

Theoretische Informatik

Prof. Dr. Barbara Staehle

HTWG Konstanz
Fakultaet für Informatik

WS 2021/2022

Teil III

Reguläre Sprachen und endliche Automaten

Teil III Typ 3 Sprachen und EA

1. Reguläre Sprachen

- 1.1 Definition und Eigenschaften
- 1.2 Pumping-Lemma für reguläre Sprachen
- 1.3 Reguläre Ausdrücke
- 1.4 Entscheidungsprobleme und Abschlusseigenschaften

2. Endliche Automaten

- 2.1 Deterministische endliche Automaten
- 2.2 Nichtdeterministische endliche Automaten
- 2.3 Endliche Automaten und reguläre Sprachen

3. Transduktoren

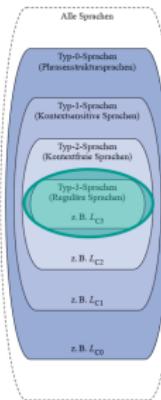
Abschnitt 1

Reguläre Sprachen

Reguläre Sprachen - Einführung

Reguläre Sprachen, Typ 3 Sprachen, Sprachen der Klasse \mathcal{L}_3

- kleinste Sprachklasse der Chomsky-Hierarchie
- relativ einfach strukturiert, daher für Computer gut zu verarbeiten
- Erzeugung: reguläre Grammatiken
- Erzeugung: reguläre Ausdrücke (siehe Abschnitt 1.3)
- Akzeptanz: endlichen Automaten (siehe Abschnitt 2)



Quelle:
[Hoffmann, 2011]

Definition

Eine formale Sprache heißt **regulär**, falls Sie von einer regulären Grammatik erzeugt wird.

Beispiel: Erzeugung der Sprache L_{C3}

Erinnerung: $L_{C3} = \{(ab)^n \mid n \in \mathbb{N}\}$.

Syntaxbaum zur Ableitung „abab“

Reguläre Grammatik für L_{C3} :

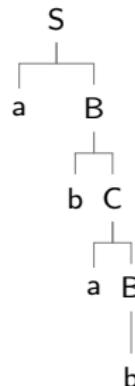
$G_{C3} = (\{S, B, C\}, \{a, b\}, P, S)$ mit
der Regelmenge P :

$$\begin{array}{lcl} S & \rightarrow & aB \\ B & \rightarrow & bC \\ C & \rightarrow & aB \mid \epsilon \end{array}$$

$$\mathcal{L}(G_{C3}) = \{ab, abab, ababab, \dots\}$$

Ableitung des Wortes „abab“:

$$\begin{array}{l} S \Rightarrow aB \\ \Rightarrow abC \\ \Rightarrow abaB \\ \Rightarrow ababC \\ \Rightarrow abab \end{array}$$



Auffällig: Die Ableitung des Wortes ist eine Rechtsableitung, der Syntaxbaum ist eine nach rechts geneigte Kette. Dies gilt für alle regulären Grammatiken, deshalb heißen diese auch **rechtslinear**.

Beispiel zum Mitdenken I

Beispiel: $L_V = \{v \mid v \text{ ist ein gültiger Variablenname in MATLAB}\}$ soll als formale Sprache über dem Alphabet Σ dargestellt werden mit $\Sigma = \{ \text{a}, \text{b}, \dots, \text{z}, \text{A}, \text{B}, \dots, \text{Z}, \text{0}, \text{1}, \dots, \text{9}, _ \}$.

- Bedingungen an einen MATLAB-Variablenamen:
 - ▶ darf Klein- und Großbuchstaben, sowie Zahlen enthalten
 - ▶ das einzige erlaubte Sonderzeichen ist „_“
 - ▶ das erste Zeichen muss ein Buchstabe sein
- Beispiele gültiger Variablennamen: x, y, temp_, v12xz, b_5

Gesucht: Grammatik G_V mit $\mathcal{L}(G_V) = L_V$.

Problem: Offensichtlich brauchen wir ganz viele langweilige ähnliche Regeln, da wir so viele Terminale zu beachten haben.

- Grund weshalb die Sprachen in der theoretischen Informatik als Terminalalphabet oft nur $\{a, b\}$ haben
- Für dieses Beispiel: Siehe nächste Folie

Beispiel zum Mitdenken II

Eine Grammatik für MATLAB-Variablennamen: $G_V = (N, \Sigma, P, S)$ mit

- $\Sigma = \{ \text{a}, \text{b}, \dots, \text{z}, \text{A}, \text{B}, \dots, \text{Z}, \text{0}, \text{1}, \dots, \text{9}, \text{_, } \}$
- $N = \{S, S_2\}$
- $P:$
 $S \rightarrow \text{a} \mid \dots \mid \text{z} \mid \text{A} \mid \dots \mid \text{Z}$
 $S \rightarrow \text{a}S_2 \mid \dots \mid \text{z}S_2 \mid \text{AS}_2 \mid \dots \mid \text{ZS}_2$
 $S_2 \rightarrow \text{a} \mid \text{b} \mid \dots \mid \text{z} \mid \text{A} \mid \dots \mid \text{Z} \mid \text{0} \mid \text{1} \mid \dots \mid \text{9} \mid \text{_}$
 $S_2 \rightarrow \text{a}S_2 \mid \dots \mid \text{z}S_2 \mid \text{AS}_2 \mid \dots \mid \text{ZS}_2 \mid \text{0S}_2 \mid \text{1S}_2 \mid \dots \mid \text{9S}_2 \mid \text{_} S_2$

- $\mathcal{L}(G_V) =$
 $\{\text{a}, \dots, \text{Z}, \text{aa}, \dots, \text{aZ}, \text{a0}, \dots, \text{a9}, \text{a_}, \dots\} = L_V \checkmark$

Bemerkung: Eigentlich gilt noch „der Name darf nicht mehr als 19 Zeichen enthalten“. Diese Anforderung ist mit Regeln umsetzbar, aber man braucht noch mehr ähnliche Regeln und Nonterminale.

Beispiel zum Mitdenken III

Beispiel: Ableitung der Variablennamen $x123$, $x12123$ und $x1212123$

Ableitung von „ $x123$ “:

$$\begin{aligned} S &\Rightarrow xS_2 \\ &\Rightarrow x1S_2 \\ &\Rightarrow x12S_2 \\ &\Rightarrow x123 \end{aligned}$$

Ableitung von „ $x12123$ “:

$$\begin{aligned} S &\Rightarrow xS_2 \\ &\Rightarrow x1S_2 \\ &\Rightarrow x12S_2 \\ &\Rightarrow x121S_2 \\ &\Rightarrow x1212S_2 \\ &\Rightarrow x12123 \end{aligned}$$

Ableitung von

$$\begin{aligned} „x1212123“: \\ S &\Rightarrow xS_2 \\ &\Rightarrow x1S_2 \\ &\Rightarrow x12S_2 \\ &\Rightarrow x121S_2 \\ &\Rightarrow x1212S_2 \\ &\Rightarrow x12121S_2 \\ &\Rightarrow x121212S_2 \\ &\Rightarrow x1212123 \end{aligned}$$

Beobachtung: $x123 \in L_V$, genauso wie $x(12)^n3 \in L_V$, für $n \in \mathbb{N}_0$.

Begründung: Für das Ziel „12“, kann das Nonterminal S_2 entweder zur Terminalkette 12, oder zu $12S_2$ abgeleitet werden. $12S_2$ kann wiederum zu 1212 oder zu $1212S_2$ abgeleitet werden ...

Man sagt, der **Mittelteil** 12 des Wortes $x123$ kann unendlich oft **aufgepumpt** werden.

Eine formellere Herleitung des Pumpinglemmas I

- Wir betrachten eine Ableitungssequenz, die mehr Schritte (k) hat, als Nonterminale zur Verfügung stehen (siehe z.B. letzte Folie).
⇒ Es muss mindestens ein Nonterminal in der Ableitung mehrfach auftauchen (Taubenschlag Prinzip).
- Sei eine Grammatik G mit $\Sigma = \{\sigma_n \mid n \in \mathbb{N}\}$ und $N = \{S, A, \dots\}$ mit $|N| < k$ gegeben, und A das mehrfach benutzte Nonterminal.
⇒ Ableitungssequenz für ein Wort der Länge k (zerlegt in die Teilworte uvw):

$$\begin{aligned} S &\xrightarrow{*} \underbrace{\sigma_1 \sigma_2 \dots \sigma_i}_u A \\ &\quad // u \text{ wird aus beliebigen NTs erzeugt} \\ &\xrightarrow{*} \underbrace{\sigma_1 \sigma_2 \dots \sigma_i}_u \underbrace{\sigma_{i+1} \sigma_{i+2} \dots \sigma_j}_v A \\ &\quad // v \text{ wird aus NT } A \text{ erzeugt} \\ &\xrightarrow{*} \underbrace{\sigma_1 \sigma_2 \dots \sigma_i}_u \underbrace{\sigma_{i+1} \sigma_{i+2} \dots \sigma_j}_v \underbrace{\sigma_{j+1} \sigma_{j+2} \dots \sigma_k}_w \\ &\quad // w \text{ wird aus NT } A \text{ erzeugt} \end{aligned}$$

Eine formellere Herleitung des Pumpinglemmas II

- Aus der Ableitung erkennt man, dass aus A das Wort

► $vA = \sigma_{i+1}\sigma_{i+2} \dots \sigma_j A$

erzeugt werden kann.

⇒ Damit können aus A auch die Wörter

► $v^2A = \sigma_{i+1}\sigma_{i+2} \dots \sigma_j\sigma_{i+1}\sigma_{i+2} \dots \sigma_j A$

► $v^3A =$

$$\sigma_{i+1}\sigma_{i+2} \dots \sigma_j\sigma_{i+1}\sigma_{i+2} \dots \sigma_j\sigma_{i+1}\sigma_{i+2} \dots \sigma_j A$$

► ...

abgeleitet werden

⇒ v kann also beliebig **aufgepumpt** werden.

⇒ Neben uvw gehören folglich auch die Worte $uv^i w$ ($i \in \mathbb{N}_0$) zu $\mathcal{L}(G)$.

- Jedes reguläre Wort (einer unendlichen Sprache) lässt sich in der Form uvw , bzw. $uv^i w$, mit $v \neq \varepsilon$ und $i \in \mathbb{N}_0$ darstellen.

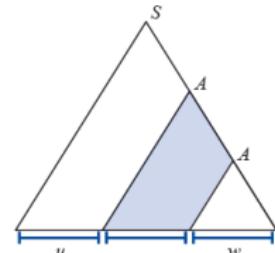


Bild: Struktur regulärer Worte
[Hoffmann, 2011]

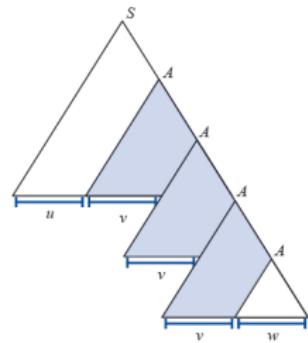


Bild: Aufpumpen des Mittelstücks
[Hoffmann, 2011]

Das Pumping-Lemma (PL) für reguläre Sprachen

Satz

Für jede reguläre Sprache L existiert ein $j \in \mathbb{N}$, so dass sich alle Wörter $\omega \in L$ mit $|\omega| \geq j$ in der folgenden Form darstellen lassen:

$$\omega = uvw \quad \text{mit } |v| \geq 1 \text{ und } |uv| \leq j .$$

Weiterhin ist für alle $i \in \mathbb{N}_0$ mit ω auch das Wort $uv^i w$ in L enthalten.

Bemerkungen:

- Für $\omega = uvw$ gilt, dass $v \neq \varepsilon$. $u = \varepsilon$ oder $w = \varepsilon$ ist zulässig.
- Alle Sprachen L die nur Wörter mit maximal Länge n haben, sind regulär. (Wähle im PL $j = n + 1$, dann ist $\{\omega \in L \mid |\omega| \geq j\} = \emptyset$.)

Zu beachten:

- Nicht jede Sprache die das PL erfüllt ist regulär.
- Jede Sprache, die das PL **nicht** erfüllt, ist **nicht** regulär.
- Das PL ist also ein gutes Instrument, um nachzuweisen, dass eine Sprache nicht regulär ist.

Beispiel zum Mitdenken

Behauptung: Die Sprache $L_{C2} = \{a^n b^n \mid n \in \mathbb{N}\}$ ist nicht regulär.

Beweis (durch Widerspruch):

- **Nehmen wir an**, L_{C2} wäre regulär.
- **Dann muss es nach dem PL** ein $j \in \mathbb{N}$ geben, so dass sich jedes Wort $\omega \in L_{C2}$ mit $|\omega| \geq j$ in der Form uvw darstellen lässt, wobei $|v| \geq 1$ und $|uv| \leq j$.
- Betrachten wir das **spezielle Wort** $\omega = a^j b^j$. Aus $|uv| \leq j$ folgt, dass für eine Zerlegung als uvw $u = a^{j-k}$ und $v = a^k$ für ein $1 \leq k \leq j$ gelten muss. Der Mittelteil muss also aus mindestens einem a (aber keinem b) bestehen.
- **Pumpen wir das Wort nun auf**, so erhalten wir z.B.
 - ▶ $\omega_2 = uv^2w = a^{j-k} a^{2k} b^j = a^{j+k} b^j$ welches gemäß dem PL in L_{C2} enthalten sein müsste.
- Da $k > 0$ und damit $j + k \neq j$, ist dies ein **Widerspruch zur Definition** von L_{C2} , **damit ist L_{C2} nicht regulär**.

Nützlichkeit regulärer Ausdrücke



Quelle: xkcd.com

Formale Definition regulärer Ausdrücke

Definition (Syntax)

Reg_{Σ} , die Menge der regulären Ausdrücke über einem Alphabet Σ , ist definiert durch:

- $\emptyset, \varepsilon \in Reg_{\Sigma}$
- $\Sigma \subset Reg_{\Sigma}$
- $r, s \in Reg_{\Sigma} \Rightarrow rs, (r|s) \in Reg_{\Sigma}$
- $r \in Reg_{\Sigma} \Rightarrow (r), r^* \in Reg_{\Sigma}$

Definition (Semantik)

Die von einem regulären Ausdruck r über Σ erzeugte Sprache $\mathcal{L}(r)$ ist definiert durch:

- $\mathcal{L}(\emptyset) = \emptyset$
- $\mathcal{L}(\varepsilon) = \{\varepsilon\}$
- $\mathcal{L}(\sigma \in \Sigma) = \{\sigma\}$
- $\mathcal{L}(rs) = \mathcal{L}(r)\mathcal{L}(s)$
- $\mathcal{L}(r|s) = \mathcal{L}(r) \cup \mathcal{L}(s)$
- $\mathcal{L}((r)) = \mathcal{L}(r)$
- $\mathcal{L}(r^*) = \mathcal{L}(r)^*$

Syntax erweiterter regulärer Ausdrücke unter Linux / UNIX

Symbol	Bedeutung
.	Beliebiges Zeichen außer dem Zeilenumbruch
[...]	Positivliste (jedes Zeichen innerhalb der spezifizierten Liste)
[^...]	Negativliste (jedes Zeichen außerhalb der spezifizierten Liste)
^	Beginn einer Zeile
\$	Ende einer Zeile
<	Beginn eines Worts
>	Ende eines Worts
+	Der vorangegangene Ausdruck kommt mindestens einmal vor
?	Der vorangegangene Ausdruck kommt höchstens einmal vor
*	Der vorangegangene Ausdruck kommt gar nicht oder beliebig oft vor
{n}	Der vorangegangene Ausdruck kommt genau n-mal vor
{n,}	Der vorangegangene Ausdruck kommt mindestens n-mal vor
{n,m}	Der vorangegangene Ausdruck kommt mindestens n-mal, höchstens m-mal vor
{,m}	Der vorangegangene Ausdruck kommt höchstens m-mal vor
	Alternative (entweder der linke oder der rechte Ausdruck)
()	Gruppierung

Tabelle: Syntax einiger erweiterter regulärer Ausdrücke für egrep oder sed [Hoffmann, 2011]

Siehe auch Artikel „Reguläre Sprachen, reguläre Ausdrücke“ des [lrz](#).

Beispiele zum Mitdenken (für $\Sigma = \{x, y, z\}$)

- Welche Sprache wird von den regulären Ausdrücken erzeugt?

regulärer Ausdruck		erzeugte Sprache
$r_1 = x$		$\mathcal{L}(r_1) = \{x\}$
$r_2 = xyz$		$\mathcal{L}(r_2) = \{xyz\}$
$r_3 = x y z$		$\mathcal{L}(r_3) = \{x, y, z\}$
$r_4 = (x y z)(x z)$		$\mathcal{L}(r_4) = \{xx, xz, yx, yz, zx, zz\}$
$r_5 = x^*$		$\mathcal{L}(r_5) = \{\varepsilon, x, xx, xxx, \dots\}$
$r_6 = (xyz)^*$		$\mathcal{L}(r_6) = \{\varepsilon, xyz, xyzxyz, xyzxyzxyz, \dots\}$
$r_7 = (x y z)^*$		$\mathcal{L}(r_7) = \{\varepsilon, x, y, z, xx, \dots, zz, xxx, \dots zzz, \dots\}$

- Welcher reguläre Ausdruck erzeugt die gegebenen Sprachen?

Sprache		erzeugender regulärer Ausdruck
$L_8 = \{xx, xy, xz\}$		$r_8 = x(x y z)$
$L_9 = \{x, xx, xy, xz, xxx, xxy, \dots\}$		$r_9 = x(x y z)^*$
$L_{10} = \{\omega \in \Sigma^* \text{ mit 3. Stelle } = z\}$		$r_{10} = (x y z)(x y z)z(x y z)^*$

Beispiel zum Mitdenken - eine Lösung zum Comic I

Gesucht: Ein regulärer Ausdruck, welcher Textmengen auf US-Straßenangaben durchsucht.

Lösung [Hopcroft et al., 2011]:

- Die meisten Straßennamen enden auf „Avenue“, „Road“ oder „Street“.
 - ▶ regulärer Ausdruck hierfür:
 $(\text{Street} \mid \text{St} \backslash . \mid \text{Avenue} \mid \text{Ave} \backslash . \mid \text{Road} \mid \text{Rd} \backslash .)$
 - ▶ Notation des Strings in Unix-Schreibweise, „|“ steht für Alternativen und „.“ muss mit einem „\“ **escaped** werden, um dessen Verwendung als „beliebiges Zeichen“ zu verhindern.
- Vorher kommt der Bezeichner der Straße, ein oder mehrere Worte, die alle mit einem Großbuchstaben anfangen und eventuell Punkte enthalten (z.B. S.W. Jackson Street).
 - ▶ regulärer Ausdruck: $' [\text{A-Z}] [\text{A-Za-z} \backslash .]^* ([\text{A-Z}] [\text{A-Za-z} \backslash .]^*)^*$,
 - ▶ „*“ bedeutet immer „0 oder mehrmals“
 - ▶ Um zu erzwingen, dass das Leerzeichen zu Beginn des hinteren Ausdrucks steht (und es nicht ignoriert wird), muss der Ausdruck in einfache Anführungszeichen „' “ gesetzt werden.

Beispiel zum Mitdenken - eine Lösung zum Comic II

- Ein amerikanischer Straßename beginnt immer mit der Hausnummer (z.B. The Simpsons - 742 Evergreen Terrace, Springfield), jedoch kann diese Nummer unter Umständen auch von einem Großbuchstaben gefolgt werden.
 - ▶ Der reguläre Ausdruck hierfür: $[0-9]^+ [A-Z] ?$
 - ▶ „+“ bedeutet „mindestens einer oder mehr“
 - ▶ „?“ steht für „einer oder keiner“
- Zusammengesetzt:
 $[0-9]^+ [A-Z] ? [A-Z] [a-zA-Z\ .\]^* ([A-Z] [a-zA-Z\ .\]^*)^*$
(Street|St\ .\ |Avenue|Ave\ .\ |Road|Rd\ .\)'
- Super - unser Ausdruck erkennt
 - ▶ 221B Baker Street
 - ▶ 1600 Pennsylvania Avenue
 - ▶ 2311 North Los Robles Avenue
- Leider erkennt er aber nicht
 - ▶ 1600 Amphitheatre Parkway
 - ▶ 15010 NE 36th Street
 - ▶ 15 Rodeo Drive

Beispiel zum Mitdenken - eine Lösung zum Comic III

Beispiel ausprobieren:

- Datei addresslist.txt aus Moodle herunterladen (Adressliste aller Touristinfo-Büros der amerikanischen Bundesstaaten)
- Unter Linux / Cygwin:
 - ▶ `egrep '[0-9]+[A-Z]? [A-Z] [a-zA-Z\.]*([A-Z] [a-zA-Z\.]*)* (Street|St\.|Avenue|Ave\.|Road|Rd\.)'` addresslist.txt
 - ▶ `egrep` statt `grep` steht für „extended regular expressions“ (bei `grep` muss man auch „+, ?, , (“ escapen)
- regexr.com
 - ▶ `[0-9]+[A-Z]? [A-Z] [a-zA-Z\.]*([A-Z] [a-zA-Z\.]*)* (Street|St\.|Avenue|Ave\.|Road|Rd\.)`
 - ▶ Inhalt von addresslist.txt ins Textfeld kopieren
- beliebiger Texteditor wie z.B. [Notepad++](#), der reguläre Ausdrücke unterstützt. Bei Notepad++ den Befehl „Hervorheben“ nutzen und „reguläre Ausdrücke“ als Suchmodus aktivieren.

Äquivalenz regulärer Ausdrücke und regulärer Grammatiken

Bemerkung: Reguläre Ausdrücke und reguläre Grammatiken erzeugen beide die Sprachklasse \mathcal{L}_3 . Es gibt also zu jedem regulären Ausdruck eine äquivalente reguläre Grammatik und umgekehrt.

Beispiele:

- $L_{C3} = \{(ab)^n \mid n \in \mathbb{N}\}$

- Regelmenge der erzeugenden Grammatik G_{C3} :
$$\begin{array}{rcl} S & \rightarrow & aB \\ B & \rightarrow & bC \\ C & \rightarrow & aB \mid \varepsilon \end{array}$$
- Äquivalenter regulärer Ausdruck: $ab(ab)^*$ oder $(ab)^+$

- $K = \{a^i b^j c^k \mid i, j, k \in \mathbb{N}\}$

- Regelmenge der erzeugenden Grammatik G_K :
$$\begin{array}{rcl} S & \rightarrow & aS \mid aB \\ B & \rightarrow & bB \mid bC \\ C & \rightarrow & cC \mid c \end{array}$$
- Äquivalenter regulärer Ausdruck: $aa^* bb^* cc^*$ oder $a^+ b^+ c^+$

Exkurs: Entscheidungsprobleme regulärer Sprachen

Erinnerung: Entscheidungsprobleme definieren Fragestellungen zu Sprachen, die je nach Sprachklasse

- **entscheidbar** sind, d.h. es existiert ein Verfahren, das entscheidet, ob die Frage mit „Ja“ oder „Nein“ beantwortet wird,
- **unentscheidbar** sind, d.h. es existiert kein Verfahren, das solch eine Antwort geben kann.

Problem	Eingabe	Fragestellung	Entscheidbar?
Wortproblem	L , Wort $\omega \in \Sigma^*$	Ist $\omega \in L$?	✓ Ja
Leerheitsproblem	L	Ist $L = \emptyset$	✓ Ja
Endlichkeitsproblem	L	Ist $ L < \infty$?	✓ Ja
Äquivalenzproblem	L_1 und L_2	Ist $L_1 = L_2$?	✓ Ja

Tabelle: Entscheidungsprobleme für Sprachen über Σ^* , $L, L_1, L_2 \in \mathcal{L}_3$

Bemerkung: Dass alle Fragen entscheidbar sind, illustriert die einfache Struktur der regulären Sprachen und ist eine Besonderheit.

Exkurs: Abschlusseigenschaften regulärer Sprachen

Bemerkung: Unter den **Abschlusseigenschaften** einer (Chomsky-Sprachklasse) versteht man die Frage, ob die Verknüpfung zweier Sprachen aus dieser Klasse wieder in der Klasse liegt oder nicht.

Beispiel: $L_{ab} = L_a \cup L_b$

$$\begin{aligned} &= \{\omega \mid \omega = a^n, n \in \mathbb{N}_0\} \cup \{\omega \mid \omega = b^n, n \in \mathbb{N}_0\} \\ &= \{\omega \mid \omega = a^n \vee b^n, n \in \mathbb{N}_0\} \in \mathcal{L}_3 \end{aligned}$$

Operation	Eingabe	Fragestellung	Antwort
Vereinigung	L_1, L_2	Ist $L_1 \cup L_2 \in \mathcal{L}_3$?	✓ Ja
Schnitt	L_1, L_2	Ist $L_1 \cap L_2 \in \mathcal{L}_3$?	✓ Ja
Komplement	L	Ist $\Sigma^* \setminus L \in \mathcal{L}_3$?	✓ Ja
Produkt	L_1, L_2	Ist $L_1 L_2 \in \mathcal{L}_3$?	✓ Ja
Stern	L	Ist $L^* \in \mathcal{L}_3$?	✓ Ja

Tabelle: Abschlusseigenschaften für Sprachen über Σ^* , $L, L_1, L_2 \in \mathcal{L}_3$

Bemerkung: Dass \mathcal{L}_3 unter allen Operationen abgeschlossen ist, belegt wieder (siehe Entscheidungsprobleme) die einfache Struktur der regulären Sprachen.

Abschnitt 2

Endliche Automaten

Was ist ein Automat?

Automat, der

Wortart: Substantiv, maskulin

Häufigkeit: ■■■■



Rechtschreibung ⓘ

Nach oben

Worttrennung:

Au|to|mat

Beispiel:

am, auf den Automaten

Bedeutungen ⓘ

Nach oben

1. a. Apparat, der nach Münzeinwurf oder nach Einstcken einer Geldkarte, eines Geldscheins o.
 A. selbsttätig etwas ab-, herausgibt oder eine Dienst- oder Bearbeitungsleistung erbringt
b. Werkzeugmaschine, die Arbeitsvorgänge nach Programm selbsttätig ausführt
c. (in elektrischen Anlagen) automatische Sicherung zur Verhinderung von Schäden durch
 Überlastung
2. (Mathematik, EDV) kybernetisches System, das Informationen an einem Eingang aufnimmt,
 selbstständig verarbeitet und an einem Ausgang abgibt

Bild: Definition laut Duden

Quelle: Automaten-Service Mahel

Für Informatiker neben Ein- und Ausgabe noch wichtig: der **Zustand** des Automaten. (siehe Vorlesung Softwaremodellierung!)

Zustandsautomaten in der Informatik

Beispiel: Pac-Man (Quelle: Eric Gribkoff, UC Davis)



Bild: Pac-Man Spielszene

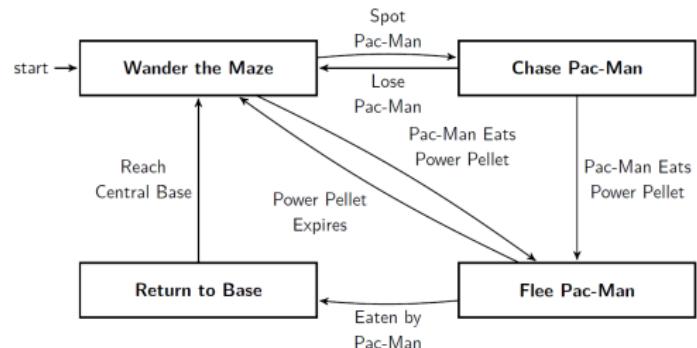


Bild: Verhalten des Pac-Man Geists

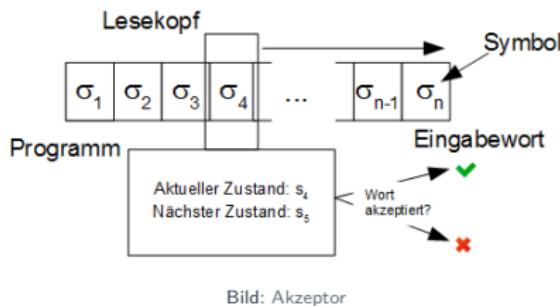
Endliche Automaten - Einführung und Begriffsbestimmung

- Erfassen und modellieren einen Zustandsraum (**siehe Vorlesung Softwaremodellierung!**)
- Arbeiten ereignisbasiert (Ursache, Wirkung)
- Bilden Ursache-Wirkungsprinzip von Zustandsübergängen ab
 - ▶ Anwendungen: Modellierung technischer Bausteine, Algorithmen, Vorgänge, Protokolle, Abbildung von Verhaltensmustern, ...
- Es existieren zwei Haupttypen von endlichen Automaten:
 - ▶ **Akzeptoren** nehmen eine Zeichenfolge entgegen und entscheiden über Gültigkeit
 - Anwendungen: Textmustererkennung, Stichwortsuche, Fehlererkennung, Spracherkennung und -analyse, Syntaxkorrektur, ...
 - ▶ **Transduktoren** setzen Eingabe in Ausgabe um
 - Anwendungen: Addier- und Rechenwerke, Korrektur- und Übersetzung, ...

Akzeptoren und Transduktoren im Vergleich

Akzeptoren nehmen ein Wort

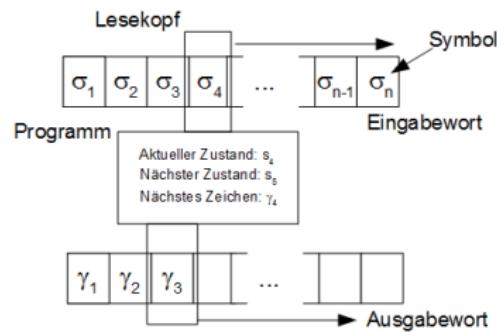
$\omega = \sigma_1\sigma_2 \dots \sigma_n$ entgegen und entscheiden, ob es gültig ist.



Beobachtung: Akzeptoren sind einfacher strukturiert, daher werden wir uns zuerst mit diesen beschäftigen.

Transduktoren nehmen ein Wort

$\omega = \sigma_1\sigma_2 \dots \sigma_n$ entgegen und generieren daraus ein Ausgabewort $\psi = \gamma_1\gamma_2 \dots \gamma_n$.



Anmerkung: Bilder verwenden „S“ für die Zustandsmenge, statt „Q“.

Deterministischer endlicher Akzeptor (DEA)

Definition

Ein deterministischer endlicher Akzeptor, kurz **DEA**, ist ein 5-Tupel $A = (Q, \Sigma, \delta, q_0, F)$ mit

- der endlichen **Zustandsmenge** Q ,
- dem endlichen **Eingabealphabet** Σ ,
- der (meist partiellen) **Zustandsübergangsfunktion** $\delta : Q \times \Sigma \rightarrow Q$,
- dem **Startzustand** $q_0 \in Q$,
- der Menge der **Finalzustände** $F \subseteq Q$.

Alternative Schreibweisen

- engl: deterministic finite state machine (FSM)
- engl: deterministic finite automaton (DFA)
- Zustandsmenge: S oder Z statt Q (engl.: state)

Anmerkungen zur Definition eines DEA

Zur Definition von $A = (Q, \Sigma, \delta, q_0, F)$ ist noch zu bemerken:

- Die Zustände $q \in Q$ von A sind die **endlich vielen „Speicherzellen“** des DEAs, mit Hilfe derer er **pro Eingabezeichen** eine Information speichern kann.
- die Zustandsübergangsfunktion δ ist
 - ▶ **meist eine partielle Funktion:** nicht für alle Kombinationen aus $Q \times \Sigma$ ist ein nächster Zustände definiert. Tritt ein solcher Fall auf, beendet A die Verarbeitung des aktuellen Eingabewortes.
 - ▶ **selten eine totale Funktion:** $\delta(q, \sigma)$ ist für alle $q \in Q, \sigma \in \Sigma$ definiert, daher wird jedes Eingabewort vollständig eingelesen, auch wenn nach wenigen Zeichen schon klar ist, dass es abgelehnt wird. Ein partielles δ ist eventuell unsauberer, aber effizienter.
- **In jedem Arbeitsschritt** liest der Automat genau ein Zeichen vom Band und führt einen Zustandsübergang durch (oder bleibt im aktuellen Zustand).

Arbeitsweise eines DEA I

- Der Automat $A = (Q, \Sigma, \delta, q_0, F)$ beginnt im Startzustand q_0 .
- Für $\omega = \sigma_1\sigma_2\dots\sigma_n$ durchläuft A nacheinander die Zustände $q_0, q_1, q_2, \dots, q_n$ wobei der nächste Zustand bestimmt wird als $q_i = \delta(q_{i-1}, \sigma_i)$.
- A hält an (stoppt das Einlesen), wenn $\delta(q_{i-1}, \sigma_i)$ undefiniert ist, es also für gelesenes Symbol und aktuellen Zustand keinen nächsten Zustand gibt.
- Das Wort ω wird genau dann **akzeptiert**, wenn es vollständig eingelesen wurde (A nicht vorher angehalten hat) und A sich in einem Finalzustand befindet ($q_n \in F$).

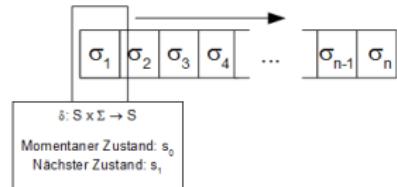


Bild: Start der Verarbeitung von ω

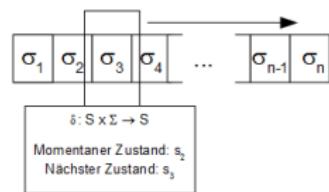


Bild: Mitten in der Verarbeitung von ω

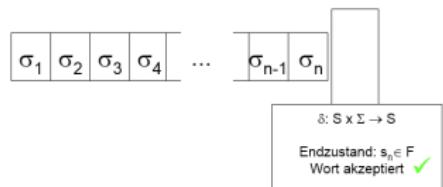


Bild: Ende der Verarbeitung von ω

Arbeitsweise eines DEA II

Beispiel:

$$A_1 = (Q, \Sigma, \delta, q_0, F)$$

- $Q = \{q_0, q_1\}$
- $\Sigma = \{a, b\}$
- $\delta : Q \times \Sigma \mapsto Q$

Q/Σ	a	b
q_0	q_0	q_1
q_1		q_1

- $F = \{q_1\}$
- Startzustand q_0

Frage: Akzeptiert A_1 das Wort $\omega = aabbb$?

Lösung: Bestimme Zustände, die bei der Abarbeitung von ω durchlaufen werden.

1. $\delta(q_0, a) = q_0$
2. $\delta(q_0, a) = q_0$
3. $\delta(q_0, b) = q_1$
4. $\delta(q_1, b) = q_1$
5. $\delta(q_1, b) = q_1$

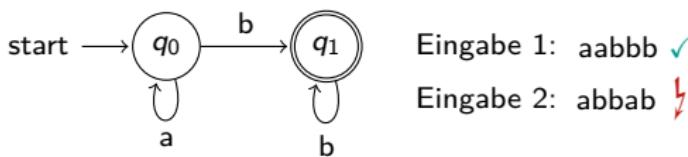
Antwort: A_1 ist nach Lesen des letzten Zeichens im Zustand $q_1 \in F$, ω wird also akzeptiert.

Bemerkung: Verfahren ist zwar korrekt, aber umständlich. Besser geeignet: Graphische Darstellung.

Arbeitsweise eines DEA III

Graphische Darstellung eines Automaten: (Zustands)Übergangsdiagramm als gerichteter kantenmarkierter Graph

- Beispiel: Automat A_1
- Zeichne Zustände als Knoten
- Zeichne Zustandsübergänge als Kanten mit Beschriftung
- Markiere Anfangszustand mit Pfeil (und „Start“)
- Umrande Endzustände doppelt
- Frage nach Akzeptanz eines Eingabewortes ω lässt sich durch Simulation der Zustandsübergänge im Graph beantworten.



Eingabe 1: aabb \checkmark

Eingabe 2: abbab $\cancel{\checkmark}$

Sprache eines DEA I

Definition

Die von einem DEA A akzeptierte Sprache ist definiert als

$$\begin{aligned}\mathcal{L}(A) &= \{\omega \in \Sigma^* \mid A \text{ akzeptiert } \omega\} \\ &= \{\omega \in \Sigma^* \mid A \text{ befindet sich nach dem vollständigen Einlesen} \\ &\quad \text{von } \omega \text{ in einem Endzustand}\} \\ &= \{\omega \in \Sigma^* \mid \hat{\delta}(q_0, \omega) \in F\}\end{aligned}$$

Rekursive Definition der **erweiterten Zustandsübergangsfunktion** $\hat{\delta}$:

$$\hat{\delta} : Q \times \Sigma^* \rightarrow Q \quad \text{mit}$$

$$\hat{\delta}(q, \varepsilon) = q, \quad \text{für } q \in Q$$

$$\hat{\delta}(q, \sigma) = \delta(q, \sigma), \quad \text{für } \sigma \in \Sigma$$

$$\hat{\delta}(q, \sigma\omega) = \hat{\delta}(\delta(q, \sigma), \omega), \quad \text{für } \sigma \in \Sigma, \omega \in \Sigma^*$$

Sprache eines DEA II

Verdeutlichung: Was kann passieren, wenn DEA A ein Wort ω liest?

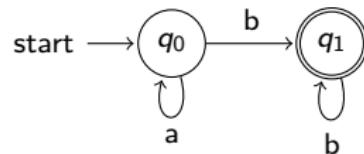
1. ω wird komplett gelesen, A ist in einem Endzustand
 $\Rightarrow \omega$ gehört zu $\mathcal{L}(A)$

- XOR 2. ω wird komplett gelesen, A ist **nicht** in einem Endzustand
 $\Rightarrow \omega$ gehört **nicht** zu $\mathcal{L}(A)$

- XOR 3. Wort wird **nicht** komplett gelesen
 $\Rightarrow \omega$ gehört **nicht** zu $\mathcal{L}(A)$

Beispiel: Sprache des DEA A_1

- Alle von A_1 akzeptierten Wort ω .
- Formal: Alle Worte ω , für die gilt
 $\hat{\delta}(q_0, \omega) = q_1$.
Alle Worte ω , für die gilt: ω wird komplett gelesen und A_1 befindet sich nach Einlesen von ω in einem Endzustand.

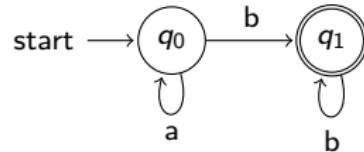


Vorgehensweise: Untersuche Testwörter, nutze $\hat{\delta}$, Tabellen oder graphische Darstellung, um Zustandsübergänge nachzuvollziehen, verallgemeinere.

Beispiel: Sprache des DEA A_1

Ansatz 1: formaler Weg für $\omega_4 = aabbb$

$$\begin{aligned}\hat{\delta}(q_0, aabbb) &= \hat{\delta}(\delta(q_0, a), abbb) = \hat{\delta}(q_0, abbb) \\&= \hat{\delta}(\delta(q_0, a), bbb) = \hat{\delta}(q_0, bbb) \\&= \hat{\delta}(\delta(q_0, b), bb) = \hat{\delta}(q_1, bb) \\&= \hat{\delta}(\delta(q_1, b), b) = \hat{\delta}(q_1, b) \\&= \delta(q_1, b) = q_1 \checkmark\end{aligned}$$



Ansatz 2: Pragmatischer Weg

- $\omega_1 = a : q_0 \xrightarrow{a} q_0 \text{ } \cancel{\checkmark}$ (komplett eingelesen, aber nicht-Finalzustand)
- $\omega_2 = b : q_0 \xrightarrow{b} q_1 \checkmark$ (komplett eingelesen und Finalzustand)
- $\omega_3 = ab : q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \checkmark$ (komplett eingelesen und Finalzustand)
- $\omega_4 = aabbb : q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \checkmark$
(komplett eingelesen und Finalzustand)

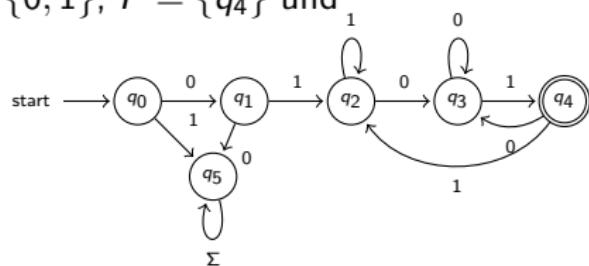
Lösung: $\mathcal{L}(A_1) = \{a^n b^m \mid n \in \mathbb{N}_0, m \in \mathbb{N}\}$

Beispiel: Sprache des DEA A_2

Wir betrachten den Automaten $A_2 = (Q, \Sigma, \delta, q_0, F)$ mit

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$, $\Sigma = \{0, 1\}$, $F = \{q_4\}$ und

δ	q_0	q_1	q_2	q_3	q_4	q_5
0	q_1	q_5	q_3	q_3	q_3	q_5
1	q_5	q_2	q_2	q_4	q_2	q_5



Fragen: Welche Zustände werden bei der Verarbeitung der folgenden Worte durchlaufen und werden diese akzeptiert?

- $\omega_1 = 1 : q_0 \xrightarrow{1} q_5$ ↗
- $\omega_2 = 01 : q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2$ ↗
- $\omega_3 = 0101 : q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{0} q_3 \xrightarrow{1} q_4$ ✓
- $\omega_4 = 0111001 : q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{1} q_2 \xrightarrow{1} q_2 \xrightarrow{0} q_3 \xrightarrow{0} q_3 \xrightarrow{1} q_4$ ✓

$$\mathcal{L}(A_2) = \{01\omega01 \mid \omega \in \Sigma^*\}$$

regulärer Ausdruck r_2 , der $\mathcal{L}(A_2)$ erzeugt: $01[01]^*01$ bzw. $01(0|1)^*01$

Beispiel: Sprache des DEA A_3

Wir betrachten den Automaten $A_3 = (Q, \Sigma, \delta, q_0, F)$

$$Q := \{q_0, q_1, q_2\}$$

$$\Sigma := \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$$\delta(q_0, \sigma) := \begin{cases} q_0 & \text{für } \sigma \in \{0, 3, 6, 9\} \\ q_1 & \text{für } \sigma \in \{1, 4, 7\} \\ q_2 & \text{für } \sigma \in \{2, 5, 8\} \end{cases}$$

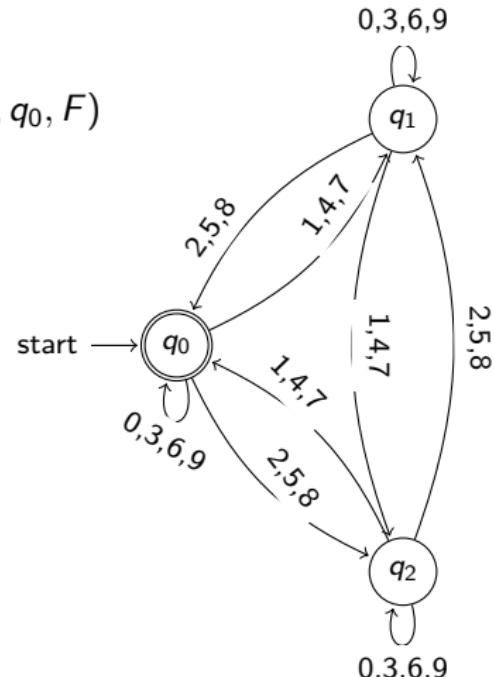
$$\delta(q_1, \sigma) := \begin{cases} q_1 & \text{für } \sigma \in \{0, 3, 6, 9\} \\ q_2 & \text{für } \sigma \in \{1, 4, 7\} \\ q_0 & \text{für } \sigma \in \{2, 5, 8\} \end{cases}$$

$$\delta(q_2, \sigma) := \begin{cases} q_2 & \text{für } \sigma \in \{0, 3, 6, 9\} \\ q_0 & \text{für } \sigma \in \{1, 4, 7\} \\ q_1 & \text{für } \sigma \in \{2, 5, 8\} \end{cases}$$

$$F := \{q_0\}$$

Sprache von A_3 ?

$$0, 3, 6, 9, 12, 15, 18, \dots \in \mathcal{L}(A_3) \Rightarrow \mathcal{L}(A_3) = \{n \in \mathbb{N}_0 \mid n \text{ ist durch 3 teilbar}\}.$$



Beispiel: Reguläre Ausdrücke und DEAs

Gegeben sei der reguläre Ausdruck $r_y = [ab]^* ab$ für $\Sigma = \{a, b\}$
 $\mathcal{L}(r_y) = \{ab, aab, bab, \dots, abbabbbaaaab, \dots\} = \{\omega ab \mid \omega \in \Sigma^*\}$

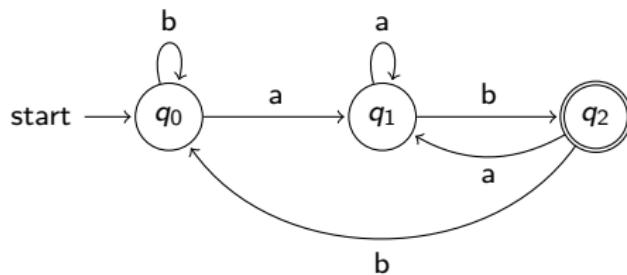
Gesucht: DEA A_y mit $\mathcal{L}(A_y) = \mathcal{L}(r_y)$.

Lösung:

δ_1 gegeben durch:

$A_y = (Q, \Sigma, \delta_1, F, q_0)$ mit

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{a, b\}$
- $F = \{q_2\}$



Beispiel: Automatenkonstruktion I

Für ein Firmen-Intranet dürfen nur Passwörter verwendet werden, die

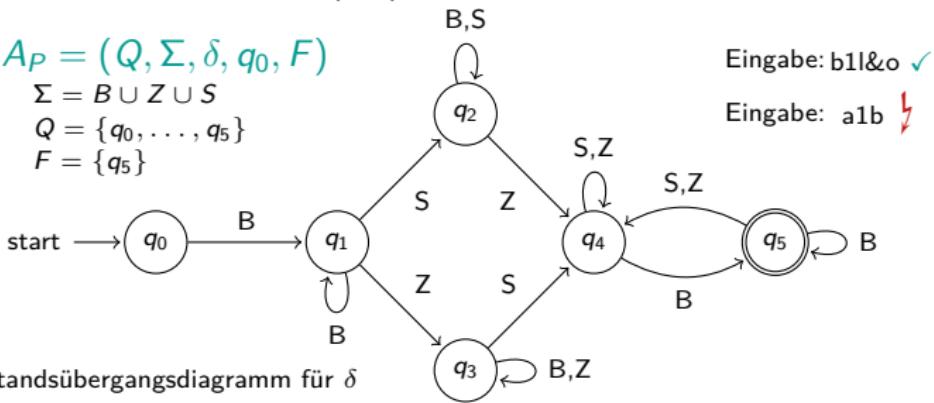
- mindestens 4 Zeichen lang sind,
- mit einem Buchstaben (B) beginnen und enden,
- mindestens eine Zahl (Z) und ein Sonderzeichen (S) beinhalten.
- Gültig: AbcD#Ef8Gh, b1l&o, ga!adri3el
- Ungültig: a1b, Fr0do?, !3golas
- Gesucht: DEA A_P mit $\mathcal{L}(A_P) = \text{Menge aller gültigen Passwörter}$

Lösung: $A_P = (Q, \Sigma, \delta, q_0, F)$

$$\Sigma = B \cup Z \cup S$$

$$Q = \{q_0, \dots, q_5\}$$

$$F = \{q_5\}$$



Beispiel: Automatenkonstruktion II

Erweiterung der Passwortanforderung: alternativ sind auch kürzere Passwörter möglich, wenn sie strenger Bedingungen gehorchen.
Erlaubt sind alle Passwörter die

- mindestens 4 Zeichen lang sind,
 - ▶ mit einem Buchstaben (B) beginnen und enden,
 - ▶ mindestens eine Zahl (Z) und ein Sonderzeichen (S) beinhalten.
- Oder mindestens 3 Zeichen lang sind,
 - ▶ mit zwei Buchstaben (B) beginnen,
 - ▶ mit einem Sonderzeichen (S) enden.
- Gültig: AbcD#Ef8Gh, b1l&o, ga!adri3el, ab?, ga!adri3e!, Fr0do?
- Ungültig: ab?c, !3golas, Frod0, xy1

Modellierung durch Erweiterung von A_P ?

- Sieht machbar aus, da eine Anforderung bestehen bleibt.
- Aber: z.B. "Fr0do?" war unter der alten Anforderung nicht gültig, unter der neuen schon.

Lösung: Ein Automat, der mehrere Optionen gleichzeitig durchspielen kann (Nichtdeterminismus).

Nichtdeterministischer endlicher Akzeptor (NEA)

Definition

Ein nichtdeterministischer endlicher Akzeptor kurz **NEA**, ist ein 5-Tupel $N = (Q, \Sigma, \delta, q_0, F)$ mit

- der endlichen **Zustandsmenge** Q ,
- dem endlichen **Eingabealphabet** Σ ,
- der **Zustandsübergangsfunktion** $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$,
- der Menge der **Finalzustände** $F \subseteq Q$ und
- dem **Startzustand** $q_0 \in Q$.

Alternative Schreibweisen

- engl: nondeterministic finite state machine (NFSM)
- engl: nondeterministic finite automaton (NFA)

Unterschied zwischen DEA und NEA: Definition von δ , da ein NEA nicht in **einen** eindeutigen nächsten Zustand übergeht, sondern in **eine Menge** von nächsten Zuständen (keiner, einer, zwei, drei, ...)!

Unterschiede zwischen DEA und NEA

DEA

- δ definiert für jedes gelesene Zeichen σ genau einen nächsten Zustand (oder keinen).
- Für jede Eingabe $\omega = \sigma_1\sigma_2\dots\sigma_n$ durchläuft A genau eine Zustandsfolge $q_0, q_1, q_2, \dots, q_n$ mit $q_i = \delta(q_{i-1}, \sigma_i)$ oder hält an, falls kein nächster Zustand definiert ist.
- Das Wort ω gilt genau dann als akzeptiert, wenn A nicht vorzeitig angehalten hat und die Zustandsfolge in einem Finalzustand endet.
- Jeder DEA ist ein NEA, die Umkehrung gilt nicht!

NEA

- δ definiert für jedes gelesene Zeichen σ einen oder mehrere nächste Zustände (oder keinen).
- Für jede Eingabe $\omega = \sigma_1\sigma_2\dots\sigma_n$ durchläuft N mehrere mögliche Zustandsfolgen $q_0, q_1, q_2, \dots, q_n$ mit $q_i \in \delta(q_{i-1}, \sigma_i)$ oder hält an, falls $\delta(q_{i-1}, \sigma_i) = \emptyset$.
- Das Wort ω gilt genau dann als akzeptiert, wenn N nicht vorzeitig angehalten hat und mindestens eine Zustandsfolge in einem Finalzustand endet.

Arbeitsweise eines NEA I

- Der Automat N beginnt im Startzustand q_0 .
- Für das Eingabewort $\omega = \sigma_1\sigma_2 \dots \sigma_n$ durchläuft N mehrere mögliche Zustandsfolgen $q_0, q_1, q_2, \dots, q_n$ mit $q_i \in \delta(q_{i-1}, \sigma_i)$.
- N hält (auf dem betroffenen Verarbeitungspfad) an (stoppt das Einlesen), wenn es für gelesenes Symbol und aktuellen Zustand keinen nächsten Zustand gibt.
- Das Wort ω gilt genau dann als akzeptiert, wenn N nicht vorzeitig angehalten hat und (mindestens) ein q_n mit $q_n \in F$ existiert.

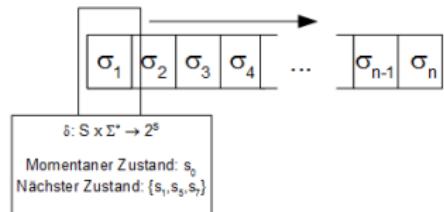


Bild: Start der Verarbeitung von ω

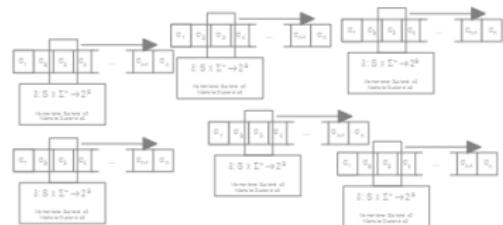


Bild: Mitten in der Verarbeitung von ω



Bild: Ende der Verarbeitung von ω

Arbeitsweise eines NEA II

Beispiel: $N_1 = (Q, \Sigma, \delta, q_0, F)$

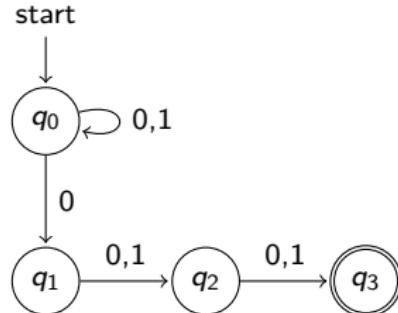
- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{0, 1\}$
- $\delta : Q \times \Sigma \mapsto \mathcal{P}(Q)$

Q/Σ	0	1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\{q_2\}$
q_2	$\{q_3\}$	$\{q_3\}$
q_3	\emptyset	\emptyset

- $F = \{q_3\}$
- Startzustand q_0

Frage: Akzeptiert N_1 die Worte
 $\omega_1 = 01011, \omega_2 = 111?$

Zustandsübergangsdiagramm:



Eingabe 1: 01011 ✓

Eingabe 2: 111 ↛

Bemerkung: Die Folge von Zuständen eines NEAs ist **keine Kette**, sondern ein sich verzweigender Baum mit **mehreren Bearbeitungspfaden**. Zu jedem Zeitpunkt befindet sich ein NEA in mehreren Zuständen **gleichzeitig!**

Sprache eines NEA

Definition (Akzeptierte Sprache)

Die von einem NEA $N = (Q, \Sigma, \delta, q_0, F)$ akzeptierte Sprache ist

$$\begin{aligned}\mathcal{L}(N) &= \{\omega \in \Sigma^* \mid N \text{ akzeptiert } \omega\} \\ &= \{\omega \in \Sigma^* \mid N \text{ hat mindestens eine Möglichkeit, } \omega \text{ vollständig einzulesen und in einem Endzustand zu stoppen}\} \\ &= \{\omega \in \Sigma^* \mid \hat{\delta}(q_0, \omega) \cap F \neq \emptyset\}\end{aligned}$$

Rekursive Definition der **erweiterten Zustandsübergangsfunktion** $\hat{\delta}$

$$\hat{\delta} : Q \times \Sigma^* \rightarrow \mathcal{P}(Q) \quad \text{mit}$$

$$\hat{\delta}(q, \varepsilon) = \{q\}, \text{ für } q \in Q$$

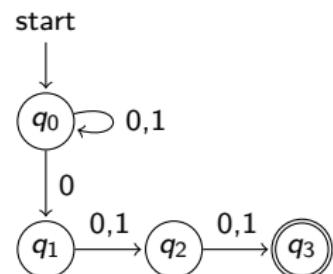
$$\hat{\delta}(q, \sigma) = \delta(q, \sigma), \text{ für } \sigma \in \Sigma$$

$$\hat{\delta}(q, \sigma\omega) = \bigcup_{r \in \delta(q, \sigma)} \hat{\delta}(r, \omega) \quad \text{mit } \sigma \in \Sigma, \omega \in \Sigma^*$$

Beispiel: Sprache des NEA N_1 I

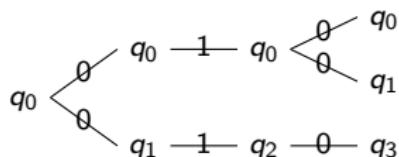
Gesucht: Sprache von N_1 :

- Alle von N_1 akzeptierten Worte ω .
- Alle Worte ω , die auf **mindestens einem** Verarbeitungspfad, der in einem Endzustand endet, komplett eingelesen werden.
- Formal: Alle Worte ω mit $q_3 \in \hat{\delta}(q_0, \omega)$.

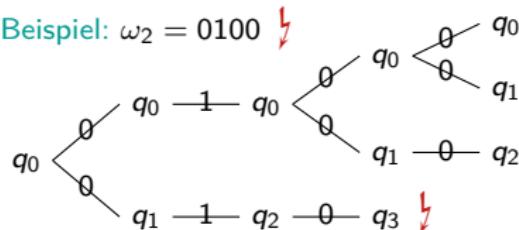


Vorgehensweise: Untersuche Testwörter, nutze $\hat{\delta}$ (siehe nächste Folie), Tabellen oder graphische Darstellung, um Zustandsübergänge nachzuvollziehen. Anschließend: Verallgemeinerung der Erkenntnis.

Beispiel: $\omega_1 = 010$ ✓



Beispiel: $\omega_2 = 0100$ ↘

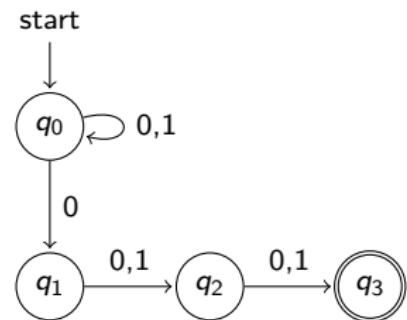


Beispiel: Sprache des NEA N_1 II

Beispiel: Bestimmung von $\hat{\delta}(q_0, \omega)$ für $\omega = 01011$

$$\hat{\delta}(q_0, 01011)$$

$$\begin{aligned} &= \hat{\delta}(\delta(q_0, 0), 1011) \\ &= \hat{\delta}(q_0, 1011) \cup \hat{\delta}(q_1, 1011) \\ &= \hat{\delta}(\delta(q_0, 1), 011) \cup \hat{\delta}(\delta(q_1, 1), 011) \\ &= \hat{\delta}(q_0, 011) \cup \hat{\delta}(q_2, 011) \\ &= \hat{\delta}(\delta(q_0, 0), 11) \cup \hat{\delta}(\delta(q_2, 0), 11) \\ &= (\hat{\delta}(q_0, 11) \cup \hat{\delta}(q_1, 11)) \cup \hat{\delta}(q_3, 11) \\ &= \hat{\delta}(\delta(q_0, 1), 1) \cup \hat{\delta}(\delta(q_1, 1), 1) \cup \hat{\delta}(\delta(q_3, 1), 1) \\ &= \hat{\delta}(q_0, 1) \cup \hat{\delta}(q_2, 1) \cup \emptyset \\ &= \delta(q_0, 1) \cup \delta(q_2, 1) \\ &= q_0 \cup q_3 \checkmark \end{aligned}$$



Lösung: $\mathcal{L}(N_1) = \{\omega \in \{0,1\}^* \mid \text{die drittletzte Stelle von } \omega \text{ ist } 0\} =: L_3$

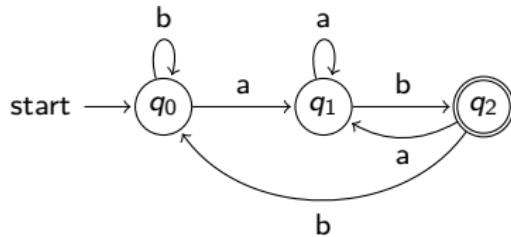
Beispiel: Reguläre Ausdrücke und NEAs

Erinnerung: Für den regulären Ausdruck $r_y = [ab]^* ab$ für $\Sigma = \{a, b\}$ mit $\mathcal{L}(r_y) = \{ab, aab, bab, \dots, abbabbbaaaab, \dots\} = \{\omega ab \mid \omega \in \Sigma^*\}$ und DEA $A_y = (\{q_0, q_1, q_2\}, \Sigma, \delta_D, \{q_2\}, q_0)$ gilt $\mathcal{L}(A_y) = \mathcal{L}(r_y)$.

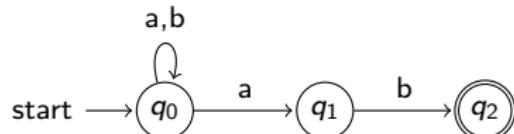
Behauptung: Es ist einfacher, einen NEA

$N_y = (\{q_0, q_1, q_2\}, \Sigma, \delta_N, \{q_2\}, q_0)$ mit $\mathcal{L}(N_y) = \mathcal{L}(r_y)$ zu konstruieren!

δ_D (für A_y) gegeben durch:



δ_N (für N_y) gegeben durch:



Allgemein gilt, dass NEAs einfacher (konzeptionell) zu konstruieren sind als DEAs.

Kleiner Nachteil: NEAs können nicht implementiert werden.

Beispiel: Automatenkonstruktion III

Erinnerung: Erweiterte Passwortanforderung. Erlaube alle Passwörter die

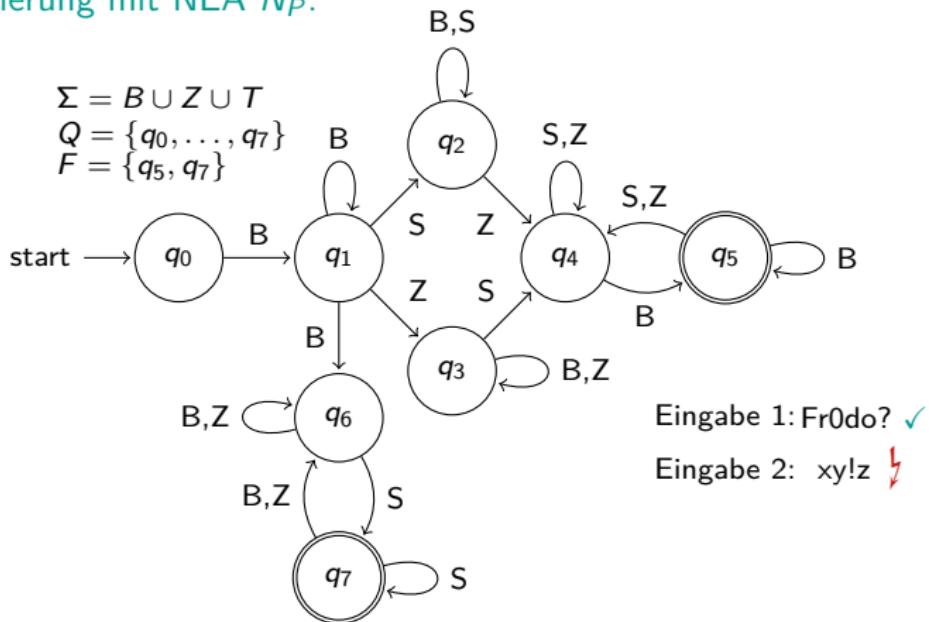
- mindestens 4 Zeichen lang sind,
 - ▶ mit einem Buchstaben (B) beginnen und enden,
 - ▶ mindestens eine Zahl (Z) und ein Sonderzeichen (S) beinhalten.
- Oder mindestens 3 Zeichen lang sind,
 - ▶ mit zwei Buchstaben (B) beginnen,
 - ▶ mit einem Sonderzeichen (S) enden.
- Gültig: AbcD#Ef8Gh, b1l&o, ga!adri3el, ab?, ga!adri3e!, Fr0do?
- Ungültig: a1b, !3golas, Frod0, xy!z

Modellierung durch Erweiterung von A_P zu N_P :

- Behalte A_P als einen Verarbeitungsast bei.
- Füge erweiterte Anforderung als zweiten Alternativ-Ast hinzu.
- Erlaube N_P , sich auf beiden Ästen zu bewegen.
- Akzeptiere Passwort, sobald auf einem Ast ein akzeptierender Zustand erreicht ist.

Beispiel: Automatenkonstruktion IV

Modellierung mit NEA N_P :



Kann man auch einen DEA konstruieren der L_3 akzeptiert?

Erinnerung: $L_3 = \{\omega \in \{0,1\}^* \mid \text{die drittletzte Stelle von } \omega \text{ ist } 0\}$

Antwort: Ja, man kann, dieser wird aber deutlich aufwändiger.

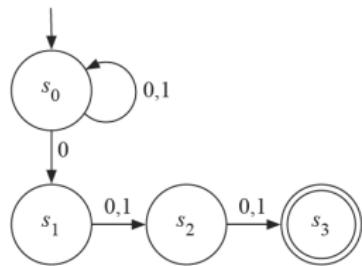


Bild: NEA N_1 mit $\mathcal{L}(N_1) = L_3$ [Hoffmann, 2011]

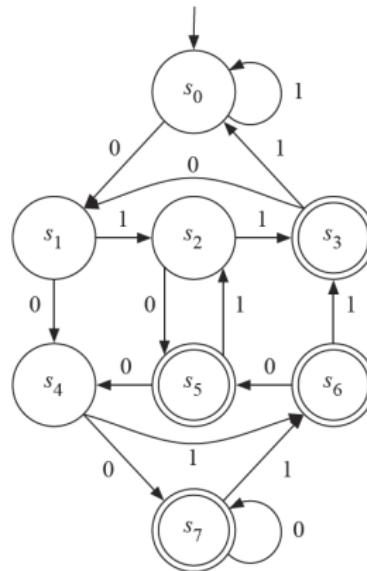


Bild: DEA A_{N_1} mit $\mathcal{L}(A_{N_1}) = L_3$ [Hoffmann, 2011]

Sind NEAs mächtiger als DEAs? I

Satz (Rabin und Scott)

Zu jedem nichtdeterministischen endlichen Automaten gibt es einen deterministischen endlichen Automaten, der die gleiche Sprache akzeptiert:

$$\mathcal{L}(\text{NEA}) = \mathcal{L}(\text{DEA})$$

Beweisidee

- Konstruiere für einen beliebigen NEA $N = (Q, \Sigma, \delta, F, q_0)$ einen DEA $A_N = (Q', \Sigma, \delta', F', q'_0)$ wie folgt
 - ▶ Q' ist die Potenzmenge von Q : Wenn sich N im Zustand q_i oder im Zustand q_j befindet, befindet sich A_N im Zustand $\{q_i, q_j\}$.
 - ▶ $Q' = \mathcal{P}(Q)$ // $q \in Q'$ ist eine Menge von Zuständen von N
 - ▶ $\delta'(\{q_1, \dots, q_n\}, \sigma) = \cup_{i=1}^n \delta(q_i, \sigma)$ // Menge von Zuständen von N ist ein Zustand von A_N
 - ▶ $F' = \{q \in Q' \mid q \cap F \neq \emptyset\}$ // F' enthält alle Zustandsmengen, die Finalzustände von N enthalten
 - ▶ $q'_0 = \{q_0\}$ // q'_0 ist die Menge, welche q_0 enthält

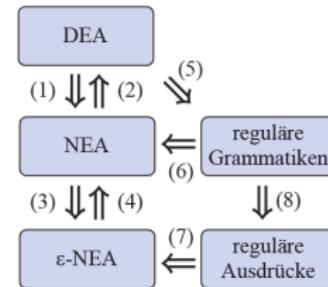
Sind NEAs mächtiger als DEAs? II

Antwort: Nein, beide akzeptieren die gleiche Sprachklasse, nämlich die der **regulären Sprachen \mathcal{L}_3** .

Satz

Die folgenden Aussagen für eine reguläre Sprache L sind äquivalent:

- Es gibt eine reguläre Grammatik G mit $L = \mathcal{L}(G)$.
- Es gibt einen regulären Ausdruck r mit $L = \mathcal{L}(r)$.
- Es gibt einen DEA A mit $L = \mathcal{L}(A)$.
- Es gibt einen NEA N mit $L = \mathcal{L}(N)$.



- (1) jeder DEA ist ein NEA
- (2) Abschnitt 5.3.2
- (3) jeder NEA ist ein ϵ -NEA
- (4) Abschnitt 5.3.3
- (5) Abschnitt 5.4
- (6) Abschnitt 5.4
- (7) Abschnitt 5.4
- (8) siehe z. B. [52]

Bild: Äquivalenzen und Beweise [Hoffmann, 2011]

Abschnitt 3

Transduktoren

Erinnerung: Akzeptoren und Transduktoren im Vergleich

Akzeptoren nehmen ein Wort $\omega = \sigma_1\sigma_2 \dots \sigma_n$ entgegen und entscheiden, ob es gültig ist.

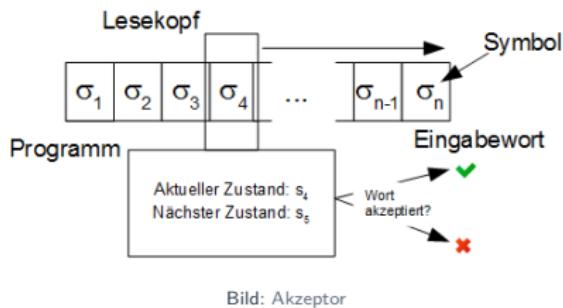


Bild: Akzeptor

Transduktoren nehmen ein Wort $\omega = \sigma_1\sigma_2 \dots \sigma_n$ entgegen und generieren daraus ein Ausgabewort $\psi = \gamma_1\gamma_2 \dots \gamma_n$.

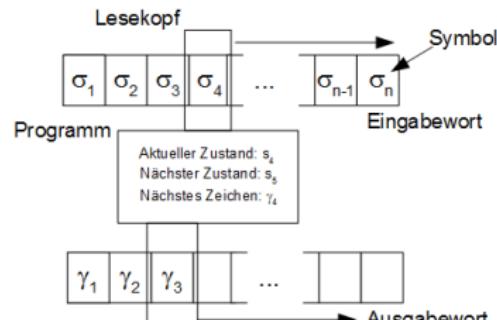


Bild: Transduktor

Deterministischer endlicher Transduktor DET

Definition

Ein **deterministischer endlicher Transduktor** (deterministic finite state transducer), kurz **DET**, ist ein 6-Tupel $T = (Q, \Sigma, \Pi, \delta, \lambda, q_0)$ mit

- der endlichen **Zustandsmenge** Q ,
- dem endlichen **Eingabealphabet** Σ ,
- dem endlichen **Ausgabealphabet** Π ,
- der **Zustandsübergangsfunktion** $\delta : Q \times \Sigma \rightarrow Q$,
- der **Ausgabefunktion** $\lambda : Q \times \Sigma \rightarrow \Pi$ und
- dem **Startzustand** q_0 .

Bemerkungen:

- DET ist ein DEA ohne Finalmenge erweitert um das Ausgabealphabet Π und die Ausgabefunktion λ
- Ein DET besitzt keine Finalmenge, da nur das Ausgabewort zählt.

Arbeitsweise eines DET

- Der Transduktor T beginnt im Startzustand q_0 das Eingabewort ω zu verarbeiten.
- Für $\omega = \sigma_1\sigma_2\dots\sigma_n$ durchläuft T nacheinander die Zustände $q_0, q_1, q_2, \dots, q_n$ mit $q_i = \delta(q_{i-1}, \sigma_i)$ und produziert die Ausgabewort $\pi_1\dots\pi_n$ mit $\pi_i = \lambda(q_{i-1}, \sigma_i)$.
- T hält an (stoppt das Einlesen), wenn $\delta(q_{i-1}, \sigma_i)$ undefiniert ist, es also für gelesenes Symbol und aktuellen Zustand keinen nächsten Zustand gibt.
- Nachdem das Wort ω vollständig eingelesen wurde (T also nicht angehalten hat), steht das Ausgabewort $\pi_1\dots\pi_n$ auf dem Ausgabeband.

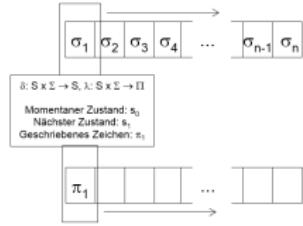


Bild: Start der Verarbeitung von ω

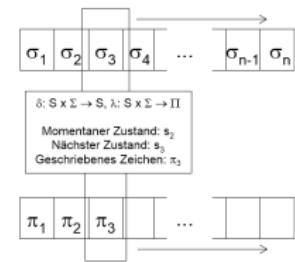


Bild: Mitten in der Verarbeitung von ω

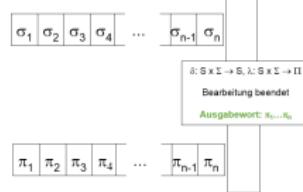


Bild: Ende der Verarbeitung von ω

Beispiel: ROT13-Rechner

Konstruktionsaufgabe: Transduktor T_{13} mit

- Eingabe: Kleinbuchstaben
- Ausgabe: Um 13 Stellen im Alphabet verschobene Großbuchstaben
- Beispiel: Eingabe = $abc \rightarrow$ Ausgabe = NOP

Lösung: $T_{13} = (Q, \Sigma, \Pi, \delta, \lambda, q_0)$ mit

- $Q = \{q_0\}$
- $\Sigma = \{a, b, \dots, z\}$
- $\Pi = \{A, B, \dots, Z\}$
- $\forall_{\sigma \in \Sigma} \delta(q_0, \sigma) = q_0$
- $\forall_{\sigma \in \Sigma} \lambda(q_0, \sigma) = \sigma_2$ mit σ_2 gegeben durch Tabelle:

σ	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
σ_2	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M

Beispiel: Eingabe = aleaiactaest \rightarrow Ausgabe = NYRNVNPGNRFG

Beispiel: Binär-Gray-Code-Wandler I

Gray-Codes sind binäre Codes zur Darstellung von Zahlen. Zwei benachbarte Zahlen unterscheiden sich immer um genau ein Bit (im Unterschied zu Dualzahlen).

Anwendungen: Robuste Codes für fehleranfällige Umgebungen (Sensordatenverarbeitung, Automatisierungs-, Nachrichtentechnik, ...).

$T_G = (Q, \Sigma, \Pi, \delta, \lambda, q_0)$ mit

- $Q = \{q_0, q_1, \dots, q_6\}$
- $\Sigma = \Pi = \{0, 1\}$
- δ und λ siehe erweitertes Überführungsdiagramm auf der nächsten Folie

setzt (dreistellige) Binärzahlen in (dreistellige) Gray-Code-Zahlen um.

Dezimal	Binär	Gray-Code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Beispiel: Binär-Gray-Code-Wandler II

Kernidee: Die i -te Stelle g_i einer Gray-Zahl $G = g_1g_2 \dots g_n$ berechnet sich aus der Binärzahl $B = b_1b_2 \dots b_n$ als

$$g_i = \begin{cases} b_{i-1} \oplus b_i & \text{für } i > 1 \\ 0 \oplus b_1 = b_1 & \text{für } i = 1. \end{cases}$$

T_G muss sich also das letzte Eingabezeichen merken, um daraus und dem aktuellen Eingabezeichen die nächste Gray-Stelle zu berechnen.

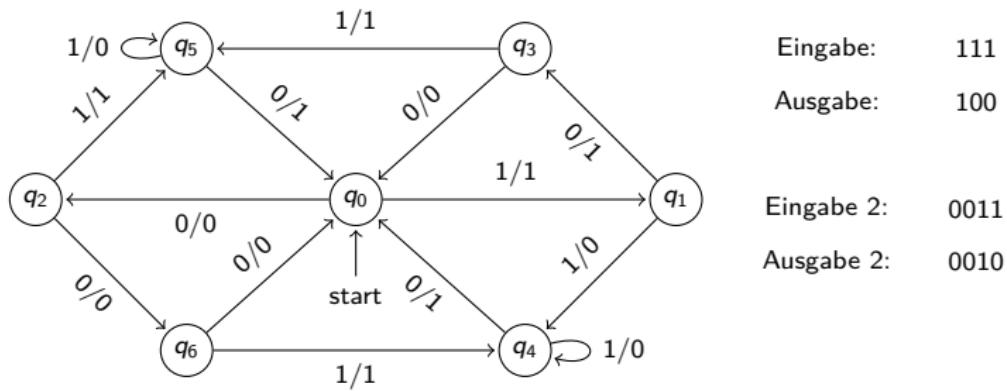


Bild: Erweitertes Übergangsdiagramm von T_G : „ x/y “ steht für „Zeichen x lesen, Zeichen y schreiben“

Mealy- und Moore-Automaten

Bemerkung: Transduktoren sind sehr beliebt für den Schaltungs-Entwurf, da Sie sich leicht in Hardware übersetzen lassen.

Weitere Bezeichnungen (für $T = (Q, \Sigma, \Pi, \delta, \lambda, q_0)$):

Mealy-Automaten sind Transduktoren

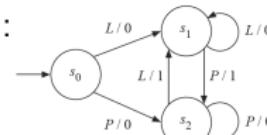
welche das Ausgabezeichens sowohl mit dem aktuellen Zustand als auch mit dem aktuellen Eingabezeichen berechnen, also

$$\pi_i = \lambda(q_i, \sigma_i).$$

Moore-Automaten sind Transduktoren

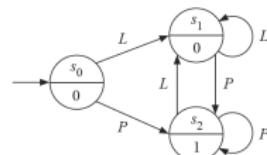
welche das Ausgabezeichens nur in Abhängigkeit vom aktuellen Zustand berechnen, also

$$\pi_i = \lambda(q_i).$$



s	s0	s1	s2
$\delta(s, L)$	s_1	s_1	s_1
$\delta(s, P)$	s_2	s_2	s_2
$\lambda(s, L)$	0	0	1
$\lambda(s, P)$	0	1	0

Bild: Ein Mealy-Automat [Hoffmann, 2011]



s	s0	s1	s2
$\delta(s, L)$	s_1	s_1	s_1
$\delta(s, P)$	s_2	s_2	s_2
$\lambda(s)$	0	0	1

Bild: Ein Moore-Automat [Hoffmann, 2011]

Beispiel: ROT13-Rechner als Moore-Automat

Konstruktionsaufgabe: Transduktor $T_{Moore13}$ mit

- Eingabe: Kleinbuchstaben
- Ausgabe: Um 13 Buchstaben verschobene Großbuchstaben
- Beispiel: Eingabe = $abc \rightarrow$ Ausgabe = NOP
- Einschränkung: Ausgabe hängt nur vom Zustand ab.

Lösung: $T_{Moore13} = (Q, \Sigma, \Pi, \delta, \lambda, q_0)$ mit

- $Q = \{q_0, q_A, q_B, \dots, q_Z\}$
- $\Sigma = \{a, b, \dots, z\}$
- $\Pi = \{A, B, \dots, Z\}$
- $\forall_{\sigma \in \Sigma} \forall_{q \in Q} \delta(q, \sigma) = q_\gamma$ mit q_γ gegeben durch Tabelle:

σ	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
q_γ	q_N	q_O	q_P	q_Q	q_R	q_S	q_T	q_U	q_V	q_W	q_X	q_Y	q_Z	q_A	q_B	q_C	q_D	q_E	q_F	q_G	q_H	q_I	q_J	q_K	q_L	q_M
q_γ	q_N	q_O	q_P	q_Q	q_R	q_S	q_T	q_U	q_V	q_W	q_X	q_Y	q_Z	q_A	q_B	q_C	q_D	q_E	q_F	q_G	q_H	q_I	q_J	q_K	q_L	q_M

- $\forall_{\gamma \in \Gamma} \lambda(q_\gamma) = \gamma$ und $\lambda(q_0) = (\text{Leerzeichen})$

Beispiel: Eingabe = aleaiactaest \rightarrow Ausgabe = NYRNVNPGNRFG
(identisch zur Ausgabe des Mealy-Automats, aber mit einem führenden Leerzeichen)

Verwendete oder empfohlene Literatur I

[Hedtstück, 2012] Hedtstück, U. (2012).

Einführung in die theoretische Informatik: formale Sprachen und Automatentheorie.

Oldenbourg Verlag.

Als eBook in der HTWG-Bibliothek verfügbar.

[Hoffmann, 2011] Hoffmann, D. W. (2011).

Theoretische Informatik.

Carl Hanser Verlag GmbH & Co. KG, 2. Auflage.

Als eBook in der HTWG-Bibliothek verfügbar.

[Hoffmann, 2015] Hoffmann, D. W. (2015).

Theoretische Informatik.

Carl Hanser Verlag GmbH & Co. KG, 3. Auflage.

Als eBook in der HTWG-Bibliothek verfügbar.

Verwendete oder empfohlene Literatur II

[Hopcroft et al., 2011] Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2011).

Einführung in die Automatentheorie, formale Sprachen und Berechenbarkeit (bzw. Komplexitätstheorie); engl.: Introduction to automata theory, languages and computation.

Pearson, 3. Auflage.

Als eBook in der HTWG-Bibliothek verfügbar.

[Kulla, 2015] Kulla, S. (2015).

Mathe für Nicht-Freaks - Grundlagen der Mathematik.

Wikibooks.

[Teschl and Teschl, 2013] Teschl, G. and Teschl, S. (2013).

Mathematik für Informatiker: Band 1: Diskrete Mathematik und Lineare Algebra.

Springer Vieweg, 4. Auflage.

Als eBook in der HTWG-Bibliothek verfügbar.