Reguläre Sprachen

Hans U. Simon (RUB)

Email: simon@lmi.rub.de

Homepage: http://www.ruhr-uni-bochum.de/lmi

Endliche Automaten

Merkmale endlicher Automaten:

- Speicher, genannt Zustandsmenge (oder auch endliche Kontrolle), hat „konstante Größe“ (d.h., er wächst nicht mit der Länge der Eingabe).
- Eingabewort wird zeichenweise von links nach rechts abgearbeitet.
- Pro Zeichen erfolgt ein „Zustandswechsel“.
- Nach Abarbeitung wird die Eingabe akzeptiert oder verworfen.
- Menge der akzeptierten Eingabeworte bildet die Sprache des Automaten.

Bezeichnungen:

DFA	=	Deterministic Finite Automaton
	=	deterministischer endlicher Automat
NFA	=	Nondeterministic Finite Automaton
	=	nicht–deterministischer endlicher Automat

Veranschaulichung

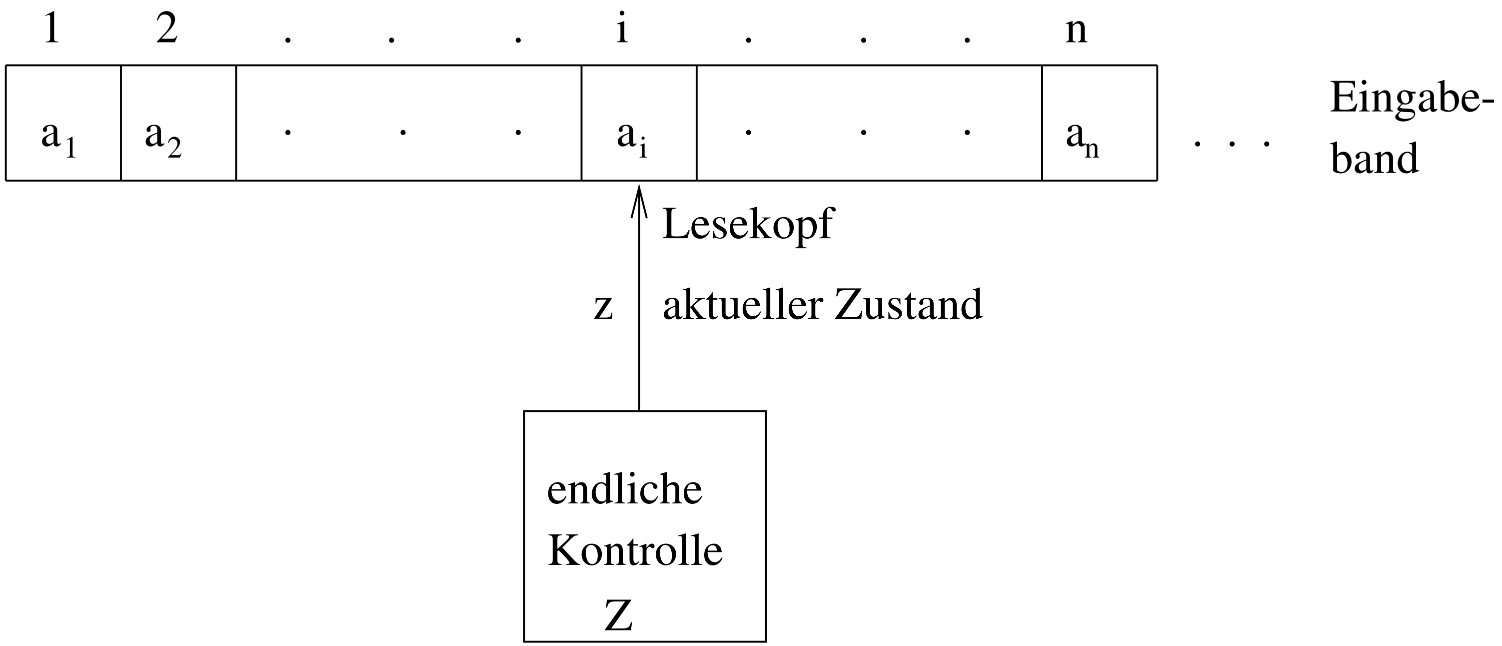


Abbildung 1: DFA mit Eingabeband (in „Zellen“ unterteilt), Lesekopf und endlicher Kontrolle.

Nach Verarbeitung von a_i erfolgt ein Zustandswechsel und der Lesekopf rückt eine Position nach rechts.

DFA (formale Definition)

Ein DFA M besteht aus den folgenden Komponenten:

- Z, die Zustandsmenge
- Σ, das Eingabealphabet
- $\delta : Z \times \Sigma \rightarrow Z$, die Überführungsfunktion
- $z_0 \in Z$, der Startzustand
- $E \subseteq Z$, die Menge der Endzustände

Arbeitsweise:

- Anfangs befindet sich M im Startzustand z_0.
- Nach Verarbeitung des Zeichens a im Zustand z geht M in den Zustand $z' = \delta(z, a)$ über.
- Nach Verarbeitung des letzten Zeichens der Eingabe wird diese genau dann akzeptiert wenn M sich in einem Endzustand befindet.

Beispiel

Betrachte den DFA M mit den folgenden Komponenten:

- $Z = \{z_0, z_1, z_2, z_3\}$.
- $\Sigma = \{a, b\}$.
- Startzustand ist z_0.
- $E = \{z_3\}$.
- δ ist gegeben durch folgende Tabelle:

δ	z_0	z_1	z_2	z_3
a	z_1	z_2	z_3	z_0
b	z_3	z_0	z_1	z_2

Eingabe *aaabbaaab* führt zur Zustandsfolge $z_0, z_1, z_2, z_3, z_2, z_1, z_2, z_3, z_0, z_3$. Akzeptiere!

Eingabe *bbabb* führt zur Zustandsfolge $z_0, z_3, z_2, z_3, z_2, z_1$. Verwerfe!

Visualisierung am Zustandsgraphen

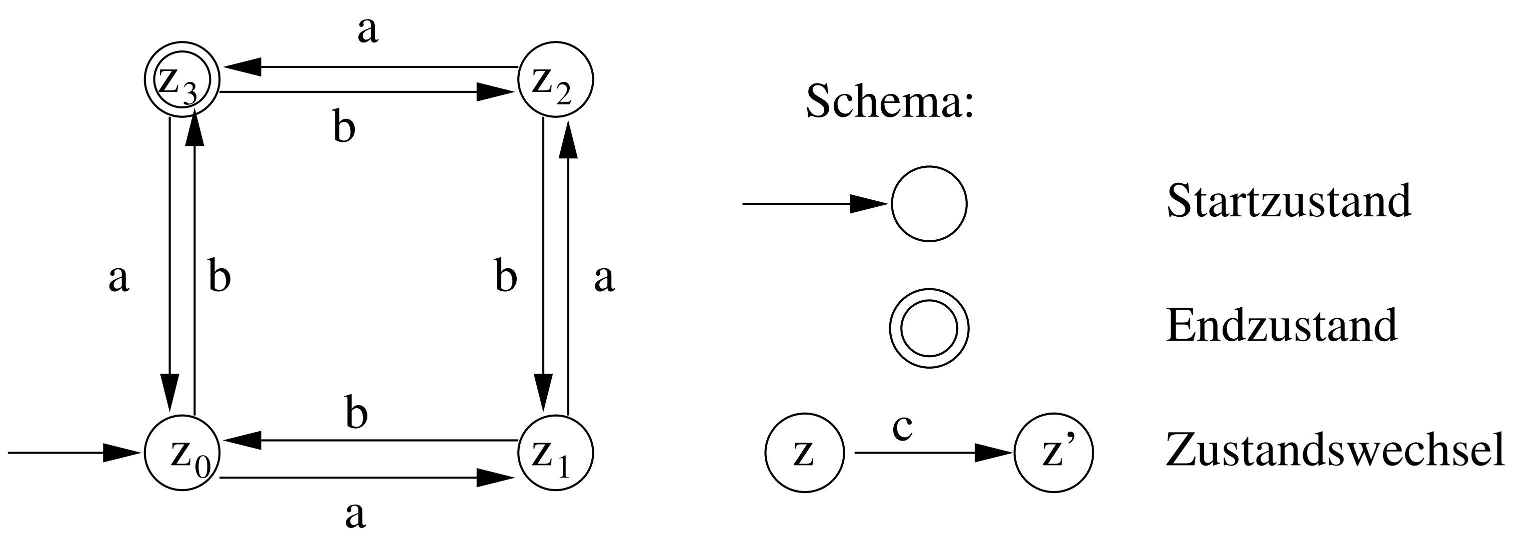


Abbildung 2: der Zustandsgraph G_M zum Beispiel–DFA M.

Rechnungen an Eingaben wie zum Beispiel $aaabbaaab$ oder $bbabb$ entsprechen Pfaden im Zustandsgraphen.

Merke wohl: Da $\delta : Z \times \Sigma \rightarrow Z$ den Folgezustand eindeutig festlegt, gibt es zu jeder Eingabe w einen eindeutigen (von z_0 startenden) Pfad P_w in G_M.

Die von einem DFA akzeptierte Sprache

Die vom DFA M akzeptierte Sprache ist gegeben durch

$$T(M) := \{w \in \Sigma^* | \text{ Pfad } P_w \text{ durch } G_M \text{ endet in einem Endzustand}\} \ .$$

Scharfes Hinsehen beim Zustandsgraph liefert für den Beispiel–DFA:

$$T(M) := \{w \in \{a,b\}^* | \ |w|_a - |w|_b \equiv 3 \pmod 4\} \ .$$

Die Funktion „delta–Hut“

Intuition: $\hat{\delta}(z,w)$ soll der Zustand sein, in dem der DFA sich befindet, wenn er gestartet im Zustand z das Wort w (zeichenweise) abgearbeitet hat.

Induktive Definition: Für alle $z \in Z$, $a \in \Sigma$, $x \in \Sigma^*$:

1. $\hat{\delta}(z,\varepsilon) := z$.
2. $\hat{\delta}(z,ax) := \hat{\delta}(\delta(z,a),x)$.

Auf $Z \times \Sigma$ stimmt $\hat{\delta}$ mit δ überein:

$$\hat{\delta}(z,a) = \hat{\delta}(\delta(z,a),\varepsilon) = \delta(z,a)$$

Es gilt:

$$T(M) = \{w \in \Sigma^* |\ \hat{\delta}(z_0,w) \in E\}$$

Elegantere Notation

Definiere für $z \in Z$ und $w \in \Sigma^*$:

$$z \cdot w := \hat{\delta}(z, w)$$

Dann gilt eine Art „Assoziativgesetz“ für alle $z \in Z, x, y \in \Sigma^*$:

$$z \cdot (x \cdot y) = (z \cdot x) \cdot y$$

- $z \cdot (x \cdot y)$ ist der Zustand, den der DFA erreicht, wenn er gestartet in z das Wort xy (Konkatenation von x und y) zeichenweise abarbeitet.
- Der Ausdruck $(z \cdot x) \cdot y$ liefert dasselbe Ergebnis, wobei $z \cdot x$ der Zwischenzustand ist, den der DFA nach Verarbeitung von x (aber noch vor der Verarbeitung von y) inne hat.

Vom DFA zur regulären Grammatik

Satz: Jeder DFA $M = (Z, \Sigma, \delta, z_0, E)$ lässt sich in eine reguläre Grammatik $G = (V, \Sigma, P, S)$ mit $L(G) = T(M)$ transformieren.

Folgerung: Jede von einem DFA akzeptierte Sprache ist regulär.

Technische Vereinfachung: Wir setzen im Beweis des Satzes $z_0 \notin E$ (gleichbedeutend zu $\varepsilon \notin T(M)$) voraus.

Randnotiz: Im Falle $z_0 \in E$ müsste die im Beweis enthaltene Konstruktion von G um die Regel $z_0 \rightarrow \varepsilon$ erweitert werden (wobei z_0 evtl. auch auf der rechten Seite von anderen Regeln vorkommen kann).

Vom DFA zur regulären Grammatik (fortgesetzt)

Konstruktiver Beweis: Setze $V := Z$, $S := z_0$ und definiere P wie folgt.

- Für jede Transition $\delta(z, a) = z'$ nimm in P die Regel $z \rightarrow az'$ auf.
- Falls dabei $z' \in E$, dann nimm zusätzlich die Regel $z \rightarrow z'$ in P auf.

Für alle $w = a_1 \cdots a_n \in \Sigma^+$ (mit $a_i \in \Sigma$) sind die folgenden Aussagen äquivalent:

1. $w \in T(M)$.
2. Es gibt eine Folge $z_0, \ldots, z_n$ von Zuständen mit Startzustand z_0, $z_n \in E$ und $z_i = \delta(z_{i-1}, a_i)$.
3. Es gibt eine Folge $z_0, \ldots, z_{n-1}$ von Variablen mit Startvariable z_0 und
$$z_0 \Rightarrow_G a_1 z_1 \Rightarrow_G a_1 a_2 z_2 \Rightarrow_G \cdots \Rightarrow_G a_1 \cdots a_{n-1} z_{n-1} \Rightarrow_G a_1 a_2 \cdots a_{n-a} a_n \ .$$
4. $w \in L(G)$.

Nicht-deterministische Zustandswechsel

Bei NFA's kann der Zustandsgraph folgenden Ausschnitt enthalten:

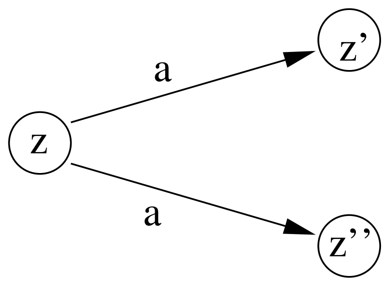


Abbildung 3: Wer die Wahl hat, hat die Qual.

- I.A. existieren mehrere mögliche Zustandswechsel pro Rechenschritt (und auch mehrere Startzustände).
- Zu einer Eingabe gibt es i.A. viele korrespondierende Pfade durch den Zustandsgraphen.
- Die Eingabe gilt als akzeptiert, wenn mindestens einer der korrespondierenden Pfade in einem Endzustand endet (in Analogie zu Grammatiken, wo eine erfolgreiche Ableitung genügt).

NFA (formale Definition)

$\mathcal{P}(Z)$ bezeichnet die Potenzmenge von Z (Menge aller Teilmengen von Z).

Ein NFA M besteht aus den folgenden Komponenten:

- Z, die Zustandsmenge
- Σ, das Eingabealphabet
- $\delta : Z \times \Sigma \rightarrow \mathcal{P}(Z)$, die Überführungsfunktion
- $S \in Z$, die Menge der Startzustände
- $E \subseteq Z$, die Menge der Endzustände

Arbeitsweise des NFA

- Anfangs befindet sich M (wahlweise) in einem der Startzustände.
- Nach Verarbeitung des Zeichens a im Zustand z kann M (wahlweise) in einen Zustand $z' \in \delta(z, a)$ übergehen.
- Die Eingabe wird akzeptiert genau dann wenn M durch geeignete Wahl eines Startzustandes und durch geeignete Zustandswechsel nach Abarbeitung der Eingabe einen Endzustand erreichen kann.

Die von einem NFA akzeptierte Sprache

Der Zustandsgraph G_M zu einem NFA M ist völlig analog definiert wie bei DFAs.

Die vom NFA M akzeptierte Sprache besteht (per Definition) aus allen Wörtern $w \in \Sigma^*$ mit folgender Eigenschaft:
Es existiert ein von einem Startzustand ausgehender Pfad in G_M, dessen Kanten (in der korrekten Reihenfolge) mit den Buchstaben von w beschriftet sind und der in einem Endzustand endet.

Beispiel

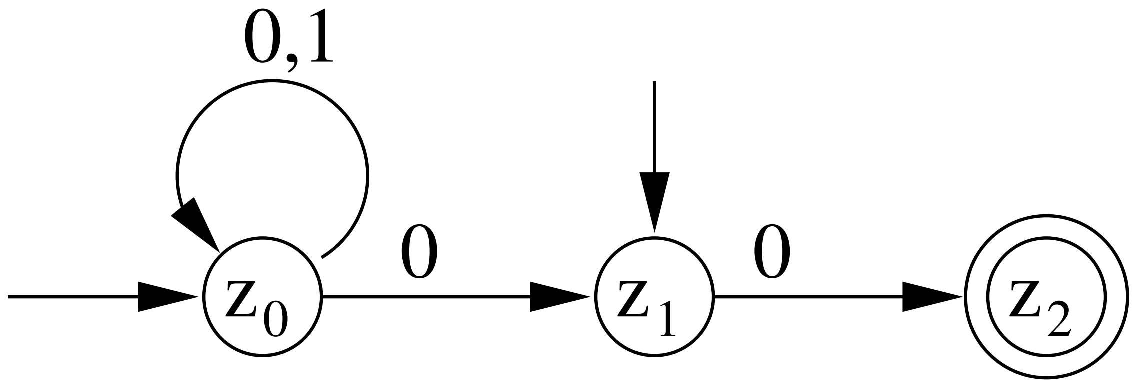


Abbildung 4: NFA M gegeben durch seinen Zustandsgraphen G_M.

Scharfes Hinsehen ergibt:

$$T(M) = \{w \in \{0,1\}^* |\ w = 0 \text{ oder } w \text{ endet mit } 00\}$$

Die Funktion „delta–Hut“

Intuition: Für eine Teilmenge $Z' \subseteq Z$ der Zustände soll $\hat{\delta}(Z', w)$ die Menge der Zustände sein, welche sich ausgehend von einem der Zustände in Z' erreichen lassen, wenn w zeichenweise abgearbeitet wird.

Induktive Definition: Für alle $Z' \subseteq Z$, $a \in \Sigma$, $x \in \Sigma^*$:

1. $\hat{\delta}(Z', \varepsilon) := Z'$.
2. $\hat{\delta}(Z', ax) := \bigcup_{z \in Z'} \hat{\delta}(\delta(z, a), x)$.

Dann gilt:

$$T(M) = \{w \in \Sigma^* |\ \hat{\delta}(\{S\}, w) \cap E \neq \emptyset\}$$

Vom NFA zum DFA

Satz: Jeder NFA $M = (Z, \Sigma, \delta, S, E)$ lässt sich in einen DFA $M' = (Z', \Sigma, \delta', z_0, E')$ mit $T(M') = T(M)$ transformieren.

Beweis (Potenzmengenkonstruktion): Wir wählen $Z' = \mathcal{P}(Z)$, d.h., jeder „Makrozustand" von M' ist eine Teilmenge der Zustände von M. M' soll M in folgendem Sinn simulieren:

$(*)$ Zu jedem Zeitpunkt der „Rechnung" soll M' in dem „Makrozustand" sein, der übereinstimmt mit der Menge der gegenwärtig von M erreichbaren Zustände.

Formal ist hierzu zu zeigen, dass

$$\hat{\delta}'(Z', w) = \hat{\delta}(Z', w)$$

fur alle $Z' \subseteq Z$ und $w \in \Sigma^*$.

Vom NFA zum DFA (fortgesetzt)

Hierzu definieren wir die restlichen Komponenten von M' wie folgt:

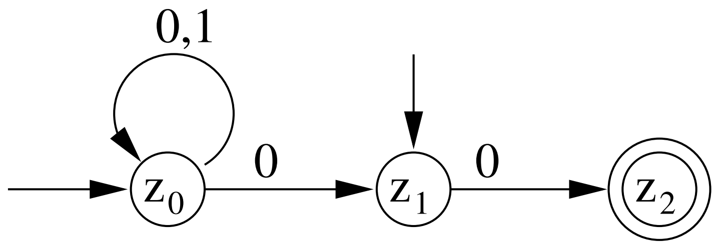
1. $z_0 = S$. (Damit gilt $(*)$ zu Anfang.)
2. $\delta'(Z', a) = \bigcup_{z \in Z'} \delta(z, a)$. (Damit bleibt $(*)$ während Verarbeitung des nächsten Zeichens richtig).
3. $E' = \{Z' \subseteq Z | \; Z' \cap E \neq \emptyset\}$.

Formal lässt sich aus den ersten beiden Bedingungen induktiv folgern:

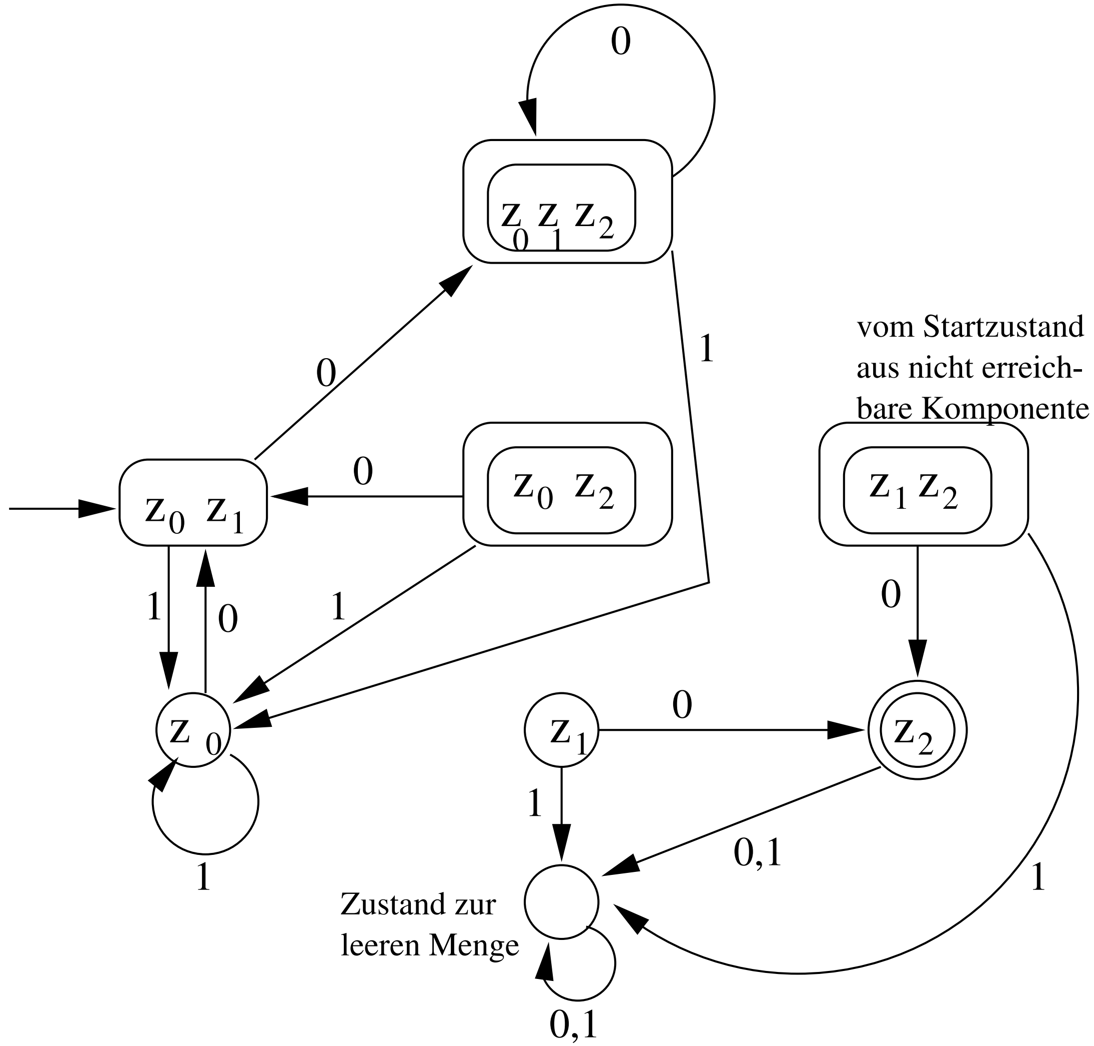
$$\hat{\delta}'(Z', w) = \hat{\delta}(Z', w) \quad .$$

Aus der Wahl von E' folgt dann, dass $T(M') = T(M)$.

Beispiel

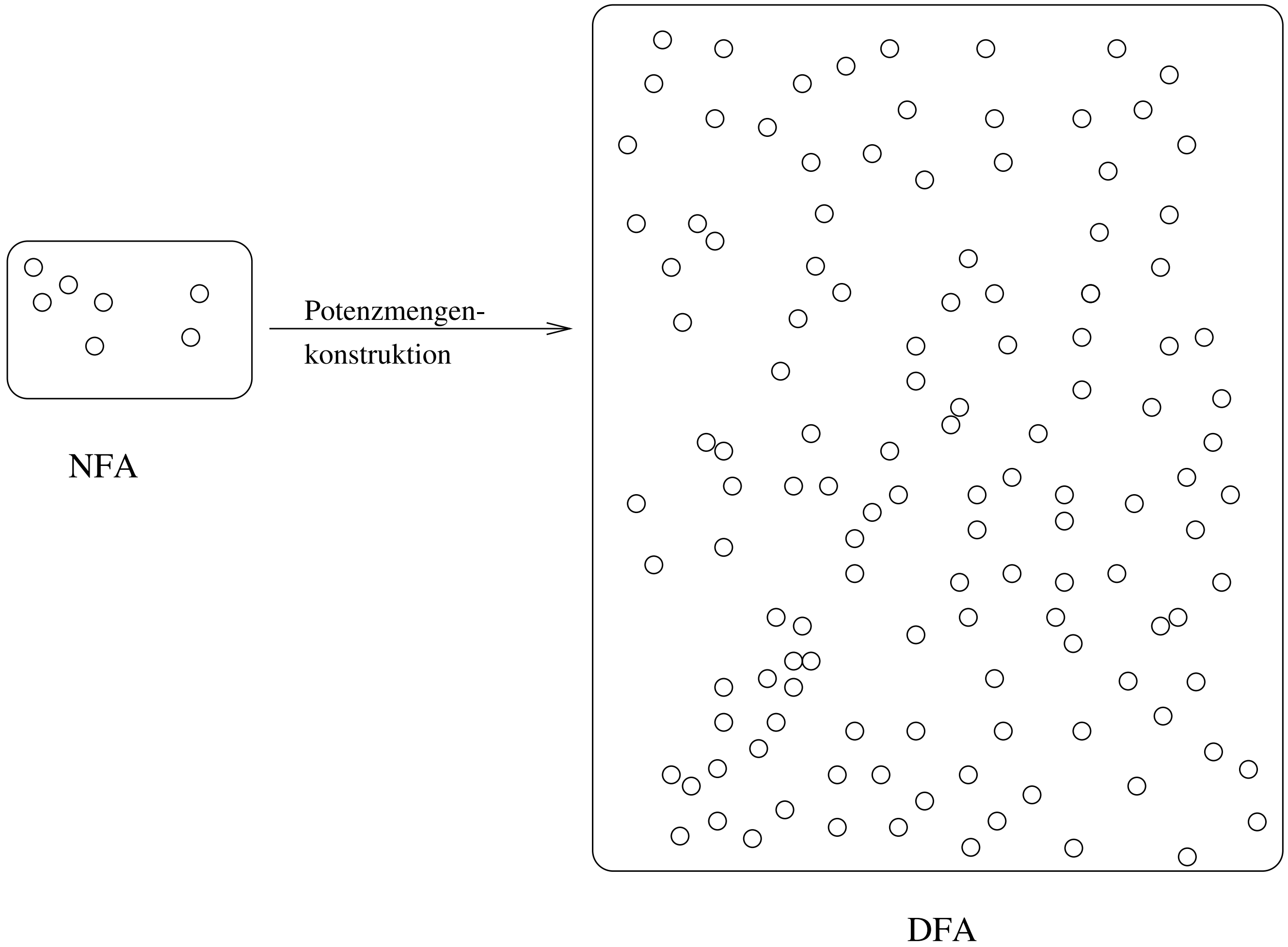


NFA M



simulierender DFA M’

Die „kombinatorische Explosion“ ...



Frage: Gibt es effizientere Simulationen ? **Antwort:** i.A. Nein !

Eine Sprache mit „kleinem“ NFA ...

Die Sprache

$$L_n = \{w \in \{0,1\}^* |\ |w| \geq n \text{ und das } n\text{–letzte Zeichen von } w \text{ ist „0“}\}$$

kann von einem NFA mit $n+1$ Zuständen erkannt werden:

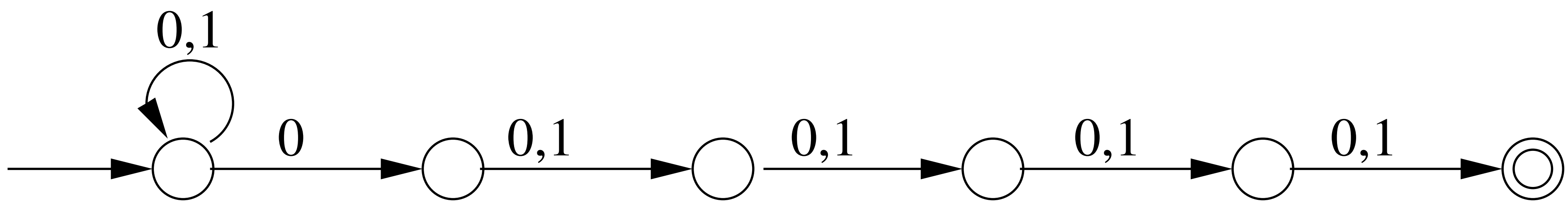


Abbildung 5: Ein NFA für L_5.

... die große DFAs benötigt

Angenommen es gäbe einen DFA M für L_n mit weniger als 2^n Zuständen?

Widerspruch (in 5 Schritten):

1. Dann gibt es 2 verschiedene Wörter $x, y \in \{0,1\}^n$ mit $z_0 \cdot x = z_0 \cdot y$ (Schubfachprinzip).
2. Betrachte eine Bit-Position i, in der x und y sich unterscheiden. Sagen wir: Bit i von x ist 0 und Bit i von y ist 1.
3. Betrachte $x' = x0^{i-1}$ und $y' = y0^{i-1}$. Bit i von x (bzw. von y) ist das n-letzte Bit von x' (bzw. von y'). Daher ist $x' \in L_n$ und $y' \notin L_n$.
4. Wegen
$$z_0 \cdot (x \cdot 0^{i-1}) = (z_0 \cdot x) \cdot 0^{i-1} = (z_0 \cdot y) \cdot 0^{i-1} = z_0 \cdot (y \cdot 0^{i-1})$$
gelangt M bei den Eingaben x' und y' zum gleichen Endzustand.
5. M macht also entweder auf Eingabe x' oder auf Eingabe y' einen Fehler.

Kleines Gedächtnis, große Fehler

Beim Widerspruchsbeweis wurde ausgenutzt:

- Wenn ein DFA zwei Eingaben x, y identifiziert (im Sinne von $z_0 \cdot x = z_0 \cdot y$), dann kann er auch Eingaben der Form xu und yu nicht auseinander halten.
- Wenn ein DFA ein zu kleines „Gedächtnis“ (= Zustandsmenge) hat, dann identifiziert er evtl. Eingaben, die er besser unterscheiden können sollte.

Vom NFA zum DFA: Zusammenfassung

- Jeder NFA mit n Zuständen lässt sich in einen äquivalenten DFA mit 2^n Zuständen transformieren.
- Es gibt Sprachen L_n, $n \geq 1$, mit n Zuständen, die von einem NFA mit $n+1$ Zuständen erkannt werden können, aber nicht von einem DFA mit weniger als 2^n Zuständen.

Andererseits kann man zeigen, dass L_n von einem DFA mit 2^n Zuständen erkannt werden kann (s. Übung).

Von der regulären Grammatik zum NFA

Satz: Jede reguläre Grammatik $G = (V, \Sigma, P, S)$ lässt sich in einen NFA $M = (Z, \Sigma, \delta, S', E)$ mit $T(M) = L(G)$ transformieren.

Zusammen mit den früheren Ergebnissen erhalten wir den Zirkelschluss

$$\text{DFA} \rightarrow \text{reguläre Grammatik} \rightarrow \text{NFA} \rightarrow \text{DFA} \quad .$$

Es ergibt sich die

Folgerung: Die Klasse der regulären Sprachen kann wahlweise durch DFAs, NFAs oder reguläre Grammatiken repräsentiert werden.

Konstruktiver Beweis: Wähle die Komponenten von M wie folgt:

- $Z = V \cup \{X\}$ und $S' = \{S\}$.
- Falls $S \to \varepsilon \notin P$, dann setze $E = \{X\}$; andernfalls $E = \{S, X\}$.
- Für jede Regel der Form $A \to aB$ in P nimm B in $\delta(A, a)$ auf.
- Für jede Regel der Form $A \to a$ in P nimm X in $\delta(A, a)$ auf.

Für alle $w = a_1 \cdots a_n \in \Sigma^+$ (mit $a_i \in \Sigma$) sind die folgenden Aussagen äquivalent:

1. $w \in L(G)$.
2. Es gibt eine Folge $A_1, \ldots, A_{n-1}$ von Variablen mit
$$S \Rightarrow_G a_1 A_1 \Rightarrow_G a_1 a_2 A_2 \Rightarrow_G \cdots \Rightarrow_G a_1 a_2 \cdots a_{n-1} A_{n-1} \Rightarrow_G a_1 a_2 \cdots a_{n-1} a_n \ .$$
3. Es gibt eine Folge von Zuständen $A_1, \ldots, A_{n-1}$ mit
$$A_1 \in \delta(S, a_1),\ A_2 \in \delta(A_1, a_2), \ldots,\ X \in \delta(A_{n-1}, a_n) \ .$$
4. $w \in T(M)$.

Kennen Sie den Unterschied
zwischen Zeichen und Bezeichnetem ?

Reguläre Ausdrücke

Induktive Definition der „Syntax" regulärer Ausdrücke:

1. $\emptyset$, ε und a für jedes $a \in \Sigma$ sind reguläre Ausdrücke.
2. Wenn α und β reguläre Ausdrücke sind, dann sind auch $\alpha \cdot \beta$, $\alpha|\beta$ und $(\alpha)^*$ reguläre Ausdrücke.

Konkatenationszeichen „$\cdot$" darf auch weggelassen werden.

$L(\alpha)$ bezeichne die durch den regulären Ausdruck α repräsentierte Sprache im Sinne der folgenden Definition.

Induktive Definition der „Semantik" regulärer Ausdrücke:

1. $L(\emptyset) = \emptyset$; $L(\varepsilon) = \{\varepsilon\}$; $L(a) = \{a\}$.
2. $L(\alpha \cdot \beta) = L(\alpha) \cdot L(\beta)$; $L((\alpha|\beta)) = L(\alpha) \cup L(\beta)$; $L((\alpha)^*) = L(\alpha)^*$.

$\emptyset, \varepsilon, a, \cdot, *$ haben also die offensichtliche Interpretation; Zeichen „$|$" wird als Vereinigung „$\cup$" interpretiert.

Beispiele

Die uns schon bekannte Sprache

$$\{w \in \{0,1\}^* |\ w = 0 \text{ oder } w \text{ endet mit } 00\}$$

ist mit regulären Ausdrücken „traumhaft elegant“ beschreibbar:

$$(0|(0|1)^*00) \ .$$

Jede endliche Sprache ist durch einen regulären Ausdruck beschreibbar. Denn:

1. Jedes einzelne Wort ergibt sich aus Konkatenation seiner (endlich vielen) Zeichen.

2. Jede endliche Sprache ist eine Vereinigung ihrer (endlich vielen) Wörter.

3. Konkatenation und Vereinigung sind Operatoren, die regulären Ausdrücken zur Verfügung stehen.

Exkurs: Induktion

- Vollständige Induktion nach einem Parameter $n \in \mathbb{N}$?
  Bestimmt schon bekannt !

- Vollständige Induktion nach der Struktur eines Objektes ?
  Wichtig, wichtig ...
  Kommt gleich zum Einsatz !

Von regulären Ausdrücken zu NFAs

Satz: Jeder reguläre Ausdruck γ kann in einen NFA M mit $T(M) = L(\gamma)$ transformiert werden.

Beweis: (Induktion nach der Struktur regulärer Ausdrücke)
Zu zeigen:

1. Zu den einfachen regulären Ausdrücken '$\emptyset, \varepsilon, a$ mit $a \in \Sigma$ finden wir jeweils einen passenden NFA.
2. Aus passenden NFAs M_1 und M_2 für reguläre Ausdrücke α und β können wir jeweils einen passenden NFA für $(\alpha|\beta)$, $\alpha \cdot \beta$ und $(\alpha)^*$ zusammenbasteln (ein „Syntheseproblem“).

Passende NFAs für einfache reguläre Ausdrücke

Endlich mal eine babyleichte Aufgabe:

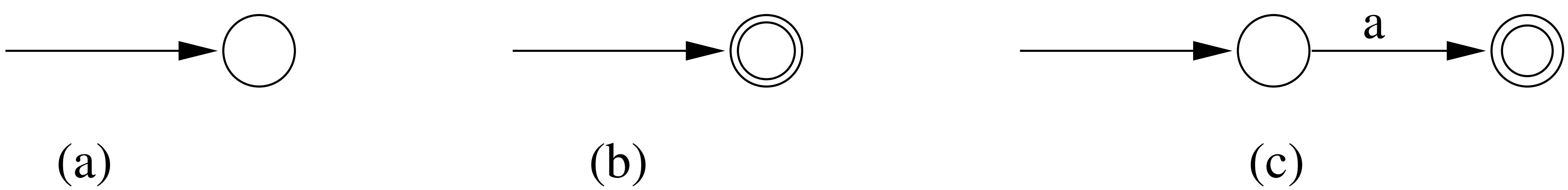


Abbildung 6: (a) ein NFA für $\emptyset$ (b) ein NFA für $\{\varepsilon\}$ (c) ein NFA für $\{a\}$

Syntheseproblem bei den Operationen $\cup, \cdot$

Grundidee in folgender Abbildung:

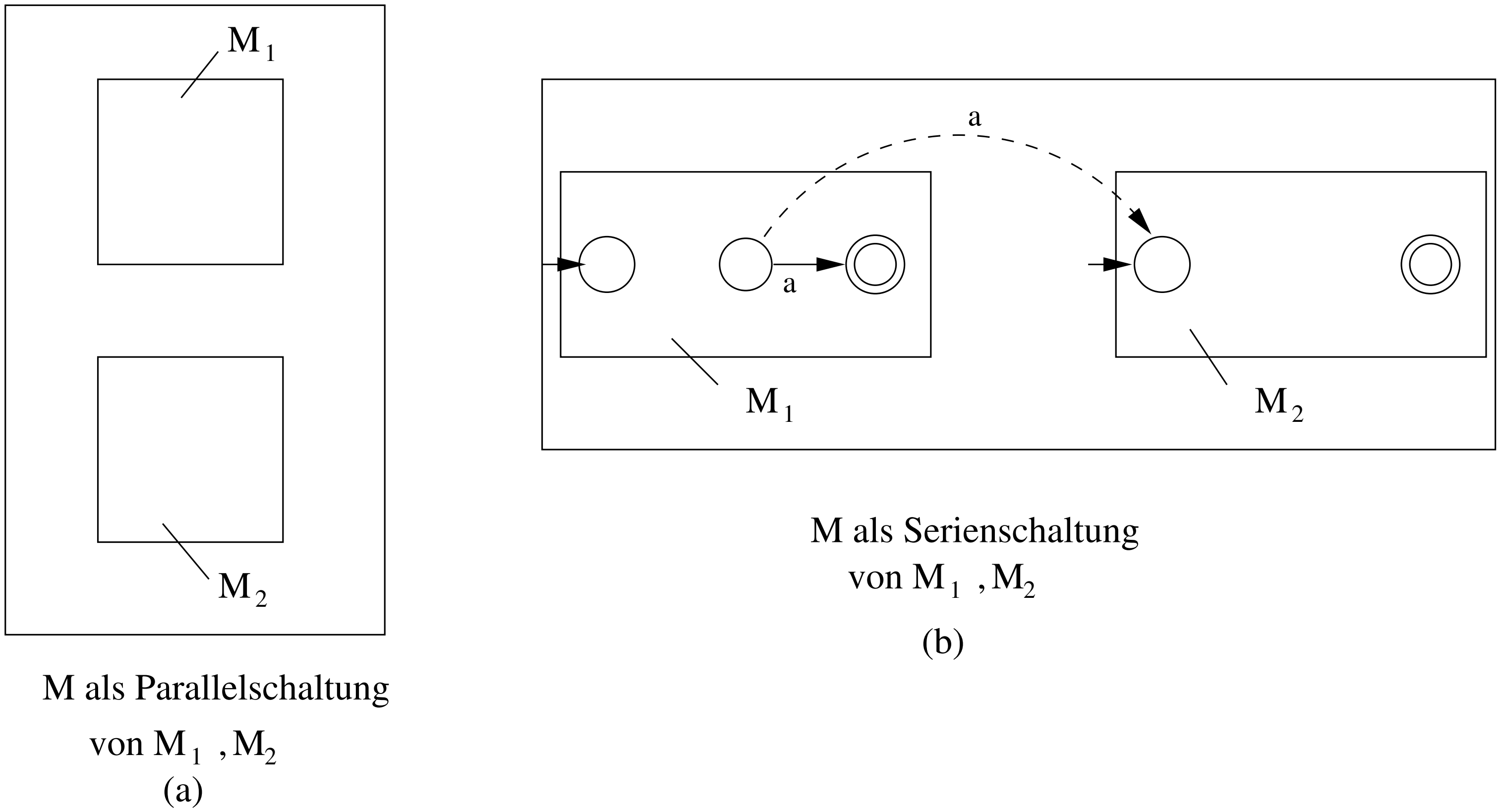


Abbildung 7: (a) ein NFA für $T(M_1) \cup T(M_2)$ (b) ein NFA für $T(M_1) \cdot T(M_2)$

Lösung des Syntheseproblems für „Vereinigung“

Voraussetzung: $T(M_1) = L(\alpha)$, $T(M_2) = L(\beta)$; $M_i = (Z_i, \Sigma, \delta_i, S_i, E_i)$.

Synthese eines Automaten M für $L(\alpha) \cup L(\beta)$:

$$
\begin{aligned}
M &:= \{Z_1 \cup Z_2, \Sigma, \delta, S_1 \cup S_2, E_1 \cup E_2\} \\
\delta(z,a) &:= \begin{cases} \delta_1(z,a) & \text{falls } z \in Z_1 \\ \delta_2(z,a) & \text{falls } z \in Z_1 \end{cases}
\end{aligned}
$$

Lösung des Syntheseproblems für „Konkatenation“

Synthese eines Automaten $M = (Z, \Sigma, \delta, S, E)$ für $L(\alpha) \cdot L(\beta)$:

$$
\begin{aligned}
Z &:= Z_1 \cup Z_2 \\
\delta(z_1, a) &:= \begin{cases} \delta_1(z_1, a) & \text{falls } \delta_1(z_1, a) \cap E_1 = \emptyset \\ \delta_1(z_1, a) \cup S_2 & \text{falls } \delta_1(z_1, a) \cap E_1 \neq \emptyset \end{cases} \\
\delta(z_2, a) &:= \delta_2(z_2, a) \\
S &:= \begin{cases} S_1 & \text{falls } \varepsilon \notin T(M_1) \\ S_1 \cup S_2 & \text{falls } \varepsilon \in T(M_1) \end{cases} \\
E &:= E_2
\end{aligned}
$$

Syntheseproblem bei der Operation $*$

Grundidee in folgender Abbildung:

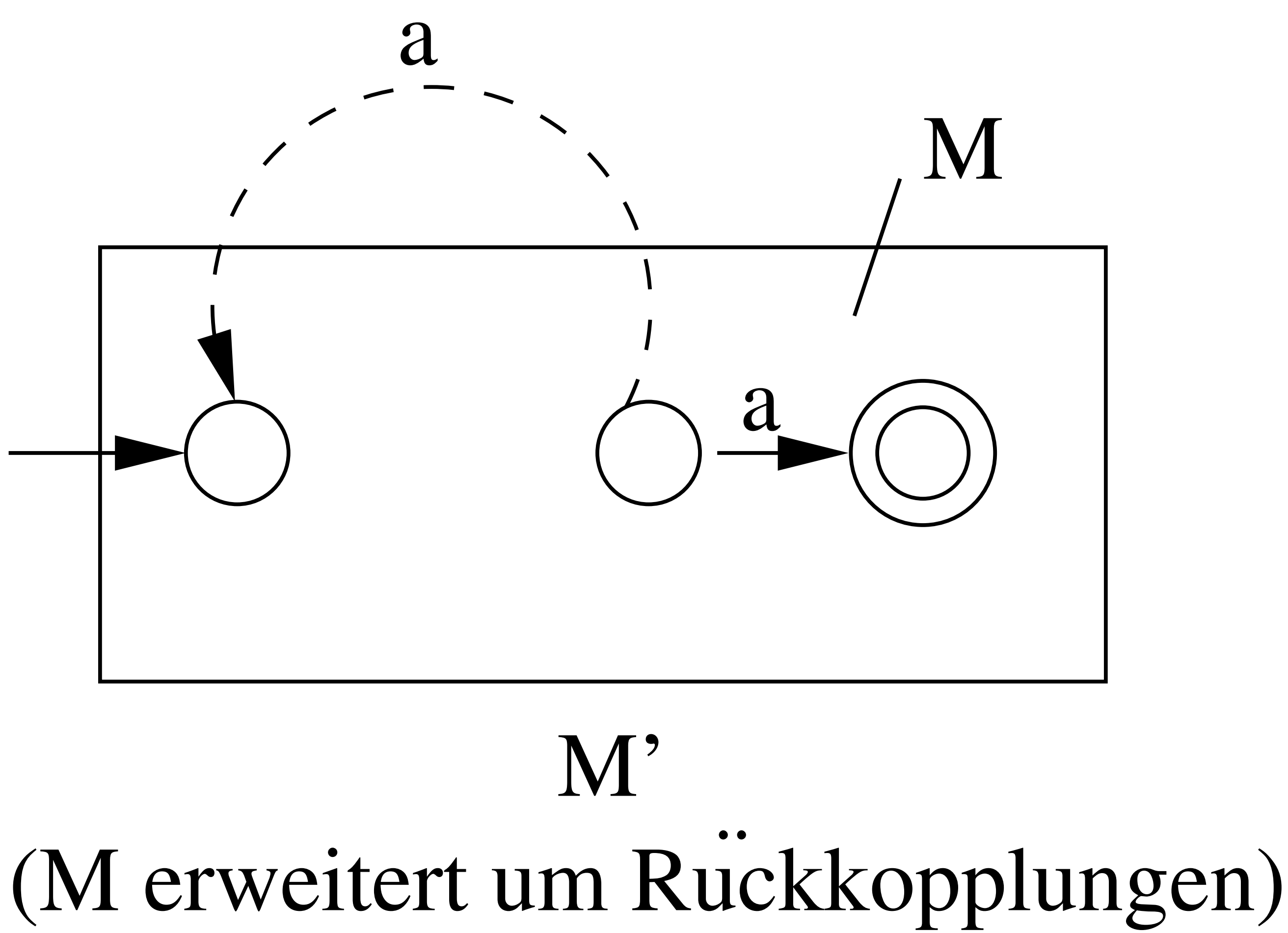


Abbildung 8: ein NFA M' für $T(M)^*$

Lösung des Syntheseproblems für Operation $*$

Voraussetzung: $T(M) = L(\alpha)$; $M = (Z, \Sigma, \delta, S, E)$.

Synthese eines Automaten $M' = (Z', \Sigma, \delta', S', E')$ für $L(\alpha)^*$ für den Fall $\varepsilon \in T(M)$:

$$
\begin{aligned}
Z' &:= Z \\
\delta'(z,a) &:= \begin{cases} \delta(z,a) & \text{falls } \delta_1(z,a) \cap E = \emptyset \\ \delta(z,a) \cup S & \text{falls } \delta(z,a) \cap E_1 \neq \emptyset \end{cases} \\
S' &:= S \\
E' &:= E
\end{aligned}
$$

Falls $\varepsilon \notin T(M)$, dann wird darüberhinaus ein neuer Zustand z_0 in die Mengen Z, S und E aufgenommen.

Von DFAs zu regulären Ausdrücken

Satz: Jeder DFA $M = \{\{z_1, \ldots, z_n\}, \Sigma, \delta, z_1, E)$ kann in einen regulären Auysdruck γ mit $L(\gamma) = T(M)$ transformiert werden.

Beweisidee: Verwende „Hilfssprachen“, die sich

- mit Hilfe der Operationen $\cup, \cdot, *$ aus einzelnen Zeichen zusammenbasteln lassen,
- so dass $T(M)$ sich als Vereinigung (geeignet ausgewählter) Hilfssprachen darstellen lässt.

Umsetzung der Beweisidee

die Hilfssprachen

$R^k_{i,j}$ sei die Menge aller Wörter $x \in \Sigma^*$ mit:

1. M gestartet in z_i ist nach Abarbeitung von x im Zustand z_j.
2. Dabei werden nur Zwischenzustände aus $\{z_1, \dots, z_k\}$ angenommen.

Beachte: Im Falle $k = 0$ sind keine Zwischenzustände erlaubt (Übergang von z_i nach z_j durch maximal einen Zustandswechsel).

Umsetzung der Beweisidee (fortgesetzt)

Die Hilfssprachen haben die gewünschten Eigenschaften:

$$
\begin{aligned}
T(M) &= \bigcup_{j:z_j \in E} R^n_{1,j} \\
R^0_{i,j} &= \{a \in \Sigma \cup \{\varepsilon\} \mid \hat{\delta}(z_i, a) = z_j\} \\
R^k_{i,j} &= R^{k-1}_{i,j} \cup R^{k-1}_{i,k} \cdot (R^{k-1}_{k,k})^* \cdot R^{k-1}_{k,j}
\end{aligned}
$$

Die letzte Gleichung gilt aus folgendem Grund:
Wenn M gestartet in z_i nach Verarbeitung von x im Zustand z_j ist und dabei nur Zwischenzustände aus $\{z_1, \ldots, z_k\}$ angenommen hat, so gilt

entweder wurden nur Zwischenzustände aus $\{z_1, \ldots, z_{k-1}\}$ angenommen

oder Zwischenzustand z_k wurde iteriert (mindestens einmal) angenommen (wobei die von z_k verschiedenen Zwischenzustände wiederum aus $\{z_1, \ldots, z_{k-1}\}$ stammen).

Das Pumping–Lemma (uvw–Theorem)

Satz: Für jede reguläre Sprache $L \subseteq \Sigma^*$ gilt:

$$\exists n \geq 1,$$
$$\forall x \in L \text{ mit } |x| \geq n,$$
$$\exists u, v, w \in \Sigma^* \text{ mit } x = uvw, 1 \leq |v| \leq |uv| \leq n, \quad .$$
$$\forall i \geq 0 \ :$$
$$uv^i w \in L$$

Beweis

Benutze einen DFA $M = (Z, \Sigma, \delta, z_0, E)$, der L erkennt, und wähle $n := |Z|$.

Zu einem Wort $x = a_1, \ldots, a_s \in L$ mit $a_i \in \Sigma$ und $s \geq n$ betrachte den resultierenden „Berechnungspfad“ P im Zustandsgraphen G_M:

$$z'_0 \xrightarrow{a_1} z'_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{s-1}} z'_{s-1} \xrightarrow{a_s} z'_s \text{ wobei } z'_0 = z_0 \text{ und } z'_s \in E \ .$$

Es muss zwei Indizes $0 \leq j < k \leq n$ mit $z'_j = z'_k$ geben (Schubfachprinzip).

Berechnugspfad P enthält daher den Zyklus

$$z'_j \xrightarrow{a_{j+1}} z'_{j+1} \xrightarrow{a_{j+2}} \cdots \xrightarrow{a_{k-1}} z'_{k-1} \xrightarrow{a_k} z'_k \ .$$

Da der Zyklus im Berechnungspfad P iteriert durchlaufen (oder auch weggelassen) werden kann, hat die Zerlegung $x = uvw$ mit

$$u = a_1 \cdots a_j \ , \ v = a_{j+1} \cdots a_k \ , \ w = a_{k+1} \cdots a_s$$

die gewünschten Eigenschaften $1 \leq |v| \leq |uv| \leq n$ und

$$\forall i \geq 0 : uv^i w \in L \ .$$

Anwendung: Nachweis der Nichtregularität

Folgerung: Falls für eine Sprache $L \subseteq \Sigma^*$ die Bedingung

$$\forall n \geq 1,$$
$$\exists x \in L \text{ mit } |x| \geq n,$$
$$\forall u, v, w \in \Sigma^* \text{ mit } x = uvw, 1 \leq |v| \leq |uv| \leq n,$$
$$\exists i \geq 0 \ :$$
$$uv^i w \notin L$$

erfüllt ist, dann ist L nicht regulär.

Beispiele für nicht–reguläre Sprachen

Die folgende Sprache $L \subseteq \{a, b\}^*$ ist nicht regulär:

$$L = \{a^n b^n \mid n \geq 1\}$$

Begründung:

1. Zu beliebig vorgegebenem $n \geq 1$ wähle $x = a^n b^n$.
   Offensichtlich gilt $x \in L$ und $|x| \geq n$.

2. Zu beliebig vorgegebener Zerlegung $x = uvw$ mit $1 \leq |v| \leq |uv| \leq n$ wähle $i = 0$. Beachte, dass uv das Zeichen b nicht enthält (wegen $|uv| \leq n$).

Nun gilt

$$uv^0 w = uw = a^{n-|v|} b^n \notin L \ ,$$

da $|uw|_a < |uw|_b$.

Beispiele für nicht–reguläre Sprachen (fortgesetzt)

Die folgende Sprache $L \subseteq \{0\}^*$ ist nicht regulär:

$$L = \{0^m \mid m \text{ ist eine Quadratzahl}\}$$

Begründung:

1. Zu beliebig vorgegebenem $n \geq 1$ wähle $x = 0^{n^2}$.
   Offensichtlich gilt $x \in L$ und $|x| \geq n$.

2. Zu beliebig vorgegebener Zerlegung $x = uvw$ mit $1 \leq |v| \leq |uv| \leq n$ wähle $i = 2$.

Nun gilt

$$uv^2w = 0^{n^2+|v|} \notin L \ ,$$

da $n^2 + |v|$ wegen

$$n^2 < n^2 + |v| \leq n^2 + n < n^2 + 2n + 1 = (n+1)^2$$

keine Quadratzahl ist.

Beispiele für nicht–reguläre Sprachen (fortgesetzt)

Die folgende Sprache $L \subseteq \{0\}^*$ ist nicht regulär:

$$L = \{0^p |\ p \text{ ist eine Primzahl}\}$$

Begründung:

1. Zu beliebig vorgegebenem $n \geq 1$ wähle $x = 0^p$ für eine Primzahl $p \geq n + 2$. Offensichtlich gilt $x \in L$ und $|x| \geq n$.

2. Zu beliebig vorgegebener Zerlegung $x = uvw$ mit $1 \leq |v| \leq |uv| \leq n$ wähle $i = |uw|$. Beachte, dass $|uw| = p - |uv| \geq (n + 2) - n = 2$.

Nun gilt

$$uv^{|uw|}w = 0^{|uw|(|v|+1)} \notin L \ ,$$

da $|uw|(|v| + 1)$ offensichtlich keine Primzahl ist.

Zwei Relationen auf Wörtern

Definition: M sei ein DFA.

$$xR_My \textbf{ gdw } z_0 \cdot x = z_0 \cdot y \ .$$

intuitiv: M „identifiziert“ die Wörter x und y: nach Verarbeitung von x erreicht er den gleichen Zustand wie nach Verarbeitung von y.

Definition (Nerode–Relation) $L \subseteq \Sigma^*$ sei eine formale Sprache.

$$xR_Ly \textbf{ gdw } (\forall z \in \Sigma^* : xz \in L \Leftrightarrow yz \in L) \ .$$

intuitiv: ein DFA dürfte x und y identifizieren, ohne dass sich daraus Fehler beim Fortsetzen der Rechnung ableiten lassen.

Eigenschaften dieser Relationen

Bemerkungen

1. Beide Relationen sind Äquivalenzrelationen,
   d.h., sie sind reflexiv, symmetrisch und transitiv.

2. Darüberhinaus gilt:

$$\begin{aligned} xR_My &\implies xwR_Myw \\ xR_Ly &\implies xwR_Lyw \end{aligned}$$

Exkurs: Index, Verfeinerung, Vergröberung

R, R_1, R_2 seien Äquivalenzrelationen.

- Index von R = Anzahl der Äquivalenzklassen von R (evtl. ∞).
- R_1 heißt Verfeinerung von R_2 (und R_2 Vergröberung von R_1) **gdw** $aR_1b \Rightarrow aR_2b$.
  In diesem Fall zerfällt jede Äquivalenzklasse von R_2 in Äquivalenzklassen von R_1 und somit gilt:

$$\text{Index von } R_2 \leq \text{ Index von } R_1 \ .$$

Der Satz von Myhill und Nerode

Satz: L ist regulär **gdw** R_L hat einen endlichen Index.

Der Beweis zerfällt in zwei Teile.

1. Beweisrichtung

Lemma: Wenn L regulär ist, dann hat R_L einen endlichen Index.

Beweis in 5 Teilschritten:

- Wähle einen DFA M, der L erkennt (und nur Zustände besitzt, die vom Startzustand z_0 aus erreichbar sind).
- Index von $R_M = |Z|$.
- $z_0 \cdot x = z_0 \cdot y \implies (\forall z \in \Sigma^* : xz \in \Leftrightarrow yz \in L)$.
  Ansonsten würde M auf einer der Eingaben xz und yz einen Fehler machen.
- R_L ist Vergröberung von R_M:

$$xR_My \implies z_0 \cdot x = z_0 \cdot y \implies (\forall z \in \Sigma^* : xz \in \Leftrightarrow yz \in L) \implies xR_Ly$$

- Index von $R_L \leq$ Index von $R_M = |Z| < \infty$.

2. Beweisrichtung

Lemma: Wenn R_L einen endlichen Index hat, dann ist L regulär.

Für ein Wort x bezeichne $[x]$ die zugehörige Äquivalenzklasse bez. R_L.
Bei endlichem Index von R_L, sagen wir Index k, gibt es k Äquivalenzklassen, sagen wir $[x_1], \ldots, [x_k]$.

Beweis erfolgt durch Angabe eines DFA (Nerode–Automat) $M = (Z, \Sigma, \delta, z_0, E)$:

1. $Z = \{[x_1], \ldots, [x_k]\}$.
2. $z_0 = [\varepsilon]$.
3. $E = \{[x] |\ x \in L\}$.
4. $\delta([x], a) = [xa]$.

Das Problem der Wohldefiniertheit wird an der Tafel diskutiert!

- M gestartet in Zustand ε ist nach Verarbeitung eines Wortes x (gemäß der Definition von δ) im Zustand $[x]$.
- Aus der Definition von E folgt nun $T(M) = L$.

Beispiel

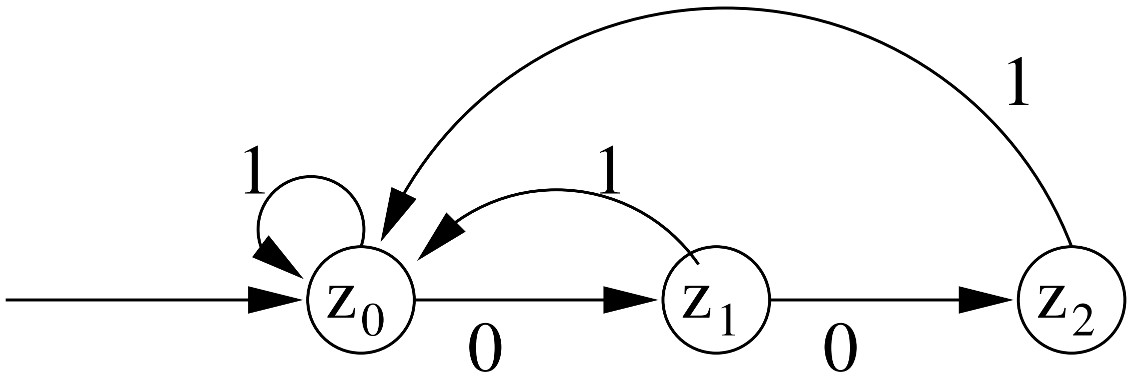
Betrachte die Sprache

$$L = \{x \in \{0,1\}^* | \; x \text{ endet mit } 00\} \; .$$

R_L hat 3 Äquivalenzklassen:

$$\begin{aligned} [\varepsilon] &= \{x | \; x \text{ endet nicht mit } 0\} \\ [0] &= \{x | \; x \text{ endet mit } 0 \text{ aber nicht mit } 00\} \\ [00] &= \{x | \; x \text{ endet mit } 00\} \end{aligned}$$

Der Nerode–Automat mit Zuständen $z_0 = [\varepsilon], z_1 = [0], z_2 = [00]$ ist dann durch folgenden Zustandsgraphen gegeben:



Minimierung eines gegebenen DFA

Eingabe: DFA $M = (Z, \Sigma, \delta.z_0, E)$, bei dem jeder Zustand vom Startzustand aus erreichbar ist

Ausgabe: Minimalautomat M_0 zu M:
der DFA mit möglichst wenigen Zuständen, der die gleiche Sprache wie M erkennt

Methode: Verschmelzung äquivalenter Zustände (Details folgen)

Äquivalente und inäquivalente Zustände

Zwei Zustände $z_1, z_2 \in Z$ heißen äquivalent **gdw**

$$\forall w \in \Sigma^* : z_1 \cdot w \in E \Leftrightarrow z_2 \cdot w \in E \ .$$

Somit sind z_1, z_2 inäquivalent **gdw**

$$\exists w \in \Sigma^* : z_1 \cdot w \in E \text{ und } z_1 \cdot w \notin E \text{ oder umgekehrt } .$$

Wir nennen w dann ein „separierendes Wort“ für z_1, z_2.

Intuition

Genau dann wenn $z_1 \sim z_2$ können z_1 und z_2 zu **einem** Zustand „verschmolzen“ werden, ohne die Sprache $T(M)$ abzuändern (Details an der Tafel).

Beobachtungen:

- Falls $z_1 \in E$ und $z_2 \notin E$ (oder umgekehrt), dann sind z_1, z_2 inäquivalent.
- Falls z_1', z_2' inäquivalent sind, und $z_1' = z_1 \cdot a, z_2' = z_2 \cdot a$ für ein $a \in \Sigma$, dann sind auch z_1, z_2 inäquivalent.

Resultierende Minimierungsprozedur

Markierung aller Paare inäquivalenter Zustände kann wie folgt geschehen:

Initialisierung: Markiere alle Paare (z_1, z_2) mit $z_1 \in E$ und $z_2 \notin E$ (oder umgekehrt).

Iteration: Solange ein unmarkiertes Paar (z_1, z_2), ein $a \in \Sigma$ und ein markiertes Paar (z_1', z_2') mit $z_1' = z_1 \cdot a, z_2' = z_2 \cdot a$ existieren, markiere (z_1, z_2) ebenfalls.

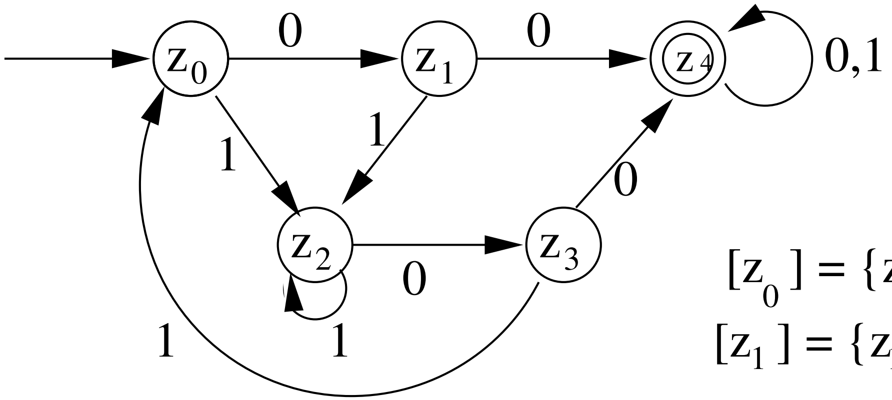
Äquivalente Zustandspaare sind gerade die unmarkierten Paare.

Berechnung des Minimalautomaten:

- Transformiere G_M zu einem „gröberen“ Zustandsgraphen, bei welchem jede Äquivalenzklasse von Knoten durch einen einzigen „Superknoten“ repräsentiert ist.
- Eine mit a markierte Kante von z nach z' wird dann zu einer mit a markierten Kante von $[z]$ nach $[z']$.
- Startknoten ist nun $[z_0]$, Endknoten sind die Superknoten $[z]$ mit $z \in E$.

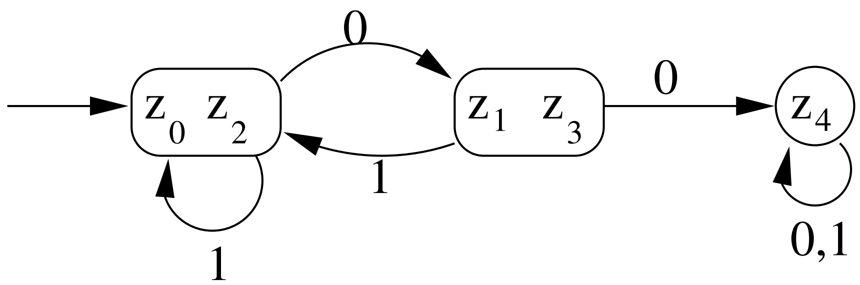
Beispiel

Links der DFA M, rechts der Minimalautomat M_0:



$[z_0] = \{z_0, z_2\}$

$[z_1] = \{z_1, z_3\}$



Die folgende Tabelle gibt die markierten Paare inäquivalenter Knoten an:

	z_0	z_1	z_2	z_3
z_1	$*$			
z_2		$*$		
z_3	$*$		$*$	
z_4	$*$	$*$	$*$	$*$

Abschlusseigenschaften

Satz: Die Klasse der regulären Sprachen ist abgeschlossen unter den Operationen $\cup, \cdot, *, \neg, *$.

Beweis:

- Abschluss unter $\cup . \cdot, *$ ist offensichtlich:
  Reguläre Sprachen sind durch reguläre Ausdrücke repräsentierbar.

- Abschluss unter $\neg$ ist einfach nachzuweisen:
  Falls $T(M) = L$ mit DFA $M = (Z, \Sigma, \delta, z_0, E)$ so folgt $T(\bar{M}) = \bar{L}$ mit DFA $\bar{M} := (Z, \Sigma, \delta, z_0, Z \setminus E)$.

- Abschluss unter $\cap$ ergibt sich jetzt mit „de Morgan“:

$$L_1 \cap L_2 = \overline{\overline{L_1 \cap L_2}} = \overline{\bar{L}_1 \cup \bar{L}_2}$$

Der „Produktautomat“

Produktautomat liefert einen alternativen Beweis für den Abschluss unter Vereinigung und Durchschnitt.

Zu gegeben DFAs $M_1 = (Z_1, \Sigma, \delta_1, z_{01}, E_1)$ und $M_2 = (Z_2, \Sigma, \delta_2, z_{02}, E_2)$ betrachte den DFA

$$M = (Z_1 \times Z_2, \Sigma, \delta, (z_{01}, z_{02}), E)$$

mit

$$\delta((z_1, z_2), a) = (\delta_1(z_1, a), \delta_2(z_2, a)) \quad .$$

Intuition: M führt die „Rechnungen“ von M_1 und M_2 parallel aus.

- Mit $E = E_1 \times E_2$ folgt $T(M) = T(M_1) \cap T(M_2)$.
- Mit $E = (E_1 \times Z_2) \cup (Z_1 \times E_2)$ folgt $T(M) = T(M_1) \cup T(M_2)$.

Das Wortproblem

Spezielles Wortproblem für eine reguläre Sprache $L \subseteq \Sigma^*$

Eingabe: $x \in \Sigma^*$

Frage: $x \in L$?

Komplexität: lösbar in Realzeit ($n = |x|$ Rechenschritte)

Methode: Benutze einen DFA M mit $T(M) = L$. Gestartet auf Eingabe x liefert er die Antwort in $|x|$ „Rechenschritten".

Das „Leerheitsproblem“

Eingabe: DFA M gegeben durch seinen Zustandsgraphen G_M

Frage: $T(M) = \emptyset$?

Komplexität: lösbar in Linearzeit

Methode: Verwende eine Graphexplorationstechnik (wie DFS oder BFS):

$$T(M) \neq \emptyset \Leftrightarrow \text{ von } z_0 \text{ ist ein Zustand aus } E \text{ erreichbar}$$

Bemerkung: Eine analoge Methode ist auch bei gegebenem NFA verwendbar.

Das „Endlichkeitsproblem“

Eingabe: DFA M gegeben durch seinen Zustandsgraphen G_M

Frage: $|T(M)| < \infty$?

Komplexität: lösbar in Linearzeit

Methode: Verwende eine Graphexplorationstechnik (wie DFS oder BFS):

- Markiere alle Zustände, die von z_0 aus erreichbar sind und von denen ausgehend sich ein Zustand aus E erreichen lässt. Es bezeichne G'_M von markierten Knoten induzierten Untergraphen von G_M.
- Nutze aus:

$$|T(M)| = \infty \Leftrightarrow G'_M \text{ enthält einen Zyklus}$$

Bemerkung: Eine analoge Methode ist auch bei gegebenem NFA verwendbar.

Das „Schnittproblem“

Eingabe: DFAs M_1, M_2 gegeben durch ihre Zustandsgraphen G_{M_1}, G_{M_2}

Frage: $T(M_1) \cap T(M_2) = \emptyset$?

Komplexität: lösbar mit quadratischem Zeitaufwand

Methode: Berechne den Produktautomaten M mit $T(M) = T(M_1) \cap T)M_2)$ und löse für M das Leerheitsproblem.

Bemerkung: Eine analoge Methode ist auch bei gegebenen NFAs verwendbar.

Das „Inklusionsproblem“

Eingabe: DFAs M_1, M_2 gegeben durch ihre Zustandsgraphen G_{M_1}, G_{M_2}

Frage: $T(M_1) \subseteq T(M_2)$?

Komplexität: lösbar mit quadratischem Zeitaufwand

Methode: Nutze aus, dass $A \subseteq B \Leftrightarrow A \cap \bar{B} = \emptyset$.

- Berechne aus M_2 den DFA $\bar{M}_2$ mit $T(\bar{M}_2) = \overline{T(M_2)}$.
- Löse für M_1 und $\bar{M}_2$ das Schnittproblem.

Das „Äquivalenzproblem“

Eingabe: DFAs M_1, M_2 gegeben durch ihre Zustandsgraphen G_{M_1}, G_{M_2}

Frage: $T(M_1) = T(M_2)$?

Komplexität: lösbar mit quadratischem Zeitaufwand

Methode: Offensichtliche Reduktion auf das Inklusionsproblem wegen

$$A = B \Leftrightarrow (A \subseteq B \text{ und } B \subseteq A)$$