit des minimalen Automaten

Für jede reguläre Sprache L gibt es (bis auf Umbenennung der Zustände) genau einen vollständigen DFA mit minimaler Zustandszahl.
Seine Zustände entsprechen den Äquivalenzklassen der Nerode-Relation.

Frage: Wie sieht dieser Automat aus?

Antwort: Der Nerode-Automat $N_L = (Q_L, \Sigma, \delta_L, q_0, F_L)$ mit

- Zustandsmenge $Q_L := \{ [u]_L \ : \ u \in \Sigma^* \}$
  d.h.: Die Zustände sind gerade die Äquivalenzklassen der Nerode-Relation $\equiv_L$.

  Idee: Definiere den Automaten so, dass für alle $u \in \Sigma^*$ gilt: Der Zustand, in dem der Automat nach dem Lesen des Worts u ist, gibt die Äquivalenzklasse von u bzgl. der Nerode-Relation $\equiv_L$ an. D.h.:

$$\delta_L(q_0, u) \ = \ [u]_L$$

- Startzustand $q_0 := [\varepsilon]_L$
- akzeptierende Zustände $F_L := \{ [u]_L \ : \ u \in L \}$
- Programm δ_L mit $\delta_L([u]_L, a) := [ua]_L$, für alle $u \in \Sigma^*$ und $a \in \Sigma$.

— Rest: siehe Tafel —

Beispiel: $L_m := \{w \in \{0,1\}^* \mid |w|_1$ ist durch m teilbar$\}$.

Wir wissen bereits:

L_m hat genau m Äquivalenzklassen, nämlich $[\varepsilon]_{L_m}, [1]_{L_m}, [11]_{L_m}, \ldots, [1^{m-1}]_{L_m}$, und

$$[1^i]_{L_m} = \{w \in \{0,1\}^* : |w|_1 \equiv i \pmod{m}\}$$

Der Nerode-Automat für L_m:

- Die Zustände des Nerode-Automaten sind genau die Äquivalenzklassen, d.h. $[\varepsilon]_{L_m}, [1]_{L_m}, [11]_{L_m}, \ldots, [1^{m-1}]_{L_m}$:
- $[\varepsilon]_{L_m}$ ist der Startzustand und der einzige akzeptierende Zustand.
- Mit Hilfe von δ_{L_m} zählen wir die Anzahl der Einsen modulo m:

$$\delta([1^i]_{L_m}, 0) = [1^i]_{L_m} \quad \text{für alle } i \in \{0, \ldots, m-1\},$$

$$\delta([1^i]_{L_m}, 1) = \begin{cases} [1^{i+1}]_{L_m} & \text{für alle } i \in \{0, \ldots, m-2\} \\ [\varepsilon]_{L_m} & \text{für } i = m-1. \end{cases}$$

Der Satz von Myhill und Nerode

Sei Σ ein endliches Alphabet und sei $L \subseteq \Sigma^*$ eine Sprache.

(a) Eine L ist genau dann regulär, wenn Index(L) endlich ist.

(b) Index(L) gibt die Anzahl der Zustände eines minimalen vollständigen DFA an, der die Sprache L akzeptiert.

Beweis:

(a): "$\Longrightarrow$": L sei regulär. Dann gibt es einen vollständigen DFA A mit $L = L(A)$. Insbesondere ist

$$\text{Index}(L) \leqslant |Q| < \infty.$$

"$\Longleftarrow$": Es gelte Index(L) $< \infty$. Der Nerode-Automat N_L akzeptiert L und hat genau Index(L) viele Zustände. Insbesondere ist L regulär.

(b): Der Nerode-Automat N_L hat genau Index(L) Zustände, und jeder Automat für L hat mindestens Index(L) Zustände. □

Nutze Myhill-Nerode, um Nicht-Regularität nachzuweisen

Satz:

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

Beweis: **Idee:** Wir zeigen, dass Index(L) unendlich ist.

Wenn $k \neq \ell$ dann ist $a^k \not\equiv_L a^\ell$, denn: $a^k b^k \in L$, aber $a^\ell b^k \notin L$.

Somit ist $\text{Index}(L) = \infty$, denn die unendlich vielen Worte ϵ, a, a^2, a^3, ... sind paarweise nicht Nerode-äquivalent. □

Weitere nicht-reguläre Sprachen:

- Die Sprache aller wohlgeformten Klammerausdrücke.
  $\Longrightarrow$ "Automaten taugen nur für einfache Aufgaben der Compilierung."
- $\{w \in \{0,1\}^* \mid w$ hat mindestens so viele Nullen wie Einsen $\}$.
  $\Longrightarrow$ "Automaten können nicht zählen."
- $\{u\#v \mid u, v \in \{0,1\}^*,\ u \neq v\}$.
  $\Longrightarrow$ "Automaten haben nur einen endlichen Speicher, nämlich ihre endliche Zustandsmenge."

Zusammenfassung

(1) Wir haben die Verschmelzungsrelation $\equiv_A$ und den Äquivalenklassenautomaten A' definiert.

(2) Es gelte $L(A) = L$. Um zu zeigen, dass A' minimal ist, haben wir
- die Nerode Relation $\equiv_L$
- und den Index von L eingeführt.

(3) L ist genau dann regulär, wenn der Index von L endlich ist.

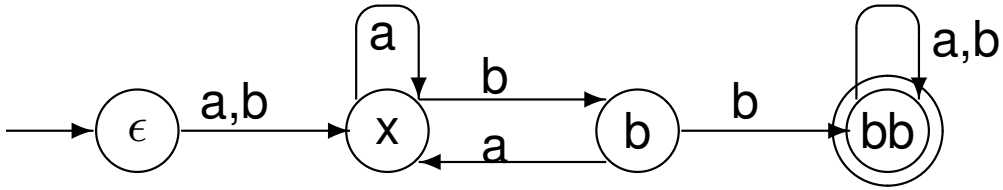
(4) Wenn L regulär ist, dann gibt es bis auf Umbenennung der Zustände genau einen minimalen vollständigen DFA für L:
Der Nerode-Automat und der Äquivalenzklassenautomat sind minimal.

(5) Um einen minimalen DFA für eine reguläre Sprache L zu konstruieren,
- konstruiere entweder den Nerode-Automaten N_L oder
- irgendeinen DFA für L, mache ihn vollständig, entferne überflüssige Zustände und bestimme den Äquivalenzklassenautomaten (finde dazu insbes. alle Paare nicht-äquivalenter Zustände).

Drei Anwendungsbeispiele des Satzes von Myhill und Nerode bzw. der DFA-Minimierung

Beispiel 1: Nachweis der Minimalität eines DFA

Der DFA



akzeptiert die Sprache $L = \{a, b\}^+\{bb\}\{a, b\}^*$. Ist der Automat minimal?

(1) Index(L) $\geqslant$ 4, denn

- die Worte abb, ab, a, ϵ sind paarweise nicht Nerode-äquivalent.

Der angegebene DFA hat 4 Zustände und ist daher minimal.

(2) Alternativ: Der Automat hat keine überflüssigen Zustände, und alle Zustände sind paarweise nicht-äquivalent.

Allgemein: Ist ein DFA A mit Zustandsmenge Q minimal?

- Zeige, dass je zwei verschiedene Zustände inäquivalent sind (bzgl. $\equiv_A$) oder
- dass Index($L(A)$) $\geqslant |Q|$.

Beide Methoden sind im Wesentlichen identisch.

Beispiel 2: Das Pattern Matching Problem (1/3)

Ein Wort $T \in \Sigma^*$ ist ein Text; ein Wort $P \in \Sigma^*$ ist ein Pattern.

Frage: Wo taucht das Pattern P im Text T auf?

Gesucht: Ein Algorithmus, der dies beantwortet.

Eine naive Lösung:

- Wir legen das Pattern sukzessive an allen Positionen $i \in \{1, \ldots, |T|-|P|+1\}$ an.
- Für Position i überprüfe, ob ein "Match" vorliegt, d.h. ob

$$T_i = P_1, \quad T_{i+1} = P_2, \quad \ldots, \quad T_{i+|P|-1} = P_{|P|}.$$

- Jede Überprüfung benötigt bis zu $|P|$ Vergleiche.
- Insgesamt werden bis zu $|P| \cdot (|T| - |P| + 1) = O(|P| \cdot |T|)$ Vergleiche durchgeführt.

Geht das schneller?

Beispiel 2: Das Pattern Matching Problem (2/3)

Idee:

Fixiere das Pattern P und betrachte die Sprache

$$L_P := \{ T \in \Sigma^* \mid P \text{ ist ein Suffix von } T \}.$$

Löse das Pattern Matching Problem, indem

- ein DFA A_P für L_P konstruiert wird.
- Dann verarbeite den Text T mit dem Automaten A_P:

  Wenn der Automat nach dem Lesen von $T_1 \cdots T_j$ einen akzeptierenden Zustand erreicht, dann ist das Pattern ein Suffix von $T_1 \cdots T_j$.
  Wir haben ein Vorkommen des Patterns gefunden!
- Statt Laufzeit $O(|P| \cdot |T|)$ nur noch Laufzeit $O(|T|)$.

Wie können wir A_P bestimmen?

Beispiel 2: Das Pattern Matching Problem (3/3)

Konstruktion des DFA A_P für $L_P := \{ T \in \Sigma^* \mid P$ ist ein Suffix von $T \}$:

Es sei $P = P_1 \cdots P_k$.

- A_P hat Zustandsmenge $\{ 0, 1, 2, \ldots, k \}$
- Startzustand: 0
- **Invariante**: Wenn Zustand i erreicht wird, dann ist $P_1 \cdots P_i$ das längste Präfix des Pattern P, das auch Suffix des Textes T ist.
- Angenommen, wir sind in Zustand i und lesen den Buchstaben X des Texts.
  - Wenn $X = P_{i+1}$, dann ist $P_1 \cdots P_{i+1}$ das längste Präfix, das auch Suffix des Texts ist. Daher setze $\delta(i, X) := i+1$.
  - Wenn $X \neq P_{i+1}$, dann setze $\delta(i, X) := j$, falls $P_1 \cdots P_j$ das längste Präfix von P ist, das auch Suffix von $P_1 \cdots P_i X$ ist.
- k ist der einzige akzeptierende Zustand von A_P.

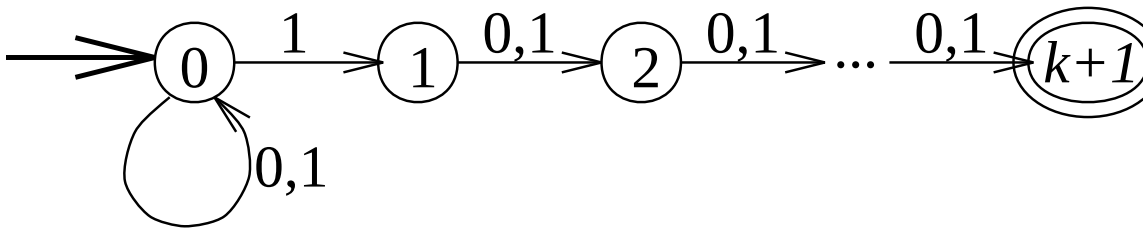
- A_P ist ein DFA für L_P mit $|P| + 1$ Zuständen.
- Der Automat ist minimal, denn
  Index$(L_P) \geqslant |P|+1$, da folgende Worte paarweise nicht Nerode-äquivalent sind:
  $P_1 \cdots P_{k-1} P_k$, $P_1 \cdots P_{k-1}$, $\ldots$, $P_1 P_2$, P_1, ε.

Beispiel 3: Nichtdeterminismus kann Zustände einsparen (1/2)

Für $k \in \mathbb{N}$ sei

$$L_k := \{0,1\}^* \cdot \{1\} \cdot \{0,1\}^k$$

Ein NFA, der L_k akzeptiert, kann die Position des $(k+1)$-letzten Buchstabens raten und dann verifizieren, dass er richtig geraten hat:



Somit gilt:

Es gibt einen NFA mit $k+2$ Zuständen, der die Sprache L_k akzeptiert.

Frage: Wie viele Zustände benötigt ein DFA, der L_k akzeptiert?

Antwort: Index(L_k) — Wie groß ist Index(L_k)?

Beispiel 3: Nichtdeterminismus kann Zustände einsparen (2/2)

Berechnung von Index(L_k) für $L_k = \{0,1\}^* \cdot \{1\} \cdot \{0,1\}^k$:

- $u, v \in \{0,1\}^k$ seien beliebige, verschiedene Worte der Länge k.
- Dann gibt es eine Position i mit $u_i \neq v_i$.
  Ohne Beschränkung der Allgemeinheit gelte

$$u_i = 0 \text{ und } v_i = 1.$$

$$\begin{aligned} \text{Es ist} \quad v\,0^i &= v_1 \cdots v_{i-1}\,1\,v_{i+1} \cdots v_k\,0^i \in L_k\,, \\ \text{aber} \quad u\,0^i &= u_1 \cdots u_{i-1}\,0\,u_{i+1} \cdots u_k\,0^i \notin L_k. \end{aligned}$$

- Die Worte u und v sind also nicht Nerode-äquivalent.
- Somit gibt es mindestens 2^k paarweise nicht Nerode-äquivalente Worte.
- Also ist Index$(L_k) \geqslant 2^k$.

Folgerung:

Jeder vollständige DFA für L_k hat mindestens 2^k Zustände.
Aber es gibt einen NFA für L_k, der nur $k+2$ Zustände besitzt.

Können auch NFAs minimiert werden?

- Es gibt Verfahren, die einen gegebenen NFA zu einem äquivalenten NFA mit möglicherweise kleinerer Zustandszahl transformieren.

- Anders als bei DFAs kann es für eine Sprache verschiedene NFAs mit minimaler Zustandszahl geben.

  Beispiel: Betrachte $L := \{a\}^+$ und konstruiere verschiedene NFAs für L mit minimaler Zustandszahl.

— Rest: siehe Tafel —

Untere Schranken für die Größe von NFAs

Untere Schranken für die Größe von Automaten

Frage:

Wie kann man für eine gegebene Sprache L zeigen, dass jeder DFA, der L erkennt, mindestens k Zustände hat?

Antwort:

Man zeigt, dass Index$(L) > k$ ist.
Denn dann hat gemäß Satz von Myhill und Nerode jeder vollständige DFA, der L erkennt, $> k$ Zustände. Somit hat jeder (nicht notwendigerweise vollständige) DFA für L mindestens k Zustände.

Frage:

Wie kann man für eine gegebene Sprache L zeigen, dass jeder **NFA**, der L erkennt, mindestens k Zustände hat?

Eine Antwort:

Man zeigt, dass Index$(L) \geqslant 2^k$ ist.
Dann hat jeder vollständige DFA für L mind. 2^k Zustände.
Aus einem NFA für L mit $k' < k$ Zuständen könnte man mittels der Potenzmengenkonstruktion einen vollständigen DFA mit $2^{k'} < 2^k$ Zuständen bauen.

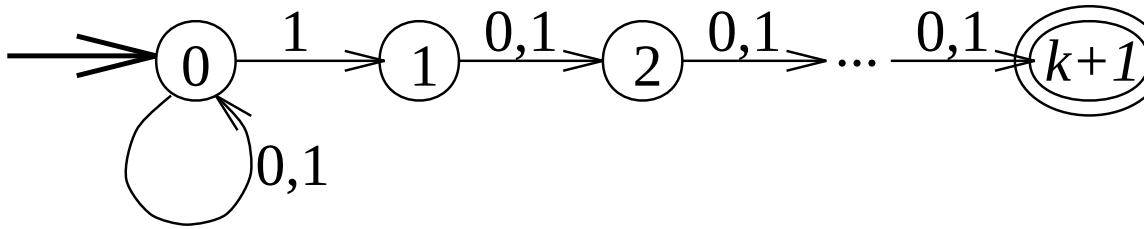
Ein Beispiel

Für $k \in \mathbb{N}$ sei $\quad L_k := \{0,1\}^* \cdot \{1\} \cdot \{0,1\}^k$

Wir wissen bereits:

- Index$(L_k) \geqslant 2^k$.
  Somit muss jeder NFA für L_k mindestens k Zustände haben.

- Es gibt einen NFA für L_k mit $k+2$ Zuständen.



Frage:

Ist das optimal?
Oder können wir einen NFA für L_k finden, der nur k oder $k+1$ Zustände hat?

Die Fooling Set Methode

Satz (Fooling Set Methode):

Sei Σ ein endliches Alphabet, $L \subseteq \Sigma^*$, $k \geqslant 1$.
Falls es Paare von Worten (u_i, v_i) für $i \in \{1, \ldots, k\}$ gibt, so dass

(1) für alle $i \in \{1, \ldots, k\}$ gilt: $u_i v_i \in L$, und

(2) für alle $i, j \in \{1, \ldots, k\}$ mit $i \neq j$ gilt: $u_i v_j \notin L$ oder $u_j v_i \notin L$,

so muss jeder NFA, der L akzeptiert, mindestens k Zustände haben.

Die Menge $\{(u_i, v_i) : i \in \{1, \ldots, k\}\}$ wird auch Fooling Set der Größe k für L genannt.

Beweis: Siehe Tafel! Details finden sich in der Arbeit

> "A lower bound technique for the size of nondeterministic finite automata"
> von I. Glaister und J. Shallit.
> *Information Processing Letters* 59, Seiten 75-77, 1996.

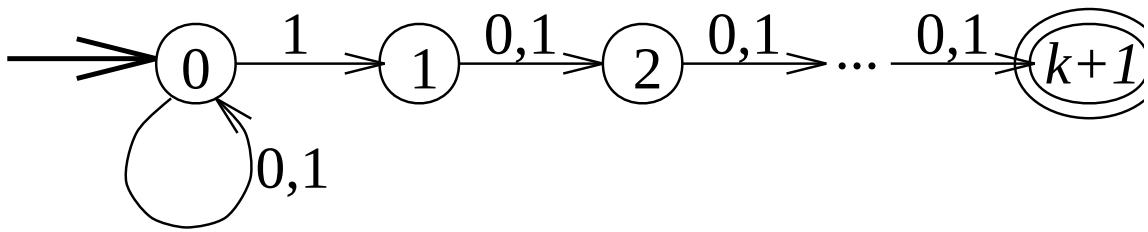
Ein Beispiel

Für $k \in \mathbb{N}$ sei $\quad L_k := \{0,1\}^* \cdot \{1\} \cdot \{0,1\}^k$

Wir wissen bereits:

- Index$(L_k) \geqslant 2^k$.
  Somit muss jeder NFA für L_k mindestens k Zustände haben.

- Es gibt einen NFA für L_k mit $k+2$ Zuständen.



Frage:

Ist das optimal?
Oder können wir einen NFA für L_k finden, der nur k oder $k+1$ Zustände hat?

Antwort:

Unter Verwendung des Satzes von Glaister und Shallit können wir zeigen, dass jeder NFA, der L_k akzeptiert, mindestens $k+2$ Zustände hat.
Der obige NFA ist also minimal. Details: Siehe Tafel!

NFAs mit Epsilon-Übergängen

NFAs mit ϵ-Übergängen

Ein NFA mit ϵ-Übergängen (kurz: ϵ-NFA) ist ein "verallgemeinerter NFA" $(Q, \Sigma, \delta, q_0, F)$, dessen Programm eine Funktion der folgenden Form ist:

$$\delta\colon Q \times \Big(\Sigma \cup \{\epsilon\}\Big) \to \mathcal{P}(Q)$$

Der Automat darf also Zustandsübergänge ausführen, ohne Buchstaben zu lesen.

Beispiele: siehe Tafel!

Gesucht: Ein ϵ-NFA, der genau die Darstellungen von Dezimalzahlen der folgenden Form akzeptiert:

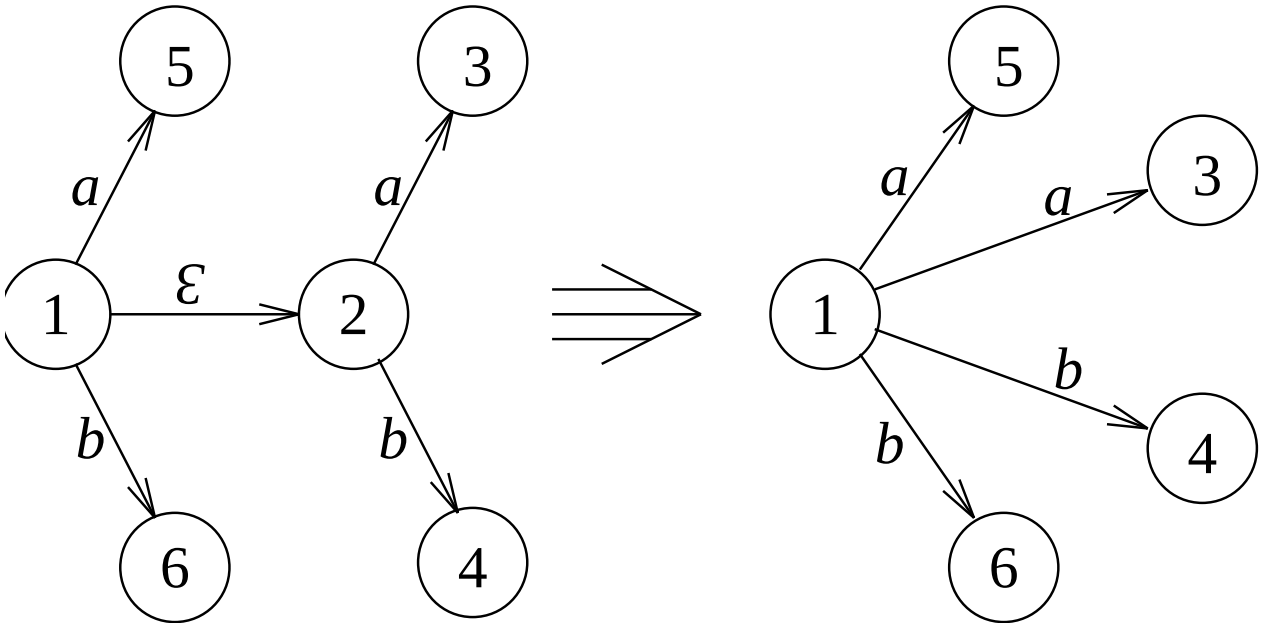
> Am Anfang kann optional ein "$+$" oder "$-$" stehen. Danach kommt eine Folge der Ziffern "0", "1", ..., "9", danach ein Dezimalpunkt ".", und danach eine weitere Folge von Ziffern"0", "1", ..., "9". Dabei muss mindestens eine der beiden Ziffernfolgen nicht-leer sein.

Frage: Können ϵ-NFAs mehr Sprachen akzeptieren als NFAs und DFAs?

Antwort: Nein!

Das Entfernen von ϵ-Übergängen: Idee

Entferne ϵ-Übergänge nach dem folgenden Schema



Die Zustandsmenge bleibt dabei gleich. Aber die Anzahl der neuen Übergänge (d.h. Pfeile in der graphischen Darstellung des NFAs) kann quadratisch anwachsen.

Das Entfernen von ϵ-Übergängen: Details

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

Ziel: Finde einen herkömmlichen NFAs A', der dieselbe Sprache wie A akzeptiert.

Dazu sind folgende Notationen nützlich:

- ϵ-Abschluss(q) := die Menge aller Zustände, die von Zustand q aus durch einen Weg erreicht werden können, der ausschließlich ϵ-Übergänge benutzt.
  Beachte: $q \in \epsilon$-Abschluss(q).
- Setze δ von Buchstaben auf Worte fort:
  $\hat{\delta}(q, w)$ ist die Menge aller Zustände, in die A von Zustand q aus durch Verarbeiten von Wort w kommen kann. — Details: Siehe Tafel! —

Definiere den NFA $A' = (Q', \Sigma, \delta', q_0', F')$ wie folgt:

- $Q' := Q$,
- $q_0' := q_0$,
- $\delta'(q, a) := \hat{\delta}(q, a)$, für alle $q \in Q'$, $a \in \Sigma$,
- $F' := \begin{cases} F \cup \{q_0\} & \text{falls } \epsilon\text{-Abschluss}(q_0) \cap F \neq \emptyset \\ F & \text{sonst} \end{cases}$

Dann ist $L(A') = L(A)$. **Beweis:** Siehe Tafel!

Zusammenfassung

- Wir können ϵ-NFAs in äquivalente NFAs umformen.
- Dabei vergrößert sich die Zustandsmenge nicht.
- Die Transformation kann bei Eingabe eines ϵ-NFAs $A = (Q, \Sigma, \delta, q_0, F)$ in Zeit $O(|Q| \cdot |\delta|)$ durchgeführt werden, wobei

$$|\delta| := \sum_{\substack{q \in Q, \\ a \in \Sigma \cup \{\epsilon\}}} |\delta(q, a)|.$$

Algorithmus: Übung!

Abschlusseigenschaften regulärer Sprachen

Abschlusseigenschaften regulärer Sprachen

Satz:

$L, L_1, L_2 \subseteq \Sigma^*$ seien reguläre Sprachen.
Dann sind auch die folgenden Sprachen regulär:

(a) $\overline{L} := \Sigma^* \setminus L$.

(b) $L_1 \cup L_2$ und $L_1 \cap L_2$.

(c) $L_1 \cdot L_2$.

(d) L^*.

Die Klasse aller regulären Sprachen ist also abgeschlossen unter Booleschen Operationen, Konkatenation und Stern-Bildung.

Beweis:

(a) Abschluss unter Komplementbildung:

Vertausche akzeptierende und verwerfende Zustände.

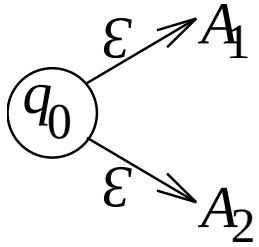
Wenn der vollständige DFA $A = (Q, \Sigma, \delta, q_0, F)$ die Sprache L akzeptiert, dann akzeptiert der Automat $B = (Q, \Sigma, \delta, q_0, Q \setminus F)$ die Komplementsprache $\overline{L}$.

Abschlusseigenschaften regulärer Sprachen

(b) Abschluss unter Vereinigung:

Verwende ϵ-Übergänge.

Wenn die DFAs $A_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ und $A_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ die Sprachen L_1 und L_2 akzeptieren, dann akzeptiert der ϵ-NFA



$A = (\{q_0\} \dot\cup Q_1 \dot\cup Q_2, \Sigma, \delta, q_0, F \cup F')$ die Sprache $L_1 \cup L_2$.

Abschluss unter Durchschnitt: folgt, da $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.

Alternative: Verwende den Produktautomaten $A = (Q, \Sigma, \delta, q_0, F)$ mit

- $Q := Q_1 \times Q_2$
- $q_0 := (q_1, q_2)$
- $\delta\big((p_1, p_2), a\big) := \big(\delta_1(p_1, a), \delta_2(p_2, a)\big)$, für alle $(p_1, p_2) \in Q$, $a \in \Sigma$
- für Durchschnitt: $F := F_1 \times F_2$
  für Vereinigung: $F := (F_1 \times Q_2) \cup (Q_1 \times F_2)$

Beachte:

Reguläre Sprachen sind natürlich nur unter endlicher Vereinigung und endlicher Durchschnittsbildung abgeschlossen.

Beispiel:

- Für jedes feste $n \in \mathbb{N}$ besteht die Sprache $L_n := \{a^n b^n\}$ aus nur einem Wort und ist damit regulär. Aber

$$\bigcup_{n \in \mathbb{N}} L_n$$

ist die Sprache $\{a^n b^n : n \in \mathbb{N}\}$, von der wir wissen, dass sie nicht regulär ist.

- Für jedes feste $n \in \mathbb{N}$ ist die Sprache $L'_n := \{a, b\}^* \setminus L_n$ regulär (da L_n regulär ist und da die regulären Sprachen unter Komplementbildung abgeschlossen sind). Aber

$$\bigcap_{n \in \mathbb{N}} L'_n$$

ist gerade die Sprache $\{a, b\}^* \setminus \{a^n b^n : n \in \mathbb{N}\}$, also das Komplement der Sprache $\{a^n b^n : n \in \mathbb{N}\}$ und daher nicht regulär.

Abschlusseigenschaften regulärer Sprachen

(c) Abschluss unter Konkatenation:

- Verbinde jeden akzeptierenden Zustand von A_1 durch einen ϵ-Übergang mit dem Startzustand von A_2.
- Die akzeptierenden Zustände von A_2 sind die akzeptierenden Zustände des neuen Automaten.
- **Skizze:** siehe Tafel. **Details:** Übung.

(d) Abschluss unter Stern-Bildung:

- Verbinde alle akzeptierenden Zustände von A durch ϵ-Übergänge mit einem neuen Startzustand q_0'.
- Verbinde q_0' durch einem ϵ-Übergang mit dem alten Startzustand q_0.
- q_0' ist der einzige akzeptierende Zustand des neuen Automaten.
- **Skizze:** siehe Tafel. **Details:** Übung.

Homomorphismen

Σ und Γ seien zwei endliche Alphabete, und h sei eine Abbildung $h : \Sigma^* \to \Gamma^*$.
Wir definieren die folgenden Sprachen:

$$h(L) := \{h(u) : u \in L\} \subseteq \Gamma^*, \quad \text{für jede Sprache } L \subseteq \Sigma^*,$$
$$h^{-1}(M) := \{u \in \Sigma^* : h(u) \in M\} \subseteq \Sigma^*, \quad \text{für jede Sprache } M \subseteq \Gamma^*.$$

Für jedes $w \in \Gamma^*$ sei $h^{-1}(w) := h^{-1}(\{w\}) = \{u \in \Sigma^* : h(u) = w\}$.

Defnition:

Eine Abbildung $h : \Sigma^* \to \Gamma^*$ heißt Homomorphismus, wenn für alle Worte $u, v \in \Sigma^*$ gilt: $h(uv) = h(u)\,h(v)$.

Beachte: Ein Homomorphismus h ist vollständig festgelegt durch die Abbildung $a \mapsto h(a)$, für alle $a \in \Sigma$. Es gilt:

- $h(\varepsilon) = \varepsilon$ und
- $h(a_1 \cdots a_n) = h(a_1) \cdots h(a_n)$, für alle $n \in \mathbb{N}_{>0}$ und $a_1, \ldots, a_n \in \Sigma$.

Beispiele

Sei $\Sigma := \{0, 1, 2\}$ und $\Gamma := \{a, b\}$.

(a) Sei $h : \Sigma^* \to \Gamma^*$ der durch

$$h(0) := a, \qquad h(1) := b, \qquad h(2) := b$$

festgelegte Homomorphismus.

Dann ist $h(0012) = aabb$, und allgemein ist $h(0^n 12^{n-1}) = a^n b^n$ für alle $n \geqslant 1$.

Somit ist

$$h(\{0^n 12^{n-1} : n \geqslant 1\}) = \{a^n b^n : n \geqslant 1\}.$$

(b) Sei $g : \Sigma^* \to \Sigma^*$ der durch

$$g(0) := 0, \qquad g(1) := 10, \qquad g(2) := 0$$

festgelegte Homomorphismus.

Dann ist $g(021) = 0010$, und $g^{-1}(0010) = \{021, 001, 221, 201\} = \{0, 2\}^2\{1\}$.

Außerdem ist $g^{-1}(0^n 10^n) = \{0, 2\}^n\{1\}\{0, 2\}^{n-1}$ und

$g^{-1}(\{0^n 10^n : n \geqslant 1\}) \cap \{0\}^*\{1\}\{2\}^* = \{0^n 12^{n-1} : n \geqslant 1\}$.

Homomorphismen und Abschlusseigenschaften regulärer Sprachen

Satz:

Seien Σ und Γ zwei endliche Alphabete und sei $h : \Sigma^* \to \Gamma^*$ ein Homomorphismus. Dann gilt:

(a) Falls $L \subseteq \Sigma^*$ regulär, so ist auch $h(L)$ regulär.

(b) Falls $M \subseteq \Gamma^*$ regulär, so ist auch $h^{-1}(M)$ regulär.

Die Klasse aller regulären Sprache ist also abgeschlossen unter Homomorphismen und unter inversen Homomorphismen.

Beweis: **Skizze:** siehe Tafel. **Details:** Übung.

Anwendungsbeispiele

Nutze Abschlusseigenschaften der Klasse aller regulären Sprachen, um die Nicht-Regularität bestimmter Sprachen nachzuweisen.

Beispiel:

Keine der folgenden Sprachen ist regulär:

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

(b) $L := \{a, b\}^* \cup \{c^k a^n b^n : k, n \geqslant 0\}$,

(c) $L := \{0^n 1 2^{n-1} : n \geqslant 1\}$,

(d) $L := \{0^n 1 0^n : n \geqslant 1\}$.

Beweis: siehe Tafel!

Weitere Abschlusseigenschaften

In den Übungen haben wir bereits einige weitere Abschlusseigenschaften der regulären Sprachen gesehen:

Satz:

$L, L_1, L_2 \subseteq \Sigma^*$ seien reguläre Sprachen; $w \in \Sigma^*$ sei ein Wort.
Dann sind auch die folgenden Sprachen regulär:

(a) $w^{-1}L := \{v \in \Sigma^* : wv \in L\}$ (Linksquotient von L bzgl. w)

(b) $L_1 \between L_2$ (Shuffle-Produkt von L_1 und L_2

(c) $L^R := \{w^R : w \in L\}$ (Rückwärtslesen von L)

Dabei ist

- $u \between v := \{u_1 v_1 \cdots u_n v_n : n \in \mathbb{N},\ u = u_1 \cdots u_n,\ v = v_1 \cdots v_n,\ u_i, v_i \in \Sigma^*\}$,
  $L_1 \between L_2 := \bigcup_{u \in L_1, v \in L_2} u \between v$

- $w^R := a_n \cdots a_1$, falls $w = a_1 \cdots a_n$ mit $a_i \in \Sigma$, $n \geqslant 0$.

Zusammenfassung

Die Klasse der regulären Sprachen ist abgeschlossen unter

- Komplementbildung
  L regulär $\Longrightarrow \overline{L} := \Sigma^* \setminus L$ regulär
- Vereinigung und Durchschnitt
  L_1, L_2 regulär $\Longrightarrow L_1 \cup L_2$ und $L_1 \cap L_2$ regulär
- Konkatenation
  L_1, L_2 regulär $\Longrightarrow L_1 \cdot L_2$ regulär
- Kleene-Stern
  L regulär $\Longrightarrow L^*$ regulär
- Homomorphismen und inversen Homomorphismen
  L regulär, h Homomorphismus $\Longrightarrow h(L)$ und $h^{-1}(L)$ regulär
- Rückwärtslesen
  L regulär $\Longrightarrow L^R$ regulär
- Shuffle-Produkt
  L_1, L_2 regulär $\Longrightarrow L_1 \between L_2$ regulär
- Linksquotienten
  L regulär, $w \in \Sigma^* \Longrightarrow w^{-1}L$ regulär

Entscheidungsprobleme

Welche Fragen über endliche Automaten sind effizient beantwortbar?

A sei ein DFA oder NFA.

Ist $L(A) \neq \emptyset$?

- Sei q_0 der Anfangszustand von A und sei F die Menge der akzeptierenden Zustände.
- $L(A)$ ist genau dann nicht-leer, wenn es einen Weg von q_0 zu einem Zustand in F gibt.
- Wende Tiefensuche auf q_0 an: $L(A)$ ist nicht-leer, wenn Tiefensuche einen akzeptierenden Zustand findet.

Die Laufzeit ist proportional zur Anzahl der Kanten im Zustandsdiagramm, also in $O(|\Sigma| \cdot |Q|)$ für DFAs und in $O(|\Sigma| \cdot |Q|^2)$ für NFAs.

Welche Fragen lassen sich effizient beantworten?

D, D_1, D_2 seien deterministische Automaten

Die folgenden Fragen lassen sich effizient beantworten (in Zeit $O(|\Sigma| \cdot |Q|)$ bzw. $O(|\Sigma| \cdot |Q_1| \cdot |Q_2|)$):

(a) Ist $L(D) = \Sigma^*$? (Universalität)

(b) Ist $L(D_1) = L(D_2)$? (Äquivalenz)

(c) Ist $L(D_1) \subseteq L(D_2)$? ("Containment")

(d) Ist $L(D)$ endlich? (Endlichkeit)

Beweis: Übungsaufgabe! Idee: Nutze jeweils aus, dass die Frage

$$\text{Ist } L(D) \neq \emptyset?$$

effizient beantwortet werden kann.

Die obigen Fragen (a)–(c) sind für NFAs extrem schwierig: Man kann zeigen, dass sie PSPACE-vollständig sind (also wahrscheinlich noch schwerer als NP-vollständige Probleme).

Das Wortproblem für NFAs

Für einen NFA A und ein Wort w entscheide, ob A die Eingabe w akzeptiert.

Wie schwierig ist das Wortproblem?

- Nicht ganz so einfach wie für DFAs,
- aber die Menge $\delta(q_0, w)$ der von q_0 aus erreichbaren Zustände lässt sich effizient berechnen: Zeit $O(|w| \cdot |Q|^2)$ reicht aus!

  Algorithmus: Übungsaufgabe!
- Akzeptiere genau dann, wenn $\delta(q_0, w) \cap F \neq \emptyset$.

Sehr schwierige Entscheidungsfragen für DFAs und NFAs

(1) Das Interpolationsproblem für DFAs:
Für gegebene Teilmengen $P, N \subseteq \Sigma^*$ und einen Schwellenwert $k \geqslant 1$,

gibt es einen DFA mit höchstens k Zuständen, der alle Worte in P akzeptiert und alle Worte in N verwirft?

Dieses Problem ist NP-vollständig. (Hier ohne Beweis)

(2) Das Minimierungsproblem für NFAs:

- Gegeben zwei NFAs A und B, entscheide, ob B ein zu A äquivalenter NFA mit minimaler Zustandszahl ist.
  Dieses Problem ist PSPACE-vollständig. (Hier ohne Beweis)
- Es ist sogar schwierig, die minimale Zustandszahl *approximativ* zu bestimmen.
- Was ist da los? Bereits die Frage "Ist $L(N) \neq \Sigma^*$?" ist extrem schwer (nämlich PSPACE-vollständig) und gehört daher wahrscheinlich nicht einmal zur Klasse NP.

  Worte in $\Sigma^* \setminus L(N)$ sind unter Umständen sehr lang.

Reguläre Ausdrücke

Reguläre Ausdrücke

Die Menge der regulären Ausdrücke über einem endlichen Alphabet Σ wird rekursiv wie folgt definiert:

Basisregel: Die Ausdrücke $\emptyset$, ϵ und a für $a \in \Sigma$ sind regulär.

Die Ausdrücke beschreiben die leere Sprache ($L(\emptyset) = \emptyset$), die Sprache des leeren Wortes ($L(\varepsilon) = \{\varepsilon\}$) und die Sprache des einbuchstabigen Wortes a ($L(a) = \{a\}$).

Rekursive Regeln: Sind R und S reguläre Ausdrücke, dann auch $(R \mid S)$, $(R \cdot S)$ und R^*.

$\mid$ beschreibt die Vereinigung und wird manchmal auch mit $+$ bezeichnet ($L((R \mid S)) = L(R) \cup L(S)$),
$\cdot$ die Konkatenation und wird manchmal auch mit $\circ$ bezeichnet ($L((R \cdot S)) = L(R) \cdot L(S)$), und
$*$ beschreibt die Kleene-Hülle ($L(R^*) = L(R)^*$).

Abkürzende Schreibweise

Zur vereinfachten Schreibweise und besseren Lesbarkeit vereinbaren wir folgende Konventionen:

- Den Punkt bei der Konkatenation darf man weglassen.
  (Schreibe (RS) statt $(R \cdot S)$)
- Bei Ketten gleichartiger Operationen darf man die Klammern weglassen.
  (Schreibe $(R_1|R_2|R_3)$ statt $((R_1|R_2)|R_3)$, und $(R_1R_2R_3)$ statt $((R_1R_2)R_3)$)
- Äußere Klammern, die einen regulären Ausdruck umschließen, dürfen weggelassen werden.
- Zur besseren Lesbarkeit dürfen zusätzliche Klammern eingefügt werden.
- Bei fehlenden Klammern gelten folgende "Präzedenzregeln":
  "$*$" bindet stärker als "$\cdot$"; "$\cdot$" bindet stärker als "$|$".

Beispiel:

$a|bbc^*$ ist eine verkürzte Schreibweise für den regulären Ausdruck $(a|((b\cdot b)\cdot c^*))$.
Die von ihm beschriebene Sprache ist $L(a|bbc^*) = \{a\} \cup (\{bb\} \cdot \{c\}^*)$.

Beispiele für reguläre Ausdrücke

(1) $L_1 = \{w \in \{a,b\}^* \mid w \text{ beginnt und endet mit dem Buchstaben } a\}$
wird beschreiben durch den regulären Ausdruck

$$a \mid a(a \mid b)^* a.$$

(2) Sei $P \in \{a,b\}^*$. Die Sprache aller Worte in $\{a,b\}^*$ mit Teilwort P, wird beschrieben durch den regulären Ausdruck

$$(a \mid b)^* P (a \mid b)^*.$$

(3) Der reguläre Ausdruck

$$b^* \mid (b^* a b^* a b^*)^*$$

beschreibt die Sprache
$L_3 = \{w \in \{a,b\}^* \mid w \text{ enthält eine gerade Anzahl von } a\text{'s}\}$.

Reguläre Ausdrücke definieren reguläre Sprachen

Satz:

Sei Σ ein endliches Alphabet.
Jeder reguläre Ausdruck R über Σ beschreibt eine reguläre Sprache $L(R)$.

Beweis: Per Induktion über den Aufbau der regulären Ausdrücke.

Induktionsanfang: Betrachte die gemäß Basisregeln gebildeten regulären Ausdrücke.

$\emptyset$, ϵ, und a, für $a \in \Sigma$ sind reguläre Ausdrücke, die die Sprachen $L(\emptyset) = \emptyset$, $L(\epsilon) = \{\varepsilon\}$ und $L(a) = \{a\}$ beschreiben. Man kann leicht DFAs konstruieren, die diese Sprachen akzeptieren.

Induktionsschritt: Betrachte die gemäß rekursiven Regeln gebildeten regulären Ausdrücke.

Seien R und S reguläre Ausdrücke, die gemäß Induktionsannahme reguläre Sprachen $L(R)$ und $L(S)$ beschreiben. Wir wissen bereits, dass die Klasse der regulären Sprachen abgeschlossen ist unter Vereinigung, Konkatenation und Kleene-Stern. Daher sind auch die folgenden Sprachen regulär:
$L(R) \cup L(S) = L((R \mid S))$, $L(R) \cdot L(S) = L((R \cdot S))$, $L(R)^* = L(R^*)$.
Somit beschreiben auch die regulären Ausdrücke $(R \mid S)$, $(R \cdot S)$ und R^* reguläre Sprachen. □

Beispiel: Regulärer Ausdruck $\Longrightarrow$ ϵ-NFA

Gegeben sei der reguläre Ausdruck $R := 01^* \mid 1$.

Aufgabe: Konstruiere einen ϵ-NFA A mit $L(A) = L(R)$.

Systematisches Vorgehen entlang des Beweises, dass reguläre Ausdrücke reguläre Sprachen definieren:

(1) Zerlege R in Teilausdrücke:

- $R_0 := 0$
- $R_1 := 1$
- $R_2 := 1^* = R_1^*$
- $R_3 := 01^* = R_0 R_2$
- $R_4 := 01^* | 1 = R_3 | R_1 = R$

(2) Konstruiere, für jedes $i \in \{0, \ldots, 4\}$, einen ϵ-NFA A_i mit $L(A_i) = L(R_i)$.

Dann ist $A := A_4$ der gesuchte Automat.

Rest: Siehe Tafel!

Zwischenstand

- Jeder reguläre Ausdruck definiert also eine reguläre Sprache.
- Und umgekehrt, gibt es zu jeder regulären Sprache auch einen regulären Ausdruck?

Reguläre Sprachen haben reguläre Ausdrücke (1/3)

Satz:

Sei Σ ein endliches Alphabet. Zu jeder regulären Sprache $L \subseteq \Sigma^*$ gibt es einen regulären Ausdruck R über Σ, der die Sprache L beschreibt.

Beweis:

Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein vollständiger DFA, der L akzeptiert.
Es gelte $Q = \{1, \ldots, n\}$.

Idee: Benutze dynamische Programmierung, um reguläre Ausdrücke $R^k_{p,q}$ zu konstruieren, die die Mengen $L^k_{p,q}$ beschreiben, wobei

$$L^k_{p,q} := \{ w \in \Sigma^* \quad : \quad \delta(p, w) = q, \text{ und alle Zwischenzustände liegen in der Menge } \{1, \ldots, k\} \},$$

für alle $p, q \in Q$ und alle $k \in \{0, 1, \ldots, n\}$.

Es gilt: $L = \bigcup_{q \in F} L^n_{q_0,q}$. Für $F = \{q_1, \ldots, q_s\}$ wird L daher durch den regulären Ausdruck $(R^n_{q_0,q_1} \mid \cdots \mid R^n_{q_0,q_s})$ beschrieben.

Reguläre Sprachen haben reguläre Ausdrücke (2/3)

Rekursiv für $k \in \{0, 1, 2, \ldots, n\}$ konstruieren wir nun reguläre Ausdrücke $L^k_{p,q}$ für alle $p, q \in Q$.

Reguläre Ausdrücke für $L^0_{p,q}$, für alle $p, q \in Q$:

- $L^0_{p,q}$ ist die Menge aller Worte, die Zustand q von Zustand p aus **ohne** Zwischenzustände erreichen. Also:
  - Falls $p \neq q$, so ist $L^0_{p,q} = \{a \in \Sigma : \delta(p, a) = q\}$
  - Falls $p = q$, so ist $L^0_{p,q} = \{\varepsilon\} \cup \{a \in \Sigma : \delta(p, a) = q\}$
- Also ist $L^0_{p,q}$ eine endliche Menge von Buchstaben bzw. dem leeren Wort. Sie wird daher von einem regulären Ausdruck $R^0_{p,q}$ beschrieben (durch das Symbol $\emptyset$ oder durch Verknüpfen der Buchstaben bzw. des leeren Wortes mit dem "|"-Operator).

Als nächstes konstruieren wir für jedes $k \geqslant 1$ reguläre Ausdrücke für die Mengen $L^k_{p,q}$, wobei wir annehmen, dass wir reguläre Ausdrücke für $L^{k-1}_{r,s}$ schon kennen (für alle $p, q, r, s \in Q$).

Reguläre Sprachen haben reguläre Ausdrücke (3/3)

Reguläre Ausdrücke für $L^k_{p,q}$:

- $L^k_{p,q}$ ist die Menge aller Worte w, die Zustand q von Zustand p aus mit Zwischenzuständen aus der Menge $\{1,\ldots,k\}$ erreichen.
  - Entweder benötigt ein Wort w nur Zwischenzustände in der Menge $\{1,\ldots,k-1\}$, d.h. es liegt in $L^{k-1}_{p,q}$,
  - oder Zustand k ist Zwischenzustand.
    - Dann führt die Berechnung von p nach k,
    - „pendelt“ zwischen k und k
    - und erreicht dann q von Zustand k aus.
  - Wie oft kann Zustand k angenommen werden? Keine Ahnung! Macht aber nichts, denn

$$L^k_{p,q} \;=\; L^{k-1}_{p,q} \;\cup\; L^k_{p,k}\left(L^{k-1}_{k,k}\right)^*\; L^{k-1}_{k,q}.$$

- Daher wird $L^k_{p,q}$ durch den folgenden regulären Ausdruck beschrieben:

$$R^k_{p,q} \;:=\; R^{k-1}_{p,q} \;\mid\; R^{k-1}_{p,k}\cdot\left(R^{k-1}_{k,k}\right)^*\cdot R^{k-1}_{k,q}.$$

□

Beispiel: vollständiger DFA $\Longrightarrow$ regulärer Ausdruck

Gegeben sei der vollständige DFA A

— siehe Tafel —

Aufgabe: Konstruiere einen regulären Ausdruck R mit $L(R) = L(A)$.

Systematisches Vorgehen entlang des Beweises, dass reguläre Sprachen von regulären Ausdrücken beschrieben werden:

- Konstruiere die regulären Ausdrücke $R^k_{p,q}$ für $k \in \{0, 1, 2\}$ und alle $p, q \in \{1, 2, 3\}$,
- konstruiere dann $R^3_{1,2}$ und $R^3_{1,3}$ und setze $R := R^3_{1,2} \mid R^3_{1,3}$.

Rest: Siehe Tafel (bzw. [Hopcroft Ullman, Example 2.12]).

Der Satz von Kleene

Wir haben somit Folgendes bewiesen:

Der Satz von Kleene

Sei Σ ein endliches Alphabet. Eine Sprache $L \subseteq \Sigma^*$ ist genau dann regulär, wenn sie von einem regulären Ausdruck beschrieben werden kann.

Fragen:

(a) Gegeben ein regulärer Ausdruck R, wie groß wird der zu R äquivalente ϵ-NFA?

Antwort: $O(|R|)$ (Beweis: Übung!)

(b) Gegeben ein vollständiger DFA A, wie groß wird der zu A äquivalente reguläre Ausdruck R?

Antwort: $2^{O(|Q|)}$ (Beweis: siehe Tafel)

Reguläre Ausdrücke für Komplement und Durchschnitt (1/2)

- Wir wissen: Die Klasse aller regulären Sprachen ist abgeschlossen unter Komplement- und Durchschnittbildung
- Obwohl die regulären Ausdrücke keine expliziten Operatoren für Komplement- und Durchschnittbildung enthalten, gibt es daher für alle regulären Ausdrücke R und S einen regulären Ausdruck
  - $\tilde{R}$, der die Sprache $\overline{L(R)}$ beschreibt, und
  - T, der die Sprache $L(R) \cap L(S)$ beschreibt.
- **Frage:** Wie können wir bei gegebenen R und S solche regulären Ausdrücke $\tilde{R}$ und T konstruieren?
- **Antwort:**

$$\begin{aligned} R,\ S &\rightsquigarrow \epsilon\text{-NFAs} && A_{L(R)},\ A_{L(S)} \\ &\rightsquigarrow \text{vollständige DFAs} && B_{L(R)},\ B_{L(S)} \\ &\rightsquigarrow \text{vollständige DFAs} && B_{\overline{L(R)}},\ B_{L(R)\cap L(S)} \\ &\rightsquigarrow \text{reguläre Ausdrücke} && \tilde{R},\ T. \end{aligned}$$

- **Laufzeit unserer Verfahren:** Für $\tilde{R}$: $2^{2^{O(|R|)}}$. Für T: $2^{2^{O(|R|+|S|)}}$.
- **Frage:** Geht das effizienter?

Reguläre Ausdrücke für Komplement und Durchschnitt (2/2)

Wir kennen Verfahren, die bei gegebenen regulären Ausdrücken R und S in Zeit

- $2^{2^{O(|R|)}}$ einen regulären Ausdruck $\tilde{R}$ für die Sprache $\overline{L(R)}$
- $2^{2^{O(|R|+|S|)}}$ einen regulären Ausdruck T für die Sprache $L(R) \cap L(S)$

liefern.

Frage: Geht das effizienter?

Antwort von Gelade und Neven (2008): (hier ohne Beweis)

- Für $\tilde{R}$ ist unser Verfahren im Wesentlichen optimal:

  Es gibt reguläre Ausdrücke R_n der Länge $O(n)$, so dass die kürzesten regulären Ausdrücke, die die Sprache $\overline{L(R_n)}$ beschreiben, Länge mindestens 2^{2^n} haben.

- Für T gibt es ein (relativ naheliegendes) Verfahren, das mit Laufzeit $2^{O(|R| \cdot |S|)}$ auskommt. Dieses Verfahren ist im Wesentlichen optimal:

  Es gibt reguläre Ausdrücke R_n, S_n der Länge $O(n^2)$, so dass die kürzesten regulären Ausdrücke, die die Sprache $L(R_n) \cap L(S_n)$ beschreiben, Länge mindestens 2^n haben.

Reguläre Grammatiken

Grammatiken

Programmiersprachen lassen sich am besten als die Sprache aller syntaktisch korrekten Programme auffassen.

Definiere die Syntax durch eine Grammatik.

Wie erzeugt man arithmetische Ausdrücke mit ganzzahligen Konstanten?
Wir arbeiten mit den Variablen

- A: erzeuge einen arithmetischen Ausdruck,
- I: erzeuge eine natürliche Zahl und
- Z: erzeuge eine Ziffer.

$$
\begin{aligned}
A &\rightarrow A + A \mid A - A \mid A * A \mid (A) \mid I \mid +I \mid -I \\
I &\rightarrow Z\,I \mid Z \\
Z &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}
$$

Wir benutzen Produktionen (oder Ersetzungsregeln).

Die Komponenten einer Grammatiken

Eine Grammatik $G = (\Sigma, V, S, P)$ besteht aus:

- einem endlichen Alphabet Σ,
- einer endlichen Menge V von Variablen (oder Nichtterminalen) mit $\Sigma \cap V = \emptyset$,
- dem Startsymbol $S \in V$ und
- einer endlichen Menge P von Produktionen der Form $u \to v$ mit

$$u \in (\Sigma \cup V)^* \, V \, (\Sigma \cup V)^* \quad \text{und} \quad v \in (\Sigma \cup V)^*$$

Beginne mit dem Startsymbol S und wende dann Produktionen an:

Eine Produktion $u \to v$ ersetzt ein Vorkommen von u durch ein Vorkommen von v.

Die von einer Grammatik erzeugte Sprache

- Für eine Produktion $u \to v$ und ein Wort $w_1 = xuy$ wird das Wort $w_2 = xvy$ abgeleitet. Wir schreiben

$$xuy \Rightarrow xvy.$$

- Für Worte $r, s \in (\Sigma \cup V)^*$ schreiben wir

$$r \overset{*}{\Rightarrow} s \quad :\Longleftrightarrow \quad \text{Es gibt Worte } w_1 = r, w_2, \ldots, w_k = s, \text{ für } k \geqslant 1, \text{ so dass } w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_k.$$

  s kann in null, einem oder mehreren Schritten aus r abgeleitet werden.

- $L(G) = \left\{ w \in \Sigma^* \ : \ S \overset{*}{\Rightarrow} w \right\}$ ist die von der Grammatik G erzeugte Sprache.

Reguläre Grammatiken

Frage: Welcher Typ von Grammatik erzeugt reguläre Sprachen?

Eine Grammatik $G = (\Sigma, V, S, P)$, bei der alle Produktionen von der Form

$$\begin{array}{lcll} X & \to & \epsilon & \text{für } X \in V \quad \text{oder} \\ X & \to & aY & \text{für } X, Y \in V \text{ und } a \in \Sigma \end{array}$$

sind, heißt regulär.

Die zentralen Fragen:

- Wird jede reguläre Sprache von einer regulären Grammatik erzeugt?
- Ist die von einer regulären Grammatik erzeugte Sprache regulär?

Reguläre Grammatiken: Ein Beispiel

Eine Grammatik für die Sprache aller Worte über $\Sigma = \{a, b\}$, die das Wort *aba* als Teilwort besitzen:

- Die Variablen:
  - S: ist das Startsymbol; es soll ein beliebiges Präfix erzeugen,
  - V_a: bedeutet, dass wir gerade ein *a* erzeugt haben und dass als nächstes ein *b* erzeugt werden soll,
  - V_{ab}: bedeutet, dass wir gerade *ab* erzeugt haben und dass als nächstes ein *a* erzeugt werden soll,
  - T: soll ein beliebiges Suffix erzeugen.

- Die Produktionen:

$$\begin{aligned} S &\to aS \mid bS \mid aV_a \\ V_a &\to bV_{ab} \\ V_{ab} &\to aT \\ T &\to aT \mid bT \mid \epsilon \end{aligned}$$

Reguläre Sprachen besitzen reguläre Grammatiken

Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein NFA, der die Sprache $L := L(A)$ akzeptiert.

Auf Eingabe $w = a_1 \cdots a_n$ führt A Zustandsübergänge

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$$

durch.

Sei $G = (\Sigma, V, S, P)$ die reguläre Grammatik mit

- Variablenmenge $V := Q$
- Startsymbol $S := q_0$
- Produktionen

$$p \to aq \quad \text{für alle } p, q, a \text{ mit } \delta(p, a) \ni q$$
$$p \to \epsilon \quad \text{für alle } p \in F.$$

Behauptung: Diese Grammatik G erzeugt die Sprache $L = L(A)$.

Beweis: Übung!

Reguläre Grammatiken erzeugen reguläre Sprachen

Sei $G = (\Sigma, V, S, P)$ eine reguläre Grammatik, die die Sprache $L := L(G)$ erzeugt.

Eine Ableitung von G hat die Form

$$S \Rightarrow a_1 V_1 \Rightarrow a_1 a_2 V_2 \Rightarrow \cdots \Rightarrow a_1 \cdots a_n V_n \Rightarrow a_1 \cdots a_n.$$

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

- Zustandsmenge $Q := V$,
- Startzustand $q_0 := S$,
- Überführungsfunktion $\delta : Q \times A \to \mathcal{P}(Q)$, wobei für alle $X \in Q$ und $a \in \Sigma$ gilt:

$$\delta(X, a) := \{Y \ : \ (X \to aY) \ \in P\}.$$

- Endzustandsmenge $F := \{X \in V \ : \ (X \to \epsilon) \ \in \ P\}$

Behauptung: Dieser NFA A akzeptiert genau die Sprache $L = L(G)$.

Beweis: Übung!

Zusammenfassung (1/2)

(1) Wir haben deterministische Automaten effizient minimiert.

- Dazu haben wir den Äquivalenzklassenautomat berechnet.
- Um Minimalität nachzuweisen, haben wir die Nerode-Relation definiert.
- Bis auf eine Umbenennung der Zustände stimmen alle minimalen Automaten mit dem Nerode-Automaten überein:
  - Der Index ist eine untere Schranke für die Anzahl benötigter Zustände.
  - Eine Sprache L ist genau dann regulär, wenn der Index von L endlich ist.

(2) Mit dem Pumping-Lemma oder mit dem Index der Sprache oder den Abschlusseigenschaften der Klasse aller regulären Sprachen kann man Nicht-Regularität nachweisen.

Zusammenfassung (2/2)

(3) Nichtdeterministische Automaten erlauben eine kompaktere Beschreibung.

- Die Potenzmengenkonstruktion haben wir für die Bestimmung äquivalenter deterministischer Automaten benutzt.
- Mit der Fooling Set Methode haben wir ein Werkzeug zum Nachweis unterer Schranken für die Größe von NFAs kennengelernt.

(4) Reguläre Grammatiken und reguläre Ausdrücke defineren ebenfalls genau die Klasse der regulären Sprachen.

(5) Reguläre Sprachen sind unter Komplement, Vereinigung, Durchschnitt, Konkatenation und Stern-Bildung, Homomorphismen, inversen Homomorphismen, ... abgeschlossen.

(6) Viele Entscheidungsfragen sind effizient für deterministische Automaten beantwortbar.
Für nichtdeterministische Automaten ist schon die Frage

$$\text{Ist } L(N) \neq \Sigma^* ?$$

außerst schwierig.
Aber das Wortproblem und das Problem $L(N) \neq \emptyset$? sind effizient lösbar.