Grundbegriffe der Theoretischen Informatik

Sommersemester 2017 - Beate Bollig

Die Folien basieren auf den Materialien von Thomas Schwentick.

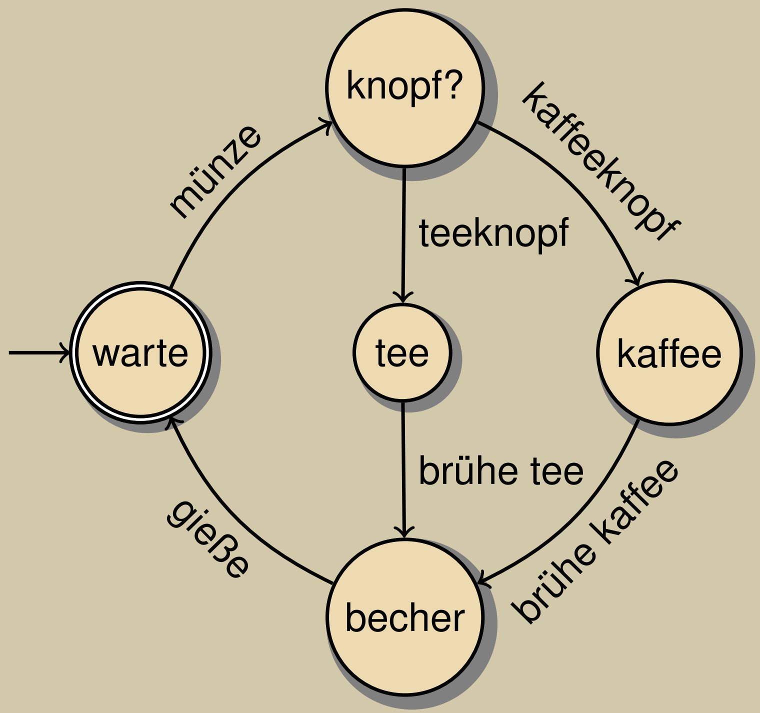
Teil A: Reguläre Sprachen

5: Abschlusseigenschaften, Grenzen und Algorithmen

Einleitung (1/2)

- Das Verhalten von vielen praktischen Systemen kann durch DFAs oder NFAs abstrahiert/beschrieben werden

Kaffeemaschine als DFA



- Oft möchten wir Eigenschaften solcher Systeme **automatisch überprüfen**

Beispiel

- Die Maschine soll nur Kaffee ausgießen, wenn (seit dem letzten Kaffee) eine Münze eingeworfen wurde
- Diese Eigenschaft lässt sich durch einen RE ausdrücken:
  - $\boldsymbol{R} \stackrel{\text{def}}{=} \left(\boldsymbol{S}^* \text{ münze } \boldsymbol{S}^* \text{ gieße } \boldsymbol{S}^*\right)^*$

    (Mit $\boldsymbol{S} = \boldsymbol{\Sigma} - \{\text{münze,gieße}\}$)

- Lässt sich automatisch überprüfen, ob der Automat die Eigenschaft $\boldsymbol{R}$ erfüllt?
- Formal führt das zur Frage: Ist $\boldsymbol{L}(\boldsymbol{A}) \subseteq \boldsymbol{L}(\boldsymbol{R})$?
  - Äquivalent: Ist $\boldsymbol{L}(\boldsymbol{A}) \cap (\boldsymbol{\Sigma}^* - \boldsymbol{L}(\boldsymbol{R})) = \varnothing$?

- Es wäre also praktisch, eine Toolbox für Automaten zu haben
  - Schnitt, Vereinigung, Komplement, etc.
  - Testalgorithmen...

Einleitung (2/2)

- Wir setzen in diesem Kapitel die Untersuchung der Klasse der regulären Sprachen fort

- Wir betrachten
  - algorithmische Methoden, mit denen sich reguläre Sprachen kombinieren und modifizieren lassen
  - eine weitere, einfache Methode zum Nachweis, dass eine Sprache nicht regulär ist
  - Algorithmen zum Testen von Eigenschaften einer durch einen Automaten gegebenen Sprache

Inhalt

Größenmaße für Automaten

- Der Aufwand der folgenden Algorithmen wird in Abhängigkeit von der Größe der Eingabe beschrieben
- Wenn die Eingabe aus Automaten besteht, stellt sich also die Frage:
  - **Wie „groß“ ist ein endlicher Automat $\mathcal{A} = (Q, \Sigma, \delta, s, F)$?**

- Es können verschiedene Größenmaße definiert werden:
  - Anzahl der Zustände: $|Q|$
  - Anzahl der Transitionen: $|\delta|$
  - Größe der Kodierung des Automaten als Bitstring

- Wir verwenden hier nur die ersten beiden Maße und geben das verwendete Maß jeweils explizit an

- Für reguläre Ausdrücke α bezeichnet $|\alpha|$ einfach die Länge des Strings
  - Also: $|(ab)^*c(d+\epsilon)| = 11$

✎ Ob Konkatenationssymbole und Klammern für die Größe eines RE gezählt werden, ist für das Folgende nicht so wichtig
  - Wir interessieren uns nur für asymptotische Abschätzungen
  - $|\alpha|$ (nach unserer Definition) ist linear in der Anzahl der Symbolvorkommen und der Vorkommen des $*$-Operators (sofern doppelte Klammerpaare ((...)) nicht vorkommen)

Synthese endlicher Automaten: Boolesche Operationen (1/3)

- Wir betrachten jetzt, auf welche Weisen aus regulären Sprachen neue reguläre Sprachen gewonnen werden können

- Uns interessiert also:
  - Unter welchen Operationen ist die Klasse der regulären Sprachen abgeschlossen?
  - Mit welchen Algorithmen lassen sich solche Operationen ausführen?

✎ Wichtig: es geht im Folgenden **nicht** um Abschlusseigenschaften von einzelnen Sprachen sondern um Abschlusseigenschaften der Klasse aller regulären Sprachen

- Wir beginnen mit Booleschen Operationen

- Reguläre Ausdrücke haben einen Operator für die Vereinigung

  ➨ Die **Vereinigung** zweier regulärer Sprachen ist regulär

- Um reguläre Sprachen algorithmisch gut „verarbeiten“ zu können, ist es wichtig, dass auch der **Durchschnitt** zweier regulärer Sprachen und das **Komplement** einer regulären Sprache wieder regulär sind

- Außerdem sollten diese Booleschen Operationen auf der Ebene endlicher Automaten möglichst effizient ausgeführt werden können

Synthese endlicher Automaten: Boolesche Operationen (2/3)

Satz 5.1

- Seien $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ und $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ DFAs
- Dann lassen sich Automaten für die folgenden Sprachen konstruieren:
  (a) für $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$
  mit $|Q_1||Q_2|$ Zuständen in Zeit $\mathcal{O}(|Q_1||Q_2||\Sigma|)$
  (b) für $L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$
  mit $|Q_1||Q_2|$ Zuständen in Zeit $\mathcal{O}(|Q_1||Q_2||\Sigma|)$
  (c) für $\Sigma^* - L(\mathcal{A}_1)$ mit $|Q_1|$ Zuständen in Zeit $\mathcal{O}(|Q_1|)$

Folgerung 5.2

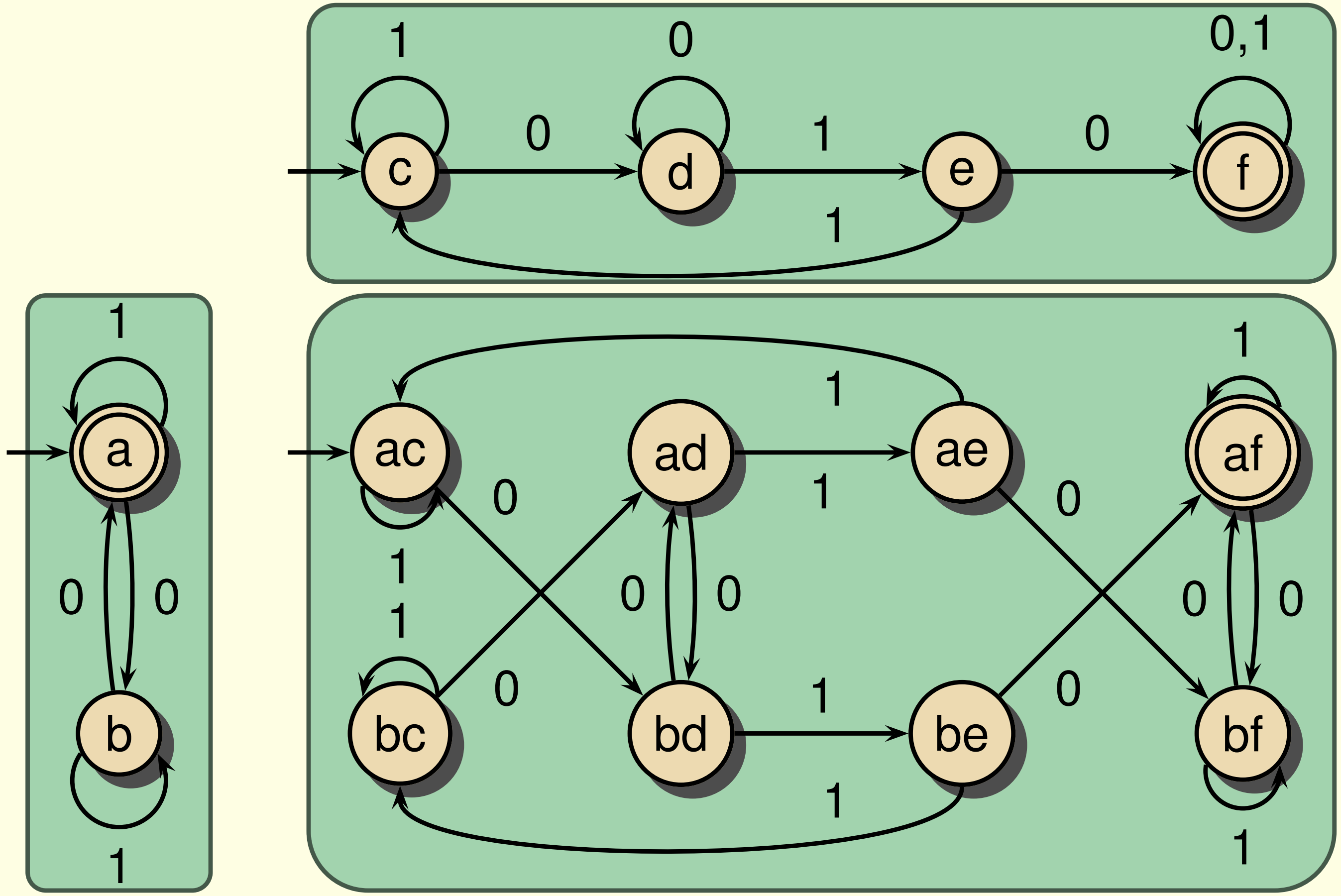
- Die regulären Sprachen sind unter Durchschnitt, Vereinigung und Komplementbildung abgeschlossen

✎ (a) und (b) gelten auch für NFAs

- Für den Beweis von (a) und (b) verwenden wir das Konzept des **Produktautomaten**

Produktautomat: Beispiel

- Ein Automat für die Menge aller Strings, die 010 als Teilstring enthalten **und** gerade viele Nullen haben:



0110100

- Um einen Automaten für die oben genannte Sprache zu erhalten, muss af **als akzeptierender Zustand** gewählt werden

Synthese endlicher Automaten: Boolesche Operationen (3/3)

Definition

- Seien $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$, $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ DFAs
- Sei $F \subseteq Q_1 \times Q_2$

- Der **Produktautomat zu $\mathcal{A}_1$ und $\mathcal{A}_2$** mit akzeptierender Menge F ist der Automat $\mathcal{B} \stackrel{\text{def}}{=} (Q_1 \times Q_2, \Sigma, \delta_{\mathcal{B}}, (s_1, s_2), F)$, wobei $\delta_{\mathcal{B}}$ komponentenweise definiert ist, d.h.:
  - Für $q_1 \in Q_1$ und $q_2 \in Q_2$ sei $\delta_{\mathcal{B}}((q_1, q_2), \sigma) \stackrel{\text{def}}{=} (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma))$

✎ Wir schreiben manchmal $\mathcal{A}_1 \times \mathcal{A}_2$ für den Produktautomaten, ohne eine akzeptierende Zustandsmenge zu spezifizieren

Beweis von Satz 5.1

- Wir beweisen zunächst Teil (a): Durchschnitt
- Sei $\mathcal{B}$ der Produktautomat zu $\mathcal{A}_1$ und $\mathcal{A}_2$ mit akzeptierender Menge $F_1 \times F_2$
- Durch Induktion lässt sich leicht zeigen, dass für alle $w \in \Sigma^*$ gilt: $\delta^*_{\mathcal{B}}((s_1, s_2), w) = (\delta^*_1(s_1, w), \delta^*_2(s_2, w))$
- Es folgt:

$$
\begin{aligned}
& w \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2) \\
\iff & \delta^*_1(s_1, w) \in F_1 \text{ und } \delta^*_2(s_2, w) \in F_2 \\
\iff & (\delta^*_1(s_1, w), \delta^*_2(s_2, w)) \in F_1 \times F_2 \\
\iff & \delta^*_{\mathcal{B}}((s_1, s_2), w) \in F_1 \times F_2 \\
\iff & w \in L(\mathcal{B})
\end{aligned}
$$

- Teil (b) kann analog bewiesen werden, mit akzeptierender Menge $F_1 \times Q_2 \cup Q_1 \times F_2$

- Teil (c) ist noch einfacher: Wähle $(Q_1, \Sigma, \delta_1, s_1, Q_1 - F_1)$ als DFA

Zum Verständnis des Produktautomaten

PINGO-Frage: `pingo.upb.de`

Wie muss die akzeptierende Menge F des Produkt-automaten gewählt werden, damit er die Menge aller Strings akzeptiert, die von einem der Automaten $\mathcal{A}_1$ und $\mathcal{A}_2$ akzeptiert wird, aber nicht vom anderen (**symmetrische Differenz**)?

(A) $(F_1 - F_2) \cup (F_2 - F_1)$

(B) $(Q_1 \times Q_2) - (F_1 \times F_2)$

(C) $(Q_1 \times F_2) \cup (F_1 \times Q_2)$

(D) $(Q_1 \times (Q_2 - F_2)) \cup ((Q_1 - F_1) \times Q_2)$

(E) $(F_1 \times (Q_2 - F_2)) \cup ((Q_1 - F_1) \times F_2)$

Synthese endlicher Automaten: Konkatenation und Iteration

- Die Definition regulärer Ausdrücke garantiert auch den Abschluss unter Konkatenation und Stern

Satz 5.3

- Seien $\mathcal{A}_1$ und $\mathcal{A}_2$ DFAs (oder NFAs) für Sprachen L_1 und L_2
- Dann lassen sich NFAs (oder DFAs) für $L_1 \circ L_2$ und L_1^* konstruieren

Beweisidee

- Ein DFA für $L_1 \circ L_2$ kann in zwei Schritten gewonnen werden:
  1. Verknüpfung von $\mathcal{A}_1$ und $\mathcal{A}_2$ durch ϵ-Übergänge von den akzeptierenden Zuständen von $\mathcal{A}_1$ zum Startzustand von $\mathcal{A}_2$
     - Wie bei der Umwandlung von REs in ϵ-NFAs
  2. Determinisierung des entstandenen ϵ-NFAs

- L_1^*: (fast) analog; eventuell neuer akzeptierender Startzustand

Abschlusseigenschaften: Homomorphismen (1/3)

- Wir betrachten nun weitere Abschlusseigenschaften der Klasse der regulären Sprachen, die vor allem für theoretische Zwecke hilfreich sind:
  - Abschluss unter Homomorphismen
  - Abschluss unter inversen Homomorphismen

Definition

- Eine Funktion $h : \Sigma^* \to \Gamma^*$ ist ein **Homomorphismus**, wenn für alle Strings $u, v \in \Sigma^*$ gilt: $h(uv) = h(u)h(v)$

- Aus der Definition folgt: $h(\epsilon) = \epsilon$

- Zur Definition eines Homomorphismus von Σ^* nach Γ^* genügt es, $h(\sigma)$ für alle $\sigma \in \Sigma$ festzulegen
- Dadurch ist $h(w)$ auch für beliebige Strings $w = \sigma_1 \cdots \sigma_n$ eindeutig festgelegt:
$$h(w) = h(\sigma_1) \cdots h(\sigma_n)$$

Beispiel

- $h : \{a, b, c, d\}^* \to \{0, 1\}^*$
- Definiert durch:
  - $h(a) \stackrel{\text{def}}{=} 00$
  - $h(b) \stackrel{\text{def}}{=} 1$
  - $h(c) \stackrel{\text{def}}{=} \epsilon$
  - $h(d) \stackrel{\text{def}}{=} 0110$
- Dann: $h(abdc) = 0010110$

- Für $L \subseteq \Sigma^*$ sei
$$h(L) \stackrel{\text{def}}{=} \{h(w) \mid w \in L\}$$
- Für $L \subseteq \Gamma^*$ sei
$$h^{-1}(L) \stackrel{\text{def}}{=} \{w \mid h(w) \in L\}$$

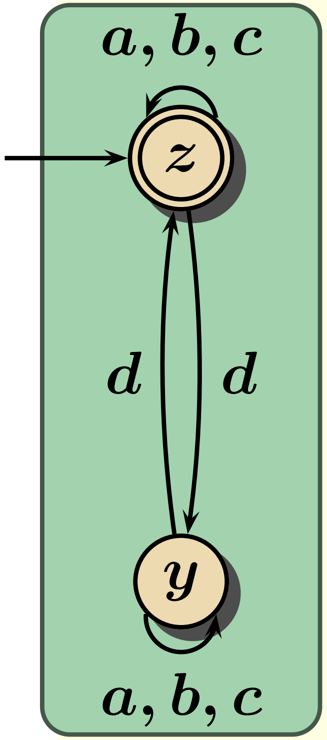
- Wir werden sehen: aus einem DFA für L lassen sich leicht konstruieren:
  - ein DFA für $h^{-1}(L)$ und
  - ein NFA für $h(L)$

➡ $h(L)$ und $h^{-1}(L)$ sind regulär

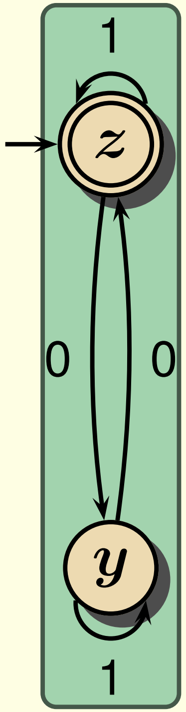
Abschlusseigenschaften: Homomorphismen (2/3)

- h_1:
  $a \mapsto 00$
  $b \mapsto 1$
  $c \mapsto \epsilon$
  $d \mapsto 0100$

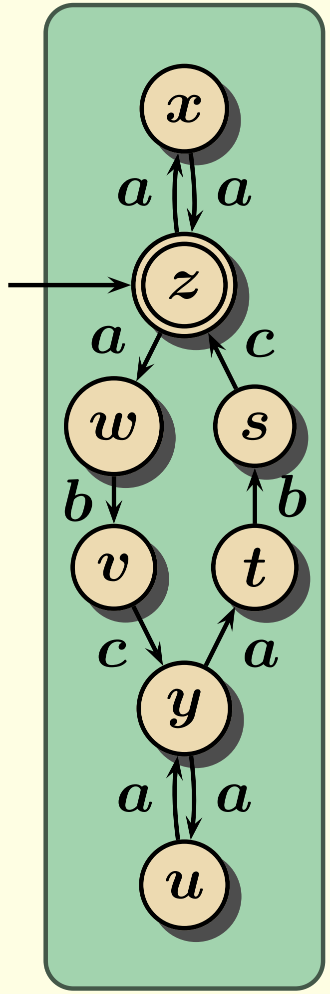
- h_2:
  $0 \rightarrow abc$
  $1 \rightarrow aa$



h_1^{-1}
$\Leftarrow$



h_2
$\Rightarrow$



Automat für $h_1^{-1}(L)$ Automat für L Automat für $h_2(L)$

Abschlusseigenschaften: Homomorphismen (3/3)

Satz 5.4

- Ist L eine reguläre Sprache über Σ und ist Γ ein Alphabet, so sind die folgenden Sprachen regulär:
  (a) $h(L)$, für jeden Homomorphismus $h : \Sigma^* \to \Gamma^*$
  (b) $h^{-1}(L)$, für jeden Homomorphismus $h : \Gamma^* \to \Sigma^*$

Beweisidee

(b) Sei $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ DFA für L

- Wir definieren $\mathcal{A}' \stackrel{\text{def}}{=} (Q, \Gamma, \delta', s, F)$ durch:
$$\delta'(q, \sigma) \stackrel{\text{def}}{=} \delta^*(q, h(\sigma))$$
- Dann gilt: $\delta'^*(s, w) = \delta^*(s, h(w))$
- Also: $w \in L(\mathcal{A}') \iff h(w) \in L(\mathcal{A})$

(a) Idee:

- Ersetze die Transition $\delta(q, \sigma)$ durch eine Folge von Transitionen für $h(\sigma)$

➞ neue Zustände einfügen

Synthese endlicher Automaten: Größe

- Die folgende Tabelle gibt eine Übersicht über die Größe der Zielautomaten (Anzahl Zustände) für die betrachteten Operationen
- Dabei spielt es eine Rolle, ob die gegebenen Automaten und der Zielautomat DFAs oder NFAs sind

- Q_1 und Q_2 bezeichnen jeweils die Zustandsmengen für Automaten für L_1 und L_2
- $|h|$ und $|S|$ bezeichnen jeweils die Größe der Repräsentation von h und S

	DFA $\longrightarrow$ DFA	DFA $\longrightarrow$ NFA	NFA $\longrightarrow$ NFA
$L_1 \cap L_2$	$\mathcal{O}(\|Q_1\| \times \|Q_2\|)$	$\mathcal{O}(\|Q_1\| \times \|Q_2\|)$	$\mathcal{O}(\|Q_1\| \times \|Q_2\|)$
$L_1 \cup L_2$	$\mathcal{O}(\|Q_1\| \times \|Q_2)\|$	$\mathcal{O}(\|Q_1\| + \|Q_2)\|$	$\mathcal{O}(\|Q_1\| + \|Q_2\|)$
$L_1 - L_2$	$\mathcal{O}(\|Q_1\| \times \|Q_2\|)$	$\mathcal{O}(\|Q_1\| \times \|Q_2\|)$	$\|Q_1\| \times 2^{\mathcal{O}(\|Q_2\|)}$
$L_1 \circ L_2$	$\|Q_1\| \times 2^{\mathcal{O}(\|Q_2\|)}$	$\mathcal{O}(\|Q_1\| + \|Q_2\|)$	$\mathcal{O}(\|Q_1\| + \|Q_2\|)$
L_1^*	$2^{\mathcal{O}(\|Q_1\|)}$	$\mathcal{O}(\|Q_1\|)$	$\mathcal{O}(\|Q_1\|)$
$h(L_1)$	$2^{\mathcal{O}(\|Q_1\|+\|h\|)}$	$\mathcal{O}(\|Q_1\| + \|h\|)$	$\mathcal{O}(\|Q_1\| + \|h\|)$
$h^{-1}(L_1)$	$\mathcal{O}(\|Q_1\|)$	$\mathcal{O}(\|Q_1\|)$	$\mathcal{O}(\|Q_1\|)$

Inhalt

Pumping-Lemma: Einleitung

- Wir haben mit dem Satz von Myhill und Nerode bereits eine Methode kennen gelernt, mit der wir überprüfen können, ob eine Sprache regulär ist
- Damit haben wir gezeigt, dass die Sprache $L_{ab} = \{a^m b^m \mid m \geqslant 0\}$ **nicht** regulär ist

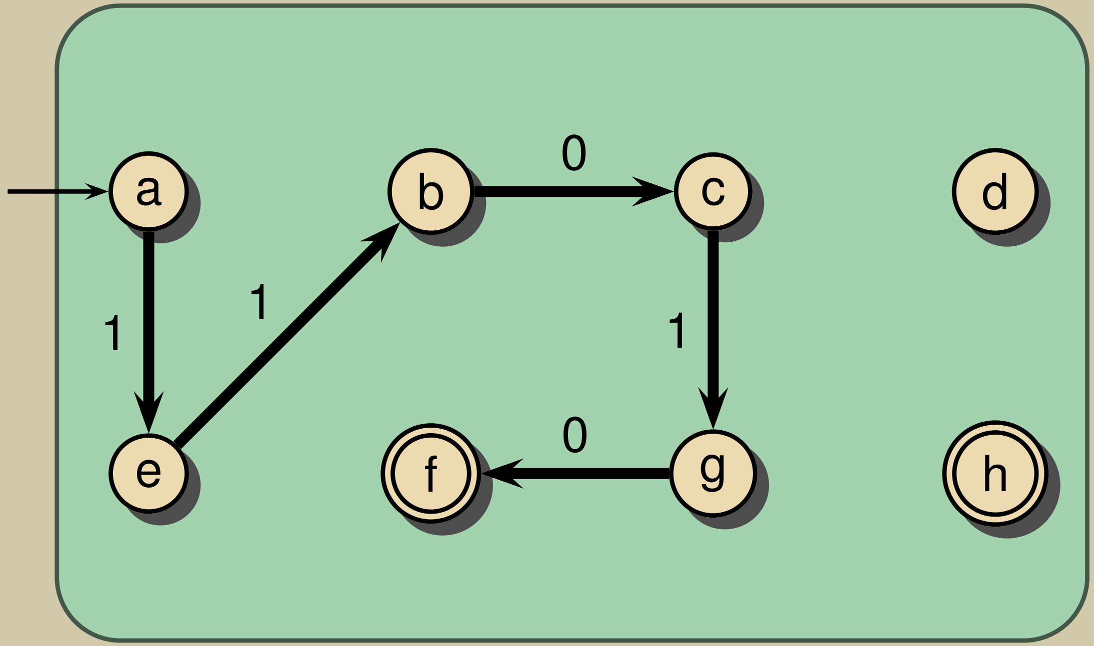
- Wir werden jetzt mit dem **Pumping-Lemma** eine weitere Methode kennen lernen, mit der sich nachweisen lässt, dass eine gegebene Sprache nicht regulär ist

- **Vorteile:**
  - Recht einfach und anschaulich
  - Lässt sich verallgemeinern für kontextfreie Sprachen
  - Liefert interessante Einsicht über reguläre Sprachen

- **Nachteile:**
  - Funktioniert nicht immer
  - Lässt sich nicht zum Nachweis von Regularität verwenden

Pumping-Lemma: Grundidee (1/2)

Beispiel

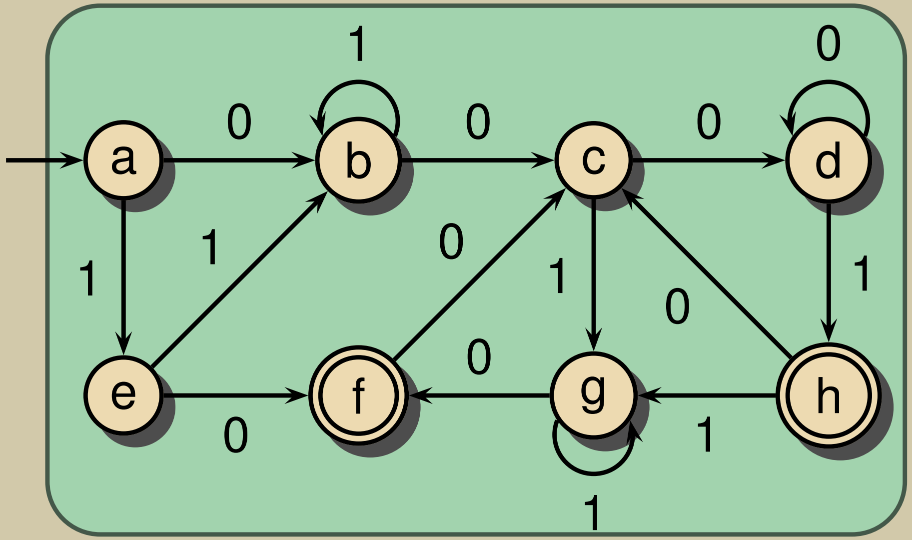


110 010 10	$x\ y\ z$
110 010 010 10	$x\ yy\ z$
110 010 010 010 10	$x\ yyy\ z$
$\vdots$	$x\ y^k\ z$
110 10	$x\ z$

- **Beobachtung:** Ein „Kreis“ in einer akzeptierenden Berechnung lässt sich beliebig oft wiederholen
- Der akzeptierte String wird dabei „aufgepumpt“ (oder „abgepumpt“)

Pumping-Lemma: Grundidee (2/2)

Beispiel



- Etwas formaler:
  - Wenn $|w| \geqslant |Q|$ gilt, muss es einen Zustand geben, den der DFA beim Lesen von w (mindestens) zweimal besucht
  - Wenn der DFA beim Lesen eines Strings $w \in L(\mathcal{A})$ einen Zustand zweimal besucht, lässt w sich so in xyz zerlegen, dass gelten:
    - $y \neq \epsilon$ und
    - für alle $k \geqslant 0$ ist $xy^kz \in L(\mathcal{A})$

➨ Wenn L regulär ist, gibt es ein n, so dass sich jedes $w \in L$ mit $|w| \geqslant n$ in xyz zerlegen lässt, so dass gelten:
- $y \neq \epsilon$ und
- für alle $k \geqslant 0$ ist $xy^kz \in L(\mathcal{A})$

✎ n ergibt sich hier als Größe $|Q|$ der Zustandsmenge eines DFAs für L

Pumping-Lemma: Aussage und Beweis

Satz 5.5

- Sei L regulär
- Dann gibt es ein n, so dass jeder String $w \in L$ mit $|w| \geqslant n$ auf mindestens eine Weise als $w = xyz$ geschrieben werden kann, so dass die folgenden Aussagen gelten:

(1) $y \neq \epsilon$

(2) $|xy| \leqslant n$

(3) für alle $k \geqslant 0$ ist $xy^k z \in L$

✎ Die Aussage des Pumping-Lemmas gilt auch in der Form:

$$w \notin L \ldots \Rightarrow \ldots xy^k z \notin L$$

Beweisskizze

- L regulär $\Rightarrow$ $L = L(\mathcal{A})$ für einen DFA $\mathcal{A}$
- Sei n die Anzahl der Zustände von $\mathcal{A}$

- Sei $w \in L$ mit $|w| \geqslant n$

➡ Beim Lesen der ersten n Zeichen von w muss sich ein Zustand wiederholen (Schubfachprinzip)

➡ $\delta^*(s, x) = \delta^*(s, xy)$ für gewisse x, y, z mit $w = xyz$ und (1) und (2)

- Sei $q \stackrel{\text{def}}{=} \delta^*(s, x)$

➡ $\delta^*(q, y) = q$

➡ $\delta^*(s, xy^k z) = \delta^*(s, xyz) \in F$, für alle $k \geqslant 0$

➡ (3)

Pumping-Lemma: Anwendung (1/2)

- Für den Nachweis, dass eine gegebene Sprache nicht regulär ist, ist die folgende äquivalente Formulierung des Pumping-Lemmas besser geeignet

Korollar 5.6

- Sei L eine Sprache
- Angenommen, für jedes $n > 0$ gibt es einen String $w \in L$ mit $|w| \geqslant n$, so dass für jede Zerlegung $w = xyz$ mit

  (1) $y \neq \epsilon$ und

  (2) $|xy| \leqslant n$

  ein $k \geqslant 0$ existiert, so dass $xy^k z \notin L$
- Dann ist L nicht regulär

- Da Korollar 5.6 die Kontraposition von Satz 5.5 ist, folgt es direkt aus Satz 5.5

Beispiel

- Sei wieder $L_{ab} \overset{\text{def}}{=} \{a^m b^m \mid m \geqslant 0\}$
- Sei n beliebig
- Wir wählen $w = a^n b^n \in L_{ab}$

  (wichtig: w hängt von n ab!)
- $|w| = 2n \geqslant n$
- Seien nun x, y, z beliebige Strings, die $w = xyz$ und die folgenden Bedingungen erfüllen:

  (1) $y \neq \epsilon$

  (2) $|xy| \leqslant n$
- Wegen (2) enthält y kein b, wegen (1) enthält es mindestens ein a

  ➡ xz hat mehr b als a
- Wähle $k = 0$:

  ➡ $xy^0z = xz \notin L_{ab}$

➡ L_{ab} ist nicht regulär

Pumping-Lemma: Anwendung (2/2)

Beispiel

- Sei wieder $L_{ab} \stackrel{\text{def}}{=} \{a^m b^m \mid m \geqslant 0\}$
- Sei n beliebig
- Wir wählen $w = a^n b^n \in L_{ab}$ (wichtig: w hängt von n ab!)
- $|w| = 2n \geqslant n$
- Seien nun x, y, z beliebige Strings, die $w = xyz$ und die folgenden Bedingungen erfüllen:
  (1) $y \neq \epsilon$
  (2) $|xy| \leqslant n$
- Wegen (2) enthält y kein b, wegen (1) enthält es mindestens ein a
  ➨ xz hat mehr b als a
- Wähle $k = 0$:
  ➨ $xy^0z = xz \notin L_{ab}$

➨ L_{ab} ist nicht regulär

- **Bemerkungen zur Anwendung des Pumping-Lemmas**
  - n dürfen Sie **nicht** wählen
    - ∗ Der Beweis muss für beliebiges n funktionieren
  - $w \in L$ dürfen Sie selbst (geschickt) wählen
    - ∗ Es muss in Abhängigigkeit von n gewählt werden ($|w| \geqslant n$)
      - · Dabei ist n für den Beweis eine Variable
  - x, y, z dürfen Sie **nicht** wählen
    - ∗ Wir wissen aber (und verwenden), dass (1) und (2) gelten
  - Zuletzt muss ein k gefunden werden, für das $xy^kz \notin L$ gilt
    - ∗ Sehr oft ist hier eine Fallunterscheidung nötig:
      - · nach den Möglichkeiten, wie x, y, z den String w unterteilen

Das Pumping-Lemma als Spiel

- Das Pumping-Lemma ist zwar im Kern recht anschaulich, der Wechsel zwischen Existenz- und Allquantoren kann jedoch durchaus zu Verwirrung führen
- Es kann deshalb hilfreich sein, die Aussage des Pumping-Lemmas in ein 2-Personen-Spiel zu fassen

- **Spiel für Sprache L:**
  - Person 1 wählt n
  - Person 2 wählt ein $w \in L$ mit $|w| \geqslant n$
  - Person 1 wählt x, y, z mit
  $$w = xyz, y \neq \epsilon, |xy| \leqslant n$$
  - Person 2 wählt k
- Falls $xy^kz \notin L$, hat Person 2 gewonnen, andernfalls Person 1

- Es gilt: **falls Person 2 eine Gewinnstrategie hat, ist L nicht regulär**
- Wenn Sie nachweisen wollen, dass L nicht regulär ist, **sind Sie Person 2**

Grenzen des Pumping-Lemmas

- Sei L die Sprache
  $\{a^m b^n c^n \mid m, n \geqslant 1\} \cup$
  $\{b^m c^n \mid m, n \geqslant 0\}$
- Klar: L ist nicht regulär
  - Das lässt sich durch eine leichte Abwandlung des Beweises aus Kapitel 4 für L_{ab} zeigen

- Aber: L erfüllt die Aussage des Pumping-Lemmas:
  - Jeder String w, lässt sich als xyz zerlegen, mit $x = \epsilon$ und $|y| = 1$

➡ dann lässt sich y beliebig wiederholen:
  - ∗ falls $y = a$: klar, dann ist das Wort in der ersten Menge und es dürfen beliebig viele a's kommen
  - ∗ falls $y \neq a$: klar, dann ist das Wort in der zweiten Menge und das Zeichen darf beliebig wiederholt werden

- Es gilt aber die folgende Verallgemeinerung des Pumping-Lemmas

Satz 5.7 [Jaffe, 78]

- Eine Sprache L ist **genau dann** regulär **wenn** es ein n gibt, so dass jeder String $w \in L$ mit $|w| \geqslant n$ auf mindestens eine Weise als $w = xyz$ geschrieben werden kann, so dass die folgenden Aussagen gelten:

(1) $y \neq \epsilon$

(2) $|xy| \leqslant n$

(3') für alle $k \geqslant 0$ gilt: $xy^kz \sim_L xyz$

Reguläre Sprachen: Grenzen

- Woran lässt sich erkennen, ob eine Sprache regulär ist?

- Intuitiv: wenn es genügt, sich beim Lesen eines Eingabewortes nur konstant viel Information zu merken, **unabhängig von der Eingabelänge**
- Beispiel: $L_{ab} = \{a^m b^m \mid m \geqslant 0\}$ ist nicht regulär, da nach Lesen von a^i „das i gemerkt sein muss“
- Insbesondere darf der Wertebereich, in dem gezählt wird, nicht mit der Eingabelänge größer werden

- Aber: es gibt auch reguläre Sprachen, die mit Zahlen zu tun haben, zum Beispiel: $L_{\text{drei}} \stackrel{\text{def}}{=} \{w \mid w \text{ ist die Binärdarstellung einer Zahl, die durch drei teilbar ist}\}$

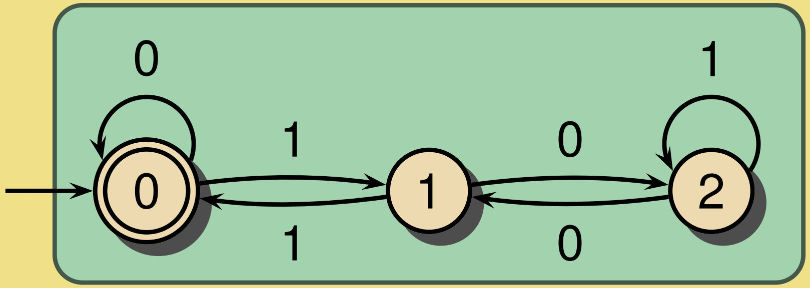
Reguläre Sprachen: Zählen modulo ...

- Ziel: Automat für $L_{\text{drei}} = \{w \mid w$ ist die Binärdarstellung einer Zahl, die durch drei teilbar ist$\}$
- Ansatz: was passiert, wenn an eine Binärzahl eine 0 oder 1 angehängt wird?

- Notation:
  - Für $w \in \{0,1\}^*$ sei $B(w)$ die Zahl, die von w repräsentiert wird
  - Also: $B(1100) = 12$

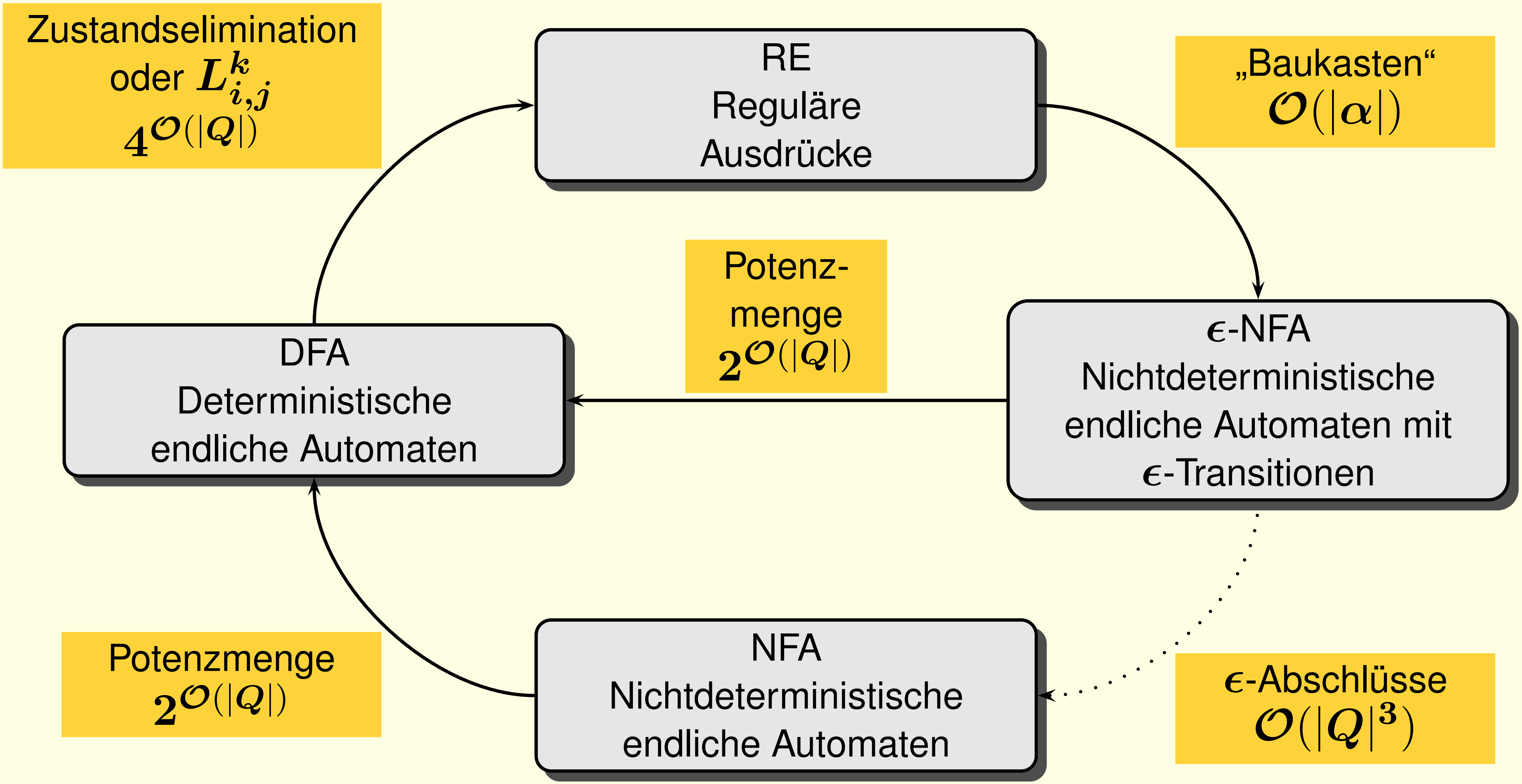
- Es gelten:
  - $B(u0) = 2B(u)$
  - $B(u1) = 2B(u) + 1$
- Also: wenn $B(u) \equiv_3 0$, dann $B(u0) \equiv_3 0$ und $B(u1) \equiv_3 1$

→ Grundidee des Automaten: der Zustand (0,1, oder 2) gibt den Rest der bisher gelesenen Binärzahl bei Division durch 3 an



Inhalt

Umwandlungsalgorithmen (Wdh.)



- Die Laufzeiten der Umwandlungsalgorithmen entsprechen im Wesentlichen den Größen der Zielobjekte

Algorithmen: Leerheitstest

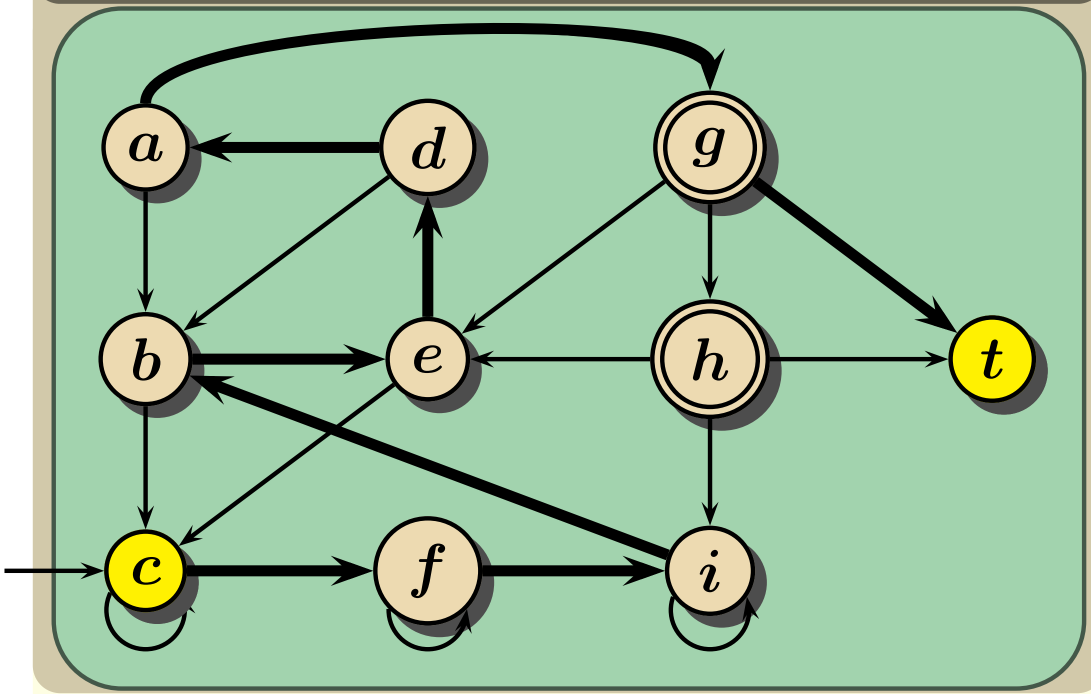
- Für Anwendungen (nicht nur) im Bereich des Model Checking ist das folgende Problem wichtig

Definition: Leerheits-Problem für DFAs

Gegeben: DFA $\mathcal{A}$

Frage: Ist $L(\mathcal{A}) \neq \emptyset$?

Beispiel



Algorithmus:

1. Vergiss die Kantenmarkierungen
2. Füge einen Zielknoten t ein und Kanten von allen akzeptierenden Knoten zu t
3. Teste, ob es einen Weg von s nach t gibt
4. Falls ja: Ausgabe „$L(\mathcal{A}) \neq \emptyset$“

- Das Leerheitsproblem für DFAs lässt sich also auf das Erreichbarkeitsproblem in gerichteten Graphen zurückführen
  - **Aufwand:** $\mathcal{O}(|\delta|)$
- Der Algorithmus funktioniert auch für NFAs

- Der Leerheitstest für reguläre Ausdrücke α ist (fast) trivial:
  - falls kein $\emptyset$ vorkommt, ist $L(\alpha) \neq \emptyset$
  - Ansonsten lässt sich α gemäß der Regeln aus Kapitel 2 vereinfachen
  - $L(\alpha) \neq \emptyset \iff$ am Schluss bleibt **nicht** $\emptyset$ übrig

Algorithmen: Wortproblem

Definition: Wortproblem für Reguläre Sprachen

Gegeben: Wort $w \in \Sigma^*$, reguläre Sprache L, repräsentiert durch DFA $\mathcal{A}$, NFA $\mathcal{A}'$ oder RE α

Frage: Ist $w \in L$?

- Der Algorithmus für das Wortproblem und damit der Aufwand hängen von der Repräsentation der Sprache ab:
  - **DFA:** Simuliere den Automaten
    Aufwand: $\mathcal{O}(|w| + |\delta|)$
  - **NFA:** Simuliere den Potenzmengenautomaten, ohne ihn explizit zu konstruieren
    - Speichere dabei immer nur die aktuell erreichte Zustandsmenge
    Aufwand: $\mathcal{O}(|w| \times |\delta|)$
  - **RE:** Wandle den RE in einen ϵ-NFA um und simuliere dann den Potenzmengenautomaten
    Aufwand: $\mathcal{O}(|w| \times |\alpha|)$

Erläuterungen:

- Für die Simulation wird jeweils zuerst die Transitionsfunktion δ in ein Array geschrieben
- Aufwand $\mathcal{O}(|\delta|)$
- Die einzelnen Übergänge können dann durch Nachschauen in der Tabelle ausgeführt werden
- Bei der Simulation von NFAs sind die Tabelleneinträge jeweils Mengen von Zuständen

Algorithmen: Äquivalenztest für DFAs (1/3)

Definition: Äquivalenzproblem für DFAs

Gegeben: DFAs $\mathcal{A}_1$ und $\mathcal{A}_2$

Frage: Ist $L(\mathcal{A}_1) = L(\mathcal{A}_2)$?

- Wir betrachten zwei Lösungsmethoden:
  (1) Mit Minimalautomaten
  (2) Mit dem Produktautomaten

(1) **Mit Minimalautomaten:**

- Konstruiere die Minimal-Automaten: $\mathcal{A}'_1$ und $\mathcal{A}'_2$ zu $\mathcal{A}_1$ und $\mathcal{A}_2$
- Teste, ob $\mathcal{A}'_1$ und $\mathcal{A}'_2$ isomorph sind:
  - ∗ Konstruiere dazu schrittweise eine Bijektion π von (den Zuständen von) $\mathcal{A}'_1$ auf $\mathcal{A}'_2$
  - ∗ Initialisierung: π bildet Startzustand auf Startzustand ab
  - ∗ Dann: Setze π gemäß der Transitionen von $\mathcal{A}'_1$ und $\mathcal{A}'_2$ fort
- Aufwand: $\mathcal{O}(|\Sigma|(|Q_1|^2 + |Q_2|^2))$

(2) **Mit dem Produktautomaten:**

- Konstruiere den Produktautomaten $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$ mit der akzeptierende Menge $F \stackrel{\text{def}}{=} \{(p_1, p_2) \mid (p_1 \in F_1, p_2 \notin F_2)$ oder $(p_1 \notin F_1, p_2 \in F_2)\}$

➡ $\delta^*_{\mathcal{A}}((s_1, s_2), w) \in F$ genau dann, wenn w genau von einem der beiden Automaten akzeptiert wird

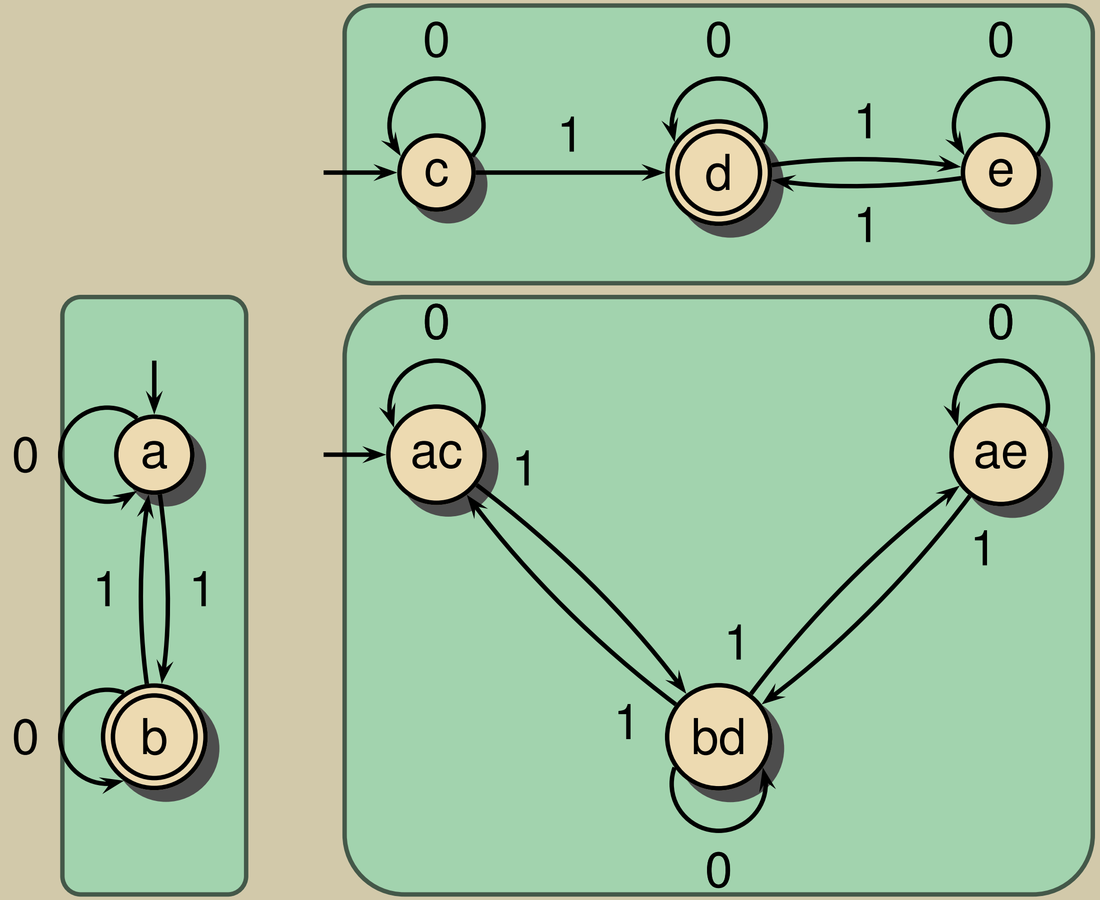
- Also: $L(\mathcal{A}_1) = L(\mathcal{A}_2)$ genau dann, wenn $L(\mathcal{A}) = \varnothing$
- Die Frage „Ist $L(\mathcal{A}_1) = L(\mathcal{A}_2)$?“ kann dann also durch den Leerheitstest für $\mathcal{A}$ entschieden werden

→ Aufwand: $\mathcal{O}(|Q_1| \times |Q_2| \times |\Sigma|)$

Algorithmen: Äquivalenztest für DFAs (2/3)

Beispiel

- Sind diese beiden DFAs äquivalent?



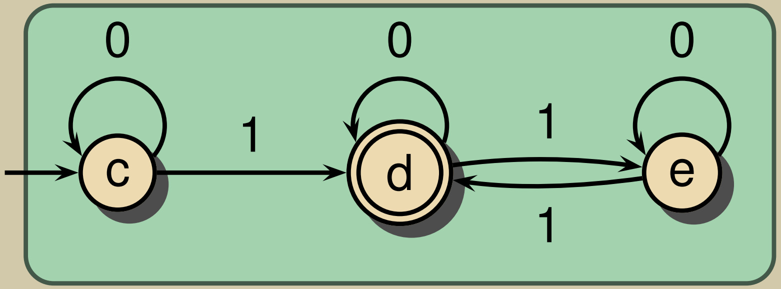
- Berechnung des Produktautomaten
- Nicht erreichbare Zustände entfernen
- Die Sprache des Automaten ist leer

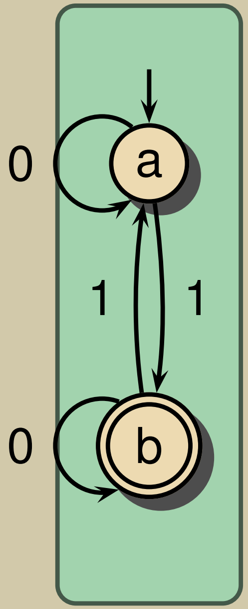
➡ Die Automaten sind äquivalent

Algorithmen: Äquivalenztest für DFAs (3/3)

Beispiel

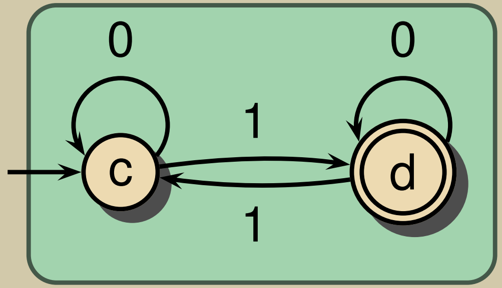
- Sind diese beiden DFAs äquivalent?





Beispiel (Forts.)

- Minimierung des oberen Automaten liefert:



- Beide Automaten sind nun isomorph gemäß
  - $a \mapsto c$
  - $b \mapsto d$

- Noch eine weitere Methode:
  - Führe den Markierungsalgorithmus auf „$\mathcal{A}_1 \cup \mathcal{A}_2$“ aus und überprüfe, ob (s_1, s_2) markiert wird

Algorithmen: Äquivalenztest für NFAs und REs

- Äquivalenztests für NFAs und REs sind zwar auch automatisierbar, aber die Komplexität ist erheblich größer
- Genauer: Zu testen, ob zwei reguläre Ausdrücke (oder zwei NFAs) äquivalent sind ist vollständig für die Komplexitätsklasse **PSPACE**
  - Das gilt sogar, wenn einer der REs gleich Σ^* ist
  - Intuitiver Grund: Die REs müssen zuerst in DFAs umgewandelt werden

- Was „vollständig für **PSPACE**“ bedeutet, werden wir im letzten Teil der Vorlesung sehen
- Hier lässt sich schon sagen: das Problem ist (wohl) noch schwieriger als **NP**-vollständige Probleme wie das Traveling Salesman Problem

Algorithmen: Endlichkeitstest

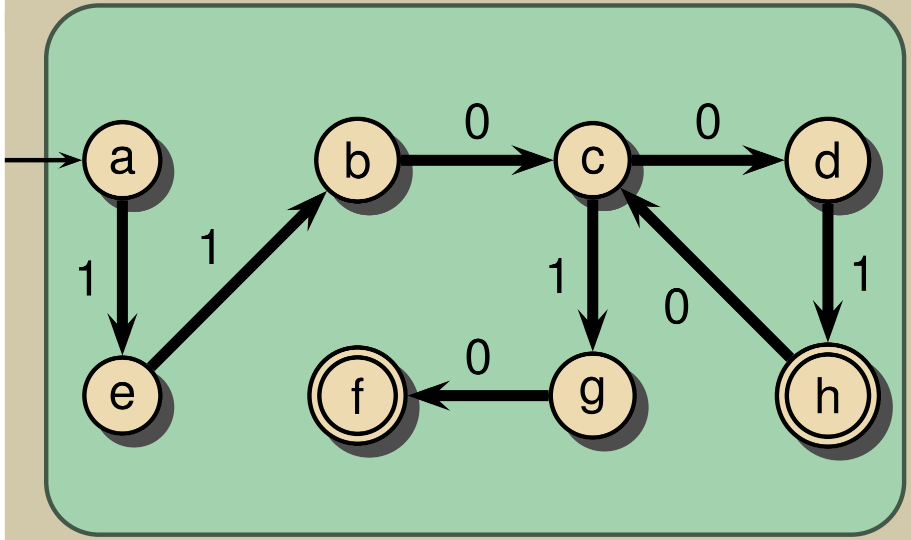
Definition: Endlichkeitsproblem für DFAs

Gegeben: DFA $\mathcal{A}$

Frage: Ist $L(\mathcal{A})$ endlich?

- Hier hilft uns die Grundidee des Pumping-Lemmas weiter:

Beispiel



Satz 5.8

- Die Sprache eines DFA $\mathcal{A}$ ist genau dann unendlich, wenn $\mathcal{A}$ einen Zustand q mit den folgenden Eigenschaften hat:

  (a) q ist von s aus erreichbar
  d.h.: $\exists x \in \Sigma^* : \delta^*(s, x) = q$

  (b) q liegt auf einem Kreis
  d.h.: $\exists y \in \Sigma^* : y \neq \epsilon$ und $\delta^*(q, y) = q$

  (c) Von q aus ist ein akzeptierender Zustand erreichbar d.h. $\exists z \in \Sigma^* : \delta^*(q, z) \in F$

Beweisidee

„$\Leftarrow$“ Dann werden die unendlich vielen Strings $xy^0z, xy^1z, xy^2z, \ldots$ von dem DFA akzeptiert

„$\Rightarrow$“ Die Existenz eines solchen Zustandes q folgt wie im Beweis des Pumping Lemmas

- Aufwand: $\mathcal{O}(|\delta|)$ (Doppelte DFS-Suche)
- Funktioniert auch für NFAs

Zusammenfassung

- Um Automaten und reguläre Ausdrücke anwenden zu können (zum Beispiel im Model Checking), benötigen wir Algorithmen für die Synthese und zum Testen von Eigenschaften regulärer Sprachen

- Synthese:
  - Die regulären Sprachen sind **unter vielen Operationen abgeschlossen**
  - In vielen Fällen lassen sich die entsprechenden Zielautomaten effizient berechnen
  - Für Boolesche Operationen spielen **Produktautomaten** eine wichtige Rolle

- Test von Eigenschaften:
  - **Leerheit** und **Endlichkeit** der Sprache eines NFA können effizient getestet werden - dabei wird im Wesentlichen ein Erreichbarkeitsproblem für gerichtete Graphen gelöst
  - **Äquivalenz zweier DFAs** kann ebenfalls effizient getestet werden
  - **Äquivalenz von NFAs und REs** ist im allgemeinen (wohl) erheblich schwieriger zu testen

- Das Pumping-Lemma liefert ein weiteres Verfahren zum Nachweis, dass eine Sprache nicht regulär ist