Theoretische Informatik I

Roland Meyer

TU Braunschweig

Inhalt I

Part B Reguläre Sprachen

1. Reguläre Sprachen und endliche Automaten

Grundbegriffe

Definition (Worte)

- Ein Alphabet ist eine *endliche* Menge sog. Buchstaben, z.B. $\Sigma = \{a, b, c\}$.
- Ein Wort über Σ ist eine endliche Folge von Buchstaben aus dem Alphabet: $w = a_1 \dots a_n$ mit $a_i \in \Sigma$ für alle $i \in [1, n]$.
- Die Länge von w ist $|w| := n$.
- Der i-te Buchstabe in w ist $w(i) := a_i$.
- Die leere Buchstabenfolge heißt leeres Wort ε mit $|\varepsilon| := 0$.
- Die Menge aller Worte über Σ ist Σ^*.
- Die Konkatenation zweier Worte $w, v \in \Sigma^*$ ist $w.v \in \Sigma^*$. Es gilt

$$|w.v| = |w| + |v|.$$

Bemerkung

Die Automatentheorie lässt sich auch betreiben über

endlichen Bäumen, unendlichen Worten und unendlichen Bäumen.

Endliche Bäume unterscheiden sich kaum von endlichen Worten.

Grundbegriffe

Definition (Sprachen und Operationen)

- Eine Sprache über Σ ist eine (meist unendliche) Menge $L \subseteq \Sigma^*$ von Worten über dem Alphabet.
- Da Sprachen Mengen sind, lassen sich mengentheoretische Operationen anwenden. Mit $L_1, L_2 \subseteq \Sigma^*$ sind die gängigsten:

$$\underset{\text{Vereinigung}}{L_1 \cup L_2} \qquad \underset{\text{Schnitt}}{L_1 \cap L_2} \qquad \underset{\text{Differenz}}{L_1 \setminus L_2} \qquad \underset{\text{Komplement}}{\overline{L_1} := \Sigma^* \setminus L_1}.$$

- Auch die Konkatenation wird auf Sprachen geliftet:

$$L_1.L_2 := \{w.v \in \Sigma^* \mid w \in L_1 \text{ und } v \in L_2\}.$$

- Die unäre Operation Kleene-Stern liefert $L^* := \bigcup_{i \in \mathbb{N}} L^i$. Dabei sind die Sprachen L^i induktiv definiert mittels

$$L^0 := \{\varepsilon\} \qquad \text{und} \qquad L^{i+1} := L.L^i$$

für alle $i \in \mathbb{N} := \{0, 1, 2, \ldots\}$.

Grundbegriffe

Bemerkung

Man nennt den Kleene-Stern auch kleenesche Hülle.

Es gilt $L^* = \{w_1 \dots w_n \in \Sigma^* \mid n \in \mathbb{N} \text{ und } w_i \in L \text{ für alle } i \in [1, n]\}$.

Man definiert auch die unäre Operation Kleene-Plus oder positive Hülle:

$$L^+ := \bigcup_{i \in \mathbb{N} \setminus \{0\}} L^i.$$

Es gilt $L^+ = L^* \setminus \{\varepsilon\}$, sofern $\varepsilon \notin L$.

Verträglichkeiten (Rechenregeln)

Zwischen den Operatoren $\cup$, . und * auf Sprachen über Σ gilt:

$$
\begin{aligned}
L_1 \cup (L_2 \cup L_3) &= (L_1 \cup L_2) \cup L_3 && \text{(Assoziativität)} \\
L_1.(L_2.L_3) &= (L_1.L_2).L_3 \\
L_1 \cup L_2 &= L_2 \cup L_1 && \text{(Kommutativität)} \\
(L_1 \cup L_2).L &= (L_1.L) \cup (L_2.L) && \text{(Distributivität)} \\
L.(L_1 \cup L_2) &= (L.L_1) \cup (L.L_2) \\
\{\varepsilon\}.L &= L = L.\{\varepsilon\} && \text{(Neutrale Elemente)} \\
\emptyset \cup L &= L \\
\emptyset.L &= \emptyset = L.\emptyset && \text{(Absorbierendes Element)} \\
L \cup L &= L && \text{(Idempotenz)} \\
L^* &= (L^*)^* \\
L^* &= \{\varepsilon\} \cup L^+ \\
L^+ &= L.L^* = L^*.L\ .
\end{aligned}
$$

Verträglichkeiten (Rechenregeln)

Bemerkung

- $(\mathbb{P}(\Sigma^*), \cup, \emptyset)$ ist ein kommutatives Monoid.
- $(\mathbb{P}(\Sigma^*), . , \{\varepsilon\})$ ist ein Monoid.
- $(\mathbb{P}(\Sigma^*), \cup, . , \emptyset, \{\varepsilon\})$ ist ein Semiring.
- Berücksichtigt man weitere (nicht genannte) Verträglichkeiten mit dem Kleene-Stern, landet man bei Kleene-Algebren.
- Mit den mengentheoretischen Operationen $\oplus \in \{\cup, \cap\}$ und einigen der unten angeführten Verträglichkeiten erhalten wir eine boolesche Algebra.

$$
\begin{aligned}
L_1 \oplus (L_2 \oplus L_3) &= (L_1 \oplus L_2) \oplus L_3 && \text{(Assoziativität)} \\
L_1 \oplus L_2 &= L_2 \oplus L_1 && \text{(Kommutativität)} \\
L \oplus L &= L && \text{(Idempotenz)} \\
(L_1 \cup L_2) \cap L &= (L_1 \cap L) \cup (L_2 \cap L) && \text{(Distributivität)} \\
(L_1 \cap L_2) \cup L &= (L_1 \cup L) \cap (L_2 \cup L)
\end{aligned}
$$

Verträglichkeiten (Rechenregeln)

(cont.)

$$\begin{aligned} \emptyset \cup L &= L && \text{(Neutrale Elemente)} \\ \Sigma^* \cap L &= L \\ \Sigma^* \cup L &= \Sigma^* && \text{(Absorbierende Elemente)} \\ \emptyset \cap L &= \emptyset \\ \overline{L_1 \cup L_2} &= \overline{L_1} \cap \overline{L_2} && \text{(De Morgan)} \\ \overline{L_1 \cap L_2} &= \overline{L_1} \cup \overline{L_2} \\ \overline{\overline{L}} &= L && \text{(Involution)} \\ L_1 \setminus L_2 &= L_1 \cap \overline{L_2} \\ L \cup \overline{L} &= \Sigma^* && \text{(Komplementarität)} \\ L \cap \overline{L} &= \emptyset \end{aligned}$$

Verträglichkeiten (Rechenregeln)

(cont.)

$$\begin{aligned} \overline{\Sigma^*} &= \emptyset && \text{(Dualität)} \\ \overline{\emptyset} &= \Sigma^* \\ L_1 \cap (L_1 \cup L_2) &= L_1 && \text{(Absorption)} \\ L_1 \cup (L_1 \cap L_2) &= L_1 \end{aligned}$$

Eine boolesche Algebra ist insbesondere ein distributiver Verband.

Reguläre Sprachen

Definition (Reguläre Sprachen)

Die Klasse der regulären Sprachen über dem Alphabet Σ, geschrieben als REG_Σ, ist die kleinste Klasse an Sprachen, die Folgendes erfüllt:

(1) $\emptyset \in \mathrm{REG}_\Sigma$ und $\{a\} \in \mathrm{REG}_\Sigma$ für alle $a \in \Sigma$ und

(2) falls $L_1, L_2 \in \mathrm{REG}_\Sigma$, dann auch $L_1 \cup L_2, L_1.L_2, L_1^* \in \mathrm{REG}_\Sigma$.

Bemerkung

Die Definition ist formuliert als kleinste Lösung einer rekursiven Gleichung.
Mit den Überlegungen im ersten Zusatzvideo ist sie äquivalent zu einer induktiven Definition mit Basisfall (1) und Induktionsschritt (2).

Jede reguläre Sprache ergibt sich durch Anwendung endlich vieler Operationen in (2) aus den Sprachen in (1).

Reguläre Sprachen

Notation

- Operatorpräzedenzen: * bindet stärker als . bindet stärker als $\cup$.
- Lasse Klammern bei Buchstaben fallen und schreibe $\{a\}$ als a.

Beispiel: $\varepsilon \cup (a \cup b)^*.b$.
Wir haben ε als reguläre Sprache, da $\{\varepsilon\} = \emptyset^*$.

Abschlusseigenschaften regulärer Sprachen

Beobachtung

- Endliche Wortmengen bilden reguläre Sprachen.
- Die regulären Sprachen sind nicht abgeschlossen unter unendlichen Vereinigungen (dann bekommt man alle Sprachen).
- Per Definition sind die regulären Sprachen abgeschlossen unter *, $.$ und $\cup$.

Ziel

- Zeige, dass REG_Σ auch unter den übrigen mengentheoretischen Operationen abgeschlossen ist: $\overline{\ }$, $\cap$ und $\setminus$.
- Im Grunde benötigen wir nur $\overline{\ }$, denn

$$\begin{aligned} L_1 \cap L_2 &= \overline{\overline{L_1} \cup \overline{L_2}} \\ L_1 \setminus L_2 &= L_1 \cap \overline{L_2}. \end{aligned}$$

- Um diesen Beweis zu führen, benötigen wir eine alternative Charakterisierung der regulären Sprachen.

Endliche Automaten

Wir charakterisieren die regulären Sprachen über endliche Automaten.

Die Charakterisierung gibt mehr her als Abschlusseigenschaften.
Sie ist eine Darstellung der Sprachen, auf der effizient gerechnet werden kann.

Außerdem ist die Darstellung an sich interessant, da Sprachen unendlich sind.
Unendliche Mengen endlich darzustellen ist in der Regel schwierig und tatsächlich eine der Sportarten der Theoretischen Informatik.

Endliche Automaten: Syntax

Wir halten das Alphabet Σ fest.

Definition (Endlicher Automat)

Ein nichtdeterministischer endlicher Automat (NFA) über Σ ist ein Tupel $A = (Q, q_0, \to, Q_F)$ mit

- einer endlichen Menge an Zuständen Q, einem initialen Zustand $q_0 \in Q$, einer Endzustandsmenge $Q_F \subseteq Q$ und
- einer Transitionsrelation $\to \subseteq Q \times \Sigma \times Q$.

Wir schreiben oft $q \xrightarrow{a} q'$ für $(q, a, q') \in \to$.

Endliche Automaten: Semantik

Definition (Ablauf und Sprache)

Ein Ablauf von $A = (Q, q_0, \to, Q_F)$ ist eine Folge von Transitionen der Form

$$q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \dots q_{n-1} \xrightarrow{a_{n-1}} q_n.$$

Der Zielzustand einer Transition ist also der Startzustand der nächsten.

Wir sprechen von einem Ablauf von A auf dem Wort $w := a_0 \dots a_{n-1}$.

Wir schreiben auch $q_0 \xrightarrow{w} q_n$ für die Tatsache, dass es einen Ablauf von A auf w gibt, der von q_0 zu q_n führt.

Der Ablauf ist akzeptierend, falls $q_n \in Q_F$.

Die Sprache von A ist

$$L(A) \quad := \quad \{w \in \Sigma^* \mid q_0 \xrightarrow{w} q \text{ mit } q \in Q_F\},$$

also die Menge an Worten, für die es einen akzeptierenden Ablauf gibt.

Falls $L = L(A)$, dann wird die Sprache L von A akzeptiert oder erkannt.

Von regulären Sprachen zu endlichen Automaten

Ziel

Zeige, dass die regulären Sprachen von NFAs erkannt werden.

Falls $L \in \mathrm{REG}_\Sigma$, dann gibt es einen NFA A mit $L = L(A)$.

Idee

Wende die Operationen von REG auf NFAs an.

Satz (Abschluss der NFA-erkennbaren Sprachen unter . und $\cup$)

Betrachte NFAs A_1 und A_2.

(1) Es gibt einen NFA $A_1.A_2$ mit $L(A_1.A_2) = L(A_1).L(A_2)$.

(2) Es gibt einen NFA $A_1 \cup A_2$ mit $L(A_1 \cup A_2) = L(A_1) \cup L(A_2)$.

Von regulären Sprachen zu endlichen Automaten

Satz (Abschluss der NFA-erkennbaren Sprachen unter *)

Betrachte einen NFA A. Es gibt einen NFA A^* mit $L(A^*) = L(A)^*$.

Konstruktion

Sei $A = (Q, q_0, \to, Q_F)$. Definiere

$$A^* \quad := \quad (Q \uplus \{q_0'\}, q_0', \to \cup \to', Q_F \uplus \{q_0'\}),$$

wobei $\uplus$ die disjunkte Vereinigung sei und die neue Transitionsrelation $\to'$ wie folgt definiert: $q_0' \xrightarrow{a}{}' q$, falls $q_0 \xrightarrow{a} q$, und $q_f \xrightarrow{a}{}' q$ falls $q_0 \xrightarrow{a} q$ für alle $q_f \in Q_F$.

Es gibt also einen neuen Initialzustand, der zugleich final (ein Endzustand) ist. Von den Endzuständen loopen wir zurück (fast auf den früheren Initialzustand). Eine Illustration findet sich in den handschriftlichen Notizen.

Theorem

Falls $L \in \mathrm{REG}_\Sigma$, dann ist L NFA-erkennbar, es gibt also A mit $L = L(A)$.

Von endlichen Automaten zu regulären Sprachen

Ziel

Zeige die Umkehrung: die NFA-erkennbaren Sprachen sind regulär.
Falls $L = L(A)$ für einen NFA A, dann gilt $L \in \mathrm{REG}_\Sigma$.

Idee

- Repräsentiere den gegebenen NFA durch ein rekursives Gleichungssystem mit einer Gleichung pro Zustand.
- Löse dieses Gleichungssystem mit Ardens Lemma.

Lemma (Arden 1960)

Seien $U, V \subseteq \Sigma^*$ Sprachen mit $\varepsilon \notin U$. Betrachte $L \subseteq \Sigma^*$. Dann gilt

$$L = U.L \cup V \quad \textit{gdw.} \quad L = U^*.V \quad .$$

Beweis.

Der Beweis findet sich in den handschriftlichen Aufzeichnungen und im Skript. □

Von endlichen Automaten zu regulären Sprachen

Beobachtung

- Die Richtung $\Rightarrow$ in der Äquivalenz von Ardens Lemma besagt, dass eine solche Gleichung eine eindeutige Lösung besitzt.
- Wir benutzen das Lemma als Werkzeug, um zu einem gegebenen NFA eine reguläre Sprache zu bestimmen.

Theorem

Falls L von einem NFA erkannt wird, dann ist L regular.

Von endlichen Automaten zu regulären Sprachen

Beweisskizze.

Sei $L = L(A)$ mit $A = (Q, q_0, \to, Q_F)$ und $Q = \{q_0, \ldots, q_{n-1}\}$.

Für jeden Zustand $q_i \in Q$ definieren wir die Sprache

$$X_i \quad = \quad L(Q, q_i, \to, Q_F).$$

Das ist die Sprache aller Worte, die von q_i aus im NFA A akzeptiert werden.

Diese Sprachen sind Lösungen des Gleichungssystems

$$X_i \quad = \quad \bigcup_{q_i \xrightarrow{a} q_j} a.X_j \cup \begin{cases} \varepsilon, & \text{falls } q_i \in Q_F \\ \emptyset, & \text{sonst.} \end{cases}$$

Durch wechselseitiges Einsetzen der Gleichungen und Anwendung von Ardens Lemma konstruieren wir Ausdrücke, die die Regularität der Sprachen X_i zeigen.

Da $L = X_0$, ist auch L regulär. □

Von endlichen Automaten zu regulären Sprachen

Beispiel

Ein Beispiel findet sich in den handschriftlichen Aufzeichnungen.

Deterministische endliche Automaten

Definition

Ein NFA $A = (Q, q_0, \to, Q_F)$ ist deterministisch oder auch ein DFA, falls

für alle Buchstaben $a \in \Sigma$ und für alle Zustände $q \in Q$ gilt:

es gibt genau einen Zustand $q' \in Q$ mit $q \xrightarrow{a} q'$.

Deterministische Automaten sind für Anwendungen überaus nützlich.
Damit man sie aber nutzen kann, muss es sie erstmal geben.

Ziel

Zeige, dass für jeden NFA A ein DFA A' existiert mit $L(A) = L(A')$.

Potenzmengenkonstruktion

Theorem (Rabin & Scott 1959)

Für jeden NFA A mit $n \in \mathbb{N}$ Zuständen gibt es einen DFA A' mit höchstens 2^n Zuständen, der $L(A) = L(A')$ erfüllt.

Idee: Der Automat A' speichert alle Zustände,
in denen sich A nach dem Lesen eines Wortes befinden kann.
Die Zustände von A' sind daher Mengen von Zuständen von A.

Konstruktion: Potenzmengenkonstruktion

Für $A = (Q, q_0, \to, Q_F)$ definieren wir $A' := (\mathbb{P}(Q), \{q_0\}, \to', Q'_F)$.
Wir nutzen Großbuchstaben $Q_1, Q_2 \in \mathbb{P}(Q)$ für die Zustände von A'.

Die Transitionsrelation $\to'$ ist definiert durch

$$Q_1 \xrightarrow{a}{}' Q_2 \quad \text{mit} \quad Q_2 := \{q_2 \in Q \mid \exists q_1 \in Q_1 . q_1 \xrightarrow{a} q_2\}.$$

Die Endzustandsmenge ist

$$Q'_F := \{Q_1 \in \mathbb{P}(Q) \mid Q_1 \cap Q_F \neq \emptyset\}.$$

Potenzmengenkonstruktion

Bemerkung

Der Automat A' ist deterministisch:

(i) Für jeden Buchstaben $a \in \Sigma$ und für jeden Potenzmengenzustand $Q_1 \in \mathbb{P}(Q)$ gibt es einen Zielzustand.

(ii) Dieser Zielzustand ist eindeutig definiert.

Beachte, dass als Spezialfall $Q_2 = \emptyset$ gilt, falls keine Transitionen existieren.

Potenzmengenkonstruktion: Ausblick

Die Akzeptanze eines Wortes hängt von der Zukunft ab, von den Transitionen, die im Automaten A später möglich sind.
Während des Lesens eines Wortes kennt der Automat A' diese Zukunft noch nicht.
Er kann daher keine Entscheidung für/gegen Nachfolgezustände treffen.
Stattdessen trackt er alle Möglichkeiten.

In der moderneren Theoretischen Informatik kennt man Alternativen.
Prophecy-Variablen (seit Abadi & Lamport 1990) sagen das richtige zukünftige Verhalten in A voraus, um ein Wort zu akzeptieren.
Seit 2011 versucht man, Prophecy-Variablen automatisiert zu finden.

Trotz Prophecy-Variablen wird die Potenzmengenkonstruktion oft implementiert.
Eine unmittelbare Optimierung ist, nur die vom initialen Zustand aus erreichbaren Potenzmengenzustände (Zustandsmengen) zu berechnen.
Für Sprachäquivalenz und Sprachinklusion gibt es weitere Tricks.

Abschluss unter Komplementierung

Eine Konsequenz der Determinisierung von Rabin & Scott ist

der Abschluss der regulären Sprachen unter Komplementierung.

Bemerkung

Solange man die Lösung nicht kennt, ist das wirklich schwierig.

Genauer, es ist wirklich schwierig, zu einem gegebenen einen NFA A einen NFA für die Sprache $\overline{L(A)}$ zu finden. Warum ist das so?

$$L(A) = \{w \in \Sigma^* \mid \exists \text{ akzeptierender Ablauf von } A \text{ auf } w\}.$$
$$\overline{L(A)} = \{w \in \Sigma^* \mid \forall \text{ Abläufe von } A \text{ auf } w \text{ sind nicht akzeptierend}\}.$$

Um also einen NFA für $\overline{L(A)}$ anzugeben, müssen wir den Allquantor in einen Existenzquantor übersetzen — das funktioniert typischerweise nicht.

Für DFAs A' klappt es, weil ein DFA genau einen Lauf auf einem Wort hat:

$$L(A') = \{w \in \Sigma^* \mid \exists \text{ akzeptierender Ablauf von } A' \text{ auf } w\}$$
$$\overline{L(A')} = \{w \in \Sigma^* \mid \exists \text{ nicht-akzeptierender Ablauf von } A' \text{ auf } w\}.$$

Abschluss unter Komplementierung

Satz (Abschluss unter $\overline{\ }$)

Betrachte einen DFA A. Es gibt einen DFA $\overline{A}$ mit $L(\overline{A}) = \overline{L(A)}$.

Konstruktion: Vertausche die Endzustände mit den Nicht-Endzuständen

Sei $A = (Q, q_0, \to, Q_F)$. Definiere $\overline{A} := (Q, q_0, \to, Q \setminus Q_F)$.

Summary

Zusammenfassung

Sei $L = L(A)$ für einen NFA A mit $n \in \mathbb{N}$ Zuständen.

Es gibt DFAs für L und $\overline{L}$ mit höchstens 2^n Zuständen.

Die Schranke ist optimal: es gibt eine Familie $(L_n)_{n \in \mathbb{N}}$ von Sprachen L_n, die

- von einem NFA mit $n+1$ Zuständen erkannt werden
- aber nicht von einem DFA mit $< 2^n$ Zuständen erkannt werden können.

2. ε-Transitionen und Homomorphismen

NFAs mit internen Transitionen

Ziel: Kompaktere Darstellung von NFAs.

Ansatz: Interne Transitionen, die den Zustand ändern,
ohne einen Buchstaben des gegebenen Wortes zu lesen.

Intuition: Diese Transitionen sind mit ε beschriftet.
Aus technischen Gründen wählen wir ein neues Symbol, oft τ.

Definition

Ein NFA mit internen Transitionen über Σ ist ein Paar $A = (\tau, A_\tau)$ mit $\tau \notin \Sigma$ und A_τ einem NFA über dem Alphabet $\Sigma \uplus \{\tau\}$.

NFAs mit internen Transitionen

Der Akzeptanzbegriff muss angepasst werden.

Ein Wort soll akzeptiert werden, wenn man τs hinzufügen kann, so dass man bei einem Wort in $L(A_\tau)$ landet.

Alternativ muss ein Wort in $L(A_\tau)$ existieren, aus dem man die τs löschen kann und dann bei dem gegebenen Wort landet.

Diese Formulierung unterliegt der folgenden Definition.

Definition

Ein NFA mit internen Transitionen $A = (\tau, A_\tau)$ akzeptiert das Wort $w \in \Sigma^*$, falls

$$\exists u \in (\Sigma \uplus \{\tau\})^*.\ u \in L(A_\tau) \text{ und } w = \pi_\Sigma(u).$$

Die Sprache L(A) ist die Menge der in diesem Sinn akzeptierten Worte über Σ.

Dabei ist $\pi_\Sigma(u)$ die Projektion von u auf Σ.
Diese Funktion behält in u nur die Buchstaben aus Σ.
Da $u \in (\Sigma \uplus \{\tau\})^*$, streicht sie also alle Vorkommen des Buchstabens τ.

NFAs mit internen Transitionen

Ziel: Zeige, dass NFAs mit internen Transitionen genau die regulären Sprachen akzeptieren.

$\Leftarrow$: Jeder NFA ist auch ein NFA mit internen Transitionen (, die es halt nicht gibt).

$\Rightarrow$: Das ist der folgende Satz.

Satz

Sei $A = (\tau, A_\tau)$ ein NFA mit internen Transitionen. Dann ist $L(A)$ regulär.

Ansatz: Die regulären Sprachen sind genau die NFA-akzeptierbaren Sprachen.
Es genügt also, einen NFA B anzugeben mit $L(B) = L(A)$.

Idee: Der NFA B überspringt die τ-Transitionen in A_τ.

NFAs mit internen Transitionen

Um die Idee des Überspringens zu präzisieren, brauchen wir einen neuen Begriff.
Der τ-Abschluss eines Zustands in einem NFA mit internen Transitionen ist die Menge an Zuständen, die mittels τ-Transitionen erreicht werden können.

Definition

Sei (τ, A_τ) ein NFA mit internen Transitionen, wobei $A_\tau = (Q, q_0, \to, Q_F)$.
Der τ-Abschluss von $q \in Q$ ist die Zustandsmenge

$$\mathit{closure}_\tau(q) \quad := \quad \{q' \in Q \mid \exists w \in \tau^*.\ q \xrightarrow{w} q'\}.$$

Beachte, dass $q \in \mathit{closure}_\tau(q)$ gilt.

Bemerkung

Der τ-Abschluss von q lässt sich als Fixpunkt berechnen.
Das funktioniert analog zur Berechnung der erreichbaren Zustände in einem gerichteten Graphen (siehe Extravideo).
Nur betrachten wir hier allein die τ-Kanten.

NFAs mit internen Transitionen

Konstruktion (zum Beweis des obigen Satzes)

Sei $A = (\tau, A_\tau)$ mit $A_\tau = (Q, q_0, \to, Q_F)$.
Wir definieren den NFA $B := (Q, q_0, \to', Q'_F)$.
Die neue Transitionsrelation $\to'$ ist definiert durch

$$q_1 \xrightarrow{a}{}' q_2, \quad \text{falls } \exists q \in Q.\ q \in \mathit{closure}_\tau(q_1) \text{ und } q \xrightarrow{a} q_2.$$

Die neue Endzustandsmenge ist $Q'_F := \{q \in Q \mid \exists w \in \tau^*.\ \exists q' \in Q_F.\ q \xrightarrow{w} q'\}$.

Die Endzustandsmenge muss tatsächlich geändert werden.
Der kritische Fall ist eine τ-Transitionssequenz vom Start- zu einem Endzustand.

Bemerkung

Der τ-Abschluss erinnert an die Potenzmengenkonstruktion von Rabin & Scott.
Mit dieser Beobachtung lässt sich ein NFA mit internen Transitionen auch direkt in einen DFA übersetzen.

Intuition zu endlichen Automaten und regulären Sprachen

Interne Transitionen erlauben uns ein intuitiveres Verständnis von Automaten.

Wir können Automaten als `while`-Programme sehen,
die über `print`-Befehle Buchstaben ausgeben.

Genauer sind Automaten die Zustandsräume der `while`-Programme.
Der Befehl `print` liefert beschriftete Transitionen.
Befehle verschieden von `print` liefern interne Transitionen.

Endlichkeit der Automaten bedeutet nun auf Programmebene,
dass alle Variablen einen endlichen Datenbereich haben.
Man spricht auch von Programmierung mit endlichem Speicher.

Wir haben also gezeigt, dass die regulären Sprachen genau die Sprachen sind,
die sich mit solch eingeschränkten Programmen erzeugen lassen.

Man muss also bei der Frage der Regularität nicht auf Automatenebene denken,
sondern kann `while`-Programme als eine Art höhere Programmiersprache nutzen.

Lösungen zu Übungs- und Klausuraufgaben sind dennoch
in der geforderten Form anzufertigen.

Homomorphismen

Idee: Homomorphismen ersetzen Buchstaben durch Worte.
Damit kann man Buchstaben als Abkürzungen oder Makros verstehen.

Ziel: Zeige, dass die regulären Sprachen unter der Bild- und Urbild-Bildung von Homomorphismen abgeschlossen sind.

Ist also eine Sprache mit Abkürzungen regulär, dann ist auch die Sprache regulär, in der die Abkürzungen durch die entsprechenden Worte ersetzt wurden.

Außerdem lässt sich zu einer gegebenen regulären Sprache die größte (und wieder reguläre) Sprache mit Abkürzungen berechnen, aus der die gegebene Sprache durch Ersetzung hervorgeht.
Das ist ein erster Schritt in Richtung Decompilation.

Homomorphismen

Definition

Seien Σ und Γ Alphabete.
Ein Homomorphismus ist eine Funktion $h : \Sigma^* \to \Gamma^*$, die in folgendem Sinn verträglich mit der Konkatenation von Worten ist:

$$\forall u, v \in \Sigma^*. \quad h(u.v) = h(u).h(v).$$

Mit dieser Gleichheit muss ε auf ε abgebildet werden.
Das ist das folgende Lemma.

Homomorphismen

Lemma

Sei $h : \Sigma^* \to \Gamma^*$ ein Homomorphismus. Dann gilt $h(\varepsilon) = \varepsilon$.

Beweis.

Es gilt $\varepsilon = \varepsilon.\varepsilon$.
Da h eine Funktion ist, folgt $h(\varepsilon) = h(\varepsilon.\varepsilon)$.
Wir messen nun die Länge der Worte:

$$|h(\varepsilon)| \overset{(1)}{=} |h(\varepsilon.\varepsilon)| \overset{(2)}{=} |h(\varepsilon).h(\varepsilon)| \overset{(3)}{=} |h(\varepsilon)| + |h(\varepsilon)|.$$

Gleichung (1) gilt, da die Länge eine Funktion ist.
Gleichung (2) ist die Anforderung an Homomorphismen.
Gleichung (3) ist eine Eigenschaft der Wortlänge.

Aus $|h(\varepsilon)| = |h(\varepsilon)| + |h(\varepsilon)|$ folgt $|h(\varepsilon)| = 0$.
Das einzige Wort mit Länge 0 ist ε. □

Darstellung von Homomorphismen

Überlegung: Homomorphismen sollten leicht anzugeben sein.
Mit der für Homomorphismen geforderten Gleichheit müssen wir nur wissen, wie sie sich auf einzelnen Buchstaben verhalten.

Das stimmt, und wird so formalisiert:

Satz

Jede Funktion auf Buchstaben $f : \Sigma \to \Gamma^*$ lässt sich eindeutig zu einem Homomorphismus $h_f : \Sigma^* \to \Gamma^*$ fortsetzen.

Fortsetzen bedeutet dabei, h_f stimmt mit f überein, wann immer f definiert ist.
Genauer:

$$\forall a \in \Sigma. \quad h_f(a) \;=\; f(a).$$

Wir beweisen zunächst den Satz, bevor wir erklären,
wie er die Überlegung rechtfertigt.

Darstellung von Homomorphismen

Beweis.

Wir unterscheiden zwei Fälle.

Fall 1: Für das Wort ε wissen wir bereits, dass $h_f(\varepsilon) = \varepsilon$ gelten muss.
Hier besteht keine Wahlmöglichkeit für die Definition von h_f.

Fall 2: Für ein Wort $w = a_1 \dots a_n$ der Länge mindestens zwei muss gelten:

$$h_f(w) \overset{(1)}{=} h_f(a_1) \dots h_f(a_n) \overset{(2)}{=} f(a_1) \dots f(a_n).$$

Gleichung (1) ist die Anforderung an Homomorphismen.
Gleichung (2) gilt, da h_f eine Fortsetzung von f ist.
Auch hier besteht keine Wahlmöglichkeit für die Definition von h_f.

Zusammen mit der angenommenen Gleichheit von h_f und f auf Buchstaben, ist also h_f eindeutig bestimmt. □

Darstellung von Homomorphismen

Um einen Homomorphismus $h : \Sigma^* \to \Gamma^*$ anzugeben,
genügt es, die Einschränkung von h auf Σ anzugeben.
Das ist die Funktion $h_{\restriction \Sigma} : \Sigma \to \Gamma^*$ mit $h_{\restriction \Sigma}(a) := h(a)$ für alle $a \in \Sigma$.

Beobachtung: Homomorphismus h ist eine Fortsetzung von $h_{\restriction \Sigma}$.

Um zu sehen, dass wir mit $h_{\restriction \Sigma}$ tatsächlich genau h angeben,
sei $g : \Sigma^* \to \Gamma^*$ ein weiterer Homomorphismus,
der auf den Buchstaben mit h übereinstimmt, also $g_{\restriction \Sigma} = h_{\restriction \Sigma}$.

Es gilt

$$g \overset{(1)}{=} h_{g_{\restriction \Sigma}} \overset{(2)}{=} h_{h_{\restriction \Sigma}} \overset{(3)}{=} h \ .$$

Die Gleichheiten (1) und (3) nutzen die obige Beobachtung und
die Eindeutigkeit der Fortsetzung aus dem obigen Satz.
Gleichheit (2) folgt schon mit der Eindeutigkeit der Fortsetzung.

Mit dieser Überlegung sind also g und h gleich.

Darstellung von Homomorphismen

Was hat man davon? Die Funktion $h_{\restriction \Sigma}$ ist (als Menge von Paaren) endlich und kann entsprechend einfach zur automatisierten Verarbeitung angegeben werden.

In der Theorie sagt man salopp, die Funktion lässt sich hinschreiben.

Die Funktion h bildet alle Worte ab, ist also als Menge unendlich und kann nicht hingeschrieben werden.

Bilder und Urbilder von Homomorphismen

Ziel: Zeige die Regularität der folgenden Sprachen
unter der Annahme, dass L_1 und L_2 regulär sind.

Definition

Seien $h : \Sigma^* \to \Gamma^*$ ein Homomorphismus und $L_1 \subseteq \Sigma^*$ sowie $L_2 \subseteq \Gamma^*$ Sprachen.

Das Bild von L_1 unter h ist die Sprache (über Γ)

$$h(L_1) \quad := \quad \{h(w) \mid w \in L_1\}.$$

Das Urbild von L_2 unter h ist die Sprache (über Σ)

$$h^{-1}(L_2) \quad := \quad \{w \in \Sigma^* \mid h(w) \in L_2\}.$$

Bilder und Urbilder von Homomorphismen

Satz (Abschluss unter Bildern von Homomorphismen)

Seien $h : \Sigma^* \to \Gamma^*$ ein Homomorphismus und $L \subseteq \Sigma^*$ regulär.
Dann ist $h(L)$ regulär.

Sei $L = L(A)$ mit A einem NFA über Σ.
Für $h(L)$ konstruieren wir einen NFA mit internen Transitionen B über Γ.

Konstruktion von B: Ersetze Buchstaben durch Transitionsfolgen

Ersetze jede Transition $q \xrightarrow{a} q'$ in A durch eine Transition $q \xrightarrow{h(a)} q'$ in B.
Falls $h(a) = \varepsilon$, ist diese Transition intern, also mit τ beschriftet.
Falls $h(a)$ mindestens zwei Buchstaben hat, füge in B neue Zwischenzustände ein.

Der Beweis $h(L) = h(L(A)) = L(B)$ ist einfach und im Skript zu finden.
Außerdem wird auf Seite 57 die Konstruktion von B an einem Beispiel gezeigt.
Das Beispiel erläutert auch noch einmal das Eliminieren interner Transitionen.

Bilder und Urbilder von Homomorphismen

Satz (Abschluss unter Urbildern von Homomorphismen)

Seien $h : \Sigma^* \to \Gamma^*$ ein Homomorphismus und $L \subseteq \Gamma^*$ regulär.
Dann ist $h^{-1}(L)$ regulär.

Sei $L = L(A)$ mit $A = (Q, q_0, \to, Q_F)$ einem NFA über Γ.
Für $h^{-1}(L)$ konstruieren wir einen NFA $h^{-1}(A) = (Q, q_0, \to', Q_F)$ über Σ.

Konstruktion von $h^{-1}(A)$: Ersetze Transitionsfolgen durch Buchstaben

Zur Definition der Transitionsrelation $\to'$ betrachten wir jeden Buchstaben $a \in \Sigma$.
Für jede mit $h(a) = b_1 \dots b_n$ beschriftete Transitionsfolge in A

$$q \xrightarrow{b_1} q_1 \xrightarrow{b_2} \dots \xrightarrow{b_n} q'$$

definieren wir eine Transition $q \xrightarrow{a}{}' q'$ in $h^{-1}(A)$.
Für den Spezialfall $h(a) = \varepsilon$ erhält jeder Zustand eine mit a beschriftete Schleife.

Der Beweis $h^{-1}(L) = h^{-1}(L(A)) = L(h^{-1}(A))$ ist wieder einfach.
Auf Seite 58 wird die Konstruktion von $h^{-1}(A)$ an einem Beispiel gezeigt.

Bilder und Urbilder von Homomorphismen

Anwendung: Urbilder von Homomorphismen helfen beim Nachweis der Regularität.

Intuition: Homomorphismen können Details einer Sprache wegabstrahieren.
Übrig bleibt eine Sprache, deren Speicheranforderungen deutlicher sind
(siehe Programmierbarkeit mit endlichem Speicher auf Folie 37).

Beispiel

Seien $\Sigma = \{a, b, c, d, e\}$, $\Gamma = \{0, 1\}$ und

$$\begin{aligned} L &= \{w \in \Sigma^* \mid |w|_a + |w|_b + |w|_c \equiv |w|_d + |w|_e \mod 3\} \\ L' &= \{w \in \Gamma^* \mid |w|_0 \equiv |w|_1 \mod 3\}. \end{aligned}$$

Die Buchstaben a, b, c sowie d, e verhalten sich gleich bzgl. der Kongruenz.
Wir können also abstrahieren mit einem Homomorphismus $h : \Sigma^* \to \Gamma^*$.
Zur Definition von h benötigen wir nur die Abbildung der Buchstaben:

$$h(a) := 0 =: h(b) =: h(c) \qquad h(d) := 1 =: h(e).$$

Es gilt $L = h^{-1}(L')$.

Bilder und Urbilder von Homomorphismen

Beispiel (cont.)

Entscheidend für die Regularität von L' ist, ob $|w|_0 \equiv |w|_1 \mod 3$
mit endlichem Speicher geprüft werden kann.

Das klappt, nutze zwei Variablen mit Datenbereich $\{0, 1, 2\}$.
Die erste Variable zählt die Vorkommen von 0 modulo 3.
Die zweite Variable zählt die Vorkommen von 1 modulo 3.
Ein Wort wird akzeptiert, wenn die Variablen denselben Wert halten.

3. Entscheidbarkeit und Komplexität

Entscheidbarkeit und Komplexität

Ziel: Untersuche, welche Probleme zu regulären Sprachen entscheidbar sind.

Probleme haben immer die Gestalt

> *PROBLEMNAME*
> *Gegeben: Die Eingabe, die gegeben ist.*
> *Frage: Was man über die Eingabe wissen möchte.*

Ein Problem ist entscheidbar, wenn es automatisch gelöst werden kann.
Genauer gibt es einen Algorithmus, der die Eingabe entgegennimmt,
endlich lange rechnet und anschließend die richtige Lösung ausgibt.

Entscheidbarkeit und Komplexität

Ziel: Sofern ein Problem entscheidbar ist, möchten wir verstehen,
mit welchen Ressourcen es gelöst werden kann (obere Schranke).

Eigentlich möchten wir auch wissen, welche Ressourcen zur Lösung auf jeden Fall benötigt werden (untere Schranke), aber das ist eine Frage für Theo II.

Ressourcen sind dabei Rechenzeit und Speicherplatz.

Der Ressourcenbedarf (obere und untere Schranken) von Problemen
wird in Relation zur Größe der Eingabe gemessen.
Es handelt sich also um eine Funktion.

Der Ressourcenbedarf wird im Forschungsgebiet Komplexitätstheorie untersucht.

Wir betrachten die Worst-Case-Komplexität, bei der die obere Schranke
für alle Eingaben gelten muss.

Entscheidbarkeit und Komplexität

Motivation: Wir betrachten Probleme, deren Eingabe reguläre Sprachen sind.

Sprachen lassen sich in der Regel nicht als endliche (Wort)Mengen angeben.
Selbst wenn, ist diese Darstellung nicht kompakt und damit nicht hilfreich,
da der Ressourcenbedarf in Relation zur Größe der Eingabe gemessen wird.

Daher nehmen wir an, die regulären Sprachen seien über NFAs gegeben.

NFAs kann man sich als (Zustandsräume von) `while`-Programme(n) vorstellen.

Entsprechend stellen die hier betrachteten Probleme zu regulären Sprachen
Fragen über das Verhalten von `while`-Programmen.

Die Lösungsalgorithmen beantworten diese Fragen.
Sie können also zur Programmanalyse eingesetzt werden,
ähnlich der Datenflussanalyse aus den ersten Kapiteln.

Derartige Compiler-Techniken zur Programmverifikation und Programmsynthese
bilden den Forschungsschwerpunkt unseres Instituts.

Wortproblem

Ziel: Löse das Wortproblem für reguläre Sprachen.
Das ist die Frage, ob ein gegebener NFA ein gegebenes Wort akzeptiert.

> *WORDREG*
> *Gegeben: Ein NFA A über Σ und ein Wort $w \in \Sigma^*$.*
> *Frage: Gilt $w \in L(A)$?*

Wir zeigen, dass WORDREG entscheidbar ist und
sogar in polynomieller Zeit (in der Größe von A und w) gelöst werden kann.

Satz

WORDREG ist entscheidbar und kann in Zeit $\mathcal{O}(|w| \cdot |Q|^2)$ gelöst werden.

Wortproblem

Lösungsversuch 1: Determinisiere A und prüfe dann,
ob das Wort w vom Start- zu einem Endzustand führt.

Ergebnis 1: Das genügt, um die Entscheidbarkeit des Problems zu zeigen.

Der Algorithmus braucht allerdings exponentielle Zeit (in der Eingabegröße),
da die Determinisierung so lange dauert.
Er liefert also nicht die gewünschte obere Schranke für die Laufzeit.

Lösungsversuch 2: Prüfe mittels Backtracking, ob w von A akzeptiert wird.

Ergebnis 2: Das Verfahren liefert wieder die Entscheidbarkeit.

Allerdings gibt es Eingaben, bei denen das Backtracking exponentiell lange läuft.
Die gesuchte polynomielle obere Schranke gilt also im Worst-Case nicht.

Wortproblem

Lösung: Determinisiere $A = (Q, q_0, \to, Q_F)$ entlang des Wortes $w = a_1 \dots a_n$.
Bestimme also zu jedem Präfix $a_1 \dots a_i$ von w die Menge Q_i an Zuständen,
in denen sich A nach dem Lesen des Präfixes befinden kann.

Algorithmus

Setze $Q_0 := \{q_0\}$.
Für $i = 1$ bis n:
  Berechne $Q_i := \{q \in Q \mid \exists q' \in Q_{i-1}.q' \xrightarrow{a_i} q\}$. //Potenzmengenkonstruktion
Falls Q_n einen Endzustand enthält, gib *ja* zurück, sonst *nein*.

Die Schleife wird $|w|$-Mal durchlaufen.
Bei jedem Durchlauf prüfen wir $|Q|^2$ viele Paare von Zuständen.

Beispiel

Seite 62 im Skript.

Leerheitsproblem

Ziel: Löse das Leerheitsproblem für reguläre Sprachen.
Das ist die Frage, ob die Sprache eines gegebenen NFAs leer ist.

EMPTYREG
Gegeben: Ein NFA A über Σ.
Frage: Gilt $L(A) = \emptyset$?

Wir zeigen, dass EMPTYREG entscheidbar ist und
sogar in polynomieller Zeit (in der Größe von A) gelöst werden kann.

Satz

EMPTYREG ist entscheidbar und kann in Zeit $\mathcal{O}(|\to|)$ gelöst werden.

Leerheitsproblem

Ansatz: Wir nehmen an, A habe nur einen Endzustand.
Falls nicht, transformieren wir A in $|\to|$-Zeit. (Wie?)

Nun gilt

$$L(A) \neq \emptyset \textit{ genau dann, wenn } \textit{vom Start- der Endzustand erreichbar ist.}$$

Letzteres ist aber eine Instanz des Problems PATH, das wir
im Zusatzvideo über Fixpunkte betrachtet haben.

Für PATH liefert die Kleene-Iteration zur Fixpunktberechnung
bereits eine polynomielle Laufzeit.

Eine genauere Analyse zeigt jedoch einen Zeitaufwand von $\mathcal{O}(|Q| \cdot |\to|)$.

Mit einer Datenstruktur, die zu einem Zustand die ausgehenden Kanten liefert,
kommt man auf die angegebene obere Schranke.

Reduktionen

Wenn man zur Lösung eines Problems
ein anderes Problem (hier PATH) zu Hilfe nehmen möchte,
benötigt man immer eine Äquivalenz zwischen den Lösungen,
also einen genau dann, wenn-Zusammenhang.

In Theo II werden wir den Begriff der Reduktion kennenlernen,
der die Idee des Zuhilfenehmens formalisiert und präzisiert.
Hier genügt uns die Äquivalenz.

Universalität

Ziel: Übe das Konzept des Zuhilfenehmens/der Reduktion anhand dreier Probleme.

UNIVERSALITYREG
Gegeben: Ein NFA A über Σ.
Frage: Gilt $L(A) = \Sigma^*$?

Ansatz: $L(A) = \Sigma^*$ gdw. $\overline{L(A)} = \emptyset$ gdw. $L(\overline{A}) = \emptyset$.
Es genügt also, den Komplementautomaten zu berechnen und dann das Leerheitsproblem (mit Eingabe $\overline{A}$) zu Hilfe zu nehen.

Satz

UNIVERSALITYREG ist entscheidbar.

Bemerkung: Das Verfahren liefert keine polynomielle Laufzeit.
Die Berechnung des Komplementautomaten erfordert eine Determinisierung, die bei einigen Eingaben viel Zeit in Anspruch nimmt.

Tatsächlich gibt es keinen Algorithmus mit polynomieller Laufzeit, nicht mal einen nicht-deterministischen, sofern nicht PSPACE=NPTIME gilt.
Das ist ähnlich unwahrscheinlich wie PTIME = NPTIME.

Inklusion

Das zweite Problem ist die Inklusion zwischen regulären Sprachen.

> *INCLUSIONREG*
> *Gegeben: Seien A und B NFAs über Σ.*
> *Frage: Gilt $L(A) \subseteq L(B)$?*

Ansatz: $L(A) \subseteq L(B)$ gdw. $L(A) \cap \overline{L(B)} = \emptyset$ gdw. $L(A \cap \overline{B}) = \emptyset$.
Es genügt also, den Komplementautomaten von B zu berechnen,
diesen mit dem NFA A zu schneiden (wie?)
und dann das obige Leerheitsproblem (mit Eingabe $A \cap \overline{B}$) zu Hilfe zu nehmen.

Satz

INCLUSIONREG ist entscheidbar.

Bemerkung: Das Verfahren liefert wieder keine polynomielle Laufzeit
und wieder sollte es kein derart schnelles Verfahren geben.

Äquivalenz

Das dritte Problem ist die Äquivalenz zwischen regulären Sprachen.

> *EQUIVALENCEREG*
> *Gegeben: Seien A und B NFAs über Σ.*
> *Frage: Gilt $L(A) = L(B)$?*

Ansatz: $L(A) = L(B)$ gdw. $L(A) \subseteq L(B)$ und $L(B) \subseteq L(A)$.
Damit können wir einfach das soeben gelöste Inklusionsproblem zu Hilfe nehmen.
Wir machen zwei Aufrufe, mit Eingabe A, B und mit Eingabe B, A.

Satz

EQUIVALENCEREG ist entscheidbar.

Bemerkung: Da EQUIVALENCEREG allgemeiner als UNIVERSALITYREG ist, wird es wieder kein schnelles Verfahren geben.

Bemerkung: Auf Seite 64 gibt das Skript einen Einblick in die Komplexitätstheorie und erklärt, welchen Ressourcenbedarf die soeben betrachteten Probleme haben.

4. Minimierung

Minimierung

Ziel: Berechne zu einem gegebenen DFA einen sprachäquivalenten DFA
mit möglichst wenigen Zuständen.

Ansatz: Verstehe zunächst, wieviele Zustände definitiv benötigt werden.

Motivation: Die obigen Algorithmen laufen kürzer, je kleiner die Automaten sind.
Außerdem sind absolute untere Schranken für Zeit-/Platzbedarf sehr selten.
Bei DFAs ist der Platzbedarf eben die Zustandszahl.
In Theo II bekommen wir mit Reduktionen nur relative untere Schranken hin.

Kritik: Warum denn nicht für NFAs, die sind für Anwendungen noch wichtiger.
Minimale NFAs sind nicht eindeutig und die Theorie schwieriger.
Es gibt nicht optimale Minimierungen für NFAs mittels Bisimulationsäquivalenz.
Diese Techniken haben wir hier nicht vorgestellt.

Äquivalenzen und Kongruenzen

Definition

Eine Relation $\equiv \subseteq \Sigma^* \times \Sigma^*$ auf Worten ist eine Äquivalenz, falls gilt:

$$\forall u \in \Sigma^*.\quad u \equiv u \qquad \text{(reflexiv)}$$

$$\forall u, v \in \Sigma^*.\quad u \equiv v \text{ impliziert } v \equiv u \qquad \text{(symmetrisch)}$$

$$\forall u, v, w \in \Sigma^*.\quad u \equiv v \text{ und } v \equiv w \text{ impliziert } u \equiv w. \qquad \text{(transitiv)}$$

Eine Relation $\equiv \subseteq \Sigma^* \times \Sigma^*$ auf Worten ist eine Kongruenz, wenn sie eine Äquivalenz ist, die zusätzlich verträglich mit der Konkatenation von rechts ist, also

$$\forall u, v, w \in \Sigma^*.\quad u \equiv v \text{ impliziert } u.w \equiv v.w. \qquad \text{(Verträglichkeit)}$$

Bemerkung: Diese Definitionen werden überall in Mathematik/Informatik genutzt.

Eine Äquivalenz ist immer eine reflexive, symmetrische und transitive Relation. Die Objekte, auf denen die Relation definiert ist, können beliebig gewählt werden.

Sofern auf diesen Objekten Operationen definiert sind, ist eine Kongruenz immer eine Äquivalenz, die unter diesen Operationen erhalten bleibt.

Für die Anwendung müssen Objekte, Operationen und Relation definiert werden.

Äquivalenzen und Kongruenzen

Definition

Seien $\equiv \subseteq \Sigma^* \times \Sigma^*$ eine Äquivalenz, $u \in \Sigma^*$ und $L \subseteq \Sigma^*$.
Die Äquivalenzklasse von u bzgl. $\equiv$ ist die Menge aller $\equiv$-äquivalenten Worte, also

$$[u]_\equiv \quad := \quad \{v \in \Sigma^* \mid u \equiv v\}.$$

Die Faktorisierung von L nach $\equiv$ ist die Menge, deren Elemente die Äquivalenzklassen von Worten in L bzgl. $\equiv$ sind, also

$$L/_\equiv \quad := \quad \{[u]_\equiv \mid u \in L\}.$$

Oft ist nur die gesamte Anzahl an Klassen von Interesse, also $\mathit{Index}(\equiv) := |\Sigma^*/_\equiv|$.
Das nennt man den Index der Äquivalenzrelation.

Äquivalenzen und Kongruenzen

Das folgende Lemma gibt Eigenschaften der Faktorisierung an.

Lemma

Sei $\equiv \subseteq \Sigma^* \times \Sigma^*$ *eine Äquivalenz.*

(i) *Für alle* $u \in \Sigma^*$ *gilt* $u \in [u]_\equiv$.

(ii) *Für alle* $u, v \in \Sigma^*$ *mit* $u \equiv v$ *gilt* $[u]_\equiv = [v]_\equiv$.

(iii) *Für alle* $u, v \in \Sigma^*$ *mit* $u \not\equiv v$ *gilt* $[u]_\equiv \cap [v]_\equiv = \emptyset$.

(iv) $\Sigma^* = \bigcup_{u \in \Sigma^*} [u]_\equiv$.

Bemerkung: Eine Äquivalenz unterteilt also die Grundmenge in disjunkte Klassen.
Diese Klassen repräsentieren die Elemente, die sie enthalten.
Wenn die Äquivalenz nun auch noch eine Kongruenz ist, lassen sich
die Operationen auf der Grundmenge auf die Klassen liften.
Es lässt sich dann mit den Klassen wie mit Elementen der Grundmenge rechnen.
Allerdings weiß man nur noch bis auf die Äquivalenz, welches Element man hat.
In diesem Sinn dienen Äquivalenzen der Abstraktion.

Bemerkung: Es gibt auch noch das Konzept der Repräsentantensysteme.
Es weist jeder Klasse ein Element aus der Klasse zu, das die Klasse repräsentiert.

Der Satz von Myhill und Nerode[1]

Ziel: Zeige eine Charakterisierung für die Regularität von Sprachen.
Eine Charakterisierung ist eine notwendige und hinreichende Bedingung.
Eine Sprache ist also regulär genau dann, wenn sie die Bedingung erfüllt.

Motivation: Wir werden der Charakterisierung Information
über die benötigte Zustandszahl entnehmen.

Bemerkung: Die Existenz dieser Charakterisierung ist kaum zu glauben.
Für die kontextfreien Sprachen hat man die Suche
nach einer solchen Charakterisierung aufgegeben.

[1] Bewiesen Ende der 1950er Jahre. Anil Nerode ist noch aktiv.

Der Satz von Myhill und Nerode

Definition

Sei $L \subseteq \Sigma^*$ eine Sprache. Die Nerode-Rechtskongruenz (bzgl. L) ist die wie folgt definierte Relation $\equiv_L \subseteq \Sigma^* \times \Sigma^*$:

$$u \equiv_L v, \quad \text{falls} \quad \forall w \in \Sigma^*.\ u.w \in L \quad \text{gdw.} \quad v.w \in L\ .$$

Lemma

Für jede Sprache $L \subseteq \Sigma^*$ ist $\equiv_L$ eine Kongruenz.
Für $u \in L$ gilt außerdem $[u]_{\equiv_L} \subseteq L$. (Warum?)

Der Satz von Myhill und Nerode

Beispiel

Betrachte $L = \{a^n.b^n \mid n \in \mathbb{N}\}$.

1. Es gilt $a \not\equiv_L aa$, denn $a.b \in L$ aber $aa.b \notin L$.
2. Es gilt $a \not\equiv_L aab$, denn $a.abb \in L$ aber $aab.abb \notin L$.
3. Es gilt $aab \equiv aaabb$, denn $aab.b \in L$ und $aaabb.b \in L$ und für alle $u \in \Sigma^* \setminus \{b\}$ gilt $aab.u \notin L$ und $aaabb.u \notin L$.

Allgemeiner sind die Äquivalenzklassen der Nerode-Rechtskongruenz:

$$[a^k]_{\equiv_L} = \{a^k\}$$
$$[a^{k+1}b]_{\equiv_L} = \{a^{l+1}b^{l+1-k} \mid l \geq k \in \mathbb{N}\}.$$

Bemerkung: Die Sprache ist nicht-regulär.
Das sehen wir später, es folgt zum Beispiel aus dem Satz von Myhill und Nerode.

Intuitiv ist der Grund, dass wir unendlichen Speicher, genauer unbeschränkte Integer-Variablen zu benötigen scheinen, um mit einem Programm genau die Worte der Sprache zu akzeptieren.

Auch die Nerode-Rechtskongruenz hat unendlichen Index.

Der Satz von Myhill und Nerode

Myhill-Nerode macht die Beziehung zwischen Speicherbedarf und Index explizit.

Der Satz kann als anwendbare (zur Widerlegung von Regularität) Version der Aussage gesehen werden, dass eine Sprache unendlichen Speicher benötigt.

Satz von Myhill und Nerode, 1957/58

Eine Sprache $L \subseteq \Sigma^*$ ist genau dann regulär, wenn $\equiv_L$ endlichen Index hat.

Der Satz von Myhill und Nerode

Beweis $\Rightarrow$:

Da L reguär ist, gibt es einen DFA $A = (Q, q_0, \rightarrow, Q_F)$ mit $L = L(A)$.
Der DFA induziert eine Relation $\equiv_A \subseteq \Sigma^* \times \Sigma^*$ wie folgt:

$$u \equiv_A v \quad \text{falls} \quad \exists q \in Q. \quad q_0 \xrightarrow{u} q \text{ und } q_0 \xrightarrow{v} q.$$

Es lässt sich zeigen, dass $\equiv_A$ eine Äquivalenz ist.
Beachte dazu, dass A deterministisch ist.
Außerdem gilt $\mathit{Index}(\equiv_A) \leq |Q|$, es gibt also nur endlich viele Äquivalenzklassen.

Nun zeigen wir, dass $\equiv_A \subseteq \equiv_L$, wann immer also zwei Worte $\equiv_A$-äquivalent sind, sind sie schon $\equiv_L$ äquivalent.
Daraus folgt dann $\mathit{Index}(\equiv_L) \leq \mathit{Index}(\equiv_A)$.
Zusammen mit $\mathit{Index}(\equiv_A) \leq |Q|$ sind wir fertig.

Für $\equiv_A \subseteq \equiv_L$, betrachte $u \equiv_A v$.
Um $u \equiv_L v$ zu zeigen, betrachte $w \in \Sigma^*$. Es gilt

$$\begin{aligned} u.w \in L = L(A) \quad &\Leftrightarrow \quad \exists q \in Q, q_f \in Q_F.\ q_0 \xrightarrow{u} q \xrightarrow{w} q_f \\ &\Leftrightarrow \quad \exists q \in Q, q_f \in Q_F.\ q_0 \xrightarrow{v} q \xrightarrow{w} q_f \qquad //A \text{ DFA und } u \equiv_A v \\ &\Leftrightarrow \quad v.w \in L(A). \end{aligned}$$

Der Satz von Myhill und Nerode

Beweis $\Leftarrow$:
Der Beweis ist die Konstruktion des sogenannten Äquivalenzklassenautomaten.
Die Zustände sind die Äquivalenzklassen von $\equiv_L$.
Intuitiv speichert ein Zustand, welche Äquivalenzklasse man gerade erreicht hat.

Konstruktion: Äquivalenzklassenautomat A_L

$A_L := (\Sigma^* /_{\equiv_L}, [\varepsilon]_{\equiv_L}, \to, L /_{\equiv_L})$ mit Transitionen

$$[u]_{\equiv_L} \xrightarrow{a} [u.a]_{\equiv_L} \quad \text{für alle } [u]_{\equiv_L} \in \Sigma^* /_{\equiv_L} \text{ und } a \in \Sigma.$$

Bemerkung: Die Zustandszahl ist endlich, da $\mathit{Index}(\equiv_L)$ endlich ist.
Der Initialzustand ist die Klasse von ε.
Die Endzustände sind die Klassen der Worte in L.
Der Automat ist deterministisch.

Lemma

$[\varepsilon]_{\equiv_L} \xrightarrow{w} [w]_{\equiv_L}$ *für alle* $w \in \Sigma^*$.

Der Satz von Myhill und Nerode

Die Korrektheit der Konstruktion, also $L = L(A_L)$, folgt mit obigem Lemma. Genauer:

$$
\begin{aligned}
w \in L(A_L) \quad &\Leftrightarrow \quad \exists [u]_{\equiv_L} \in L/_{\equiv_L}.\ [\varepsilon]_{\equiv_L} \xrightarrow{w} [u]_{\equiv_L} \quad \text{//Akzeptanz und Definition } A_L \\
&\Leftrightarrow \quad \exists u \in L.\ [w]_{\equiv_L} = [u]_{\equiv_L} \qquad\qquad \text{//} A_L \text{ DFA und obiges Lemma} \\
&\Leftrightarrow \quad w \in L.
\end{aligned}
$$

□

Minimale DFAs

Ziel: Zeige, dass der Äquivalenzklassenautomat von Myhill und Nerode der kleinste (in der Zustandszahl) DFA ist, der die Sprache akzeptiert.

Zeige außerdem, dass jeder andere DFA mit derselben Zustandszahl und Sprache bereits isomorph zum Äquivalenzklassenautomaten ist.

Dass aus schwachen Bedingungen wie Sprachgleichheit, Größe und Determinismus bereits Isomorphie folgt, ist schon ein Wunder.

Satz

Sei L regulär. Dann gibt es einen eindeutigen minimalen DFA A mit $L = L(A)$. Dieser DFA hat $\mathit{Index}(\equiv_L)$ Zustände.

Minimale DFAs

Der Satz macht drei Aussagen, die wir jetzt beweisen.

Existenz: Es gibt einen DFA A mit $L = L(A)$ und Zustandszahl $\mathit{Index}(\equiv_L)$.

Beweis: Wähle den Äquivalenzklassenautomaten, $A := A_L$.
Der Satz von Myhill und Nerode zeigt $L(A_L) = L$.
Außerdem ist die Zustandszahl von A_L genau $\mathit{Index}(\equiv_L)$.

Minimale DFAs

Minimalität: Es gibt keinen DFA B für L mit weniger Zuständen.
Wir schieben die Negation nach innen, um die Aussage zu beweisen:
Für alle DFAs $B = (Q, q_0, \to, Q_F)$ mit $L(B) = L$ gilt $|Q| \geq \mathit{Index}(\equiv_L)$.

Beweis: Sei B solch ein DFA.
Sei $\equiv_B$ die von B induzierte Äquivalenz aus dem Beweis von Myhill und Nerode.
In dem Beweis wird gezeigt, dass $|Q| = \mathit{Index}(\equiv_B)$ (Richtung $\Rightarrow$).
Ferner gilt $\equiv_L \subseteq \equiv_B$ und so $\mathit{Index}(\equiv_L) \leq \mathit{Index}(\equiv_B)$.
Auch das stammt aus Myhill und Nerode.
Damit folgt $|Q| \geq \mathit{Index}(\equiv_L)$, wie gefordert.

Minimale DFAs

Um die Eindeutigkeitsaussage formal zu machen, benötigen wir eine Definition.

Definition

Zwei NFAs $A = (Q, q_0, \to_A, Q_F)$ und $B = (P, p_0, \to_B, P_F)$ über Σ sind isomorph, geschrieben $A \approx B$, falls es eine bijektive Abbildung $\beta : Q \to P$ gibt mit:

- $\beta(q_0) = p_0$,
- $\beta(Q_F) = P_F$ und
- $\forall q, q' \in Q.\ \forall a \in \Sigma.\ q \xrightarrow{a}_A q' \ \Leftrightarrow\ \beta(q) \xrightarrow{a}_B \beta(q')$.

Man nennt β einen Isomorphismus zwischen A und B.

Erinnerung: Eine Abbildung $\beta : Q \to P$ ist bijektiv, wenn sie injektiv $(\forall q, q'.\beta(q) = \beta(q') \Rightarrow q = q')$ und surjektiv $(\forall p \in P.\exists q \in Q.p = \beta(q))$ ist.

Bemerkung: Isomorphie $\approx$ ist eine Äquivalenzrelation auf NFAs über Σ.
Zwei NFAs sind isomorph, wenn sie bis auf die Namen der Zustände gleich sind.

Minimale DFAs

Eindeutigkeit: Es gibt keine verschiedenen DFAs für L mit $\mathit{Index}(\equiv_L)$ Zuständen.
Für den Beweis formen wir die Aussage wieder äquivalent um.
Für alle DFAs $A = (Q, q_0, \to, Q_F)$ mit $L(A) = L, |Q| = \mathit{Index}(\equiv_L)$ gilt $A \approx A_L$.

Beweis: Sei A solch ein DFA.
Wir definieren eine Abbildung β der Zuständen in A_L auf Q.
Zur Erinnerung, die Zustände in A_L sind die Äquivalenzklassen $[u]_{\equiv_L}$.
Solch eine Klasse wird abgebildet auf den Zustand q mit $q_0 \xrightarrow{u} q$.
Dieser Zustand ist eindeutig, da A ein DFA ist.

Es ist eine Übungsaufgabe, zu zeigen, dass β ein Isomorphismus ist.

Damit ist der Beweis des obigen Satzes abgeschlossen. □

Kompaktheit von NFAs

Bemerkung: NFAs können exponentiell kompakter sein als DFAs.

Problem: Wie zeigt man, dass der Zusammenhang exponentiell ist?
Eine Sprache stellt nicht genügend Anforderungen an den Zusammenhang.
20 und 20.000 Zustände können über ein Polynom verbunden sein.
Wir benötigen eine Sprachfamilie!

Satz

Es gibt eine Familie $(L_i)_{i \in \mathbb{N} \setminus \{0\}}$ an Sprachen über $\{0, 1\}$, für die Folgendes gilt.
Jede Sprache L_n wird von einem NFA mit $n + 1$ Zuständen akzeptiert.
Der minimale DFA für L_n hat 2^n Zustände.

Wir definieren die Sprachen

$$L_n := \{w \in \{0,1\}^* \mid w = u.1.v \text{ mit } |v| = n - 1\}.$$

An der n-ten Stelle von rechts steht eine 1.

Kompaktheit von NFAs

Beweis:
Der NFA für L_n rät einfach die richtige 1.

Für die Größe des DFAs benutzen wir den Satz von Myhill und Nerode.
Wir müssen also die Klassen von $\equiv_{L_n}$ bestimmen.
Betrachte Worte der Länge n.
Abhängig davon, wo eine 1 steht, verhalten sie sich unterschiedlich bzgl. $\equiv_{L_n}$.
Das gibt uns bereits 2^n Klassen. □

Konstruktion des minimalen DFA

Stand: Der Äquivalenzklassenautomat ist der minimale DFA für eine Sprache.
Wir haben aber noch keinen Algorithmus, um ihn auszurechnen.

Ziel: Löse das folgende algorithmische Problem.

> *MINIMIZE*
> *Gegeben: Sei A ein DFA.*
> *Problem: Berechne den minimalen DFA B mit $L(A) = L(B)$.*

Annahme: Alle Zustände in $A = (Q, q_0, \to, Q_F)$ seien erreichbar.
Ein Zustand $q \in Q$ ist erreichbar, wenn es ein Wort $w \in \Sigma^*$ gibt mit $q_0 \xrightarrow{w} q$.
Wenn das nicht gilt, streiche die unerreichbaren Zustände.
Berechnung der erreichbaren Zustände: Zusatzvideo über Fixpunkte.

Konstruktion des minimalen DFA

Ansatz: Bestimme die Nerode-Rechtskongruenz $\equiv_{L(A)}$.

Problem: Die ist über einer unendlichen Zahl an Worten definiert.
Das macht sie algorithmisch kaum handhabbar.

Schritt 1: Definieren eine syntaktischere Variante von $\equiv_{L(A)}$.
Sie bezieht sich auf Zustände in A anstatt auf Worte in $L(A)$.

Definition

Zustände $q_1, q_2 \in Q$ heißen äquivalent, geschrieben $q_1 \sim q_2$, falls für alle $w \in \Sigma^*$:

$$\exists q_f \in Q_F.\ q_1 \xrightarrow{w} q_f \quad \text{gdw.} \quad \exists q_f' \in Q_F.\ q_2 \xrightarrow{w} q_f'.$$

Äquivalenz $\sim$ charakterisiert $\equiv_{L(A)}$ wie folgt.

Lemma

Betrachte $u, v \in \Sigma^*$ mit $q_0 \xrightarrow{u} q_1$ und $q_0 \xrightarrow{v} q_2$. Es gilt $u \equiv_{L(A)} v$ gdw. $q_1 \sim q_2$.

Konstruktion des minimalen DFA

Überlegung: Der minimale DFA für $L(A)$ ist der Äquivalenzklassenautomat.
In diesem Automaten sind die Zustände die $\equiv_{L(A)}$-Klassen.
Mit dem Lemma liegen zwei Worte genau dann in derselben $\equiv_{L(A)}$-Klasse, wenn sie zu $\sim$-äquivalenten Zuständen führen.
Die Klassen von $\equiv_{L(A)}$ stimmen also mit den Klassen von $\sim$ überein.

Konsequenz: Um den minimalen DFA zu berechnen, müssen wir genau die $\sim$-äquivalenten Zustände von A zusammenlegen.

Definition

Definiere $A_\sim := (Q/\sim, [q_0]_\sim, \to', Q_F/\sim)$ mit

$$[q_1]_\sim \xrightarrow{a}{}' [q_2]_\sim, \quad \text{falls } q_1 \xrightarrow{a} q_2 \text{ in } A.$$

Lemma

Es gilt $L(A_\sim) = L(A)$ und $A_\sim$ ist minimal.

Konstruktion des minimalen DFA

Schritt 2: Die Definition von $A_\sim$ nimmt $\sim$ als gegeben an.
Wir müssen die Relation noch berechnen.

Ansatz: Nutze einen kleinsten Fixpunkt über $(\mathbb{P}(Q \times Q), \supseteq)$.
Das ist eine umgekehrte Powerset-Domäne, das kleinste Element ist also $Q \times Q$.

Die Funktion $f : \mathbb{P}(Q \times Q) \to \mathbb{P}(Q \times Q)$, deren Fixpunkt wir suchen, ist

$$f(X) \quad := \quad \{(q_1, q_2) \mid \mathrm{cond}_X(q_1, q_2) \wedge \mathrm{cond}_0(q_1, q_2)\}$$

mit den Bedingungen

$$\mathrm{cond}_X(q_1, q_2) := \forall a \in \Sigma.\, \forall q_1', q_2' \in Q. \quad q_1 \xrightarrow{a} q_1' \wedge q_2 \xrightarrow{a} q_2' \quad \Rightarrow \quad (q_1', q_2') \in X$$
$$\mathrm{cond}_0(q_1, q_2) := q_1 \in Q_F \Leftrightarrow q_2 \in Q_F.$$

Lemma

$\sim = \mathrm{lfp}\ f$, *es gilt also* $q_1 \sim q_2$ *gdw.* $(q_1, q_2) \in X$ *mit* $X = \mathrm{lfp}\ f$.

Konstruktion des minimalen DFA

Intuition: Es gilt $q_1 \sim q_2$, falls

$$\forall w \in \Sigma^*. \quad \exists q_f \in Q_F.\ q_1 \xrightarrow{w} q_f \quad \Leftrightarrow \quad \exists q_f' \in Q_F.\ q_2 \xrightarrow{w} q_f'.$$

Wir ordnen die Worte nach ihrer Länge und schreiben die Bedingung um.
Dann gilt $q_1 \sim q_2$, falls

$$\forall n \in \mathbb{N}. \forall w \in \Sigma^* \text{ mit } |w| = n. \quad \exists q_f \in Q_F.\ q_1 \xrightarrow{w} q_f \quad \Leftrightarrow \quad \exists q_f' \in Q_F.\ q_2 \xrightarrow{w} q_f'.$$

Worte w der Länge n ergeben sich aus Worten w' der Länge $n-1$, $w = a.w'$.
Damit erhalten wir eine Rekursion, es gilt $q_1 \sim q_2$, falls

$$\forall n \in \mathbb{N}.\ \mathrm{cond}_n(q_1, q_2).$$

Die Bedingung für $n = 0$ haben wir oben eingeführt.
Für $n > 0$ lautet die Bedingung:

$$\mathrm{cond}_n(q_1, q_2) \ := \ \forall a \in \Sigma. \forall q_1', q_2' \in Q.\ q_1 \xrightarrow{a} q_1' \wedge q_2 \xrightarrow{a} q_2' \quad \Rightarrow \quad \mathrm{cond}_{n-1}(q_1', q_2').$$

Konstruktion des minimalen DFA

Die Mengen $X^0, X^1, \ldots$ aus der Fixpunkt-Iteration erfüllen Folgendes.

Lemma

Für $i > 0$ gilt $(q_1, q_2) \in X^i$ gdw. $\mathrm{cond}_i(q_1, q_2)$.

Der Offset 1 liegt am Start der Fixpunktiteration bei $\bot = Q \times Q$.

Bemerkung: Wenn man eine Menge oder eine Bedingung
durch das Einführen von Parametern in Scheiben schneidete,
spricht man von einer Stratifizierung.
Wir haben also die Menge der Worte
entlang der Wortlänge stratifiziert.

Konstruktion des minimalen DFA

Optimierung: Da die Relation $\sim$ symmetrisch und reflexiv ist, genügt es, Zustandspaare (q_i, q_j) zu betrachten mit $i < j$.

Anschauung: Die Fixpunkt-Iteration streicht sukzessive Paare aus $Q \times Q$. Das sind Kandidaten für $\sim$, die dann doch nicht in $\sim$ liegen.

Dieses Streichen kann man sich als Ankreuzen von Zellen einer Matrix vorstellen. Wegen der Symmetrie ist das eine obere Dreiecksmatrix.

Initial werden alle Paare markiert, bei denen nur ein Zustand final ist.
Anschließend wird geprüft, ob (q_1, q_2) für einen Buchstaben in ein markiertes Paar übergeht.
In dem Fall wird auch (q_1, q_2) markiert.

Die Illustration ist im Skript in den Beispielen 6.15 und 6.16 ausgearbeitet.

5. Das Pumping-Lemma

Das Pumping-Lemma

Motivation: Mit der Nerode-Kongruenz ist es schwierig, Regularität zu widerlegen.
Immerhin müssen wir unendlich viele Klassen identifizieren.
Außerdem lässt sich der Ansatz nicht auf größere Sprachfamilien verallgemeinern.

Ziel: Entwickle eine Technik für das Widerlegen von Regularität, die
(i) einfacher zu nutzen und (ii) besser zu verallgemeinern ist.

Ansatz: Finde eine notwendige Bedingung für Regularität,
also eine Bedingung cond auf Sprachen, so dass für alle Sprachen L gilt:

$$L \text{ regulär} \quad \Rightarrow \quad \text{cond } L \text{ gilt.}$$

Mit so einer Bedingung lässt sich Regularität widerlegen, indem man zeigt,
dass cond L falsch ist.

Das Pumping-Lemma

Ziel: Finde eine gute Bedingung cond L.
Gut bedeutet, dass sie für viele nicht reguläre Sprachen falsch wird.

Idee: Reguläre Sprachen haben eine bestimmte Struktur.
Sobald Worte lang genug werden, gibt es Infixe, die man wiederholen kann.
Die resultierenden Worte müssen ebenfalls in der Sprache liegen.

Intuition: Irgendwann läuft man durch Schleifen eines NFA für die Sprache.

Das Pumping-Lemma

Satz (Das Pumping-Lemma für reguläre Sprachen)

Sei L eine reguläre Sprache. Dann gibt es eine Pumping-Konstante $p_L \in \mathbb{N}$, so dass für alle Worte $x \in L$ mit $|x| \geq p_L$ Folgendes gilt.
Es gibt eine Zerlegung $x = u.v.w$ mit

1. $|v| \geq 1$
2. $|u.v| \leq p_L$
3. $u.v^i.w \in L$ für alle $i \in \mathbb{N}$.

Das Pumping-Lemma

Beweis:
Sei $L = L(A)$ mit $A = (Q, q_0, \to, Q_F)$ einem NFA.
Wähle $p_L := |Q|$, die Zahl der Zustände.
Betrachte ein Wort $x \in L$ mit $x = a_1 \dots a_d$ und $d \geq p_L$.
Da $x \in L$, gibt es einen akzeptierenden Ablauf von A auf x:

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_d} q_d \in Q_F.$$

Der Lauf enthält $d + 1 > p_L = |Q|$ Zustände.
Also wiederholt sich ein Zustand.
Betrachte die erste solche Wiederholung, $q_j = q_k$ mit $j < k$.
Es sind also q_0 bis q_{k-1} alle verschieden.

Das Pumping-Lemma

Beweis (cont.):
Definiere

$$\begin{aligned} u &:= a_1 \dots a_j \\ v &:= a_{j+1} \dots a_k \\ w &:= a_{k+1} \dots a_d. \end{aligned}$$

Diese Worte können das Geforderte:

1. $|v| \geq 1$, da $j < k$.
2. $|u.v| \leq p_L$, da die Zustände q_0 bis q_{k-1} verschieden sind.
3. $u.v^i.w \in L$ für alle $i \in \mathbb{N}$, da wir eine Schleife im NFA haben. □

Das Pumping-Lemma

Ziel: Wende das Pumping-Lemma an, um Regularität zu widerlegen.

Lemma

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

Beweis.

Angenommen L wäre regulär.
Dann erfüllt L das Pumping-Lemma.
Sei $p_L \in \mathbb{N}$ die Konstante aus dem Pumping-Lemma.
Betrachte das Wort $x = a^{p_L}.b^{p_L}$.
Es gilt $x \in L$ und $|x| \geq p_L$.
Sei nun $x = u.v.w$ die Zerlegung, die das Pumping-Lemma liefert.
Es gilt $|u.v| \leq p_L$, also besteht v nur aus Buchstaben a.
Außerdem gilt $|v| \geq 1$, das heißt, v hat auch tatsächlich Buchstaben.
Betrachte nun das Wort $u.w$.
Nach dem Pumping-Lemma müsste $u.w \in L$ gelten.
Das stimmt aber nicht, wir haben $u.w = a^{p_L - |v|}.b^{p_L}$.
Also muss die Annahme, L sei regulär, falsch gewesen sein. □