Minimalautomat für reguläre Sprache → Abschnitt 2.4.3

$L \subseteq \Sigma^*$ regulär
$\Rightarrow$ der Äquivalenzklassen-Automat zu $\sim_L$ ist
ein DFA mit minimaler Zustandszahl ($= \mathrm{index}(\sim_L)$)
unter allen DFA, die L akzeptieren

Satz **(Lemma 2.4.13)**

jeder DFA mit $\mathrm{index}(\sim_L)$ Zuständen, der L erkennt,
ist isomorph zum Äquivalenzklassen-Automat zu $\sim_L$

Äquivalenzklassen-Automat zu $\sim_L$: *der* Minimalautomat zu L

Minimierung eines gegebenen DFA

wie kann man redundante Zustände erkennen/eliminieren?

- zunächst eliminiere alle nicht erreichbaren Zustände
- dann: $q \not\sim q'$ *wesentlich verschieden*, wenn
  für ein $x \in \Sigma^*$ *nicht* $(\hat\delta(q,x) \in A) \leftrightarrow (\hat\delta(q',x) \in A)$
- Zusammenfassung von $\sim$-Klassen von Zuständen liefert minimierten DFA, isomorph zum Minimalautomat

algorithmisch: induktive Verfeinerung

Tests für x der Länge $i = 0, 1, \ldots$

$q \not\sim_0 q'$ gdw. *nicht* $(q \in A) \Leftrightarrow (q' \in A)$

$q \not\sim_{i+1} q'$ gdw. $q \not\sim_i q'$ oder $(\exists a \in \Sigma)\ \delta(q,a) \not\sim_i \delta(q',a)$

sobald $\sim_{i+1} = \sim_i$ ($=: \sim$), fasse Zustände in $\sim$-Klassen zusammen

Das "pumping lemma" → Abschnitt 2.5

Pumping Lemma **(Lemma 2.5.2)**

$L \subseteq \Sigma^*$ regulär $\Rightarrow$

existiert $n \in \mathbb{N}$ sodass sich jedes $x \in L$ mit $|x| \geqslant n$ zerlegen lässt in $x = uvw$, $v \neq \varepsilon$, $|uv| \leqslant n$ und für alle $m \in \mathbb{N}$

$$u \cdot v^m \cdot w = u \cdot \underbrace{v \cdots v}_{m \text{ mal}} \cdot w \in L$$

Beweis: $L = L(\mathcal{A})$, DFA $\mathcal{A} = (\Sigma, Q, q_0, \delta, A)$, $n := |Q|$

$|x| \geqslant n$: existiert Zustandswiederholung $q_i = q_j = q$ im Lauf auf x:

$$x = \underset{\uparrow q_0}{} \underbrace{a_1 \cdots a_i}_{u} \underbrace{a_{i+1} \cdots a_j}_{v} \underbrace{a_{j+1} \cdots a_\ell}_{w}$$

(q nach a_i und nach a_j)

lässt sich pumpen

Beispiele nicht-regulärer Sprachen

$L = \{a^n b^n : n \in \mathbb{N}\}$ für $a, b \in \Sigma$, $a \neq b$

$L = \{w \in \{(,)\}^* : w$ korrekte Klammerschachtelung $\}$

$\textsc{Palindrom} = \{w \in \Sigma^* : w = w^{-1}\}$ für $|\Sigma| \geqslant 2$

$L = \{u\#v\#w : u, v, w \in \{0,1\}^*, u + v = w$ (binäre Addition) $\}$

Nachweis: Pumping Lemma!

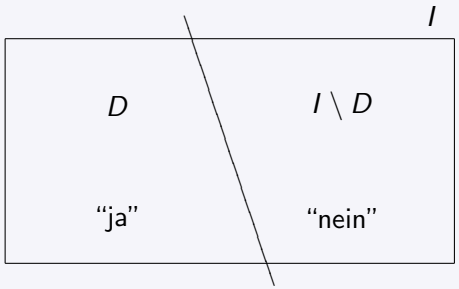
Algorithmische Entscheidungsprobleme → Abschnitt 2.6

Entscheidungsproblem (ja/nein Problem)

spezifiert durch

- Menge von zugelassenen Eingaben $x \in I$ (Instanzen)
- Teilmenge $D \subseteq I$ der positiven Instanzen (Antwort "ja")

d.h.: *wohldefinierte* ja/nein Antwort für alle $x \in I$



Entscheidungsproblem:
gegeben $x \in I$
entscheide, ob $x \in D$

Entscheidungsprobleme für reguläre Sprachen Beispiele

Wortproblem zu Sprache $L \subseteq \Sigma^*$:

für $w \in \Sigma^*$ entscheide ob $w \in L$

$I = \Sigma^*$, $D = L$

Leerheitsproblem:

für DFA $\mathcal{A}$ entscheide ob $L(\mathcal{A}) = \emptyset$

$I = \{\mathcal{A} \colon \mathcal{A} \text{ DFA }\}$, $D = \{\mathcal{A} \colon L(\mathcal{A}) = \emptyset\}$

analog mit $I = \mathrm{REG}(\Sigma)$

Sprachgleichheit/Automatenäquivalenz:

für $\alpha, \beta \in \mathrm{REG}(\Sigma)$ entscheide ob $L(\alpha) = L(\beta)$

$I = \big(\mathrm{REG}(\Sigma)\big)^2$, $D = \{(\alpha, \beta) \in (\mathrm{REG}(\Sigma))^2 \colon L(\alpha) = L(\beta)\}$

analog für DFA/NFA

das Wichtigste aus Kapitel 2

reguläre Sprachen

REG **DFA** **NFA**

Abschlusseigenschaften / Automatenkonstruktionen

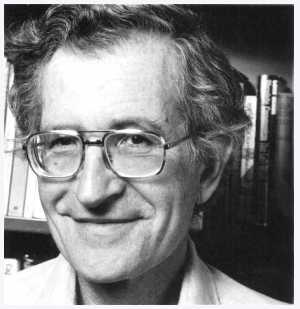
Satz von Kleene

Satz von Myhill-Nerode

Pumping Lemma für reguläre Sprachen

Minimalautomat

Kapitel 3: Grammatiken
Chomsky Hierarchie: Komplexitätsniveaus für formale Sprachen



Noam Chomsky (geb. 1928)

Grammatiken → Abschnitt 3.1

Idee: Spezifikation einer Sprache durch *Erzeugungsprozess*

$G = (\Sigma, V, P, S)$

Σ	*Terminalalphabet*,
V	endliche Menge von *Variablen*, $V \cap \Sigma = \emptyset$
$S \in V$	*Startvariable/Startsymbol*
P	endliche Menge von *Produktionen/Regeln*

$P \subseteq \big((V \cup \Sigma)^* V (V \cup \Sigma)^*\big) \times (V \cup \Sigma)^*$
$(v, v') \in P$ eine Produktion/Regel von G

Regel
$v \to v'$

linke Seite: $v \in (V \cup \Sigma)^* V (V \cup \Sigma)^*$
rechte Seite: $v' \in (V \cup \Sigma)^*$

erlaubt in $(V \cup \Sigma)$-Wörtern { *Ersetzung von v durch v'* / Übergang von uvw nach $uv'w$

Ableitbarkeit

zu Produktionen $v \to v'$ in P:

$uvw \to_G uv'w$
direkter *Ableitungsschritt* in G

zwei Ableitbarkeitsrelationen über $(\Sigma \cup V)^*$:

1-Schritt Ableitbarkeit, $\to_G$:
$x \to_G x' \;:\Leftrightarrow\; x = uvw,\ x' = uv'w$ wobei $(v, v') \in P$

Ableitbarkeit, $\to_G^*$:
$x \to_G^* x' \;:\Leftrightarrow$ { es existiert ein $\to_G$-Pfad von x nach x': / $x = x_1 \to_G x_2 \to_G \ldots \to_G x_n = x'$.

"endliche Iteration von $\to_G$": reflexiver transitiver Abschluss

- $w \in (\Sigma \cup V)^*$ *ableitbar in* G gdw. $S \to_G^* w$
- $L(G) = \{w \in \Sigma^* : S \to_G^* w\}$ die *von G erzeugte Σ-Sprache*

Beispiel Beispiel 3.1.3

Palindrom (3.1.3)

z.B. $\Sigma = \{0, 1\}$,
$G = (\Sigma, V, P, X)$ mit $V = \{X\}$ und Produktionen

$$\begin{aligned} P:\quad & X \to \varepsilon \\ & X \to 0 \\ & X \to 1 \\ & X \to 0X0 \\ & X \to 1X1 \end{aligned}$$

erzeugt die Σ-Sprache der Σ-Palindrome ($w \in \Sigma^*$: $w = w^{-1}$)

beachte: nicht regulär

Beispiele Beispiele 3.1.4

korrekte Klammerungen (3.1.4)

$\Sigma = \{(,)\}$, $G = (\Sigma, V, P, X)$ mit $V = \{X\}$ und Produktionen

$$\begin{aligned} P:\quad & X \to (\;) \\ & X \to (X) \\ & X \to XX \end{aligned}$$

erzeugt die Sprache der korrekt geschachtelten
nicht-leeren Klammerwörter [Induktionsbeweis!]

beachte: nicht regulär

Beispiele: reguläre Sprachen

Beispiel 3.1.7

Automaten 'fressen' Wörter, Grammatiken 'spucken' sie aus.

Jede reguläre Sprache wird auch von einer Grammatik erzeugt.

Trick: übersetze NFA $\mathcal{A} = (\Sigma, Q, q_0, \Delta, A)$ in Grammatik:

$$G_{\mathcal{A}} := (\Sigma, V, P, S) \quad \begin{cases} V & := \{X_q : q \in Q\}, \\ S & := X_{q_0}, \\ P & : X_q \to aX_{q'} \quad \text{für } (q, a, q') \in \Delta \\ & \quad X_q \to \varepsilon \quad \text{für } q \in A \end{cases}$$

dann ist $L = L(\mathcal{A}) = L(G_{\mathcal{A}})$: Ableitungen in G simulieren akzeptierende Berechnungen von $\mathcal{A}$

Beispiel $a^n b^n$

(3.1.10)

$\Sigma = \{a, b\}$, $G = (\Sigma, V, P, S)$, $V = \{S\}$,

$$P: \quad S \to \varepsilon, \quad S \to aSb$$

erzeugt die Sprache $L = \{a^n b^n : n \in \mathbb{N}\}$. **nicht regulär**

Beispiel $a^n b^n c^n$

(3.1.11)

$\Sigma = \{a, b, c\}$, $G = (\Sigma, V, P, S)$, $V = \{S, B, C\}$,

$$P: \quad S \to \varepsilon, \quad S \to aSBC, \quad CB \to BC, \quad aB \to ab, \quad bB \to bb, \quad bC \to bc, \quad cC \to cc$$

erzeugt die Sprache $L = \{a^n b^n c^n : n \in \mathbb{N}\}$. **nicht regulär**

Beispiel

Übung 3.1.5

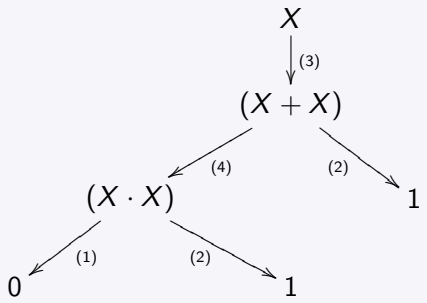
Baumdarstellung von Ableitungsschritten

$\Sigma = \{+, \cdot, (,), 0, 1\}$

$G = (\Sigma, \{X\}, P, X)$:

$$P: \quad X \to 0 \ (1), \quad X \to 1 \ (2), \quad X \to (X + X) \ (3), \quad X \to (X \cdot X) \ (4)$$

Baum zu $((0 \cdot 1) + 1) \in L(G)$:



Backus-Naur Form, BNF

(Kompaktschreibweise)

$v \to v_1' \mid v_2' \mid \ldots \mid v_n'$ anstelle von $v \to v_1'$, $\vdots$, $v \to v_n'$

lies "|" als "oder"; oft auch ::= anstelle von $\to$

erweiterte BNF, EBNF

weitere eliminierbare Abkürzungen

z.B. $X \to u[v]w$ für $X \to uw \mid uvw$

"v *kann* zwischen u und w eingefügt werden"

z.B.

$X \to u\{v\}w$ für $X \to uw \mid uZw$ [Z neu!]

$Z \to ZZ \mid v$

"jedes $v' \in \{v\}^*$ kann zwischen u und w eingefügt werden"

Chomsky-Hierarchie → Abschnitt 3.2

Klassifikation von Grammatiken (und Sprachen)

X, Y Variablen in V, $a \in \Sigma$, $v, v' \in (\Sigma \cup V)^*$

regulär, Typ 3
Produktionen *rechtslinear*: $X \to \varepsilon$, $X \to a$, $X \to aY$

kontextfrei, Typ 2
alle Produktionen haben als linke Seite
eine einzelne *Variable*: $X \to v$

kontextsensitiv, Typ 1
alle Produktionen *nicht verkürzend*: $v \to v'$, $|v'| \geqslant |v|$
bis auf ggf. eine *harmlose* ε-Produktion

allgemein, Typ 0
allgemeinste von Grammatik erzeugte Sprachen (!)

reguläre Grammatiken – endliche Automaten Satz 3.2.7

reguläre Grammatiken charakterisieren reguläre Sprachen

$L \subseteq \Sigma^*$ regulär gdw. $L = L(G)$ für eine reguläre Grammatik G.

Beweis:

"$\Leftarrow$": NFA aus Typ 3 Grammatik $G = (\Sigma, V, P, S)$
mit Produktionen $X \to aY$ und $X \to \varepsilon$

(zur Vorbereitung: ersetze $X \to a$ durch $X \to aZ$ und $Z \to \varepsilon$, Z neu)

Ableitung $S \to_G a_1X_1 \to_G \cdots \to_G a_1 \ldots a_nX_n \to_G a_1 \ldots a_n$

simuliert durch $\mathcal{A} := \big(\Sigma, V, S, \Delta, A\big)$ mit

$$\Delta = \big\{(X, a, Y) \colon X \to aY \in P\big\},$$
$$A = \big\{X \colon X \to \varepsilon \in P\big\}$$

regulär ⇒ kontextfrei ⇒ kontextsensitiv Beobachtung 3.2.5

- offensichtlich: Jede reguläre Grammatik ist kontextfrei.
- ebenso: G kontextfrei $\begin{cases} \textit{ohne } \varepsilon\text{-Produktionen} \\ \text{oder mit } \textit{harmloser } \varepsilon\text{-Produktion} \end{cases}$

  $\Rightarrow$ G kontextsensitiv.
- kontextfreies G *mit* ε-Produktionen
  äquivalent zu kontextfreiem G' mit *harmloser* ε-Produktion.

  Begr.: (Lemma 3.2.4) o.B.d.A. hat G S nur als Startsymbol.

  – für $X \to uYv$ mit $Y \to_G^* \varepsilon$ nehme $X \to uv$ hinzu
  – entferne alle ε-Produktionen, bis auf $S \to \varepsilon$

Chomsky-Hierarchie

regulär ⇒ kontextfrei ⇒ kontextsensitiv

Typ 3 ⊆ Typ 2 ⊆ Typ 1 ⊆ Typ 0 ⊆ beliebige Sprachen

wir werden sehen: alle Inklusionen echt / *strikte Hierarchie*

trennende Beispiele (die wir schon kennen):

kontextfrei, nicht-regulär: $\{a^nb^n \colon n \in \mathbb{N}\}$
Palindrom
korrekte Klammerungen

kontextsensitiv, nicht kontextfrei: $\{a^nb^nc^n \colon n \in \mathbb{N}\}$
noch zu zeigen

Methoden für Nachweis der Nicht-Zugehörigkeit:

vgl. Pumping Lemma (für Typ 3; Variante für Typ 2: siehe unten)
allgemein: aus Analyse der einzelnen Klassen