Rheinisch-Westfälische Technische Hochschule Aachen

Skript zur Vorlesung

Formale Systeme, Automaten, Prozesse

Letzte Änderung:
8. März 2018

Autor:
Niklas Rieken

Hinweise

Dieses Skript entstand aus der Vorlesung Formale Systeme, Automaten, Prozesse an der RWTH Aachen von Prof. Dr. Wolfgang Thomas und Prof. Dr. Martin Grohe vom Lehrstuhl Informatik 7 in den Sommersemestern 2015 und 2016. Ein paar Notationen und Definitionen sind außerdem adaptiert aus dem Skript zu den Diskreten Strukturen und Lineare Algebra I für Informatiker von Dr. Timo Hanke und Prof. Dr. Gerhard Hiß vom Lehrstuhl D für Mathematik.

Inhaltsverzeichnis

1 Mathematisches Vorwissen

In diesem ersten Kapitel fixieren wir einige mathematischen Notationen und geben elementare Sätze aus der diskreten Mathematik, die im weiteren verlauf der Vorlesung benötigt werden. In der Regel sollten nahezu alle Begriffe und Notationen aus dem ersten Semester bereits bekannt sein. Deshalb ist dieses Kapitel eher nur für Sommersemesteranfänger bestimmt. Die in diesem Abschnitt gesammelten Definitionen und Sätze sind nicht alle relevant, aber dennoch nützlich zu kennen im weiteren Verlauf des Studiums.

1.1 Mengen

Der Begriff Menge geht auf Georg Cantor aus dem 19. Jahrhundert zurück und wurde (verglichen mit späteren Definitionen in diesem Skript) informell beschrieben.

> Unter einer “Menge“ verstehen wir jede Zusammenfassung M von bestimmten wohlunterschiedenen Objekten m unserer Anschauung oder unseres Denkens (welche die “Elemente“ von M genannt werden) zu einem Ganzen.

Wir definieren eine Menge wie folgt

Definition 1.1. Eine *Menge* M ist etwas, zu dem jedes beliebige Objekt x entweder *Element* der Menge ist ($x \in M$), oder nicht ($x \notin M$).

Mengen selbst können auch wieder als Objekte aufgefasst werden, also Elemente anderer Mengen sein. Wir vermeiden jedoch Aussagen über “Mengen, die sich selbst enthalten“, da so schnell Widersprüche entstehen können (vgl. Russel'sche Antinomie). Wir schließen uns der weit verbreiteten *Zermelo-Fraenkel-Mengenlehre* an, dazu geben wir jedoch keine Details (diese findet man zum Beispiel in der Logik 2-Vorlesung im Wahlpflichtbereich). Wir schauen uns nur an, wie wir Mengen im allgemeinen betrachten können. Folgende Definition sind dabei elementar.

Definition 1.2. Seien M, N zwei Mengen. N ist eine *Teilmenge* von M ($N \subseteq M$) bzw. M eine *Obermenge* von N ($M \supseteq N$), wenn für alle $x \in N$ gilt, dass auch $x \in M$.
Wir sagen N ist eine *echte Teilmenge* von M ($N \subset M$) bzw. M eine *echte Obermenge* von N ($M \supset N$), wenn es zusätzlich ein $y \in M$ gibt mit $y \notin N$.
M und N sind *gleich* ($M = N$), wenn sowohl $M \subseteq N$ als auch $N \subseteq M$ gilt.

Wir kommen nun zum Mächtigkeitsbegriff der Mengenlehre, der für die Anzahl der Elemente einer Menge beschreibt.

Definition 1.3. Sei M eine Menge. M heißt *endlich*, wenn M nur endlich viele Elemente besitzt, dann beschreibt $|M|$ die Anzahl der Elemente von M. Andernfalls heißt M *unendlich* und wir schreiben $|M| = \infty$. Man nennt $|M|$ die *Mächtigkeit* von M.

Um eine konkrete Menge zu zu benennen gibt es im Wesentlichen vier verschiedene Möglichkeiten:

(i) *Aufzählen.* Die Elemente der Menge werden aufgelistet und in Mengenklammern $(\{,\})$ eingeschlossen. Reihenfolge und Wiederholungen spielen keine Rolle.

$$\{3, 4.5, \pi, \diamondsuit\} = \{\pi, 4.5, \diamondsuit, \diamondsuit, 3\} \subseteq \{\diamondsuit, \pi, 4.5, 3, \clubsuit\}.$$

(ii) *Beschreiben.* Mengen können durch Worte beschrieben werden.

$$\text{Menge der natürlichen Zahlen} = \{0, 1, 2, 3, \ldots\} =: \mathbb{N}.$$

Aber Achtung: Natürliche Sprache neigt zu Uneindeutigkeit!

(iii) *Aussondern.* Sei M eine Menge, dann ist

$$\{x \in M \mid \varphi(x)\}$$

die Menge aller Elemente aus M, die die Eigenschaft φ erfüllen. Zum Beispiel:

$$\mathbb{P} := \{n \in \mathbb{N} \mid n \text{ hat genau zwei Teiler}\}$$

als Menge aller Primzahlen.

(iv) *Abbilden.* Sei M eine Menge und f ein Ausdruck, der für jedes $x \in M$ definiert ist. Dann ist

$$\{f(x) : x \in M\}$$

die Menge aller Ausdrücke $f(x)$, wobei jedes $x \in M$ in f eingesetzt wird. Zum Beispiel:

$$\{n^2 : n \in \mathbb{N}\}$$

als Menge aller Quadtratzahlen.

Wir können Abbilden und Aussondern auch kombinieren, zum Beispiel mit:

$$\{n^2 : n \in \mathbb{N} \mid n \text{ ungerade}\}$$

als Menge aller Quadratzahlen von ungeraden natürlichen Zahlen. Man würde hier jedoch abkürzend schreiben:

$$\{n^2 : n \in \mathbb{N} \text{ ungerade}\}.$$

Mit der Definition, dass eine Zahl n ungerade ist, wenn ein k existiert mit $2k+1 = n$ (und umgekehrt n gerade, wenn ein k exisitiert mit $n = 2k$) könnten wir auch schreiben:

$$\{n^2 : n \in 2\mathbb{N} + 1\}.$$

Eine wichtige Menge haben wir bisher außen vor gelassen: die *leere Menge*. Wir schreiben $\varnothing := \{\}$. Gelegentlich verwenden wir außerdem folgende Notation, wenn wir nur eine endliche geordnete Menge benötigen: $[\ell] := \{0, 1, \ldots, \ell - 1\}$. Ein-elementige Mengen (z.B. $[1] = \{0\}$) nennt man auch *Singleton*.

Bemerkung. Im Allgemeinen gibt es keine Elemente in Mengen, die mehrfach vorkommen. Man kann dies explizit zulassen, durch *Multimengen*, z.B.: $\{{}^*1, 1, 2, \pi, \clubsuit, \pi^*\}$.

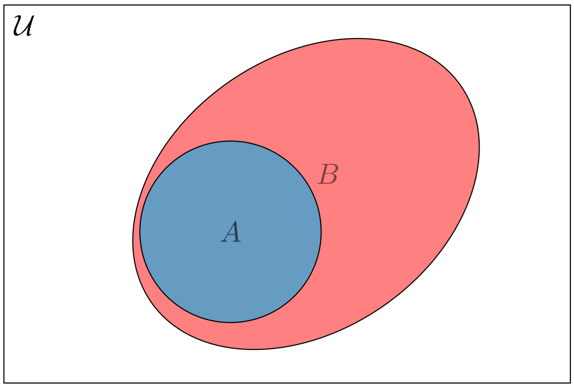


Abbildung 1: Venn-Diagramm für $A \subseteq B$.

1.2 Operationen auf Mengen

Im folgendem betrachten wir Mengen immer als Teilmenge eines *Universums* (oder auch *Grundemenge*) $\mathcal{U}$. In der Analysis ist das typischerweise die Menge der reellen Zahlen $\mathbb{R}$ (oder die Menge der komplexen Zahlen $\mathbb{C}$), die betrachteten Teilmengen sind oftmals Intervalle in denen zum Beispiel Funktionen auf Stetigkeit hin untersucht werden. Vorweg: Wir betrachten später im Allgemeinen das Universum Σ^* und Sprachen als Teilmenge von eben diesem. Genaueres folgt im nächsten Kapitel.
Um die Operatoren auf den Mengen zu veranschaulichen gibt es die sogenannten *Venn-Diagramme*, bei denen Kreise oder Ellipsen die Mengen visualisieren. In Abbildung 1 finden wir zum Beispiel für die Inklusion ($\subseteq$) ein entsprechendes Diagramm. Wir definieren nun einige Operationen auf Mengen ähnlich wie Addition und Multiplikation usw. auf Zahlen. Zusätzlich zur formalen Definition befindet sich in Abbildung 2 auch ein passendes Venn-Diagramm. Die jeweils eingefärbte Fläche kennzeichnet die resultierende Menge. $\mathcal{U}$ sei ein beliebiges aber festes Universum.

Definition 1.4. Seien A, B Mengen.

(a) Die *Vereinigung* von A und B ist definiert als

$$A \cup B \coloneqq \{a \in \mathcal{U} \mid a \in A \text{ oder } a \in B\}.$$

Für endliche und unendliche Vereinigungen (z.B. gegeben durch eine Indexmenge $I = \{0, 1, \ldots\}$) schreiben wir abkürzend

$$\bigcup_{i \in I} A_i = A_0 \cup A_1 \cup \ldots$$

(b) Der *Schnitt* von A und B ist definiert als

$$A \cap B \coloneqq \{a \in A \mid a \in B\}.$$

Für endlichen und unendlichen Schnitt (z.B. gegeben durch eine Indexmenge $I = \{0, 1, \ldots\}$) schreiben wir abkürzend

$$\bigcap_{i \in I} A_i = A_0 \cap A_1 \cap \ldots$$

(c) Das *Komplement* von A is definiert als

$$\overline{A} \coloneqq \{a \in \mathcal{U} \mid a \notin A\}.$$

(d) Die *Differenz* (auch: *relatives Komplement*) von A und B ist definiert als

$$A \setminus B \coloneqq A \cap \overline{B}.$$

(e) Das *kartesische Produkt* zwischen A und B ist definiert als

$$A \times B \coloneqq \{(a, b) : a \in A, b \in B\}.$$

Für ein endliches Produkt einer Menge A auf sich selbst schreiben wir abkürzend

$$A^k \coloneqq A \times A^{k-1}, \qquad A^1 \coloneqq A, \qquad A^0 \coloneqq \{\bullet\},$$

wobei $\bullet$ ein beliebiges Platzhaltersymbol ist, d.h. A^0 ist für jedes A ein Singleton. Die Elemente eines kartesischen Produkts $(x_1, \ldots, x_k)$ heißen *k-Tupel*. Für $k = 2, 3, 4, \ldots$ kann man auch *Paar, Tripel, Quadrupel, …* sagen.

(f) Die *Potenzmenge* von A ist definiert als

$$2^A \coloneqq \{M \subseteq \mathcal{U} \mid M \subseteq A\}.$$

1.3 Relationen

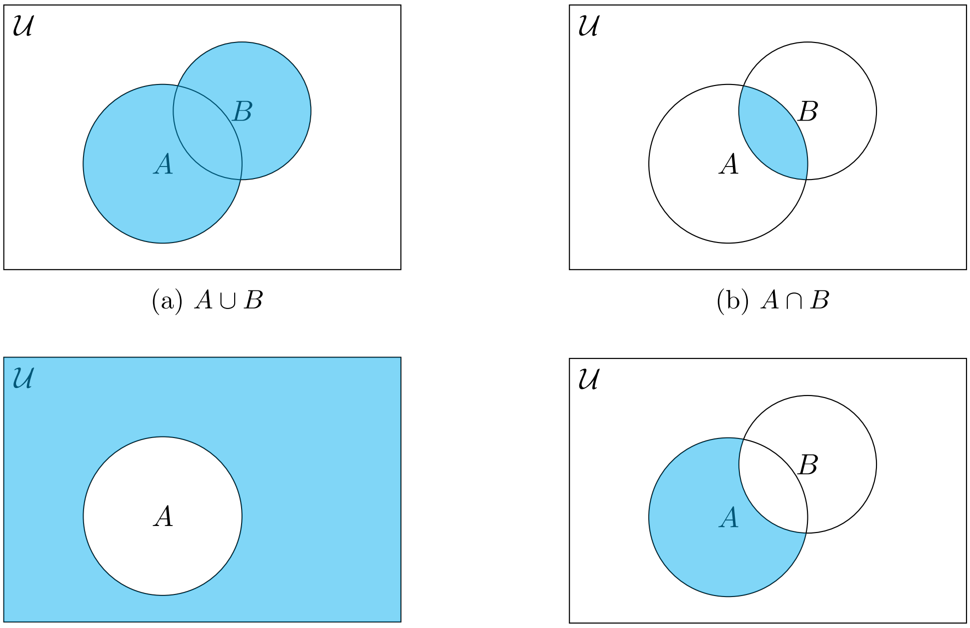
Relationen drücken Beziehungen oder Zusammenhänge zwischen Elementen aus. Im Allgemeinen können dies Beziehungen zwischen beliebig vielen Elementen sein und wir werden verschieden stellige Relationen auch im Laufe der Vorlesung benutzen. In diesem Abschnitt legen wir aber ein besonderes Augenmerk auf 2-stellige Relationen.

Definition 1.5. Es seien $M_1, \ldots, M_k$ nicht-leere Mengen. Eine Teilmenge $R \subseteq M_1 \times \ldots \times M_k$ heißt *Relation* zwischen $M_1, \ldots, M_k$ (oder auf M, falls $M = M_1 = \ldots = M_k$).

Für 2-stellige Relationen verwenden wir oft Symbole wie $\sim, \prec$ und schreiben dann statt $(a, b) \in \sim$ intuitiver $a \sim b$.

Definition 1.6. Sei $\sim \subseteq M \times M$ eine 2-stellige Relation. $\sim$ heißt

- *reflexiv*, falls $x \sim x$ für alle $x \in M$,



(a) $A \cup B$ (b) $A \cap B$ (c) $\overline{A}$ (d) $A \setminus B$

Abbildung 2: Venn-Diagramme für Mengen-Operationen.

- *symmetrisch*, falls für alle $x, y \in M$ mit $x \sim y$ auch $y \sim x$ gilt,
- *antisymmetrisch*, falls für alle $x, y \in M$ mit $x \sim y$ und $y \sim x$ gilt, dass $x = y$,
- *transitiv*, falls für alle $x, y, z \in M$ mit $x \sim y$ und $y \sim z$ gilt, dass $x \sim z$.

Wir klassifizieren außerdem 2-stellige Relationen, falls sie bestimmte Eigenschaften haben.

Definition 1.7. Sei $\sim \subseteq M \times M$ eine 2-stellige Relation. $\sim$ heißt

- *Äquivalenzrelation*, falls sie reflexiv, symmetrisch und transitiv ist,
- *(partielle) Ordnung*, falls sie reflexiv, antisymmetrisch und transitiv ist,
- *Totalordnung*, falls sie partielle Ordnung ist und für alle $x, y \in M$ entweder $x \sim y$ oder $y \sim x$ gilt.

Beispiel.

(i) Die Relation $\leq$ ist auf $\mathbb{N}$ eine Totalordnung.

(ii) Die Relation $\{(a, b) \in \mathbb{R}^2 \mid a^2 = b^2\}$ ist eine Äquivalenzrelation auf $\mathbb{R}$.

Definition 1.8. Sei $\sim$ eine Äquivalenzrelation auf einer Menge M. Für $x \in M$ heißt

$$[x]_\sim \coloneqq x/_\sim \coloneqq \{y \in M \mid x \sim y\}$$

die *Äquivalenzklasse* von x. Die Menge aller Äquivalenzklassen von $\sim$ wird notiert mit $M/_\sim := \{[x]_\sim : x \in M\}$. Der *Index* einer Äquivalenrelation (index($\sim$)) ist die Anzahl ihrer Äquivalenzklassen (möglicherweise ∞).

Definition 1.9. Sei A eine Menge. Eine *Partition* von A ist eine Menge $\mathcal{P} \subset 2^A \setminus \{\varnothing\}$, sodass

(i) $P \cap Q = \varnothing$ für alle $P, Q \in \mathcal{P}$ mit $P \neq Q$,

(ii) $\bigcup_{P \in \mathcal{P}} P = A$.

Die Mengen $P \in \mathcal{P}$ heißen *Teile* oder *Klassen* der Partition $\mathcal{P}$.

Bemerkung. Sei A eine Menge.

(i) Ist $\sim$ eine Äquivalenzrelation auf A, so ist $\{[x]_\sim : x \in A\}$ eine Partition von A.

(ii) Ist $\mathcal{P}$ eine Partition auf A, so ist $\sim_\mathcal{P}$ mit

$$x \sim_\mathcal{P} y \text{ g.d.w. } x, y \in P \text{ für ein } P \in \mathcal{P}$$

eine Äquivalenzrelation auf A und es gilt $\mathcal{P} = \{[x]_{\sim_\mathcal{P}} : x \in A\}$.

1.4 Gesetze für Mengen

In diesem Abschnitt sammeln wir ein paar Gesetzmäßigkeiten, die für Mengen gelten. Manche davon sind offensichtlich, wir werden aber auch zu ein paar Aussagen die Beweise geben, manche bleiben als Übung.

Bemerkung. Für die Inklusion gilt offensichtlich für jede Menge M

$$\varnothing \subseteq M \subseteq M \subseteq \mathcal{U}.$$

Insbesondere ist die Relation $\subseteq$ reflexiv. Sie ist außerdem transitiv und per Definition der Gleichheit von Mengen antisymmetrisch, also eine partielle Ordnung.
Schnitt und Vereinigung sind per Definition offenbar *assoziativ* (d.h. $A \cup (B \cup C) = (A \cup B) \cup C$ und $A \cap (B \cap C) = (A \cap B) \cap C$) und *kommutativ* (d.h. $A \cup B = B \cup A$ und $A \cap B = B \cap A$). Außerdem sind diese beiden Operationen zueinander *distributiv*, was wir im folgenden einmal zeigen wollen.

Das Beweisschema für solche Aufgaben ist stets das selbe und sollte deshalb auch ruhig übernommen werden für Übungsaufgaben. Tricks sind selten notwendig, es ist meist

Definition anwenden – triviale Umformung – Definition anwenden – Profit.

Bemerkung. Für Mengen A, B, C gilt:

(i) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$,

(ii) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.

Beweis. Wir zeigen nur Aussage (i), die zweite Hälfte geht analog. Wir müssen zwei Richtungen beweisen.
"$\subseteq$": Sei $a \in A \cup (B \cap C)$. D.h. $a \in A$ oder $a \in B \cap C$.

- $a \in A$. Dann ist a auch in $A \cup B$ und $A \cup C$ (da $\cup$ die Mengen nicht verkleinert). Also ist a auch im Schnitt dieser beiden Mengen.
- $a \in B \cap C$. Dann ist $a \in B$ und $a \in C$. Also (da wie oben $\cup$ die Menge nicht verkleinert) ist $a \in A \cup B$ und $a \in A \cup C$. Somit ist a auch wieder im Schnitt beider Mengen.

"$\supseteq$": Sei $a \in (A \cup B) \cap (A \cup C)$. Dann ist $a \in A \cup B$ und $a \in A \cup C$ $(*)$. Wir unterscheiden zwei Fälle:

- $a \in A$. Unabhängig von B, C ist dann $a \in A \cup (B \cap C)$.
- $a \notin A$. Dann muss $a \in B$ und $a \in C$ gelten, sonst würde $(*)$ nicht gelten. Somit ist $a \in B \cap C$ und damit auch wieder $a \in A \cup (B \cap C)$.

Wir haben also $A \cup (B \cap C) \subseteq (A \cup B) \cap (A \cup C)$ und $A \cup (B \cap C) \supseteq (A \cup B) \cap (A \cup C)$ gezeigt. Somit muss Gleichheit zwischen diesen beiden Mengen vorliegen. $\square$

Weiterhin nützlich sind noch folgende Bemerkungen.

Bemerkung (DeMorgan'sche Gesetze)**.** Für Mengen A, B gilt:

- $\overline{(A \cup B)} = \overline{A} \cap \overline{B}$,
- $\overline{(A \cap B)} = \overline{A} \cup \overline{B}$.

Bemerkung (Absorptionsgesetz)**.** Für Mengen A, B gilt:

- $A \cup (A \cap B) = A$,
- $A \cap (A \cup B) = A$.

Die Beweise hierfür bleiben als Übung.

1.5 Funktionen

Definition 1.10. Seien M, N Mengen. Eine *Funktion* (oder *Abbildung) f* von M nach N ist eine Vorschrift (z.B. eine Formel), die jedem $x \in M$ genau ein $f(x) \in N$ zuordnet. Wir schreiben dazu

$$f\colon M \to N, \quad x \mapsto f(x).$$

M heißt der *Definitionsbereich* (auch Domäne) von f, N heißt der *Wertebereich* von f. $f(x)$ ist das *Bild* von x unter f und x ist ein *Urbild* von $f(x)$ unter f. Die Menge aller Funktionen von M nach N wird mit N^M bezeichnet. Falls $M = A^n, N = A$ für ein nichtleeres A ist, sagen wir auch, dass f eine *n-stellige* Funktion über A ist. Desweitere nennen wir eine 0-stellige Funktion auch *Konstante*.

Definition 1.11. Sei $f\colon M \to N$ eine Funktion.

- Für jede Teilmenge $X \subseteq M$ ist $f(X) \coloneqq \{f(x) : x \in X\}$ das *Bild von X unter f*. Falls $X = M$ sprechen wir auch nur vom *Bild* von f.
- Für jede Teilmenge $Y \subseteq N$ ist $f^{-1}(Y) \coloneqq \{x \in M \mid f(x) \in Y\}$ das *Urbild von Y unter f*.

Definition 1.12. Zu einer Menge M ist $\mathrm{id}_M\colon M \to M, x \mapsto x$ die *Identität*.

Definition 1.13. Eine Funktion $f\colon M \to N$ heißt

- *surjektiv*, falls für alle $y \in N$ ein $x \in M$ mit $f(x) = y$ existiert,
- *injektiv*, falls für alle $x, x' \in M$ mit $x \neq x'$ gilt, dass $f(x) \neq f(x')$,
- *bijektiv*, falls f surjektiv und injektiv ist.

Beispiel. Die Addition zweier natürlicher Zahlen kann als Funktion aufgefasst werden:

$$+\colon \mathbb{N}^2 \to \mathbb{N}, \quad (m, n) \mapsto m + n.$$

$+$ ist surjektiv (jedes $y \in \mathbb{N}$ wird z.B. durch $(y, 0) \in \mathbb{N}^2$ getroffen), aber nicht injektiv ($1 \in \mathbb{N}$ wird sowohl von $(1, 0)$ als auch $(0, 1)$ getroffen).

Für Funktionen $f\colon [k] \to M$ für beliebige $k \in \mathbb{N}$ in beliebige M können wir auch abkürzend die Tupelschreibweise $(y_0, \ldots, y_{k-1})$ verwenden. Dann ist $y_i = f(i)$ für alle $i \in [k]$.
Wir fügen nun noch ein paar nützliche Definitionen an um dieses Skript in sich abgeschlossen zu halten.

Definition 1.14. Sei $f\colon M \to N$ eine Funktion und $A \subseteq M$. Dann ist $f|_A\colon A \to N$ die *Einschränkung von f auf A* mit $f|_A(x) = f(x)$ für alle $x \in A$.

Definition 1.15. Wir schreiben für zwei Funktionen $f, g\colon M \to N$, dass sie *identisch* sind ($f \equiv g$), wenn für alle $x \in M$ gilt, dass $f(x) = g(x)$.

Definition 1.16. Die *Komposition* zweier Funktionen $f\colon M \to M', g\colon M' \to N$ ist bezeichnet mit $g \circ f\colon M \to N, x \mapsto g(f(x))$.

Definition 1.17. Seien $f\colon M \to N, g\colon N \to M$ Funktionen. Dann heißt g eine *linksseitige (rechtseitige) Umkehrfunktion* von f, wenn $g \circ f \equiv id_M$ ($f \circ g \equiv id_N$). Wenn g sowohl links- als auch rechtsseitige Umkehrfunktion ist sprechen wir schlicht von einer *Umkehrfunktion* von f und bezeichnen diese mit f^{-1}.

Bemerkung. Die Schreibweise f^{-1} benutzen wir sowohl für das Urbild als auch für Umkehrabbildungen, was jedoch zwei verschiedene Begriffe sind!

1.6 Strukturen

Wir haben Mengen, Relationen und Abbildungen jeweils eigenständig kennengelernt. Tatsächlich benutzen wir sie allerdings nur zusammen. Beispielsweise sind die natürlichen Zahlen für sich allein nicht sehr interessant, wenn man auf ihnen nicht addieren könnte. Umgekehrt sind auch Funktionsvorschriften wertlos, wenn sie nicht auf Elemente einer Menge angewendet werden kann. Wir fassen alles nun unter dem Begriff einer *Struktur* zusammen.

Definition 1.18. Sei $\tau = \{R_1, \dots, R_m, f_1, \dots, f_n\}$ eine Menge von Relationssymbolen und Funktionssymbolen, eine sogenannte *Signatur*. Eine *τ-Struktur* ist ein Paar $\mathfrak{A} = (A, \mathfrak{a})$, wobei A eine nicht-leere Menge ist, die *Universum* (oder *Träger*) heißt und $\mathfrak{a}$ ist eine Funktion, die jedem k-stelligem Relationssymbol $R \in \tau$ eine k-stellige Relation und jedem k-stelligem Funktionssymbol $f \in \tau$ eine k-stellige Funktion zuordnet.

Statt $\mathfrak{a}(R), \mathfrak{a}(f)$ schreibt man auch häufig $R^{\mathfrak{A}}, f^{\mathfrak{A}}$. In der mathematischen Logik ist es wichtig eine Unterscheidung zwischen Relations- und Funktionssymbolen und ihren Interpretationen durch konkrete Relationen $R^{\mathfrak{A}}$ bzw. Funktionen $f^{\mathfrak{A}}$ zu machen. Deshalb sollten Schreibweisen wie $\mathfrak{N} = (\mathbb{N}, +)$ nur als Abkürzungen verstanden werden für $\mathfrak{N} = (\mathbb{N}, +^{\mathfrak{N}})$ mit $+^{\mathfrak{N}} \colon \mathbb{N}^2 \to \mathbb{N}, (m, n) \mapsto +^{\mathfrak{N}}(m, n) = m + n$, wobei wir mit $m + n$ den gewohnten n-ten Nachfolger von m bezeichnen. Der Autor hofft, dass es klar ist worauf er hinaus wollte.
Wir können Strukturen wieder klassifizieren nach sogenannten *Axiomen*, die sie erfüllen.

Definition 1.19. Eine Struktur vom Typ $(A, \bullet)$ heißt *Magma*. Ein Magma $(A, \bullet)$ heißt *abelsch* (oder *kommutativ)*, falls für alle $a, b \in A$ gilt: $a \bullet b = b \bullet a$. Es heißt *assoziativ* (oder *Halbgruppe*) falls für alle $a, b, c \in A$ gilt: $(a \bullet b) \bullet c = a \bullet (b \bullet c)$.

Definition 1.20. Sei $\mathfrak{A} = (A, \bullet)$ eine Struktur mit $\bullet \colon A \times A \to A$. Wir sagen $\mathfrak{A}$ ist ein *Monoid*, wenn folgende Axiome gelten:

(i) *Assoziativität*, für alle $a, b, c \in A$ gilt $(x \bullet y) \bullet z = x \bullet (y \bullet z)$.

(ii) *neutrales Element*, es gibt ein $e \in A$, sodass für alle $a \in A$ gilt $a \bullet e = e \bullet a = a$.

Gilt zusätzlich

(iv) *Kommutativität*, für alle $a, b \in A$ gilt $a \bullet b = b \bullet a$.

so heißt das Monoid *abelsch*.

Beispiel 1.21. Sei A eine beliebige nicht-leere Menge und $\mathbb{B} = \{0, 1\}$.

(i) $(\mathbb{N}, +)$ ist abelsches Monoid mit neutralem Element 0,

(ii) $(\mathbb{R}, \cdot)$ ist abelsches Monoid mit neutralem Element 1,

(iii) $(A^A, \circ)$ ist Monoid mit neutralem Element id_A,

(iv) $(\mathbb{B}, \wedge)$ ist abelsches Monoid mit neutralem Element 1.

Es ist bei Monoiden auch üblich, dass das neutrale Element mit in die Struktur geschrieben wird also, z.B. $(\mathbb{R}, \cdot, 1)$.

Definition 1.22. Sei $\mathfrak{A} = (A, \bullet)$ ein Monoid. Eine Teilmenge $E \subseteq A$ heißt *Erzeugendensystem* von $\mathfrak{A}$ falls jedes $a \in A$ als $a = e_0 \bullet \ldots \bullet e_{n-1}$ mit $e_i \in E$ für alle $i \in [n]$ dargestellt werden kann. E heißt *frei*, falls diese Darstellung eindeutig ist. In diesem Fall nennen wir $\mathfrak{A}$ auch das von E *frei erzeugte Monoid* (oder unter Missbrauch der Notation auch *freies Monoid*).

Bemerkung. Das neutrale Element wird stets durch eine *leere* Anwendung dargestellt. Beispielsweise ist für $(\mathbb{N}, +)$ die Menge $\{1\}$ ein (freies) Erzeugendensystem, denn jede natürliche Zahl n lässt sich als Summe von n 1en darstellen $n = \sum_{i=1}^{n} 1$, für $n = 0$ erhalten wir eben genau die leere Summe ohne Summanden.

Definition 1.23. Sei $(A, \bullet, e)$ Monoid und $a \in A$.

- Gibt es ein $b \in A$ mit $a \bullet b = e$ so heißt a *rechtsinvertierbar*.
- Gibt es ein $b \in A$ mit $b \bullet a = e$ so heißt a *linksinvertierbar*.
- Ist a rechts- und linksinvertierbar, dann heißt a eine *Einheit*.
- Gibt es ein $b \in A$ mit $b \bullet a = a \bullet b = e$ so heißt a *invertierbar* und b *invers* zu a.

Die *Menge aller Einheiten* von A bezeichnen wir mit $A^\times$.

Wir bezeichnen die Inversen zu Einheiten a in der Regel mit a^{-1} oder $-a$.

Definition 1.24. Sei $\mathfrak{A} = (A, \bullet, e)$ ein Monoid. Falls in $\mathfrak{A}$ zusätzlich

(iii) *Invertierbarkeit*, für alle $a \in A$ existiert $b \in A$ mit $a \bullet b = b \bullet a = e$,

dann ist $\mathfrak{A}$ eine *Gruppe* (bzw. *abelsche Gruppe*).

Lemma 1.25. *Ist $(A, \bullet, e)$ Monoid, so ist $(A^\times, \bullet, e)$ Gruppe (Einheitengruppe).*

Beweis. Per Definition ist jedes Element in $A^\times$ eine Einheit. Außerdem ist e stets eine Einheit, da es invers zu sich selbst ist. Es bleibt zu zeigen, dass $A^\times$ auch abgeschlossen ist unter der Operation $\bullet$. Sei dazu $a, b \in A^\times$ und a^{-1}, b^{-1} die jeweiligen Inversen $(*)$. Wir zeigen das $a \bullet b$ Einheit ist mit Inversem $b^{-1} \bullet a^{-1}$:

$$(a \bullet b) \bullet (b^{-1} \bullet a^{-1}) \overset{\text{(i)}}{=} a \bullet (b \bullet b^{-1}) \bullet a^{-1} \overset{(*)}{=} a \bullet e \bullet a^{-1} \overset{\text{(ii)}}{=} a \bullet a^{-1} \overset{(*)}{=} e. \qquad \square$$

Beispiel 1.26. Sei $\mathbb{B} = \{0, 1\}$. Das Symbol $\not\leftrightarrow$ steht für das Boolsche exklusive Oder (xor).

(i) $(\mathbb{B}, \not\leftrightarrow)$ ist abelsche Gruppe,

(ii) $(\mathbb{R} \setminus \{0\}, \cdot)$ ist abelsche Gruppe,

(iii) $(\mathbb{Z}, -)$ ist Gruppe.

Definition 1.27. Sei $\mathfrak{A} = (A, \bullet)$ eine Gruppe und $B \subseteq A$ nicht-leer. $\mathfrak{B} = (B, \bullet)$ ist eine *Untergruppe* von $\mathfrak{A}$, wenn für alle $a, b \in B$ auch $a \bullet b^{-1} \in B$ gilt.

Natürlich ist es auch möglich eine solche Klassifizierung von Strukturen mit mehr als einer Funktion vorzunehmen.

Definition 1.28. Eine Struktur $(A, \oplus, \odot)$ heißt *Ring*, falls gilt

(i) $(A, \oplus)$ ist abelsche Gruppe,

(ii) $(A, \odot)$ ist Halbgruppe,

(iii) *Distributivität*, für alle $a, b, c \in A$ gilt $a \odot (b \oplus c) = a \odot b \oplus a \odot c$ und $(a \oplus b) \odot c = a \odot c \oplus b \odot c$.

Das neutrale Element von $(A, \oplus)$ heißt *Nullelement* des Rings. Der Ring heißt *kommutativ*, falls er bzgl. $\odot$ kommutativ ist, ansonsten *nicht-kommutativ*. Ist $(A, \odot)$ Monoid, so heißt der Ring *unitär*. Das neutrale Element von $(A, \odot)$ heißt dann *Einselement*. Ist $(A, \oplus)$ nur eine abelsche Halbgruppe, so heißt $(A, \oplus, \odot)$ *Halbring*. Ist $(A, \oplus)$ Halbgruppe, aber $(A, \odot)$ Gruppe, so heißt $(A, \oplus, \odot)$ *Halbkörper*.

Definition 1.29. Eine Struktur $(A, \oplus, \odot)$ heißt *Körper*, wenn gilt

(i) $(A, \oplus)$ ist abelsche Gruppe,

(ii) $(A \setminus \{0\}, \odot)$ ist abelsche Gruppe (mit 0 neutralem Element bzgl. $\oplus$),

(iii) Distributivität ist erfüllt.

Ist $(A \setminus \{0\}, \odot)$ nur Gruppe (ohne Kommutativität), so ist $(A, \oplus, \odot)$ *Schiefkörper*.

Bemerkung. Jeder Körper ist ein kommutativer Ring und auch ein Schiefkörper. Jeder Schiefkörper ist ein Ring.

1.7 Graphen

Eine weitere wichtige Klasse von Strukturen sind Graphen. Dise werden häufig genutzt um Operatoren wie Funktionen und Relationen zu visualisieren, allerdings untersucht man Graphen auch selbst als mathematische Objekte.

Definition 1.30. Ein *Graph* ist eine Struktur $G = (V, E)$ mit *Knotenmenge* V und *Kantenmenge* (oder Multimenge) $E \subseteq V \times V$.

Definition 1.31. Sei $G = (V, E)$ ein Graph, $v, w \in V$ und $e = (v, w) \in E$.

- v, w heißen *adjazent* (oder *benachbart*) durch e.
- e ist *inzident* zu v und w.
- Zwei Kanten heißen *inzident*, wenn sie einen geimensamen Endknoten haben.

- e heißt *Schleife* (oder *Loop*), falls $v = w$.
- Zwei Kanten e, e' heißen *parallel*, falls $e = e'$.
- Ein Graph heißt *einfach*, falls er keine parallelen Kanten enthält.
- Ein Graph heißt *ungerichtet*, falls $(v, w) \in E$ impliziert, dass $(w, v) \in E$, ansonsten *gerichtet*.

Für einen ungerichteten Graphen können wir auch $e = \{v, w\}$ schreiben (statt zwei Kanten $(v, w), (w, v)$ zu betrachten. Für einen fixierten Knoten v bezeichnet man die Menge aller Nachbarn von v häufig mit $\Gamma(v)$.

Definition 1.32. Sei $G = (V, E)$ ein Graph und $v \in V$. Der *Knotengrad* von v ist definiert durch

$$\deg(v) = |\{e \in E \mid e = \{v, w\}, w \neq v\}| .$$

Für gerichtete Graphen können wir auch vom *Eingangs-* und *Ausgangsgrad* sprechen:

$$\deg^-(v) = |\{e \in E \mid e = (v, w), w \neq v\}| ,$$
$$\deg^+(v) = |\{e \in E \mid e = (w, v), w \neq v\}| .$$

Definition 1.33. Ein Graph $G = (V, E)$ heißt *k-regulär*, falls für alle $v \in V$ gilt $\deg(v) = k$. G heißt *vollständig*, falls für alle Knotenpaare $v \neq w \in V$ gilt: $(v, w) \in E$. Mit K_n bezeichnen wir den vollständigen Graphen mit n Knoten. Ist der Teilgraph $G[U]$ vollständig, so heißt $U \subseteq V$ *Clique* in G. G heißt *bipartit*, falls sich V als disjunkte Vereinigung aus U, W darstellen lässt, sodass für alle $(u, v) \in E$ entweder $u \in U, v \in W$ oder $u \in W, v \in U$ gilt. Er heißt *vollständig bipartit*, wenn zudem gilt, dass $E = \{(u, w) \mid u \in U, w \in W\}$. Wir schreiben $K_{p,q}$ mit $p = |U|$ und $q = |W|$ für den vollständig bipartiten Graphen mit Knotenmenge $V = U \cup W, U \cap W = \varnothing$.

Bemerkung. Den Graphen $S_n \coloneqq K_{1,n-1}$ nennt man auch *Stern*.

Definition 1.34. Sei $G = (V, E)$ ein Graph. Ein *Teilgraph* von G ist ein Graph $G' = (V', E')$, wenn gilt $V' \subseteq V$ und $E' \subseteq E$. Ein *induzierter Teilgraph* von G ist ein ein Graph $G' = (V', E')$, wenn gilt $V' \subseteq V$ und $E' = E \cap V' \times V'$. Wir schreiben für den durch V' induzierten Teilgraphen von G auch $G[V']$.

Definition 1.35. Sei $G = (V, E)$ ein Graph.

- Ein *Kantenzug* der Länge ℓ ist ein Tupel $(v_0, v_1, \ldots, v_\ell)$ von Knoten mit $(v_i, v_{i+1}) \in E$ für alle $i \in [\ell]$.
- Ein Kantenzug $(v_0, \ldots, v_\ell)$ heißt *Pfad*, falls die Knoten paarweise verschieden sind.
- Ein *Kreis* ist ein Kantenzug $(v_0, \ldots, v_\ell)$, falls $\ell \geq 3$, $(v_0, \ldots, v_{\ell-1})$ ein Pfad ist und $v_0 = v_\ell$.

- Eine *Tour* ist ein Kantenzug $(v_0, \ldots, v_\ell)$, falls die Kanten (v_i, v_{i+1}) für alle $i \in [\ell]$ paarweise verschieden sind und $v_0 = v_\ell$.

Wir nutzen selbsterklärende, abkürzende Schreibweisen, wie (u, v)-Kantenzug und Bezeichnungen wie Anfangs- oder Endknoten bzgl. Kantenzügen oder Pfaden.

Definition 1.36. Sei $G = (V, E)$ ein Graph. G heißt *zusammenhängend*, falls für alle $u, v \in V$ gilt, dass es einen (u, v)-Kantenzug gibt. Ein zusammenhängender Teilgraph $G[U]$ heißt *Zusammenhangskomponente* von G, falls für alle $v \notin U$ der Graph $G[U \cup \{v\}]$ nicht zusammenhängend ist.

Definition 1.37. Ein kreisfreier Graph heißt *Wald*. Ein zusammenhängender Wald heißt *Baum*. Die Knoten eines Waldes mit $\deg(v) \leq 1$ heißen *Blätter*. In einem gerichteten Baum heißt ein ausgezeichneter Knoten $r \in V$ *Wurzel*, wenn $\deg^+(r) = 0$.

1.8 Beweismethoden

Wir sind fast am Ende des Einführungsabschnitts angekommen. Bekanntlich wollen wir in der Mathematik Sätze, d.h. mathematische Eigenschaften auf verschiedenen Strukturen beweisen. Ein Beweis ist dabei eine endliche Folge logischer Schlüsse. Die mathematische Logik gibt dazu genaue Einblicke wie Beweissysteme funktionieren. Im Wesentlichen geht es darum mit einer möglichst kleinen Menge von *Axiomen* (das sind Aussagen, die als wahr angenommen werden, zum Beispiel fordert man, dass jede natürliche Zahl einen Nachfolger hat) Schritt für Schritt die Sätze herzuleiten. Genauer werden wir darauf aber nicht eingehen, wir verwenden vielmehr die Verfahren, die sich Logiker über die Jahrhunderte überlegt haben. Wir unterscheiden zwischen vier verschiedenen Beweisarten: Deduktion, Widerspruchsbeweis, Beweis der Kontraposition und Beweis durch vollständige Induktion. Dabei sei gesagt, dass diese Abgrenzung nicht immer eindeutig ist. Diese vier Methoden lassen sich (zum Beispiel in Beweisen größerer Theoreme) auch kombinieren. Gelegentlich liefern Beweise auch eine Konstruktion mit dem die Aussage klar wird. Wir starten mit der Deduktion oder dem direktem Beweis. Wir betrachten dafür Aussagen der Form:

Wenn Aussagen A gilt, dann auch Aussage B.

Zum Beweis solcher Aussagen nehmen wir an, dass A tatsächlich wahr ist und verwenden sogenannte *Schlussregeln*, dass dann auch B wahr sein muss. Wir zeigen dies an einem Beispiel.

Satz 1.38. *Seien $a, b \in \mathbb{Z}$ Falls a gerade ist, so ist auch ab gerade.*

Beweis. Da a gerade ist existiert ein $k \in \mathbb{Z}$ mit $2 \cdot k = a$. Dann ist aber $a \cdot b = (2 \cdot k) \cdot b = 2 \cdot \underbrace{(k \cdot b)}_{\in \mathbb{Z}}$ und somit ist $a \cdot b$ gerade. □

Der zweite Weg, der Widerspruchsbeweis, ist ein indirekter Beweis (wie auch die Kontraposition später). Das Schema diesmal ist

Angenommen Aussage A wäre falsch, dann folgt Unsinn.

Im Lateinischen spricht man auch von *reductio ad absurdum* (Zurückführung auf das Sinnlose). Man zeigt damit, dass wenn A nicht gilt, dass dann eine bereits andere Aussage (von der man den Wahrheitsgehalt kennt) nicht stimmen könnte. Mit diesem Widerspruch kann man dann folgern, dass A doch wahr ist. Wir machen wieder ein Beispiel.

Satz 1.39. *Es gibt unendlich viele Primzahlen.*

Beweis. Angenommen es gäbe nur endlich viele Primzahlen. Sei dann $P := \{p_0, \ldots, p_\ell\}$ die Menge aller vielen Primzahlen, d.h. $\ell \in \mathbb{N}$. Wir definieren nun die Zahl

$$q := 1 + \prod_{i=0}^{\ell} p_i.$$

Wir untersuchen zwei Fälle:

- q prim. Dann war P jedoch noch nicht die Menge aller Primzahlen wie oben agenommen, denn $q > p$ für alle $p \in P$. $\lightning$
- q nicht prim. Dann existiert eine eindeutige Primfaktorzerlegung von q. Jedoch lässt jede Primzahl $p \in P$ einen Rest von 1 bei ganzzahliger Division. Also muss es noch mehr Primzahlen geben als die in P. $\lightning$

In beiden Fällen erhalten wir also einen Widerspruch zur ursprünglichen Aussage, dass ein endliches P alle Primzahlen enthalten würde. Also muss es unendlich viele Primzahlen geben. $\square$

Eine weitere Form des indirekten Beweises ist es die *Kontraposition* der Aussage zu zeigen. D.h. statt wie beim direkten Beweis aus Aussage A Aussage B zu folgern zeigt man, dass aus der Negation von B die Negation von A folgt.

Satz 1.40. *Sei a^2 gerade, dann ist auch a gerade.*

Beweis. Angenommen a wäre ungerade, d.h. es gibt ein $k \in \mathbb{Z}$ mit $a = 2k + 1$. Betrachte dann

$$\begin{aligned} a^2 &= (2k+1)^2 \\ &= 4k^2 + 4k + 1 \\ &= 2\underbrace{(2k^2 + 2k)}_{\in \mathbb{Z}} + 1. \end{aligned}$$

Also ist a^2 ebenfalls ungerade. Es folgt vermöge Kontraposition, dass für gerade a^2 auch a gerade ist. $\square$

Zuletzt betrachten wir das Beweisprinzip der vollständigen Induktion. Hiermit lassen sich leicht Aussagen für natürliche Zahlen zeigen. Man zeigt dafür zunächst, dass eine Aussage für einen Basisfall erfüllt ist (in dieser Vorlesung gewöhnlich für $n = 0$). Danach beweist man, dass die Aussage, sofern sie für ein $n \in \mathbb{N}$ erfüllt ist, dass sie dann auch für $n + 1$ erfüllt ist. Dieses Beweisschema beruht auf dem Induktionsprinzip der natürlichen Zahlen, welches besagt: Für jede Teilmenge $M \subseteq \mathbb{N}$ gilt: Falls $0 \in M$ und für jedes $n \in M$ auch $n + 1 \in M$, dann ist $M = \mathbb{N}$. Wollen wir also eine Aussage A für alle $n \in \mathbb{N}$ beweisen, zeigen wir also lediglich, dass die Menge $M \coloneqq \{n \in \mathbb{N} \mid A \text{ ist wahr für } n\}$ glich $\mathbb{N}$ ist.

Satz 1.41. *Für jede endliche Menge M gilt $|2^M| = 2^{|M|}$.*

Beweis. Wir machen eine Induktion über die Größe von M.
Induktionsverankerung: $|M| = 0$, d.h. $M = \varnothing$. Es gilt $|2^M| = |2^\varnothing| = |\{\varnothing\}| = 1$. ✓
Induktionshypothese (IH): Angenommen die Aussage gelte für ein M der Größe $n \in \mathbb{N}$.
Induktionsschritt: Sei $M' \coloneqq M \cup \{a\}$ wobei a ein beliebiges Element ist, welches nicht in M vorkommt. Die Menge wird also um ein Element größer. Dann erhalten wir

$$\begin{aligned}
\left|2^{M\cup\{a\}}\right| &= \Bigg| \overbrace{2^M}^{\text{Teilmengen ohne } a} \cup \overbrace{\{A \cup \{a\} : A \subseteq M\}}^{\text{Teilmengen mit } a} \Bigg| \\
&= |2^M| + |\{A \cup \{a\} : A \in 2^M\}| \\
&= |2^M| + |2^M| \\
&= 2 \cdot |2^M| \\
&\overset{\text{IH}}{=} 2 \cdot 2^{|M|} \\
&= 2^{|M|+1} \\
&= 2^{|M\cup\{a\}|}.
\end{aligned}$$

□

2 Alphabete, Wörter, Sprachen

Das erste Kapitel hat uns mit den nötigen mathematischen Grundlagen versorgt, die wir als Modellierungswerkzeuge in der theoretischen Informatik verwenden wollen. Wir definieren dazu später abstrakte Rechenmodelle, sogenannte Automaten, um das Verhalten von konkreten Rechenmodellen (z.B. Computern) formal zu erfassen. Zunächst sehen wir uns an, wie wir ganz allgemein diese konkreten Rechenmodelle funktionieren und wie sich diese Funktionsweisen möglichst knapp und allgemein (d.h. abstrakt, "vereinfacht auf das wesentliche") darstellen lassen. Nach diesem Kapitel haben wir das Hauptthema der Vorlesung, die *abstrakte Automatentheorie*, vorbereitet.

2.1 Grundlegende Definitionen

Wir fassen Aktionen eines Computers (oder eines Getränkeautomaten, …) in unserer Abstrakion als *Symbole* (Buchstaben) auf, im Rahmen dieser Vorlesung sind das immer nur endlich viele, d.h. das *Alphabet* ist endlich. Aktionsfolgen (z.B. vom Einwurf einer Münze bis zur Ausgabe des Getränkes) entsprechen somit einem *Wort*. Die Menge aller gültigen Aktionsfolgen (solche, die für das betrachtete System "sinnvoll" sind) bezeichnen wir dann als *Sprache des Automaten*.

Definition 2.1. Ein *Alphabet* ist eine nicht-leere endliche Menge, deren Elemente als *Symbole* bezeichnet sind.

Alphabete werden durch griechische Großbuchstaben Σ, Γ oder Variationen wie Σ_1, Γ' bezeichnet. Symbole werden durch kleine lateinische Buchstaben $a, b, c, \ldots$ oder arabische Ziffern $0, 1, \ldots$ bezeichnet.

Beispiel 2.2.

(i) Das *Boole'sche Alphabet* $\{0, 1\}$.

(ii) Das *Morsealphabet* $\{\cdot, -, \ \}$.

(iii) Das *ASCII-Alphabet* für zum Beispiel Textdateien.

Definition 2.3. Ein *Wort* über einem Alphabet Σ ist eine Abbildung

$$w\colon [n] \to \Sigma.$$

Für $n = 0$ ist $w\colon \varnothing \to \Sigma$ das *leere Wort*, was wir als ε bezeichnen. Die *Länge* des Wortes w ist bezeichnet mit $|w| = n$. Mit $|w|_a \coloneqq |\{i \in [n] \mid w(i) = a\}|$ bezeichnen wir die *Häufigkeit des Symbols a* im Wort w.

Wie in Abschnitt 1.5 lässt sich w auch als Tupel $(a_0, \ldots, a_{n-1})$ schreiben. Wir gehen hier sogar noch einen Schritt weiter und benutzen $a_0 a_1 \ldots a_{n-1}$ als Abkürzung für die langen Schreibweisen. Für Wörter verwenden wir in der Regel u, v, w und Varianten als Bezeichner. In der Literatur sind auch kleine griechische Buchstaben $\alpha, \beta, \ldots$ gebräuchlich.

Definition 2.4. Sei Σ ein Alphabet. Dann ist $\Sigma^n := \Sigma^{[n]}$ die *Menge aller Wörter mit Länge n* über Σ. Die *Menge aller Wörter* ist definiert als

$$\Sigma^* := \bigcup_{n \in \mathbb{N}} \Sigma^n$$

und $\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$.

Wie in Abschnitt 1.2 angekündigt wird für ein fixiertes Σ die Menge Σ^* unser Universum sein.

Definition 2.5. Eine *(formale) Sprache* über einem Alphabet Σ ist eine Teilmenge von Σ^*.

Sprachen bezeichnen wir in der Regel mit $L, K, \ldots$ und Varianten.

Beispiel 2.6.

- Die leere Sprache $\varnothing$.
- Die Sprache, die nur das leere Wort enthält $\{\varepsilon\}$.
- Die Sprache aller Binärdarstellungen von Primzahlen $\{\mathrm{bin}(n) : n \in \mathbb{P}\}$.
- Die Menge aller grammatikalisch korrekten deutschen Sätze.

2.2 Operationen und Relationen auf Wörtern und Sprachen

Durch diese vorgegangen Definitionen haben wir das Fundament für die theoretische Informatik bereits definiert. Da dies nur mithilfe von Funktionen und Mengen passiert ist lassen sich Beweismethoden und Ergebnisse aus der Mathematik einfach übertragen. Wir definieren nun noch ein paar Operationen auf Wörtern und erweitern diese Definitionen auf Sprachen.

Definition 2.7. Seien $u, v \in \Sigma^*$ mit $m = |u|, n = |v|$. Die *Konkatenation* (Verkettung) ist definiert als

$$(u \cdot v) \colon [m{+}n] \to \Sigma \text{ mit}$$
$$(u \cdot v)(i) = \begin{cases} u(i), & i < m \\ v(i-m), & i \geq m. \end{cases}$$

Außerdem ist $u^0 := \varepsilon$ und $u^n := u \cdot u^{n-1}$.

Aus Bequemlichkeitsgründen wird der Punkt auch weggelassen. Für Sprachen erhalten wir noch die Definitionen.

Definition 2.8. Seien $L, K \subseteq \Sigma^*$ Sprachen.

(i) *Konkatenation.* $L \cdot K := \{uv : u \in L, v \in K\}$ und $L^0 := \{\varepsilon\}, L^n := L \cdot L^{n-1}$.

(ii) *Inklusion.* Wie in Definition 1.2.

(iii) *Vereinigung, Schnitt, Komplement, Differenz.* Wie in Definiton 1.4.

(iv) *Kleene'scher Abschluss.* (auch *Iteration*, *Kleene-Stern*)

$$L^* := \bigcup_{n \in \mathbb{N}} L^n.$$

Definition 2.9. Seien u, v Wörter. Dann ist u

- *Präfix* von v (geschrieben: $u \sqsubseteq v$), falls es ein Wort w gibt mit $v = uw$,
- *Infix* von v, falls es Wörter w, w' gibt mit $v = wuw'$,
- *Suffix* von v, falls es ein Wort w gibt mit $v = wu$.

Beispiel 2.10. Sei $v = aaba$.

- Die Präfixe von v sind $\varepsilon, a, aa, aab, aaba$.
- Die Suffixe von v sind $\varepsilon, a, ba, aba, aaba$.
- Die Infixe von v sind alle Präfixe und Suffixe, sowie ab, b.

Definition 2.11. Sei Σ ein durch $\prec$ geordnetes, endliches Alphabet. Die *lexikographische Ordnung* $\prec_{\text{lex}}$ auf Wörtern $u = a_0 \ldots a_{n-1}, v = b_0 \ldots b_{m-1}$, über Σ ist definiert durch:

$$\begin{aligned} u \prec_{\text{lex}} v \text{ g.d.w. } & u \sqsubseteq v \\ & \text{oder es gibt ein } k, \text{ s.d. für alle } i < k \text{ gilt } a_i = b_i \text{ und } a_k \prec b_k. \end{aligned}$$

Gewöhnlich betrachten wir ja eine Teilmenge des lateinischen Alphabets $\Sigma_{\text{lat}} = \{a, b, \ldots, z\}$. Wir nehmen im Folgenden immer die Ordnung $a \prec b \prec \ldots \prec z$ an.

Beispiel 2.12. Sei $\Sigma = \{a, b, c\}$.

1) $abc \prec_{\text{lex}} abcc$,
2) $aaaa \prec_{\text{lex}} ab$,
3) $\varepsilon \prec_{\text{lex}} u$ für alle $u \in \Sigma^*$.

2.3 Gesetze für Wörter und Sprachen

In diesem Abschnitt wollen wir einige Gesetzmäßigkeiten, die bei der Anwendung von Operationen auf Wörtern und Sprachen gelten, herausarbeiten. Einige Eigenschaften übertragen sich sofort aus denen für Mengen aus Abschnitt 1.4. Bei anderen ist etwas mehr zu zeigen und bei wieder anderen gibt es vielleicht auch zunächst unintuitive Unterschiede.

Bemerkung. Für das leere Wort ε gilt:

(i) Es ist für jedes Wort sowohl Präfix, Infix als auch Suffix.

(ii) Es ist das *neutrale Element* der Konkatenation, d.h. für alle $w \in \Sigma^*$ gilt $\varepsilon w = w = w\varepsilon$.

(iii) Daran anknüpfend gilt für jede Sprache L, dass $\{\varepsilon\}L = L = L\{\varepsilon\}$.

(iv) Für jede Menge A (inklusive dem Fall $A = \varnothing$) ist $A^0 = \{\varepsilon\}$.

(v) ε ist eindeutig, d.h. es gibt kein zweites Wort mit diesen Eigenschaften.

(vi) Weil es häufig durcheinander gebracht wird: $\{\varepsilon\} \neq \varnothing$.

Bemerkung. Für Vereinigung, Schnitt, Differenz, ... von Sprachen gelten die selben Regeln (Assoziativ-, Kommutativ-, Distributivgesetze, deMorgan, Absorption, ...) wie für Mengen.

Bemerkung. Für jede Sprache L gilt, dass $\varnothing L = L\varnothing = \varnothing$.

Beweis. Wir zeigen, dass $L\varnothing$ leer ist. Der andere Fall geht analog. Angenommen es existiert ein $w \in L\varnothing$. Dann lässt sich w zerlegen in $w = uv$ mit $u \in L, v \in \varnothing$. Da ein solches v nicht existieren kann (leere Menge), kann auch die gesamte Zerlegung und somit auch w nicht existieren. Also ist $L\varnothing$ leer. □

3 Endliche Automaten und Reguläre Sprachen

Wir haben formale Sprachen eingeführt als Modellierung für Prozessabläufe auf z.B. Computern. Diese Ansicht werden wir zunächst aber beiseite legen und erst in Kapitel 6 wieder aufgreifen. In der Zwischenzeit untersuchen wir Sprachen auf verschiedene Eigenschaften und klassifizieren unter anderem nach der sogenannten *Chomsky-Hierarchie*. Wir prüfen, wie sich Sprachen von formalen Systemen (Automaten, abstrakte Rechenmodelle) erkennen lassen. Diese Systeme wirken manchmal etwas künstlich, sind aber sehr sinnvoll, da sie sich mit den uns zu Verfügungen Werkzeugen aus der Mathematik gut handhaben lassen. Die daraus entstehenden Resultate haben außerdem auch einen nicht zu vernachlässigenden ästhetischen Wert für die theoretische Informatik. Zugegeben, der Praxisbezug offenbart sich bei einigen Sätzen nicht sofort und ist vielleicht auch gar nicht überalll vorhanden. Dennoch sollte der Wert dieser Ergebnisse auch nicht unterschätzt werden, denn wir liefern hier auch die Grundlagen zur Untersuchung, was sich prinzipiell mit Computern überhaupt berechnen lässt und die Erkenntnis, dass es Probleme in der Informatik gibt, die von einem Computer mehr Funktionalität beansprucht als andere Probleme (und vielleicht sogar mehr als ein Computer prinzipiell haben kann), sollte Motivation genug sein, sich mit theoretischer Informatik auseinanderzusetzen.

3.1 Deterministische Endliche Automaten

Wir beginnen mit der Art Automaten, die “die einfachste“ Klasse formaler Sprachen erkennen kann.

Definition 3.1. Ein *deterministischer endlicher Automat (DFA)* (von engl.: deterministic finite automaton) ist ein 5-Tupel

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

mit

- Q eine nicht-leere, endliche Menge von *Zuständen*,
- Σ ein nicht-leeres, endliches *Eingabealphabet*,
- $\delta\colon Q \times \Sigma \to Q$ die *Transitionsfunktion*,
- $q_0 \in Q$ der *Startzustand*,
- $F \subseteq Q$ die Menge der *akzeptierenden Zustände* (oder *Endzustände*).

DFAs lassen sich auch problemlos als Strukturen wie in Abschnitt 1.6 auffassen. Aus Gründen der Lesbarkeit verzichten wir jedoch darauf. Wir bezeichnen DFAs stets mit $\mathcal{A}, \mathcal{B}, \ldots$, Zustände mit p, q, r, s und Variationen.
Es lassen sich auch durchaus noch einfacherere Rechenmodelle definieren (z.B. durch Restriktionen gegenüber der Größe der Zustandsmenge), dies ist jedoch vorerst nicht sinnvoll. Auf die bereits angesprochene Chomsky-Hierarchie werden wir noch genauer eingehen,

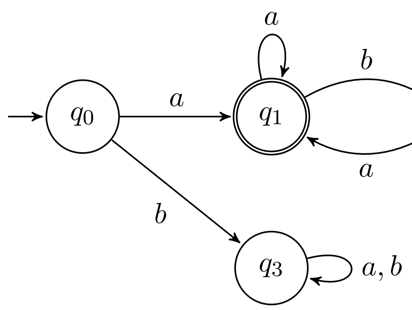


Abbildung 3: DFA für die Sprache aus Beispiel 3.5.

aber auch dort sind die Sprachen, die durch DFAs erkannt werden können, als die einfachste Klasse bezeichnet.
Möchte man einen DFA konkret angeben, so ist die Darstellung als Transitionsgraph sinnvoller, als die Angabe des 5-Tupels.

Definition 3.2. Sei $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ ein DFA. Der *Transitionsgraph* von $\mathcal{A}$ ein beschrifteter Graph $G_{\mathcal{A}} = ((Q, E), \lambda, q_0, F)$ mit

$$E = \{(p, q) \mid \text{es ex. } a \in \Sigma \text{ mit } \delta(p, a) = q\}$$

und

$$\lambda \colon E \to 2^{\Sigma}, \quad (p, q) \mapsto \{a \mid \delta(p, a) = q\}.$$

Aus Gründen der Bequemlichkeit, lassen wir die Mengenklammern bei der Beschriftung der Transitionen weg. Der Startzustand q_0 bekommt einfach eine eingehende Kante ohne Beschriftung und ohne Startknoten. Die Endzustände werden zusätzlich eingekreist. Ein Beispieltransitionsgraph ist in Abbildung 3. Wir werden die Begriffe Automat und Transitionsgraph gelegentlich synonym verwenden, da sich sowohl im Graphen als auch in der ursprünglichen Automatenstruktur alle Informationen befinden. Wir greifen dann auf den Begriff zurück, der für das aktuelle Thema die bequemere Anschauung hat.
Wir schauen uns nun das Verhalten eines DFA auf einem Wort an.

Definition 3.3. Sei $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ ein DFA. Ein *Lauf* von $\mathcal{A}$ auf einem Wort $w = a_0 \dots a_{n-1}$ für ein $n \in \mathbb{N}$ ist eine endliche Folge

$$(r_0, a_0, r_1, a_1, \dots, a_{n-1}, r_n),$$

wobei $r_0, \dots, r_n \in Q$ und $a_0, \dots, a_{n-1} \in \Sigma$, sodass

(i) $r_0 = q_0$,

(ii) $\delta(r_i, a_i) = r_{i+1}$ für $i \in [n]$.

Wir sagen ein Lauf ist *akzeptierend*, wenn zusätzlich $r_n \in F$ gilt.

Wir bezeichnen Läufe in der Regel mit ϱ bzw. Variationen. Einen Lauf $(r_0, a_0, r_1, a_1, \dots, a_{n-1}, r_n)$ kürzen wir gelegentlich durch $(r_0, r_1, \dots, r_n)$ ab, wenn die Symbole nicht relevant für unsere Betrachtungen sind.

Bemerkung. Zu jedem Wort $w \in \Sigma^*$ existiert genau ein Lauf von $\mathcal{A}$ auf w.

Definition 3.4.

(i) Ein DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *akzeptiert* ein Wort $w \in \Sigma^*$, wenn der Lauf von $\mathcal{A}$ akzeptierend ist. Andernfalls *verwirft* $\mathcal{A}$ das Wort w.

(ii) Die von einem DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *erkannte Sprache* ist

$$L(\mathcal{A}) \coloneqq \{w \in \Sigma^* \mid \mathcal{A} \text{ akzeptiert } w\}.$$

(iii) Eine Sprache L heißt *DFA-erkennbar*, wenn es einen DFA $\mathcal{A}$ gibt, sodass $L = L(\mathcal{A})$.

Beispiel 3.5. Betrachte erneut den Automaten $\mathcal{A}$ in Abbildung 3.

- Sei $w_1 = abaaba$. Der Lauf von $\mathcal{A}$ auf w_1 ist

  $$\varrho_1 = (q_0, q_1, q_2, q_1, q_1, q_2, q_1).$$

  D.h. $\mathcal{A}$ akzeptiert w_1.

- Sei $w_2 = baa$. Der Lauf von $\mathcal{A}$ auf w_2 ist

  $$\varrho_2 = (q_0, q_3, q_3, q_3).$$

  D.h. $\mathcal{A}$ verwirft w_2.

- $\mathcal{A}$ erkennt die Sprache

  $$L = \{w \in \{a, b\}^* \mid w \text{ beginnt und endet mit } a\}.$$

Die letzte Aussage sagt etwas über das Verhalten des Automaten auf allen, d.h. unendlich vielen, Wörtern aus. Man kann also nicht für jedes Wort einzeln zeigen, dass sich der Automat korrekt verhält. Stattdessen beweisen wir die Aussage per Induktion.

Beweis. Wir zeigen die folgende Aussage:

$$\mathcal{A} \text{ akzeptiert das Wort } w \text{ g.d.w. } w \text{ beginnt und endet mit } a.$$

Beweis per vollständige Induktion über Wortlänge $n \in \mathbb{N}$. Ist $r = (r_0, \dots, r_n)$ der Lauf auf dem Wort $w = a_0 \dots a_{n-1}$, so ist

$$r_n = \begin{cases} q_0, & w = \varepsilon \\ q_1, & a_0 = a_{n-1} = a \\ q_2, & a_0 = a \text{ und } a_{n-1} = b \\ q_3, & a_0 = b. \end{cases}$$

Wir wollen also zeigen, dass ein Lauf von $\mathcal{A}$ auf einem Wort dann und nur dann in q_1, dem einzigen akzeptierendem Zustand, endet, wenn w mit a beginnt und endet.
Induktionsverankerung: $n = 0$. Dann ist $w = \varepsilon$ und der Lauf von $\mathcal{A}$ auf w ist $r = (q_0)$, also auch $r_n = r_0 = q_0$.✓
Induktionshypothese (IH): Für $w = a_0 \dots a_{n-1}$ sei $r = (r_0, \dots, r_n)$ der Lauf auf $\mathcal{A}$ mit r_n wie in der Behauptung.
Induktionsschritt: Betrachte nun das Wort $w = a_0 \dots a_n$.

- $a_0 \dots a_{n-1} = \varepsilon$. Dann ist nach IH $r_n = q_0$ und somit

$$r_{n+1} = \delta(r_n, a_n) = \begin{cases} q_1, & a_n = a \\ q_3, & a_n = b. \end{cases}$$

- $a_0 = a_{n-1} = a$. Dann ist nach IH $r_n = q_1$ und somit

$$r_{n+1} = \delta(r_n, a_n) = \begin{cases} q_1, & a_n = a \\ q_2, & a_n = b. \end{cases}$$

- $a_0 = a$ und $a_{n-1} = b$. Dann ist nach IH $r_n = q_2$ und somit

$$r_{n+1} = \delta(r_n, a_n) = \begin{cases} q_1, & a_n = a \\ q_2, & a_n = b. \end{cases}$$

- $a_0 = b$. Dann ist nach IH $r_n = q_3$ und somit

$$r_{n+1} = \delta(r_n, a_n) = q_3$$

  unabhängig von a_n.

Insgesamt gilt also:

$$\begin{aligned} w \text{ beginnt und endet mit } a. &\text{ g.d.w. Ist } r = (r_0, \dots, r_n) \text{ der Lauf von} \\ &\qquad \mathcal{A} \text{ auf } w \text{ so gilt } r_n = q_1 \in F. \\ &\text{ g.d.w. } \mathcal{A} \text{ akzeptiert } w. \\ &\text{ g.d.w. } L(\mathcal{A}) = L. \end{aligned}$$

□

So ausführlich wie hier werden wir später nicht mehr beweisen, dass ein gegebener Automat “das richtige tut“, sollte dies nicht klar sein werden wir die Funktionsweise nur grob erläutern. Ausführliche Beweise sind im Wesentlichen dann gefordert, wenn man zum Beispiel zeigen möchte, dass eine Sprache nicht DFA-erkennbar ist.

3.2 Abschlusseigenschaften DFA-erkennbarer Sprachen

Bei einer Einteilung aller möglichen Sprachen in Klassen will man in einer möglichst sinnvollen Weise vorgehen. Damit meint man u.a., dass die einzelnen Klassen in sich abgeschlossen sind bzgl. verschiedener Operationen oder auch andere Eigenschaften haben, die in einer wissenschaftlichen Weise "schön" sind. Die folgenden Abschnitte sind dazu da um zu zeigen, dass DFA-erkennbare Sprachen dies in vielerlei Hinsicht erfüllen. In diesem Abschnitt beginnen wir damit zu zeigen, dass DFA-erkennbare Sprachen unter den üblichen Mengenoperationen (Komplementbildung, Vereinigung und Schnitt) abgeschlossen sind. Im späteren Verlauf werden wir noch einige andere Operationen betrachten.
Wir zeigen als erstes, dass die DFA-erkennbaren Sprachen unter Komplementbildung abgeschlossen sind.

Satz 3.6. *Sei $L \subseteq \Sigma^*$ eine DFA-erkennbare Sprache. Dann ist auch $\overline{L}$ DFA-erkennbar.*

Beweis. Sei $L \subseteq \Sigma^*$ eine beliebige DFA-erkennbare Sprache. D.h. es existiert ein DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, der L erkennt. Wir konstruieren aus $\mathcal{A}$ den DFA

$$\overline{\mathcal{A}} = (Q, \Sigma, \delta, q_0, Q \setminus F),$$

welcher die Sprache $\overline{L}$ erkennt. Die Konstruktion vertauscht also lediglich akzeptierende und nicht-akzeptierende Zustände. Wir müssen nun noch zeigen, dass diese Konstruktion korrekt ist, also formal, dass $\overline{L(\mathcal{A})} = L(\overline{\mathcal{A}})$ gilt. Dazu zeigen wir folgende Aussage, aus der offensichtlich die Behauptung folgt.

$$\text{Für alle } w \in \Sigma^* \text{ gilt: } \mathcal{A} \text{ akzeptiert } w \text{ g.d.w. } \overline{\mathcal{A}} \text{ verwirft } w.$$

Sei also $w \in \Sigma^*$. Da $\mathcal{A}$ und $\overline{\mathcal{A}}$ den selben Startzustand q_0 und die selbe Transitionsfunktion δ haben, haben beide Automaten den selben (eindeutigen) Lauf $(r_0, \ldots, r_n)$ auf w. $\mathcal{A}$ akzeptiert w falls $r_n \in F$. Dann gilt $r_n \notin Q \setminus F$ und somit $\overline{\mathcal{A}}$ verwirft w. Die andere Richtung geht analog. Also gilt die Behauptung. □

Diese Abschlusseigenschaft war einfach zu beweisen, da wir nur eine kleine Änderung am ursprünglichen DFA machen mussten und dann wieder einfach über den eindeutigen Lauf argumentieren konnten. Die Idee hinter der Konstruktion ist auch sehr intuitiv, da wir genau die Wörter akzeptieren wollen, die vom ursprünglichen Automaten verworfen wurden, also deren Läufe in nicht-akzeptierenden Zuständen enden. Der Ansatz die Zustände einfach zu vertauschen drängt sich also geradezu auf.
Für den Abschluss unter Vereinigung ist etwas mehr Arbeit zu machen. Wir haben also nun zwei DFA-erkennbare Sprachen (und damit die zugehörigen Automaten) und wollen nun prüfen ob ein Wort in wenigstens einer der beiden Sprachen liegt. Wir müssen nun also einen Automaten konstruieren, der zwei gegebene Automaten simuliert. Diese Simulation muss synchron bzw. parallel stattfinden, eine sequentielle (d.h. Hintereinander-)Ausführung beider Automaten ist nicht möglich (da DFAs bereits gelesene Symbole "vergessen", ein explizites "Abspeichern" ist nur möglich für konstant viele Symbole – das reicht nicht für beliebig große Eingabelängen). Wir präsentieren für die parallele Ausführung nun die Produktkonstruktion im Rahmen des nächsten Satzes.

Satz 3.7. *Seien $L_1, L_2 \subseteq \Sigma^*$ DFA-erkennbare Sprachen. Dann ist auch $L_1 \cap L_2$ DFA-erkennbar.*

Beweis. Seien $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_0^1, F_1)$ und $\mathcal{A}_1 = (Q_2, \Sigma, \delta_2, q_0^2, F_2)$ die DFAs für L_1, L_2. Wir betrachten den *Produktautomaten*

$$\mathcal{A} \coloneqq (Q_1 \times Q_2, \Sigma, \delta, (q_0^1, q_0^2), F)$$

mit $\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$ für alle $p \in Q_1, q \in Q_2, a \in \Sigma$ und $F = F_1 \times F_2$.
Wir zeigen nun, dass $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ ist. Sei dazu $w = a_0 \dots a_{n-1}$ gegeben. Wir zeigen, dass $\mathcal{A}$ akzeptiert w g.d.w. $\mathcal{A}_1$ und $\mathcal{A}_2$ das Wort w akzeptiert.
"Wenn $\mathcal{A}_1$ und $\mathcal{A}_2$ akzeptiert, dann akzeptiert $\mathcal{A}$": Es gibt also einen Lauf $(r_0, \dots, r_n)$ mit $r_n \in F_1$ von $\mathcal{A}_1$ und einen Lauf $(p_0, \dots, p_n)$ mit $p_n \in F_2$ von $\mathcal{A}_2$ jeweils auf w. Dann ist der Lauf auf auf $\mathcal{A}$

$$((r_0, p_0), \dots, (r_n, p_n)),$$

da in jedem Schritt

$$\begin{aligned} \delta((r_i, p_i), a_i) &= (\delta_1(r_i, a_i), \delta_2(p_i, a_i)) \\ &= (r_{i+1}, p_{i+1}). \end{aligned}$$

Wegen $r_n \in F_1$ und $p_n \in F_2$ ist auch $(r_n, p_n) \in F_1 \times P_2 = F$. Also akzeptiert $\mathcal{A}$.
"Wenn $\mathcal{A}$ akzeptiert, dann akzeptieren $\mathcal{A}_1$ und $\mathcal{A}_2$": Wir zeigen die Kontraposition dieser Aussage. $\mathcal{A}_1$ oder $\mathcal{A}_2$ verwirft. Die Läufe von $\mathcal{A}_1, \mathcal{A}_2$ sind also $(r_0, \dots, r_n)$ und $(p_0, \dots, p_n)$ mit $r_n \notin F_1$ oder $p_n \notin F_2$. Wie oben ist $((r_0, p_0), \dots, (r_n, p_n))$ der Lauf von $\mathcal{A}$ und es gilt $(r_n, p_n) \notin F_1 \times F_2 = F$. Somit verwirft auch $\mathcal{A}$ das Wort. □

Eine Beispielkonstruktion für einen Produktautomaten für den Schnitt zweier DFA-erkennbarer Sprachen befindet sich in Abbildung 4. Aus den Sätzen 3.6 und 3.7 erhält man nun einfach alle übrigen Abschlusseigenschaften für die üblichen Mengenoperationen, aber auch mit der Produktkonstruktion lassen sich diese Eigenschaften zeigen.

Korollar 3.8. *Seien L_1, L_2 DFA-erkennbare Sprachen. Dann sind auch die Sprachen $L_1 \cup L_2, L_1 \setminus L_2$ DFA-erkannbar.*

Beweis. Wegen Satz 3.6 sind auch $\overline{L_1}, \overline{L_2}$ DFA-erkennbar und damit nach Satz 3.7 $\overline{L_1} \cap \overline{L_2}$. Mit erneuter Anwendung von Satz 3.6 ist auch $\overline{\overline{L_1} \cap \overline{L_1}}$ DFA-erkennbar, was nach den DeMorgan'schen Gesetzen $L_1 \cup L_2$ entspricht. Alternativ kann man in der Produktkonstruktion auch $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$ setzen und erhält einen DFA für $L_1 \cup L_2$.
Für $L_1 \setminus L_2$ setzt man $F = F_1 \times (Q_2 \setminus F_2)$ und erhält einen DFA für die Differenz. □

Die Produktkonstruktion ist eine sehr grundlegende Konstruktion, die in ähnlicher Form immer wieder Anwendungen findet. Wir können auch eigene Sprachoperatoren erfinden und DFA-erkennbare Sprachen daraufhin untersuchen ob sie abgeschlossen sind bzgl. dieser Operationen.

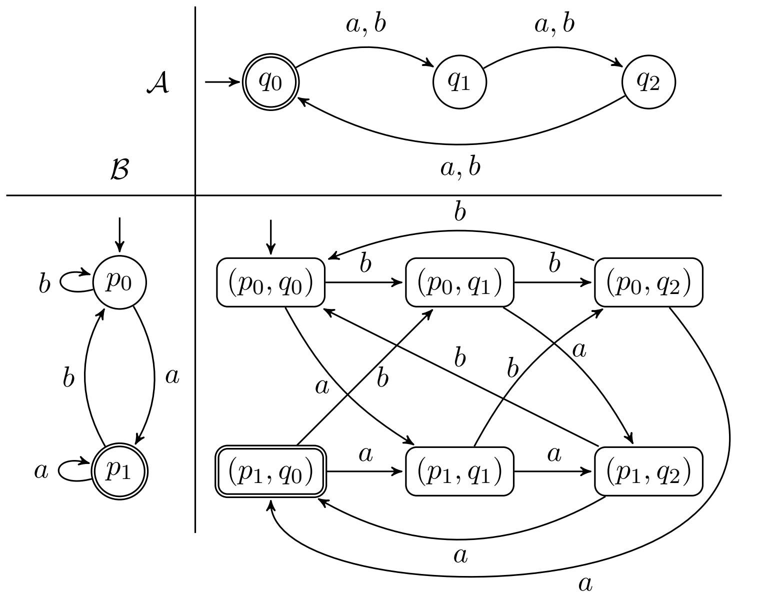


Abbildung 4: Produktkonstruktion: Der Automat $\mathcal{A}$ erkennt die Sprache $\{w \in \{a,b\}^* \mid |w|_a + |w|_b \text{ teilbar durch } 3\}$, $\mathcal{B}$ erkennt die Sprache $\{ua : u \in \{a,b\}^*\}$. Der Produktautomat erkennt den Schnitt.

Beispiel 3.9. Seien $L, K \subseteq \Sigma^*$. Wir definieren die Operation *Perfect Shuffle* wie folgt:

$$L \shuffle K := \{a_0 b_0 \dots a_{n-1} b_{n-1} : a_0 \dots a_{n-1} \in L, b_0 \dots b_{n-1} \in K\}.$$

Satz 3.10. *Seien $L_1, L_2 \subseteq \Sigma^*$ DFA-erkennbare Sprachen. Dann ist auch $L_1 \shuffle L_2$ DFA-erkennbar.*

Beweis. Seien $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_0^1, F_1), \mathcal{A}_2 = (Q_2, \Sigma, \delta_2, q_0^2, F_2)$ die DFAs für L_1, L_2. Wir definieren wieder einen Produktautomaten

$$\mathcal{A} = (Q_1 \times Q_2 \times [2], \Sigma, \delta, (q_0^1, q_0^2, 0), F)$$

mit

$$\delta((p,q,i),a) = \begin{cases} (\delta_1(p,a), q, 1), & i = 0 \\ (p, \delta_2(q,a), 0), & i = 1 \end{cases}$$

und $F = F_1 \times F_2 \times \{0\}$. Die Idee ist diesmal nur eine *quasi-parallele* Simulation von $\mathcal{A}_1$ und $\mathcal{A}_2$. Die dritte Komponente des Zustands gibt an in welchem Automat $\mathcal{A}$ den nächsten Schritt simulieren soll. Wir akzeptieren, wenn beide Simulationen in einem Endzustand sind und der zuletzt durchgeführte Simulationsschritt auf $\mathcal{A}_2$ war. Wir zeigen nun noch, dass die Konstruktion funktioniert.
"$L(\mathcal{A}) \subseteq L_1 \shuffle L_2$": Sei $w = c_0 \dots c_{n-1} \in L(\mathcal{A})$. D.h. es existiert ein Lauf

$$\varrho = ((p_0, q_0, i_0), \dots, (p_n, q_n, i_n))$$

von $\mathcal{A}$ auf w mit $p_n \in F_1, q_n \in F_2$ und $i_j = j \mod 2$ und $i_0 = i_n = 0$ (insbesondere ist $|w|$ gerade), wobei abwechselnd in jedem Schritt j die $1 + i_j$te Komponenten gleich bleibt (s. Definition von δ). Aus den geraden Positionen in ϱ (die mit der dritten Komponente 0) ergeben dann einen Lauf von $\mathcal{A}_1$ auf dem Wort $c_0c_2 \ldots c_{n-2}$. Umgekehrt sind die ungeraden Positionen induziert durch den Lauf von $\mathcal{A}_2$ auf $c_1c_3 \ldots c_{n-1}$. Wegen $p_n \in F_1$ und $i_n = 0$ ist $p_{n-1} = p_n \in F_1$ und somit akzeptiert $\mathcal{A}_1$ das Wort $c_0c_2 \ldots c_{n-2}$. Analog für $\mathcal{A}_2$. Also ist $w \in L(\mathcal{A}_1) \shuffle L(\mathcal{A}_2)$.
"$L(\mathcal{A}) \supseteq L_1 \shuffle L_2$": Sei $w = a_0b_0 \ldots a_{n-1}b_{n-1} \in L_1 \shuffle L_2$. Dann exitieren Läufe $\varrho_1 = (p_0, \ldots, p_n), \varrho_2 = (q_0, \ldots, q_n)$ mit $p_n \in F_1, q_n \in F_2$ von $\mathcal{A}_1, \mathcal{A}_2$. Nach Definition von δ ist der Lauf auf $\mathcal{A}$ dann

$$((p_0, q_0, 0), (p_1, q_0, 1), \ldots, (p_n, q_{n-1}, 1), (p_n, q_n, 0))$$

und $\mathcal{A}$ akzeptiert w. □

3.3 Nichtdeterministische Endliche Automaten

Im letzten Abschnitt haben wir gesehen, dass unter einigen Operationen abgeschlossen sind. Die Mengenoperationen wirken dabei ohnehin von Vorteil für weitere Betrachtungen unter mathematischen Aspekten. Perfect Shuffle könnte man dagegen als eine Routine sehen, ob zum Beispiel ein Prozessor zwei Prozessen wirklich abwechselnd Berechnungszeit gibt. Wir würden gerne weitere Abschlusseigenschaften kennenlernen, besonders die Konkatenation unter der Kleene-Stern wären nun interessant. Eine simultane Ausführung zweier Automaten bringt hier jedoch nichts mehr, da beide Operationen eher von sequentieller Natur sind (Hintereinanderausführung, Wiederholung). DFAs sind für diese Aufgaben zunächst nicht sonderlich sinnvoll. Denkbar wäre ein Modell, dass nach Erreichen eines Endzustandes den nächsten Automaten (im Falle der Konkatenation) auf dem Rest des Wortes startet. Ein DFA weiß aber nicht ad hoc wie das Wort zerlegt ist, es kann also sein, dass nach Erreichen eines Endzustandes der erste Automat noch weiterlaufen soll und erst bei erneutem Besuch eines akzeptierenden Zustandes das zweite Wort losgeht. Zu diesem Zweck führen wir das Prinzip des Nichtdeterminismus ein. Ein Automat kann dann "raten" in welchen Zustand er wechseln soll (bzw. ob er "den nächsten Automaten startet"). Dieses Konzept wirkt etwas unnatürlich, da es nicht von einem Computer simuliert werden kann (Zufall $\neq$ Nichtdeterminismus!), in unserem Modell rät der Automat stets "richtig".

Definition 3.11. Ein *nicht-deterministischer endlicher Automat (NFA)* (von engl.: non-deterministic finite automaton) ist ein 5-Tupel

$$(Q, \Sigma, \Delta, q_0, F),$$

mit

- Q eine nicht-leere, endliche Menge von *Zuständen*,
- Σ ein nicht-leeres, endliches *Eingabealphabet*,

- $\Delta \subseteq Q \times \Sigma \times Q$ die *Transitionsrelation*,
- $q_0 \in Q$ der *Startzustand*,
- $F \subseteq Q$ die Menge der *akzeptierenden Zustände* (oder *Endzustände*).

Wir bezeichnen NFAs genau wie DFAs mit $\mathcal{A}, \mathcal{B}$ usw. Die Transitionsgraphen sehen ebenfalls genauso aus, nur, dass nun Zustände mehrere Kanten haben können, die gleich beschriftet sind. Außerdem ist es erlaubt, dass Transitionen komplett fehlen.

Bemerkung. Eine äquivalente und ebenfalls verbreitete Definition benutzt statt einer Transitionsrelation wieder eine Transitionsfunktion $\delta\colon Q \times \Sigma \to 2^Q$.

Bemerkung. Auch wenn es widersprüchlich klingt, aber jeder DFA kann auch als ein NFA gesehen werden. Genauer gesagt ist ein DFA ein NFA bei dem die Transitionsrelation der Graph einer totalen Funktion $Q \times \Sigma \to Q$ ist.

Bisher ist noch nicht klar, wie das Akzeptanzverhalten von NFAs sein soll.

Definition 3.12. Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NFA. Ein *Lauf* von $\mathcal{A}$ auf einem Wort $w = a_0 \dots a_{n-1}$ für ein $n \in \mathbb{N}$ ist eine endliche Folge

$$(r_0, a_0, r_1, a_1, \dots, a_{n-1}, r_n),$$

wobei $r_0, \dots, r_n \in Q$ und $a_0, \dots, a_{n-1} \in \Sigma$, sodass

(i) $r_0 = q_0$,

(ii) Für alle $i \in [n]$ gilt, dass $(r_i, a_i, r_{i+1}) \in \Delta$.

Wir sagen ein Lauf ist *akzeptierend*, wenn zusätzlich $r_n \in F$ gilt.

Bemerkung. Für NFAs müssen Läufe nicht mehr eindeutig sein. Es kann zu einem Wort mehrere Läufe geben oder auch gar keine Läufe, wenn entsprechende Transitionen fehlen.

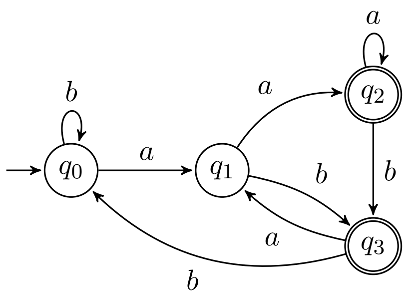
Bemerkung. Produktkonstruktionen um zum Beispiel den Schnitt zweier NFA-erkennbaren Sprachen zu erkennen funktioniert für NFAs genauso wie für DFAs. Schwieriger ist es aber schon das Komplement zu erkennen – ein einfaches vertauschen von akzeptierenden und nicht-akzeptierenden Zuständen genügt nicht, da weiterhin Wörter aus der Sprache rausgelassen werden für die kein Lauf existiert.

Definition 3.13.

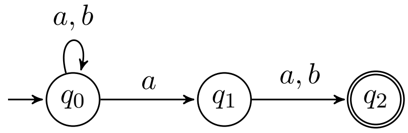
(i) Ein NFA $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ *akzeptiert* ein Wort $w \in \Sigma^*$, wenn es mindestens einen akzeptierenden Lauf von $\mathcal{A}$ gibt. Andernfalls *verwirft* $\mathcal{A}$ das Wort w.

(ii) Die von einem NFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *erkannte Sprache* ist

$$L(\mathcal{A}) \coloneqq \{w \in \Sigma^* \mid \mathcal{A} \text{ akzeptiert } w\}.$$



(a) DFA



(b) NFA

Abbildung 5: Ein DFA und ein NFA für die Sprache aus Beispiel 3.14.

(iii) Eine Sprache L heißt *NFA-erkennbar*, wenn es einen NFA $\mathcal{A}$ gibt, sodass $L = L(\mathcal{A})$.

Beispiel 3.14. Wir betrachten die Sprache

$$L = \{w \in \{a, b\}^* \mid \text{das vorletzte Symbol in } w \text{ ist } a\}.$$

In Abbildung 5 befindet sich ein DFA und ein NFA für die Sprache. Wir sehen, dass der NFA weniger Zustände hat. Dies ist nicht überraschen, denn der DFA muss sich stets das zuletzt gelesene Symbol im Zustand merken, während der NFA dies nicht muss und einfach "rät" welches Symbol das vorletzte ist.

Für NFAs ist das *Wortproblem*, das Problem ob ein Automat ein gegebenes Wort akzeptiert, nicht ganz so offensichtlich zu lösen wie für DFAs. Bei DFAs müssen wir lediglich die eindeutige Transitionsfolge anwenden und prüfen ob der letzte Zustand ein akzeptierender ist. Für NFAs können wir natürlich alle möglichen Läufe ausprobieren und prüfen, ob ein akzeptierender dabei ist. Doch es gieht effizienter mit der Erreichbarkeitsrelation.

Definition 3.15. Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NFA und $w = a_0 \dots a_{n-1}$. Ein Zustand $q \in Q$ heißt *erreichbar* von p über w (geschrieben $\mathcal{A} : p \xrightarrow{w} q$), wenn es einen Lauf $(r_0, \dots, r_n)$ gibt mit

- $r_0 = p, r_n = q$ und
- $(r_i, a_i, r_{i+1}) \in \Delta$ für alle $i \in [n]$.

Diese Notation können wir auch selbstverständlich auf DFAs anwenden. Wir lassen gelegentlich, dass $\mathcal{A}$ weg, wenn der Automat aus dem Kontext klar ist.

Definition 3.16. Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NFA und $w \in \Sigma^*$. Die *Menge der in $\mathcal{A}$ über w erreichbaren Zustände* ist definiert als

$$E(\mathcal{A}, w) := \{q \in Q \mid \mathcal{A} : q_0 \xrightarrow{w} q\}.$$

Wir haben zwei Lemmata, die uns die Anwendung dieser Definition nahelegen.

Lemma 3.17. *Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NFA und $w \in \Sigma^*$. $w \in L(\mathcal{A})$ g.d.w. $E(\mathcal{A}, w) \cap F \neq \varnothing$.*

Beweis. "wenn, dann": Sei $w \in L(\mathcal{A})$. Dann exitiert ein Lauf $(r_0, \ldots, r_n)$ von $\mathcal{A}$ auf w mit $r_n =: q \in F$, also $q_0 \xrightarrow{w} q$. Somit ist $q \in E(\mathcal{A}, w)$. Also ist $\varnothing \neq \{q\} \subseteq E(\mathcal{A}, w) \cap F$.
"genau dann": Es exitiert ein $q \in E(\mathcal{A}, w) \cap F$, also $q \in F$ und $q \in E(\mathcal{A}, w)$. Also gibt es einen Lauf von $\mathcal{A}$ auf w, der in q_0 startet und in q endet. Dieser ist akzeptierend, also ist $w \in L(\mathcal{A})$. □

Lemma 3.18. *Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NFA.*

(i) $E(\mathcal{A}, \varepsilon) = \{q_0\}$,

(ii) Für alle $u \in \Sigma^*$ und $a \in \Sigma$ gilt

$$E(\mathcal{A}, ua) = \bigcup_{p \in E(\mathcal{A}, u)} \{q \in Q \mid (p, a, q) \in \Delta\}.$$

Beweis. Induktionsverankerung: $q \in E(\mathcal{A}, \varepsilon)$ g.d.w. $q_0 \xrightarrow{\varepsilon} q$ g.d.w. $q = q_0$. ✓
Induktionsschritt: $q \in E(\mathcal{A}, ua)$, also $q_0 \xrightarrow{ua} q$. Damit gibt es ein $p \in Q$, sodass $q_0 \xrightarrow{u} p \xrightarrow{a} q$. Wir folgern, dass $p \in E(\mathcal{A}, u)$ und $(p, a, q) \in \Delta$ und somit $q \in \bigcup_{p \in E(\mathcal{A}, u)} \{r \in Q \mid (p, a, r) \in \Delta\}$. Rückrichtung analog. □

Mit Hilfe der Erreichbarkeitsrelation und Lemmata 3.17 und 3.18, erhalten wir für das Wortproblem für NFAs einen Algorithmus.

Algorithmus 1 : Wortproblem für NFAs

NFA-Akzeptanz$(\mathcal{A}, w)$
Input : NFA $\mathcal{A}$, Wort w
Output : Ja g.d.w. $\mathcal{A}$ akzeptiert $w = a_0 \ldots a_{n-1}$.
$u \coloneqq \varepsilon$
$E(\mathcal{A}, u) \coloneqq \{q_0\}$
for $i = 0, \ldots, n-1$ **do**
  Bestimme $E(\mathcal{A}, ua_i)$ aus $E(\mathcal{A}, u)$ (Lemma 3.18)
  $u \coloneqq ua_i$
Prüfe, ob $E(\mathcal{A}, w) \cap F \neq \varnothing$.

Beispiel 3.19. Wir betrachten erneut den NFA in Abbildung 5b und die Erreichbarkeitsmengen für die Präfixe von $w = abbabaa$.

$$\begin{aligned} E(\mathcal{A}, \varepsilon) &= \{q_0\}, \\ E(\mathcal{A}, a) &= \{q_0, q_1\}, \\ E(\mathcal{A}, ab) &= \{q_0, q_2\}, \\ E(\mathcal{A}, abb) &= \{q_0\}, \\ E(\mathcal{A}, abba) &= \{q_0, q_1\}, \\ E(\mathcal{A}, abbab) &= \{q_0, q_2\}, \\ E(\mathcal{A}, abbaba) &= \{q_0, q_1\}, \\ E(\mathcal{A}, abbabaa) &= \{q_0, q_1, q_2\}. \end{aligned}$$

Wegen $E(\mathcal{A}, w) \cap F \neq \varnothing$ wird w akzeptiert.

3.4 Äquivalenz von NFAs und DFAs

Auf den ersten Blick wirkt es vermutlich so, dass NFAs “mehr“ können als DFAs, da NFAs durch das stets richtige Raten der Transition einen Blick in die Zukunft werfen können. Dieser Abschnitt widmet sich jedoch der Äquivalenz von NFAs und DFAs, d.h. auch wenn wie in Beispiel 3.14 NFAs mit weniger Zuständen die gleichen Sprachen erkennen können wie DFAs, so können auch DFAs jede NFA-erkennbare Sprache erkennen. Dies ist mit einer einfachen Überlegung auch gar nicht so unintuitiv: Die Erreichbarkeitsmenge für einen NFA und ein Wort ist stets endlich, da auch ein NFA lediglich endlich viele Zustände hat. In einer Simulation eines NFA in einem DFA könnte man also also einen Zustand durch die Menge der erreichbaren Zustände wählen und würde endlich bleiben. Details dazu folgen in diesem Abschnitt.

Definition 3.20. Seien $\mathcal{A}, \mathcal{B}$ zwei endliche Automaten (deterministisch oder nicht-deterministisch). $\mathcal{A}$ und $\mathcal{B}$ heißen *äquivalent*, wenn $L(\mathcal{A}) = L(\mathcal{B})$.

Eine Bemerkung aus dem vorigen Abschnitt greifen wir nochmal im folgendem Lemma auf.

Lemma 3.21. *Zu jedem DFA exitiert ein äquivalenter NFA.*

Beweis. Sei $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ ein DFA. Wir definieren einen NFA

$$\mathcal{A}' = (Q, \Sigma, \Delta, q_0, F),$$

mit $\Delta = \{(p, a, q) \mid \delta(p, a) = q\}$. Wir zeigen, dass $\mathcal{A}'$ und $\mathcal{A}$ äquivalent sind. Dazu sei $w = a_0 \dots a_{n-1} \in L(\mathcal{A})$. Also ist der eindeutige Lauf $\varrho = (r_0, \dots, r_n)$ akzeptierend auf $\mathcal{A}$. Der Lauf ist nach Konstruktion auch ein Lauf auf $\mathcal{A}'$, also akzeptiert auch $\mathcal{A}'$. Rückrichtung analog. $\square$

Diese Richtung war relativ einfach mit den eingangs genannten Bemerkungen. Man könnte den Beweis auch aufwändig wieder über Induktion führen, dies ist aber nicht sinnvoll, da das Ergbenis recht klar ist.

Lemma 3.22. *Zu jedem NFA existiert ein äquivalenter DFA.*

Beweis. Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NFA. Wir konstruieren einen DFA, der äquivalent ist, durch die sogenannte *Potenzmengenkonstruktion*:

$$\mathcal{A}' = (2^Q, \Sigma, \delta, \{q_0\}, \{P \subseteq Q \mid P \cap F \neq \varnothing\})$$

mit $\delta(P, a) = \{q \mid \text{es ex. } p \in P \text{ mit } (p, a, q) \in \Delta\}$. Wir zeigen, dass beide Automaten äquivalent sind durch die Behauptung, dass für alle $w \in \Sigma^*$ und $P \subseteq Q$ mit $\mathcal{A}' : \{q_0\} \xrightarrow{w} P$ gilt, dass $P = E(\mathcal{A}, w)$, d.h. der Zustand, den der Potenzmengenautomatauf dem Wort w erreicht, entspricht der Erreichbarkeitsmenge auf dem selben Wort für den NFA. Dies zeigen wir per Induktion über alle Wortlängen n.
Induktionsverankerung: $n = 0$. Dann ist $w = \varepsilon$ und es gilt $P = \{q_0\} = E(\mathcal{A}, \varepsilon)$ nach Lemma 3.18.
Induktionsschritt: $n \to n + 1$. Dann ist $w = ua$ für ein $u \in \Sigma^n$ und $a \in \Sigma$. Sei $P' \subseteq Q$, sodass $\mathcal{A}' : \{q_0\} \xrightarrow{u} P'$. Nach Induktionshypothese ist $P' = E(\mathcal{A}, u)$. Somit gilt

$$\begin{aligned} P &= \delta(P', a) \\ &= \{q \in Q \mid \text{es ex. } p \in P' \text{ mit } (p, a, q) \in \Delta\} \\ &= \bigcup_{p \in P' = E(\mathcal{A}, u)} \{q \in Q \mid (p, a, q) \in \Delta\} \\ &= E(\mathcal{A}, w) \end{aligned}$$

nach Definition von δ und Lemma 3.18. Insgesamt gilt also, dass $w \in L(\mathcal{A})$ g.d.w. $E(\mathcal{A}, w) \cap F \neq \varnothing$ (Lemma 3.17). Nach der oben gezeigten Behauptung gilt dies g.d.w. $P \cap F \neq \varnothing$ und nach Definition des Konstruktion ist P genau dann akzeptierend im Potenzmengenautomaten. Somit akzeptiert $\mathcal{A}'$ das Wort w. □

In der Potenzmengenkonstruktion haben wir als Zustandsmenge stets die Potenzmenge der Zustandsmenge des NFA genutzt. Offenbar ist es aber so, dass dann einige Zustände unerreichbar sind, diese können auch weggelassen werden.

Definition 3.23. Sei $\mathcal{A} = (Q, \Sigma, \frac{\delta}{\Delta}, q_0, F)$ ein DFA bzw. NFA.

(i) Ein Zustand $q \in Q$ ist *erreichbar*, wenn es ein $w \in \Sigma^*$ gibt, sodass $\mathcal{A} : q_0 \xrightarrow{w} q$.

(ii) Der *reduzierte Automat* (auf die erreicharen Zustände) ist:

$$\mathcal{A}' = (Q', \Sigma, \frac{\delta'}{\Delta'}, q_0, F')$$

wobei

- $Q' := \{q \in Q \mid q \text{ erreichbar}\}$,
- $\delta' := \delta|_{Q' \times \Sigma}$ bzw. $\Delta' := \Delta \cap (Q' \times \Sigma \times Q')$,
- $F' := F \cap Q'$.

Wir verwenden die Potenzmengenkonstruktion aus Lemma 3.22 zur Determinisierung eines NFA. Wir können dabei schrittweise vom Anfangszustand die Zustände und Transitionen konstruieren um so direkt auf einen reduzierten Automaten gemäß Definition 3.23 zu kommen. Dies ist in Algorithmus 2 beschrieben.

Algorithmus 2 : NFA-Determinisierung

Potenzmengenkonstruktion($\mathcal{A}$)
Input : NFA $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$
Output : äquivalenter, reduzierter DFA $\mathcal{A}' = (Q', \Sigma, \delta, q_0', F')$
$Q' := \{\{q_0\}\}$
$q_0' := \{q_0\}$
$F' := \varnothing$
Queue $P := Q'$
while $P \neq \varnothing$ **do**
    $S := P$.dequeue()
    for $a \in \Sigma$ **do**
        $R := \{\bigcup_{p \in S}\{r \mid (p, a, r) \in \Delta\}$
        P.enqueue(R)
        $Q' := Q' \cup R$
        $\delta(S, a) = R$
for $P \in Q'$ **do**
    if $P \cap F \neq \varnothing$ **then**
        $F' := F' \cup \{P\}$

Beispiel 3.24. Wir betrachten den NFA in Abbildung 6a. In Abbildung 6b ist ein äquivalenter, reduzierter DFA, erhalten durch Potenzmengenkonstruktion.

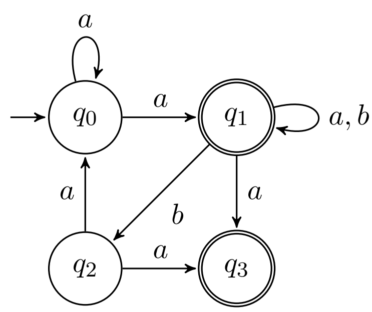
Satz 3.25. *Eine Sprache ist genau dann DFA-erkennbar, wenn sie NFA-erkennbar ist.*

Beweis. Die Lemmata 3.21 und 3.22 zusammen geben den Beweis. □

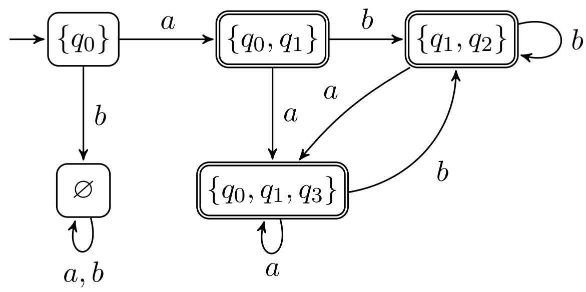
In diesem Sinne ist es sinnvoll von nun an nur noch von FA-erkennbaren Sprachen zu sprechen statt zwischen DFA- und NFA-erkennbaren Sprachen zu unterscheiden.

3.5 NFAs mit ε-Transitionen

Mit Blick auf spätere Abschnitte erweitern wir das Modell der NFAs um die Möglichkeit den Zustand zu wechseln ohne ein Symbol dabei zu lesen.



(a) NFA



(b) Potenzmengenautomat

Abbildung 6: Beispiel zur Potenzmengenkonstruktion.

Definition 3.26. Ein *nicht-deterministischer endlicher Automat mit ε-Transitionen (ε-NFA)* ist ein 5-Tupel

$$(Q, \Sigma, \Delta, q_0, F),$$

mit

- Q eine nicht-leere, endliche Menge von *Zuständen*,
- Σ ein nicht-leeres, endliches *Eingabealphabet*,
- $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ die *Transitionsrelation*,
- $q_0 \in Q$ der *Startzustand*,
- $F \subseteq Q$ die Menge der *akzeptierenden Zustände* (oder *Endzustände*).

Wir bezeichnen ε-NFAs wieder mit $\mathcal{A}, \mathcal{B}$ usw. Transitionsgraphen lassen sich ebenfalls analog zeichnen wie für NFAs und DFAs, wobei ε-Transitionen durch Kanten gekennzeichnet werden, die mit ε beschriftet sind. Der Vollständigkeit halber definieren wir auch wieder Läufe auf ε-NFAs und das Akzeptanzverhalten.

Definition 3.27. Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein ε-NFA. Ein *Lauf* von $\mathcal{A}$ auf einem Wort $w = a_0 \dots a_{n-1}$ für ein $n \in \mathbb{N}$ ist eine endliche Folge

$$(r_0, \sigma_0, r_1, \sigma_1, \dots, sigma_{m-1}, r_m),$$

wobei $r_0, \dots, r_m \in Q$ und $\sigma_0, \dots, \sigma_{m-1} \in \Sigma \cup \{\varepsilon\}$, sodass

(i) $r_0 = q_0$,

(ii) Für alle $i \in [m]$ gilt, dass $(r_i, \sigma_i, r_{i+1}) \in \Delta$.

Wir sagen ein Lauf ist *akzeptierend*, wenn zusätzlich $r_n \in F$ gilt.

Definition 3.28.

(i) Ein ε-NFA $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ *akzeptiert* ein Wort $w \in \Sigma^*$, wenn es mindestens einen akzeptierenden Lauf von $\mathcal{A}$ gibt. Andernfalls *verwirft* $\mathcal{A}$ das Wort w.

(ii) Die von einem ε-NFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *erkannte Sprache* ist

$$L(\mathcal{A}) \coloneqq \{w \in \Sigma^* \mid \mathcal{A} \text{ akzeptiert } w\}.$$

(iii) Eine Sprache L heißt *ε-NFA-erkennbar*, wenn es einen ε-NFA $\mathcal{A}$ gibt, sodass $L = L(\mathcal{A})$.

Beispiel 3.29. Wir betrachten die Sprache $L = \{a^n b^m : n, m \in \mathbb{N}\} \cup \{a^n c^m : n, m \in \mathbb{N}\}$. Der ε-NFA in Abbildung 7 erkennt die Sprache L mit drei Zuständen.

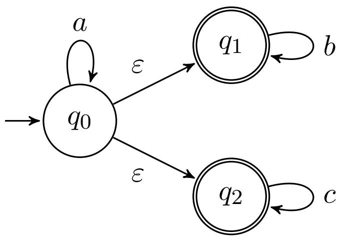


Abbildung 7: ε-NFA, der die Sprache aus Beispiel 3.29 erkennt.

Für ε-NFAs zeigen wir wie im vorigen Abschnitt, dass auch sie die Klasse der FA-erkennbaren Sprachen nicht vergrößern. Der Äquivalenzbegriff aus Definition 3.20 überträgt sich natürlich auf ε-NFAs.

Lemma 3.30. *Zu jedem NFA existiert ein äquivalenter ε-NFA.*

Beweis. Dies ist sofort klar, da jeder NFA als ein ε-NFA betrachtet werden kann, der keine ε-Transitionen besitzt. □

Bevor wir den anderen Teil der Äquivalenz zeigen fehlt uns eine Definition.

Definition 3.31. Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein ε-NFA. Der *ε-Abschluss* (auch ε-Hülle) eines Zustands $p \in Q$ ist

$$\overline{\varepsilon}(p) \coloneqq \{q \in Q \mid \text{es ex. } p_1, \ldots, p_k \text{ mit } (p, \varepsilon, p_1), (p_i, \varepsilon, p_{i+1}), (p_k, \varepsilon, q) \in \Delta\}.$$

Für Mengen $P \subseteq Q$ ist der *ε-Abschluss* dann erweitert durch

$$\overline{\varepsilon}(P) \coloneqq \bigcup_{p \in P} \overline{\varepsilon}(p).$$

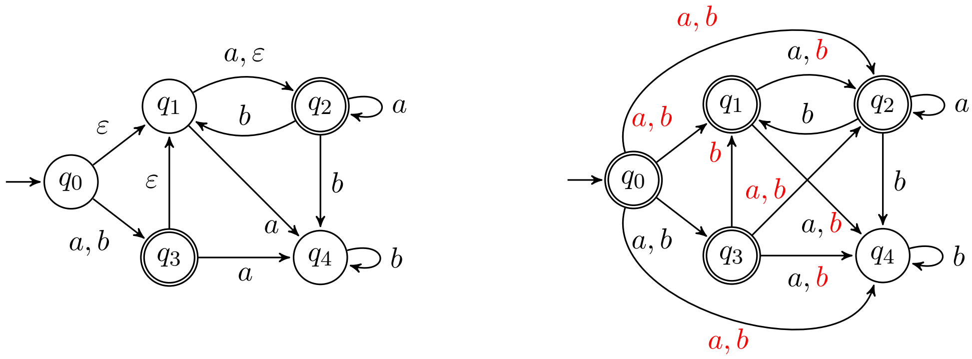
Auf diese Weise lässt sich die Erreichbarkeitsrelation, die wir schon aus Definition 3.15 kennen erweitern.

Definition 3.32. Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NFA und $w = a_0 \ldots a_{n-1}$. Ein Zustand $q \in Q$ heißt *erreichbar* von p über w (geschrieben $\mathcal{A} : p \xrightarrow{w} q$), wenn es ein $m \geq n$, Indizes $0 \leq i_0 < \ldots < i_{n-1} \leq m$ und Zustände $r_0, \ldots, r_m \in Q$ gibt, sodass

- $r_0 = p, r_m = q$,
- $(r_{i_j}, a_j, r_{i_j+1}) \in \Delta$ für alle $j \in [n]$ und
- $(r_i, \varepsilon, r_{i+1}) \in \Delta$ für alle $i \in [m] \setminus \{i_1, \ldots, i_{n-1}\}$.

Man beachte, dass $p \xrightarrow{\varepsilon} q$ nicht $p = q$ bedeutet wie bei NFAs.

Lemma 3.33. *Zu jedem ε-NFA existiert ein äquivalenter NFA.*



(a) mit ε-Transitionen

(b) ohne ε-Transitionen

Abbildung 8: Eliminieren von ε-Transitionen

Beweis. Wir betrachten einen ε-NFA $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ und überführen ihn in einen äquivalenten NFA $\mathcal{A}' = (Q, \Sigma, \Delta', q_0, F')$. Dabei ist

$$\Delta' \coloneqq \{(p, a, q) \in Q \times \Sigma \times Q \mid \mathcal{A} : p \xrightarrow{a} q\}$$

und $F' \coloneqq \{q \in Q \mid q \in \overline{\varepsilon}(F)\}$. Die Korrektheit ist einfach einzusehen. Sei $w = a_0 \ldots a_{n-1} \in L(\mathcal{A})$. Dann gibt es ein $q \in F$ mit $q_0 \xrightarrow{w} q$. Also gibt es Zustände $r_0 = q_0, \ldots, r_m = q$ wie in Definition 3.32. Nach Konstruktion von Δ' gibt es somit Zustände $r_{i_0} = q_0, \ldots, r_{i_n} = q$, sodass $r_{i_j} \xrightarrow{a_j} r_{i_{j+1}}$ für alle $j \in [n]$. Damit ist $w \in L(\mathcal{A}')$. Rückrichtung analog. □

Beispiel 3.34. Wir betrachten den ε-NFA in Abbildung 8a. In Abbildung 8b sehen wir den NFA den wir nach Lemma 3.33 erhalten, wobei die neu eingefügten Transitionen rot markiert sind. Beachte auch, dass sich die Menge der akzeptierenden Zustände geändert hat, da sonst das leere Wort nicht mehr akzeptiert werden würde.

Bemerkung. Beachte, dass ein Zusammenziehen der Zustände, die durch ε-Transitionen verbunden sind nicht funktioniert, da das im Allgemeinen die erkannte Sprache verändert. Ein solches Vorgehen würde zum Beispiel dazu führen, dass der entstehende Automat zu Beispiel 7 nur noch einen Zustand besitzt mit einem durch a, b, c beschrifteten Loop. Die erkannte Sprache wäre dann $\{a, b, c\}^*$.

Korollar 3.35. *Zu jedem ε-NFA existiert ein äquivalenter DFA.*

Beweis. Lemma 3.33 liefert zunächst einen NFA, der mit Lemma 3.22 in einen DFA umgewandelt werden kann. □

Wir werden also auch weiterhin von FA-erkennbaren Sprachen sprechen, solange sie von einem DFA, NFA oder ε-NFA erkannt werden, da alle drei Modelle gleich mächtig sind. In mancher Fachliteratur wird wegen ihrer Äquivalenz auch zwischen NFAs und ε-NFAs gar nicht unterschieden – wir werden im Folgenden die Unterscheidung dennoch weiterhin machen.

3.6 Reguläre Ausdrücke

Bisher haben wir Sprachen betrachtet, die sich von endlichen Automaten erkennen lassen. Diese Art von Sprachen liefern uns eine eigene Klasse von Sprachen, die unter verschiedenen Operationen abgeschlossen ist. Auch wenn bisher noch nicht alles gezeigt wurde (Konkatenation, Kleene'sche Hülle) lässt sich mit einiger Berechtigung sagen, dass die Klasse der FA-erkennbaren Sprachen gute Eigenschaften hat. Wir untersuchen nun welche Sprache wir mit sogenannten *regulären Ausdrücken* beschreiben können.

Definition 3.36. Sei Σ ein endliches Alphabet. Ein *regulärer Ausdruck* ist induktiv definiert mit:

- $\boldsymbol{\varnothing}$ ist ein regulärer Ausdruck.
- Für jedes $a \in \Sigma$ ist $\boldsymbol{a}$ ein regulärer Ausdruck.
- Falls r, r' reguläre Ausdrücke sind, dann ist auch $(r + r')$ ein regulärer Ausdruck.
- Falls r, r' reguläre Ausdrücke sind, dann ist auch $(r \cdot r')$ ein regulärer Ausdruck.
- Falls r reguläre Ausdrücke sind, dann ist auch $r*$ ein regulärer Ausdruck.

Die *Menge aller regulären Ausdrücke* über Σ bezeichnen mit RE_Σ.

Das ist bisher lediglich die Syntax der regulären Ausdrücke gewesen. Nun definieren wir eine Sematik für diese Ausdrücke, d.h. wir ordnen jedem regulären Ausdruck eine Sprache zu.

Definition 3.37. Sei Σ ein endliches Alphabet. Die *Interpretation eines regulären Ausdrucks* ist die Abbildung

$$[\![\cdot]\!] \colon \mathsf{RE}_\Sigma \to 2^{\Sigma^*}$$

mit

- $[\![\boldsymbol{\varnothing}]\!] = \varnothing$,
- $[\![\boldsymbol{a}]\!] = \{a\}$ für jedes $\boldsymbol{a} \in \mathsf{RE}_\Sigma$,
- $[\![(r + r')]\!] = [\![r]\!] \cup [\![r']\!]$,
- $[\![(r \cdot r')]\!] = [\![r]\!] \cdot [\![r']\!]$,
- $[\![r*]\!] = [\![r]\!]^*$.

Eine Sprache $L \subseteq \Sigma^*$ heißt *regulär*, wenn ein regulärer Ausdruck $r \in \mathsf{RE}_\Sigma$ existiert mit $[\![r]\!] = L$.

Wir erlauben in der Regel auch als Abkürzung die Ausdrücke $\boldsymbol{\varepsilon}$ für $\boldsymbol{\varnothing}*$ und $r+$ für $r \cdot r*$. Außerdem lassen wir den Punkt ($\cdot$) weg, es sei denn er dient der Lesbarkeit. Die Klammern können wir auch weglassen unter der Konvention, dass $*$ stärker bindet als $\cdot$, was wiederum stärker bindet als $+$. Statt $[\![r]\!]$ ist auch die Schreibweise $L(r)$ gebräuchlich.
Reguläre Ausdrücke kennt man auch für Computerprogramme wie `grep`, `awk` oder `sed`. Diese sind syntaktisch etwas anders aufgebaut, jedoch lässt sich jeder POSIX-Ausdruck (das sind die regulären Ausdrücke für oben genannte Programme) auch als regulärer Ausdruck wie in Definition 3.36 umschreiben.

Beispiel 3.38. Die POSIX-Syntax für reguläre Ausdrücke geht wie folgt.

- $r \,|\, r'$ für $r + r'$,
- Für $(\boldsymbol{a} + \boldsymbol{b} + \ldots + \boldsymbol{z})$ geht auch `[a-z]`
  (analog: `[A-Z]`, `[0-9]`, `[a-z0-9,\.:;\?!]` usw.),
- `.` für ein beliebiges Zeichen,
- r? für $r + \boldsymbol{\varepsilon}$,
- r+ und r* für $r+, r*$,
- r`{m,n}` für $r^m + r^{m+1} + \ldots + r^n$.

Wir können zum Beispiel die Sprache aller E-Mail-Adressen als Ausdruck

```
[a-zA-Z0-9\.-]+@[a-zA-Z0-9\.-]+\.(de|com|net)
```

darstellen.

Für unsere Analyse von regulären Sprachen benutzen wir nur reguläre Ausdrücke der Form wie in Definition 3.36 vorgestellt. Dadurch lassen sich viele Beweise über den minimalistischen induktiven Aufbau der Ausdrücke führen.

Beispiel 3.39. Wir betrachten das Alphabet $\Sigma = \{a, b\}$.

- $L(((\boldsymbol{a} + \boldsymbol{b})(\boldsymbol{a} + \boldsymbol{b}))*) = \{w \in \Sigma^* \mid |w| \text{ gerade}\}$.
- $L((\boldsymbol{a} + \boldsymbol{b})(\boldsymbol{a} + \boldsymbol{b})(\boldsymbol{a} + \boldsymbol{b})*) = \{w \in \Sigma^* \mid |w| \geq 2\}$.
- $L(\boldsymbol{a}* + \boldsymbol{b}*) = \{w \in \Sigma^* \mid |w|_a = 0 \text{ oder } |w|_b = 0\}$.

Aus Gründen der Einfachheit werden wir von nun an bei der Notation von regulären Ausdrücken auf eine Unterscheidung zu Alphabetsymbolen, Kleene-Stern usw. verzichten, d.h. wir verwenden $\varnothing, \varepsilon, a, +, \cdot, ^*$ statt $\boldsymbol{\varnothing}, \boldsymbol{\varepsilon}, \boldsymbol{a}, +, \cdot, *$. Außerdem verwenden wir Shortcuts, z.B. $\sum_{a \in \Sigma} a = \Sigma^*$. Formal ist aber wichtig weiterhin eine Unterscheidung zwischen Sprachen und regulären Ausdrücken zu haben.

(a) $r = \varnothing$ (b) $r = \varepsilon$ (c) $r = a$

Abbildung 9: Basisfälle der Thompson-Konstruktion

Definition 3.40. Zwei reguläre Ausdrücke $r, e \in \mathsf{RE}_\Sigma$ heißen *äquivalent* (geschrieben $r \equiv e$), wenn $[\![r]\!] = [\![e]\!]$.
Ein regulärer Ausdruck $r \in \mathsf{RE}_\Sigma$ und ein endlicher Automat $\mathcal{A}$ (DFA, NFA) heißen *äquivalent*, wenn $[\![r]\!] = L(\mathcal{A})$.

Beispiel 3.41. Die regulären Ausdrücke $(a+b)^*$ und $(a^*b^*)^*$ sind äquivalent.

Wir kommen nun zu einem der wichtigsten Ergebnisse der Automatentheorie und dieser Vorlesung.

Satz 3.42 (Äquivalenzsatz von Kleene)**.** *Eine Sprache ist genau dann regulär, wenn sie FA-erkennbar ist.*

Aus Gründen der Übersichtlichkeit werden wir den Beweis des Satzes aufteilen auf zwei Lemmata.

Lemma 3.43. *Jede reguläre Sprache ist FA-erkennbar.*

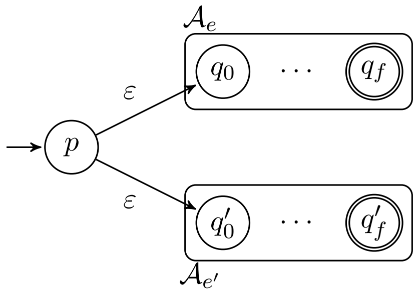
Beweis. Wir stellen im Rahmen dieses Beweises eine Variante der sogenannten Thompson-Konstruktion vor, die einen regulären Ausdruck in einen äquivalenten ε-NFA umwandelt. Die Konstruktion nutzt den induktiven Aufbau von regulären Ausdrücken aus.
Basisfälle (s. Abbildung 9): Wir geben drei ε-NFAs für die drei Basisfälle für reguläre Ausdrücke:

$$\begin{aligned}
\mathcal{A}_\varnothing &= (\{q_0\}, \Sigma, \varnothing, q_0, \varnothing),\\
\mathcal{A}_\varepsilon &= (\{q_0\}, \Sigma, \varnothing, q_0, \{q_0\}),\\
\mathcal{A}_a &= (\{q_0, q_1\}, \Sigma, \{(q_0, a, q_1)\}, q_0, \{q_1\}).
\end{aligned}$$

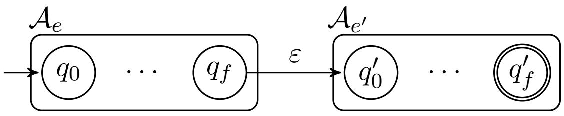
Rekursive Fälle (s. Abbildung 10): Seien für $e, e' \in \mathsf{RE}_\Sigma$ die ε-NFAs $\mathcal{A}_e = (Q, \Sigma, \Delta, q_0, F)$ und $\mathcal{A}_{e'} = (Q', \Sigma, \Delta', q_0', F')$ gegeben mit $q_{-1} \notin Q \cup Q'$ und $Q \cap Q' = \varnothing$. Dann erhalten wir ε-NFAs für die Ausdrücke $e + e', e \cdot e'$ und e^*:

$$\begin{aligned}
\mathcal{A}_{e+e'} &= (Q \cup Q' \cup \{q_{-1}\}, \Sigma, \Delta \cup \Delta' \cup \{(q_{-1}, \varepsilon, q_0), (q_{-1}, \varepsilon, q_0')\}, q_{-1}, F \cup F'),\\
\mathcal{A}_{e\cdot e'} &= (Q \cup Q', \Sigma, \Delta \cup \Delta' \cup \{(q, \varepsilon, q_0') \mid q \in F\}, q_0, F'),\\
\mathcal{A}_{e^*} &= (Q \cup \{q_{-1}\}, \Sigma, \Delta \cup \{(q_{-1}, \varepsilon, q_0)\} \cup \{(q, \varepsilon, q_{-1}) \mid q \in F\}, q_{-1}, \{q_{-1}\}).
\end{aligned}$$

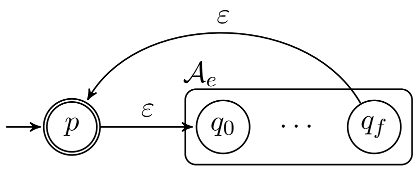
Wir zeigen nun noch die Korrektheit der Konstruktion. Dies geschieht per Induktion über den Aufbau des regulären Ausdrucks.



(a) $r = e + e'$



(b) $r = e \cdot e'$



(c) $r = e^*$

Abbildung 10: Rekursive Fälle der Thompson-Konstruktion

Induktionsverankerung: $e \in \{\varnothing, \varepsilon, a\}$. Der Automat $\mathcal{A}_e$ wie in Abbildung 9 erkennt offensichtlich $[\![e]\!]$.✓
Induktionshypothese: Für die regulären Ausdrücke $e, e' \in \mathsf{RE}_\Sigma$ erkennen die Automaten $\mathcal{A}_e = (Q, \Sigma, \Delta, q_0, F)$ und $\mathcal{A}_{e'} = (Q', \Sigma, \Delta', q_0', F')$ die Sprachen $[\![e]\!]$ bzw. $[\![e']\!]$.
Induktionsschritt: Wir zeigen nur den Fall für $e + e'$, die anderen beiden gehen analog. Sei $w \in [\![e + e']\!] = [\![e]\!] \cup [\![e']\!]$. OBdA sei $w \in [\![e]\!]$. Demnach existiert ein Lauf $(r_0, \ldots, r_n)$ mit $r_n \in F$. Dann ist $(q_{-1}, r_0, \ldots, r_n)$ akzeptierender Lauf auf $\mathcal{A}_{e+e'}$ und damit $w \in L(\mathcal{A}_{e+e'})$. Sei umgekehrt $w \in L(\mathcal{A}_{e+e'})$. Das heißt es gibt einen Lauf $(q_{-1}, r_0, \ldots, r_n)$ mit $r_n \in F \cup F'$. Da q_{-1} nur zwei ε-Transitionen hat muss $(r_0, \ldots, r_n)$ ebenfalls ein akzeptierender Lauf sein mit Startzustand r_0. Nach Konstruktion ist r_0 Startzustand von oBdA $\mathcal{A}_e$ und da $Q \cap Q' = \varnothing$ muss $r_n \in F$ sein. Damit ist $w \in [\![e]\!] \subseteq [\![e + e']\!]$. □

Bemerkung. Wir sehen, dass die Thompson-Konstruktion einige ε-Transitionen verwendet und auch gelegentlich den Zustandsraum unnötig vergrößert. Abseits von einigen Optimierungen dazu gibt es auch Verfahren, die ganz ohne ε-Transitionen auskommt (s. Glushkov-Konstruktion). In dieser Vorlesung wird diese aber nicht betrachten.

Bemerkung. Der ursprüngliche Beweis von Lemma 3.43 von Stephen C. Kleene ging übrigens von einem regulären Ausdruck zu einem DFA, das Konzept "Nichtdeterminismus“ wurde erst später eingeführt. Man kann sich vorstellen, dass der Beweis um einiges schwieriger war als diese recht anschauliche Konstruktion, u.a. waren auch algebraische Konzepte und Halbgruppentheorie involviert.

Ein Nebenprodukt der Thompson-Konstruktion aus dem Beweis ist folgendes Korollar.

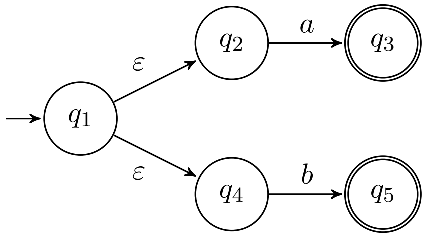
Korollar 3.44. *Seien $L, K \subseteq \Sigma^*$ FA-erkennbare Sprachen. Dann sind auch $L \cdot K$ und L^* FA-erkennbar.*

Beispiel 3.45. Wir betrachten den regulären Ausdruck $r = (a + b)^*c$ und bauen nun einen ε-NFA für r. Die Automaten für die atomaren Teilausdrücke a, b, c sind klar. In Abbildungen 11a bis 11c gehen wir Schritt für Schritt die Konstruktion durch. Man sieht wie der Automat aus dem vorigen Schritt in den Automaten aus dem aktuellen Schritt eingebettet ist. In Abbildung 11d finden wir außerdem einen zustandsminimalen NFA, der sich recht schnell nur durch Ablesen des Ausdrucks hinschreiben lässt. Dies zeigt, dass auch bei einem solchen Minimalbeispiel bereits viele unnötige Zustände und Transitionen eingefügt werden in der Thompson-Konstruktion. Der Vorteil ist aber natürlich, dass ein Algorithmus immer funktioniert.

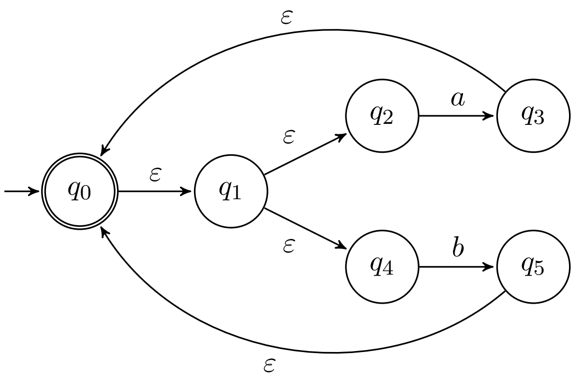
Wir fahren nun fort mit dem zweiten Teil von Satz 3.42. Wir schauen uns hier ein Verfahren an, was aus jedem endlichen Automaten einen regulären Ausdruck macht. Auch dazu gibt es verschiedene Varianten, u.a. auch grafische, die grundlegende Idee hinter diesen Verfahren ist jedoch meist die selbe.

Lemma 3.46. *Jede FA-erkennbare Sprache ist regulär.*

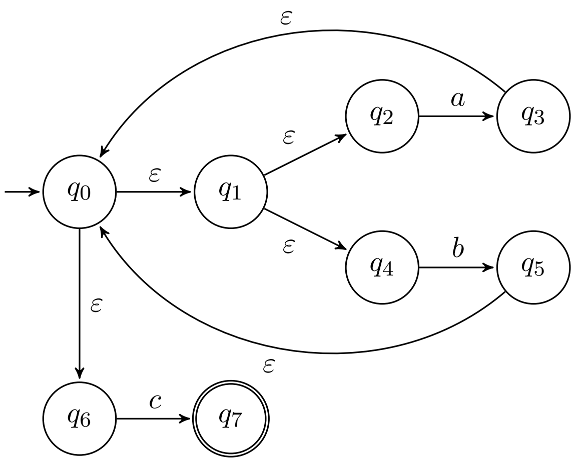
Beweis. Wir gehen von einem NFA $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ aus und erzeugen einen äquivalenten regulären Ausdruck $r_\mathcal{A} \in \mathsf{RE}_\Sigma$. Die Idee des Verfahrens ist recht einfach. Wir suchen einen



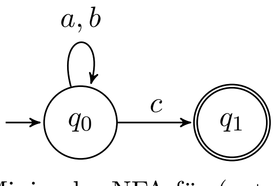
(a) Automat für $(a + b)$.



(b) Automat für $(a + b)^*$.



(c) Automat für $(a + b)^*c$.



(d) Minimaler NFA für $(a + b)^*c$.

Abbildung 11: Thompson-Konstruktion für $r = (a + b)^*c$.

regulären Ausdruck der alle Wörter beschreibt mit denen man vom Startzustand q_0 zu einem der akzeptierenden Zustände $q \in F$ kommt $(*)$. Wir benutzen nun folgende Notation: Für $P \subseteq Q$ und $p, q \in Q$ ist $r_P(p, q)$ der reguläre Ausdruck, der alle Wörter $w \in \Sigma^*$ beschreibt mit den man von p nach q kommt und dazwischen nur Zustände aus P benutzt (für die Korrektheit später schreiben wir dafür $p \xrightarrow[P]{w} q$). Anders gesagt ist $r_P(p, q)$ der reguläre Ausdruck zum Automaten $\mathcal{A} = (P, \Sigma, \Delta \cap ((P \cup \{p\}) \times \Sigma \times (P \cup \{q\})), p, \{q\})$. Bzgl. $(*)$ suchen wir also den regulären Ausdruck

$$r_{\mathcal{A}} = \sum_{q \in F} r_Q(q_0, q).$$

Das Verfahren lässt sich rekursiv beschreiben nun mit folgender Überlegung: Ein regulärer Ausdruck $r_\varnothing(p, q)$ ist $\sum_{(p,a,q) \in \Delta} a$, also alle Symbole $a \in \Sigma$ mit denen eine direkte Transition von p nach q möglich ist (oder noch anschaulicher: Alle Symbole die im Transitionsgraphen auf der Kante von p nach q stehen). Sollte keine Transition existieren, dann ist der reguläre Ausdruck entsprechend $\varnothing$ für $p \neq q$ und ε für $p = q$. Einmal formal aufgeschrieben:

$$r_\varnothing(p, q) = \begin{cases} \displaystyle\sum_{(p,a,q) \in \Delta} a, & p \neq q \\ \varepsilon + \displaystyle\sum_{(p,a,q) \in \Delta} a, & p = q. \end{cases}$$

Der Rekursionsschritt funktioniert nun folgendermaßen: Für ein $P \neq \varnothing$ wählen wir einen Zustand $s \in P$ und entfernen diesen mit folgender Umformung

$$r_P(p, q) = r_{P'}(p, q) + r_{P'}(p, s) r_{P'}(s, s)^* r_{P'}(s, q),$$

wobei $P' := P \setminus \{s\}$. Auch dies einmal in einfache Worte gefasst: Um von p nach q zu kommen über Zustände ausschließlich aus P können wir entweder (links vom $+$) von p nach q gehen ohne einen Zustand s zu besuchen oder (rechts vom $+$) von p nach s gehen, Pfade von s nach s benutzen und schließlich von s nach q gehen, wobei auch hier nur Zustände in $P \setminus \{s\}$ benutzt werden (s. Abbildung 12). $\square$

Bemerkung. Das Verfahren könnte einem auch aus Algorithmik-Vorlesungen bekannt vorkommen. Es handelt sich dabei um eine (zugegeben umständlich aufgeschriebene) Variante des Floyd-Warshall-Algorithmus mit dem sich zum Beispiel auch kürzeste Wege in gerichteten Graphen finden lassen.

Beispiel 3.47. Wir betrachten den NFA in Abbildung 13. Wir suchen also einen regulären Ausdruck, der die Worte beschreibt mit denen man von q_0 nach q_0 (q_0 ist einziger akzeptierender Zustand) kommt mit allen Zuständen, also den Ausdruck $r_Q(q_0, q_0)$. Wir wählen zunächst $s = q_1$ (eliminieren q_1):

$$r_{\mathcal{A}} = r_Q(q_0, q_0) = r_{\{q_0\}}(q_0, q_0) + r_{\{q_0\}}(q_0, q_1) r_{\{q_0\}}(q_1, q_1)^* r_{\{q_0\}}(q_1, q_0)$$

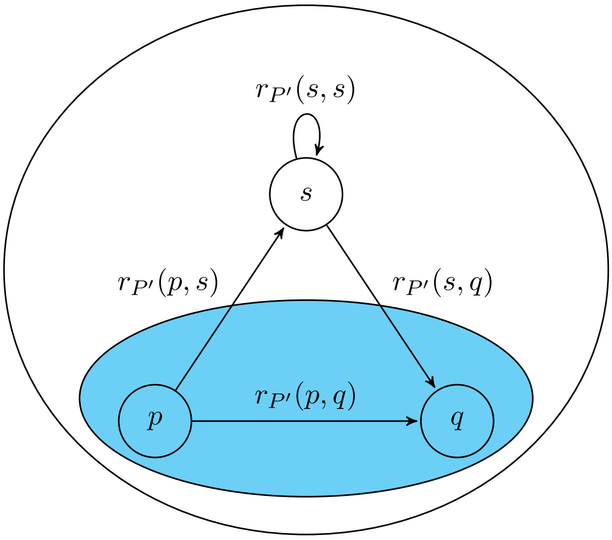


Abbildung 12: Rekursionsschritt um einen Zustand rauszuwerfen. Die blau getönte Fläche ist $P' := P \setminus \{s\}$. Die größere Fläche ist P. Wichtig: Die Kanten in diesem Graphen sind keine Transitionen, sondern sind mit einem regulären Ausdruck beschriftet, der die Sprache zwischen den Knoten beschreibt mit Zuständen aus dem Index!

Noch zu berechnen sind $r_{\{q_0\}}(q_0,q_0), r_{\{q_0\}}(q_0,q_1), r_{\{q_0\}}(q_1,q_1), r_{\{q_0\}}(q_1,q_0)$. Wir berechnen $r_{\{q_0\}}(q_0,q_0)$ und eliminieren jetzt auch q_0 und können dann direkt den Basisfall einsetzen. Dabei vereinfachen wir die regulären Ausdrücke noch:

$$\begin{aligned}
r_{\{q_0\}}(q_0,q_0) &= r_\varnothing(q_0,q_0) + r_\varnothing(q_0,q_0) r_\varnothing(q_0,q_0)^* r_\varnothing(q_0,q_0)\\
&= \varepsilon + \varepsilon\varepsilon^*\varepsilon\\
&\equiv \varepsilon.
\end{aligned}$$

Wir berechnen nun $r_{\{q_0\}}(q_0,q_1)$ und eliminieren wieder q_0:

$$\begin{aligned}
r_{\{q_0\}}(q_0,q_1) &= r_\varnothing(q_0,q_1) + r_\varnothing(q_0,q_0) r_\varnothing(q_0,q_0)^* r_\varnothing(q_0,q_1)\\
&= c + \varepsilon\varepsilon^* c\\
&\equiv c.
\end{aligned}$$

Jetzt berechnen wir: $r_{\{q_0\}}(q_1,q_1)$ und eliminieren wieder q_0:

$$\begin{aligned}
r_{\{q_0\}}(q_1,q_1) &= r_\varnothing(q_1,q_1) + r_\varnothing(q_1,q_0) r_\varnothing(q_0,q_0)^* r_\varnothing(q_0,q_1)\\
&= (a+b+\varepsilon) + a\varepsilon^* c\\
&\equiv a+b+\varepsilon+ac.
\end{aligned}$$

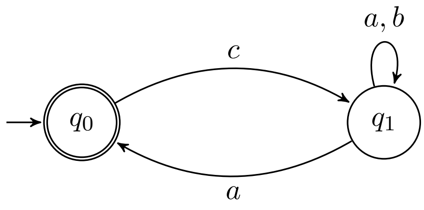


Abbildung 13: NFA $\mathcal{A}$ für Beispiel 3.47.

Zuletzt berechnen wir $r_{\{q_0\}}(q_1, q_0)$ und eliminieren wieder den einzigen verbleibenden Zustand q_0:

$$\begin{aligned} r_{\{q_0\}}(q_1, q_0) &= r_\varnothing(q_1, q_0) + r_\varnothing(q_1, q_0) r_\varnothing(q_0, q_0)^* r_\varnothing(q_0, q_0) \\ &= a + a\varepsilon^*\varepsilon \\ &\equiv a. \end{aligned}$$

Diese Ausdrücke können wir nun rückwärts wieder einsetzen:

$$\begin{aligned} r_\mathcal{A} = r_Q(q_0, q_0) &= \varepsilon + c(a + b + \varepsilon + ac)^* a \\ &\equiv \varepsilon + c(a + b + ac)^* a. \end{aligned}$$

Da der Automat recht einfach ist, lässt sich hier natürlich auch ein einfacher Ausdruck einfach ablesen:

$$r'_\mathcal{A} = (c(a + b)^* a)^* \equiv r_\mathcal{A}.$$

Die beiden Lemmata 3.43 und 3.46 ergeben zusammen den Äquivalenzsatz von Kleene. Mit den Sätzen 3.6 und 3.8 können wir auch schließen, dass sich die regulären Ausdrücke auch um Operatoren für Komplement und Schnitt erweitern ließen, ohne dadurch zusätzliche Sprachen zu beschreiben. Dies ist nicht offensichtlich: Dazu versuche man einmal Komplement und Schnitt nur mit Hilfe von Vereinigung (+), Konkatenation ($\cdot$) und Iteration (*) zu simulieren.
Wir werden nun anstelle von FA-erkennbaren Sprachen auch nur noch von regulären Sprachen sprechen.

3.7 Weitere Abschlusseigenschaften

Wir haben bis hier hin reguläre Sprachen eine umfangreiche Charakterisierung gegeben, nämlich als Sprachen, die sich durch reguläre Ausdrücke oder Sprachen von endliche Automaten beschreiben lassen – auf die Äquivalenz dieser Charakterisierungen haben wir lange hingearbeitet und mit dem Äquivalenzsatz von Kleene einen schweren Brocken der theoretischen Informatik bewiesen. Dieser Abschnitt wird nun wieder deutlich einfacher. Wir haben bereits einige Abschlusseigenschaften regulärer Sprachen bewiesen: Schnitt, Vereinigung, Komplement, Konkatenation, Iteration usw., aber auch unter "erfundenen" Operatoren wie *Perfect Shuffle*. Diese Abschlüsse allein zusammen mit dem Äquivalenzsatz von Kleene ergeben bereits genug Gründe um reguläre Sprachen als eine in wissenschaftlicher Hinsicht *sinnvolle* Sprachklasse zu sehen. Wir werden in diesem Abschnitt nun ein paar weitere sinnvolle Abschlusseigenschaften beweisen.

Definition 3.48. Seien Σ, Γ Alphabete. Die Abbildung $h\colon \Sigma^* \to \Gamma^*$ heißt *(Wort-)Homomorphismus*, wenn für alle $a_0 \dots a_{n-1} \in \Sigma^*$ gilt, dass

$$h(a_0 \dots a_{n-1}) = h(a_0) \dots h(a_{n-1}).$$

Insbesondere gilt bei einem Homomorphismus h, dass $h(uv) = h(u)h(v)$ und somit auch $h(\varepsilon) = \varepsilon$.

Definition 3.49. Sei $L \subseteq \Sigma^*$ eine Sprache und $h\colon \Sigma^* \to \Gamma^*$ ein Homomorphismus. Dann ist

$$h(L) \coloneqq \{h(w) : w \in L\}$$

die *Sprache unter dem Homomorphismus* h von L (über Γ).

Satz 3.50. *Sei $L \subseteq \Sigma^*$ eine reguläre Sprache und $h\colon \Sigma^* \to \Gamma^*$ ein Homomorphismus. Dann ist auch $h(L) \subseteq \Gamma^*$ regulär.*

Beweis. Sei $L \subseteq \Sigma^*$ regulär und h Homomorphismus auf Σ. Da L regulär existiert ein regulärer Ausdruck $e \in \mathsf{RE}_\Sigma$ mit $[\![e]\!] = L$. Wir können auf dem regulären Ausdruck den Homomorphismus anwenden, wobei wir Klammern, Punkte und Sterne ignorieren, dazu definieren wir die Erweiterung $\hat{h}$ von h mit $\hat{h}|_\Sigma \equiv h$ und $\hat{h}(\sigma) = \sigma$ für $\sigma \in \{(,),\boldsymbol{+},\boldsymbol{\cdot},\boldsymbol{*}\}$. Aus der Definition 3.48 geht hervor, dass es genügt den Homomorphismus auf den einzelnen Symbolen $a \in \Sigma$ anzuwenden. Im Detail heißt das:

- $\hat{h}(\boldsymbol{\varnothing}) = \boldsymbol{\varnothing}$,
- $\hat{h}(\boldsymbol{a}) = \boldsymbol{b}_0 \dots \boldsymbol{b}_{n-1}$, wenn $h(a) = b_0 \dots b_{n-1}$,
- $\hat{h}((r \boldsymbol{+} r')) = ((\hat{h}(r)) \boldsymbol{+} (\hat{h}(r')))$,
- $\hat{h}((r \boldsymbol{\cdot} r')) = ((\hat{h}(r)) \boldsymbol{\cdot} (\hat{h}(r')))$,
- $\hat{h}((r)\boldsymbol{*}) = (\hat{h}(r))\boldsymbol{*}$.

Es bleibt zu zeigen, dass $[\![\hat{h}(e)]\!] = h(L)$. □

Wir betrachten nun eine weitere Abbildung:

Definition 3.51. Die *Reverse*-Operation ist definiert als

$$\cdot^{\mathcal{R}}\colon \Sigma^* \to \Sigma^*, \qquad a_0 a_1 \dots a_{n-1} \mapsto a_{n-1} \dots a_1 a_0.$$

Wir können sie auf Sprachen $L \subseteq \Sigma^*$ natürlich erweitern durch

$$L^{\mathcal{R}} \coloneqq \{w^{\mathcal{R}} : w \in L\}.$$

Beachte, dass $w^{\mathcal{R}}$ wohldefiniert ist für jedes $w \in \Sigma^*$ da jedes w eine eindeutige Darstellung besitzt im freien Monoid $(\Sigma^*, \cdot)$.

Bemerkung. Die Reverse-Operation ist kein Homomorphismus für $|\Sigma| > 1$, denn es gilt für z.B: $abb = (bba)^{\mathcal{R}}$. Wäre $\cdot^{\mathcal{R}}$ Homomorphismus müsste aber gelten $(bba)^{\mathcal{R}} = b^{\mathcal{R}}b^{\mathcal{R}}a^{\mathcal{R}} = bba \neq abb$. Eine zeichenweise Anwendung von $\cdot^{\mathcal{R}}$ ist also nicht möglich.

Satz 3.52. *Sei $L \subseteq \Sigma^*$ regulär. Dann ist auch $L^{\mathcal{R}}$ regulär.*

Beweis. Übung. □

3.8 Nicht-reguläre Sprachen

Bisher haben wir nur sehr einfache Sprachen betrachtet, die alle regulär waren. D.h. wir konnten zu jeder bisher betrachteten Sprache einen regulären Ausdruck angeben oder einen endlichen Automaten. Man könnte auf die Idee kommen, dass alle formalen Sprachen regulär sind. Dies ist aber nicht der Fall. Dass es mehr Sprachen als nur reguläre Sprachen geben muss folgt bereits aus kombinatorischen Argumenten. Dies heben wir uns aber für ein späteres Kapitel noch auf. Schauen wir uns zunächst ein Beispiel für eine nicht-reguläre Sprache an:

$$L_{\mathbb{P}} := \{a^p : p \in \mathbb{P}\}.$$

$L_{\mathbb{P}}$ ist die Sprache aller (mit a) unär codierten Primzahlen, d.h. $L_{\mathbb{P}} = \{a^2, a^3, a^5, a^7, a^{11}, \ldots\}$. Dass $L_{\mathbb{P}}$ nicht regulär ist, ist auch nicht so schwer einzusehen: Intuitiv, wie würde ein regulärer Ausdruck aussehen? Natürlich könnten wir eine unendliche Vereinigung bilden (dies wäre jedoch nicht regulär, da reguläre Ausdrücke endlich sind):

$$r_{\mathbb{P}} = \sum_{p \in \mathbb{P}} a^p = a^2 + a^3 + a^5 + a^7 + a^{11} + \ldots$$

Es erscheint klar, dass es jedoch keine Möglichkeit gibt diesen Ausdruck zu vereinfachen. In einem Automaten wird es noch deutlicher. Wenn wir mit endlich vielen Zuständen auskämen um genau alle unendlich vielen Primzahlcodierungen zu erkennen würden wir Kreise im Transitionsgraphen und somit eine gewisse Regelmäßigkeit in der Wiederholung von Primzahlen bekommen. Dies würde bedeuten, dass die Generierung neuer Primzahlen viel einfacher wäre als bisher angenommen, da wir ab einer bestimmten Primzahl einfach immer eine Konstante addieren könnten um die nächste zu bekommen. Das ist natürlich nicht der Fall. Eine schematische Erklärung dieses Arguments ist in Abbildung 14. Egal an welcher und an wie vielen Stellen wir eine Zustandwiederholung zulassen (durch gestrichtelte Kante angedeutet), wir würden stets Codierungen akzeptieren, die nicht zu Primzahlen gehören können. In diesem Abschnitt werden wir uns ansehen, wie wir dieses Argument formalisieren und auf weitere nicht-reguläre Sprachen anwenden können. Vorab: Mit diesem Abschnitt der Vorlesung haben viele Studenten Probleme, da das später vorgestellte Lemma etwas sperrig ist. Tatsächlich basiert dieses Lemma jedoch auf dem wahrscheinlich einfachsten Prinzip der Mathematik. Wir führen die Idee des Lemmas anhand dieses Prinzips einmal vor. Man sollte dieses Beispiel zunächst verstehen, dann sollte der Schritt zum Lemma geradezu trivial wirken.

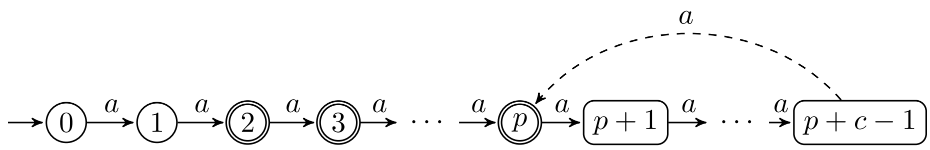


Abbildung 14: Wenn $L_{\mathbb{P}}$ regulär wäre gäbe es unendlich viele Primzahlen, die sich als $p + ck$ darstellen lassen, wobei $p \in \mathbb{P}, c \in \mathbb{N}, k = 0, 1, 2, \ldots$.

Beispiel 3.53. Die Sprache $L = \{a^n b^n : n \in \mathbb{N}\}$ ist nicht regulär.

Beweis. Angenommen L wäre regulär. Dann existiert ein endlicher Automat $\mathcal{A}$ mit $L(\mathcal{A}) = L$. Wir nehmen an, dass $\mathcal{A}$ n Zustände besitzt. Betrachten wir nun das Wort $w = a^n b^n$ Offenbar ist $w \in L$, also gibt es einen akzeptierenden Lauf $\varrho = (r_0, \ldots, r_{2n})$. Das Wort hat die Länge $|w| = 2n \geq n$. Das heißt aber, dass spätestens nach Lesen des letzten as sich im Lauf von $\mathcal{A}$ ein Zustand wiederholt haben muss. Dies folgt aus dem sogenannten *pigeonhole principle* (oder deutsch: *Schubfachprinzip*): Der Lauf auf den ersten n Zeichen ist $(r_0, \ldots, r_n)$, d.h. er besteht aus $n + 1$ Zuständen. Da $\mathcal{A}$ nur n Zustände hat, muss mindestens einer doppelt im Lauf vorkommen. Wir nehmen an, dass $r_i = r_j$ mit oBdA $i < j$. Es ist auch klar, dass bis zu dieser Wiederholung im j-ten Schritt nur as gelesen wurden, da die ersten n Symbole von w alle a sind (d.h. $j \leq n$). Betrachten wir nun folgenden Lauf

$$\varrho' = (r_0, \ldots, r_i, r_{j+1}, \ldots, r_{2n})$$

auf dem Wort $w' = a^{n-k} b^n$ mit $k := j - i > 0$. Der Lauf entsteht also durch das Weglassen der as die sonst zwischen der Zustandswiederholung gelesen worden wären. Dieser Lauf ist gültig auf dem Automaten, denn $r_i = r_j$, d.h. die selben Transitionen die im j-ten Schritt möglich waren sind auch bereits im i-ten Schritt möglich. Außerdem gilt, dass ϱ' ein akzeptierender Lauf ist, da er auf $r_{2n} \in F$ endet. Also akzeptiert $\mathcal{A}$ das Wort w', welches aber echt weniger as als bs besitzt (denn $k > 0$, weil $i < j$), also ist $w' \notin L$. Der Automat $\mathcal{A}$ erkennt also nicht L. Da $\mathcal{A}$ und die Anzahl seiner Zustände n belieibig gewählt wurde, existiert also kein endlicher Automat für L. Somit ist L nicht regulär. □

Bemerkung. Statt die as zwischen der Zustandswiederholung wegzulassen hätten wir natürlich auch beliebig viele weitere Wiederholungen hinzufügen können. Wir würden also den Lauf

$$\varrho'' = (r_0, \ldots, r_i, \ldots, r_j, r_{i+1} \ldots r_j, \ldots, r_{j+1}, \ldots, r_{2n})$$

betrachten zu dem Wort $w'' = a^{n+hk} b^n$ für ein beliebiges $h > 0$.

Dieses Beispiel ist sehr ausführlich jetzt besprochen worden. Sobald man diese Überlegungen verstanden hat, hat man diesen gesamten Abschnitt eigentlich auch schon verstanden und sollte auch beim folgendem Lemma nicht mehr allzu große Schwierigkeiten haben. Tatsächlich bildet das Beispiel den Beweis des Lemmas auf einen einzelnen Automaten bzw. eine einzelne Sprache ab.

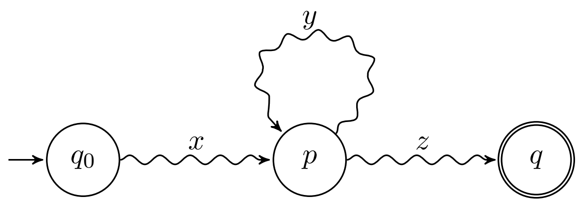


Abbildung 15: Zustandswiederholung in p. Es gibt also einen Lauf von q_0 nach p und einen Lauf von p nach $q \in F$. Den Zwischenlauf von p nach p kann man also weglassen (oder auch beliebig oft wiederholen).

Lemma 3.54 (Pumping-Lemma). *Sei $L \subseteq \Sigma^*$ eine reguläre Sprache. Dann existiert ein $n \in \mathbb{N}_+$, sodass für alle $w \in L$ mit $|w| \geq n$ eine Zerlegung $w = xyz$ existiert mit den Eigenschaften:*

(i) $|xy| \leq n$,

(ii) $y \neq \varepsilon$,

(iii) $xy^iz \in L$ *für alle* $i \in \mathbb{N}$.

Beweis. Sei $L \subseteq \Sigma^*$ eine beliebige reguläre Sprache und $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NFA mit $L(\mathcal{A}) = L$. Wir setzen $n := |Q|$. Betrachte nun ein belibiges Wort $w = a_0 \dots a_{m-1} \in L$ mit Länge $|w| =: m \geq n$. Falls kein solches Wort existiert sind wir bereits fertig. Ansonsten betrachten wir den Lauf $\varrho = (r_0, \dots, r_m)$ von $\mathcal{A}$ auf w. Da $w \in L = L(\mathcal{A})$ ist $r_m \in F$. Wegen $m \leq n$ muss eine Zustandswiederholung in ϱ vorkommen, spätestens nach Lesen des n-ten Zeichens. Seien also die Zustände r_k und r_j mit $0 \leq k < j \leq n$ gleich. Wir wählen die Zerlegung $x := a_0 \dots a_{k-1}, y := a_k \dots a_{j-1}, z := a_j \dots a_m$ und zeigen, dass diese die gewünschten Eigenschaften hat:

(i) $|xy| = |a_0 \dots a_{j-1}| = j \leq n$. ✓

(ii) $y = a_k \dots a_{j-1} \neq \varepsilon$, da nach Annahme $j - k > 0$. ✓

(iii) Die Aussage gilt offensichtlich für $i = 1$. Für $i = 0$ betrachten wir nun den Lauf für das Wort $xz = xy^0z$: $(r_0, \dots, r_k, r_{j+1}, \dots, r_m)$. Dieser Lauf ist tatsächlich korrekt, denn $r_k = r_j =: r$ und somit gilt $(r_k, a_j, r_{j+1}) = (r_j, a_j, r_{j+1}) \in \Delta$ (alle anderen Transitionen sind nach Annahme trivialerweise ebenfalls in Δ). Außerdem gilt weiterhin $r_m \in F$. Also akzeptiert $\mathcal{A}$ das Wort xz. Ansonsten haben wir, dass $r \xrightarrow{y} r$ und somit auch $r \xrightarrow{y^i} r$ für $i \geq 2$.✓ □

Bemerkung. Wir stellen wieder fest, dass der Beweis des Pumping-Lemmas eine Verallgemeinerung des Vorgehens aus Beispiel 3.53 ist. Wir betrachten auch wieder einen Automaten mit n Zuständen und akzeptierte Wörter, die zu lang sind, als dass sie dann ohne Wiederholung erkannt werden könnten. Das Weglassen oder Wiederholen von Infixen (*pumpen*) entspricht dem Weglassen oder Wiederholen im Lauf. Eine Visualisierung des Prinzip ist in Abbildung 15.

Bemerkung. Wir können mit dem Pumping-Lemma nicht zeigen, dass eine Sprache regulär ist. Es ist lediglich eine notwendige Bedingung für Regularität, aber keine hinreichende Bedingung. D.h. es gibt Sprachen, die nicht regulär sind, aber dennoch dem Pumping-Lemma genügen, z.B.

$$K = \{a^m b^n c^n : m \in \mathbb{N}_+, n \in \mathbb{N}\} \cup \{b^m c^n : m, n \in \mathbb{N}\}.$$

Beispiel 3.55. Wir betrachten die Sprache

$$L = \{ww^{\mathcal{R}} : w \in \Sigma^*\}$$

aller Palindrome gerader Länge über $\Sigma = \{a, b\}$. L ist nicht regulär.

Beweis. Angenommen L wäre regulär. Dann gilt das Pumping-Lemma. Sei also $n \in \mathbb{N}_+$ beliebig und wir betrachten das Wort $w = a^n bba^n \in L$. Es hat die Länge $|w| = 2n+2 \geq n$. Wir betrachten nun Zerlegungen, die die Eigenschaften (i) und (ii) erfüllen und zeigen, dass (iii) dann nicht gelten kann. Wegen (i) ist $y = a^k$ für ein $k \leq n$ ($k = |y| \leq |xy| \leq n$, da die ersten n Zeichen alle a sind, kommt also auch kein b in y vor). Wegen (ii) gilt außerdem, dass $k > 0$. Betrachte nun das Wort $w' := xyyz = a^{n+k} bba^n$. Es liegt nicht in L, da w' kein Palindrom ist. Damit gilt die Eigenschaft (iii) für $i = 2$ nicht (gilt sogar für gar kein $i \neq 1$). Dies ist ein Widerspruch zum Pumping-Lemma, also kann L nicht regulär sein. □

Es ist auch möglich, das Pumping-Lemma als Spiel aufzufassen. Das Vorgehen ist dabei identisch zu Model-Checking-Spielen für First Order Logic (*Prädikatenlogik*), die man in der Vorlesung Mathematische Logik (normalerweise im 4. Semester) sieht. Wir beginnen mit den Spielern:

Alice versucht die Beweisführung der Nicht-Regualarität zu sabotieren,
(Verifiziererin, Defender)

Bob möchte zeigen, dass eine Sprache nicht regulär ist.
(Falsifizierer, Attacker)

Die Spielregeln gehen wie folgt:

1) Alice wählt eine Zahl $n \in \mathbb{N}_+$.

2) Bob wählt ein Wort $w \in L$ mit $|w| \geq n$.

3) Alice wählt eine Zerlegung $xyz = w$, mit $|xy| \leq n$ und $y \neq \varepsilon$.

4) Bob wählt eine Zahl $i \in \mathbb{N}$.

Alice gewinnt, wenn $xy^iz \in L$ oder Bob in Schritt 2 nicht ziehen kann (das wäre der Fall wenn jedes Wort aus L kürzer ist als das n, welches Alice vorgegeben hat – die Sprache wäre dann endlich und somit regulär). Bob gewinnt, wenn $xy^iz \notin L$.
Falls L regulär ist, so hat Alice eine Gewinnstrategie, d.h. sie kann ein n wählen, sodass sie für jedes Wort, mit dem Bob in Schritt 2 antworten könnte eine Zerlegung findet, die Bob in Schritt 4 nicht aus der Sprache rauspumpen kann. Äquivalent dazu: Falls Bob eine Gewinnstrategie hat, so ist L nicht regulär.

Bemerkung. Beachte, dass in Schritt 3 *Alice* diejenige ist, die eine Zerlegung wählt. Wenn wir zeigen wollen, dass Bob eine Gewinnstrategie hat, heißt das, dass wir für *alles* was Alice wählen könnte im 4. Schritt noch aus der Sprache herausgepumpt werden kann. Es wäre keine Gewinnstrategie, wenn Bob nur auf manche Züge von Alice eine passende Antwort findet.

Beispiel. Wir kehren einmal zur Sprache $L_{\mathbb{P}}$ vom Anfang des Abschnitts zurück und geben eine Gewinnstrategie für Bob an um zu zeigen, dass $L_{\mathbb{P}}$ nicht regulär ist:

1) Alice wählt eine Zahl $n \in \mathbb{N}_+$.
2) Bob wählt das Wort a^q mit $q \geq n$ und $q \in \mathbb{P}$ (ein solches q existiert wegen Satz 1.39).
3) Alice wählt eine Zerlegung $xyz = a^q$, mit $|xy| \leq n$ und $y \neq \varepsilon$, d.h. $y = a^k$ für ein $0 < k \leq n$.
4) Bob wählt die Zahl $i = q + 1$.

Wir zeigen, dass $xy^{q+1}z \notin L_{\mathbb{P}}$:

$$\begin{aligned} |xy^{q+1}z| &= |xyz| + |y^q| \\ &= |a^q| + q|y| \\ &= q + qk \\ &= q(1+k). \end{aligned}$$

Da $k > 0$ ist $1 + k > 1$ und somit $q(1 + k)$ eine zusammengesetzte Zahl (d.h. nicht prim). Also ist $xy^{q+1}z \notin L_{\mathbb{P}}$. Wir erhalten also eine Gewinnstrategie für Bob und somit ist $L_{\mathbb{P}}$ nicht regulär.

3.9 Myhill-Nerode-Äquivalenz

Bisher haben wir reguläre Sprachen durch Formalismen wie reguläre Ausdrücke und endliche Automaten charakterisiert. Nun wollen wir eine Eigenschaft finden, die reguläre Sprachen auf einer stark mathematisch-strukturellen Ebene charakterisiert. Dazu erinnern wir uns an unser freies Monoid $(\Sigma^*, \cdot, \varepsilon)$, welches bereits in Abschnitt 3.7 uns einmal angesehen haben. Wir definieren uns eine Äquivalenzrelation (siehe Definition 1.7), d.h. wir teilen zu einem Alphabet Σ die Wörter aus Σ^* in Äquivalenzklassen ein. Tatsächlich definiert die gleich eingeführte Relation sogar noch etwas mehr, deshalb schieben wir eine weitere Definition hier ein.

Definition 3.56. Sei A eine Menge und $\bullet$ eine zweistellige Funktion auf A. Eine Relation $\sim \subseteq A \times A$ heißt *rechtsseitige Kongruenz* (bezüglich $\bullet$), wenn

(i) $\sim$ eine Äquivalenzrelation ist und

(ii) $\bullet$ diese Relation respektiert (d.h. für alle $a \sim b$ und alle $c \in A$ gilt $a \bullet c \sim b \bullet c$).

Wir definieren nun wie angekündigt unsere Äquivalenzrelation auf Wörtern.

Definition 3.57. Sei Σ ein Alphabet und $L \subseteq \Sigma^*$. Seien $u, v \in \Sigma^*$. u und v heißen *Myhill-Nerode-äquivalent bezüglich* L ($u \sim_L v$) genau dann, wenn für alle $w \in \Sigma^*$ gilt, dass $uw \in L$ g.d.w. $vw \in L$.

Diese Relation ist tatsächlich sogar nicht nur eine Äquivalenz sondern auch eine rechtsseitige Kongruenz bezüglich $\cdot$, deswegen sind auch die Begriffe *Myhill-Nerode-Rechtskongruenz* und – nicht ganz korrekterweise – *Myhill-Nerode-Kongruenz* gebräuchlich. Dies wollen wir im folgenden zeigen. Eine einfacher Folgerung aus der Definition ist

Lemma 3.58. *Sei $L \subseteq \Sigma^*$ eine Sprache. Für alle $u, v, w \in \Sigma^*$ mit $u \sim_L v$ gilt $uw \sim_L vw$.*

Beweis. Wir zeigen zunächst, dass $\sim_L$ eine Äquivalenzrelation ist. Offensichtlich ist $\sim_L$ reflexiv und symmetrisch. Angenommen nun $u \sim_L v$ und $v \sim_L w$, aber $u \not\sim_L w$, d.h. es gibt ein trennendes Wort x mit oBdA $u \cdot x \in L$ und $w \cdot x \notin L$. Dann ist $v \cdot x \notin L$ wegen $v \sim_L w$. Daraus folgt aber auch, dass $u \not\sim_L v$↯.
Sei nun $u \sim_L v$ und $w \in \Sigma^*$ beliebig. Angenommen $uw \not\sim_L vw$, d.h. es gibt ein trennendes Wort $x \in \Sigma^*$ mit oBdA $uw \cdot x \in L$ und $vw \cdot x \notin L$. Dann ist aber wx ein trennendes Wort für u und v und somit $u \not\sim_L v$↯.
Ingesamt gilt also, dass $\sim_L$ rechtsseitige Kongruenzrelation bezüglich $\cdot$ ist. □

Wir verwenden etwas vereinfachte Schreibweisen für Myhill-Nerode-Äquivalenz: Statt $[w]_{\sim_L}$ schreiben wir nur $[w]_L$ und für $\Sigma^*/_{\sim_L}$ schreiben wir $\Sigma^*/_L$. Außerdem schreiben wir $\mathrm{index}(L)$ für $\mathrm{index}(\sim_L)$.

Beispiel 3.59. Wir bestimmen die Myhill-Nerode-Äquivalenzklassen und den Index der Sprache
$$L = \{w \in \{a, b\}^* \mid w \text{ hat Infix } ab\}.$$
Wir betrachten $\sim_L$ und die Wörter über $\{a, b\}$:

- $\varepsilon \not\sim_L a$, denn b ist trennendes Wort: $\varepsilon \cdot b = b \notin L$, aber $a \cdot b = ab \in L$.
- $\varepsilon \sim_L b$, denn es gilt $\varepsilon \cdot w \in L$ g.d.w. $b \cdot w \in L$, nämlich genau dann, wenn w das Infix ab beinhaltet.
- $a \sim_L aa$, denn es gilt $a \cdot w \in L$ g.d.w. $aa \cdot w \in L$, nämlich genau dann, wenn w mit b beginnt oder das Infix ab beinhaltet.
- $a \not\sim_L ab$, denn ε ist trennendes Wort: $a \cdot \varepsilon = a \notin L$, aber $ab \cdot \varepsilon = ab \in L$.
- $\varepsilon \not\sim_L ab$, denn ε ist trennendes Wort: $\varepsilon \cdot \varepsilon = \varepsilon \notin L$, aber $ab \cdot \varepsilon = ab \in L$.

Wir behaupten nun, dass $[\varepsilon]_L, [a]_L, [ab]_L$ alle Myhill-Nerode-Äquivalenzklassen sind (und damit, dass $\mathrm{index}(L) = 3$ ist).

Beweis. Bereits gezeigt haben wir, dass diese drei Myhill-Nerode-Äquivalenzklassen jeweils verschieden sind. Es bleibt zu zeigen, dass $[\varepsilon]_L \cup [a]_L \cup [ab]_L = \{a,b\}^*$, also dass $\{[\varepsilon]_L, [a]_L, [ab]_L\}$ eine Partition von $\{a,b\}^*$ ist. Wir zeigen, dass für jedes Wort $v \in \{a,b\}^*$ eine der drei Möglichkeiten gilt $v \sim_L \varepsilon, v \sim_L a, v \sim_L ab$.

1) v enthält ab als Infix. Dann gilt $v \cdot x \in L$ für alle $x \in \{a,b\}^*$. Auch $ab \cdot x \in L$ gilt für alle $x \in \{a,b\}^*$, somit $v \sim_L ab$.

2) v enthält ab nicht als Infix und endet mit a. Dann $v \cdot x \in L$ g.d.w. x mit b beginnt oder ab als Infix enthält. Für genau die selben x gilt $a \cdot x \in L$. Also ist $v \sim_L a$.

3) Die oberen beiden Fälle treffen nicht zu, d.h. v enthält ab nicht als Infix und endet auch nicht mit a (d.h. $v \in [\![b^*]\!]$). Dann ist $v \cdot x \in L$ g.d.w. x das Infix ab enthält. Offensichtlich sind das genau die selben x mit denen man auch von ε in L landet, somit $v \sim_L \varepsilon$.

Jedes Wort fällt in eines dieser drei Fälle, also haben wir die Behauptung gezeigt. Außerdem liefert uns diese Betrachtung eine Beschreibung der Äquivalenzklassen:

1) $[ab]_L = L = [\![(a+b)^*ab(a+b)^*]\!]$,

2) $[a]_L = \{w \mid w \text{ enthält kein } ab \text{ und endet auf } a\} = [\![b^*a^+]\!]$,

3) $[\varepsilon]_L = \{w \mid w \text{ enthält kein } ab \text{ und endet nicht auf } a\} = [\![b^*]\!]$.

Außerdem folgt natürlich, dass $\mathrm{index}(L) = 3$. □

In dem Beispiel haben wir bereits auf eine sehr algorithmische Weise die Myhill-Nerode-Äquivalenzklassen bestimmt. Auch wenn wir nochmal bewiesen haben, dass die drei angegebenen Äquivalenzklassen tatsächlich die geforderte Partition bilden, so war dies nicht mehr wirklich nötig mit der vorangegangenen Betrachtung. Denn Lemma 3.58 lieferte bereits die Aussage, dass es nicht mehr Äquivalenzklassen geben kann:

- $\varepsilon \sim_L bb$, denn $\varepsilon \sim_L b$ und (wegen Lemma 3.58) $\varepsilon \cdot b \sim_L b \cdot b$. Mit Transitivität folgt auch $\varepsilon \sim_L bb$.
- $\varepsilon \not\sim_L aa$, denn $aa \sim_L a \not\sim_L \varepsilon$. Aus Symmetriegründen folgt dann auch $aa \not\sim_L \varepsilon$.
- $a \sim_L aaa$, denn $a \sim_L aa$ und damit auch $a \cdot a \sim_L aa \cdot a$. Mit Transitivität erhalten wir dann auch wieder $a \sim_L aaa$ und induktiv auch $a \sim_L a^k$ für beliebige $k \in \mathbb{N}_+$.

Das bedeutet, wenn man bereits für alle Wörter bis zu einer bestimmten Länge alle Myhill-Nerode-Äquivalenzklassen gefunden hat und für Wörter der nächstgrößeren Länge keine neuen mehr hinzugekommen sind, so hat man bereits alle Klassen gefunden. Alle noch längeren Wörter sind dann bereits ebenfalls äquivalent zu einem kürzeren Wort.

Lemma 3.60. *Sei $L \subseteq \Sigma^*$ eine Sprache und $w \in \Sigma^*$. Falls w kleinster Repräsentant (bzgl. $\prec_{lex}$) einer Myhill-Nerode-Äquivalenzklasse $[w]_L$ ist, so ist bereits jedes Präfix von w ebenfalls kleinster Repräsentant seiner Myhill-Nerode-Äquivalenzklasse.*

Beweis. Sei $u \sqsubseteq w$, d.h. es gibt ein $x \in \Sigma^*$ mit $ux = w$, sodass $u \in [v]_L$ für ein $v \prec_{\text{lex}} u$. Dann ist aber $vx \in [w]_L$, aber $vx \prec_{\text{lex}} ux = w \lightning$. □

Korollar 3.61. *Sei $\mathcal{P}_L$ eine Menge von Myhill-Nerode-Äquivalenzklassen bzgl. einer Sprache $L \subseteq \Sigma^*$. Falls für alle $[u]_L \in \mathcal{P}_L$ und alle $a \in \Sigma$ gilt, dass $ua \sim_L w$ für ein $w \prec_{lex} ua$ mit $[w]_L \in \mathcal{P}_L$, so gilt bereits für alle $v \in a\Sigma^+, a \in \Sigma$, dass $uv \sim_L w$ für ein $w \prec_{lex} uv$ mit $[w]_L \in \mathcal{P}_L$. (Und somit gilt, dass $\bigcup_{w \in \mathcal{P}} [w]_L = \Sigma^*$.)*

Beweis. Angenommen für alle $[u]_L \in \mathcal{P}_L$ und alle $a \in \Sigma$ ist ua bereits Myhill-Nerode-äquivalent zu einem kleineren Repräsentanten $w \in \mathcal{P}$, aber es gibt ein $v \in a\Sigma^+$ mit $uv \notin [w]_L$ für alle $[w]_L \in \mathcal{P}$. Wähle v dabei minimal. uv repräsentiert also eine neue Myhill-Nerode-Äquivalenzklasse, d.h. es gibt kein lexikographisch kleineres Wort, welches zu $[uv]_L$ gehört. Nach vorigem Lemma ist dann aber auch jedes Präfix von uv kleinster Repräsentant einer Myhill-Nerode-Äquivalenzklasse. Insbesonder gilt das für ua. $\lightning$ zur Annahme, dass $[ua]_L$ einen kleineren Repräsentanten bereits hat. □

Dieses Korollar liefert intuitiv den Algorithmus 3 zum Finden von Myhill-Nerode-Äquivalenzklassen, denn wir wissen nun, dass wir alle Klassen gefunden haben, wenn ein Anhängen eines(!) beliebigen weiteren Symbols keine neue Klasse bringt. Beachte, dass

Algorithmus 3 : Induktives Finden von Myhill-Nerode-Äquivalenzklassen

Myhill-Nerode-Klassifizierer$L$
Input : Sprache $L \subseteq \Sigma^*$
Output : Myhill-Nerode-Äquivalenzklassen $\mathcal{P} = \{[w]_L : w \in \Sigma^*\}$.
$\mathcal{P} := \{[\varepsilon]_L\}$
for $[u]_L \in \mathcal{P}$ **do**
    for $a \in \Sigma$ **do**
        if $ua \not\sim_L v$ *für alle* $[v]_L \in \mathcal{P}$ **then**
            $\mathcal{P} := \mathcal{P} \cup \{[ua]_L\}$

dieses Verfahren nur für reguläre Sprachen terminiert. Nicht-reguläre Sprachen haben unendlich viele Äquivalenzklassen. Dies ist ein sehr starkes Resultat, denn es gibt uns eine notwendige und hinreichende Bedingung für Regularität und ist neben dem Äquivalenzsatz von Kleene sicher das schönste Theorem in dieser Vorlesung. Die Details geben wir im folgenden Satz.

Satz 3.62 (Satz von Nerode)**.** *Eine Sprache ist genau dann regulär, wenn die Anzahl der Myhill-Nerode-Äquivalenzklassen endlich ist.*

Die Sprache L im Beispiel 3.59 hat $\text{index}(L) = 3 < \infty$ und ist somit regulär. Wir betrachten nun ein Beispiel für eine nicht-reguläre Sprache.

Satz 3.63. *Der minimale DFA zu einer reguläre Sprache L ist isomorph zum Myhill-Nerode-DFA:*

$$\mathcal{A}_L = (\Sigma^*/_L, \Sigma, \delta, [\varepsilon]_L, \{[u]_L : u \in L\}),$$

mit $\delta([u]_L, a) = [ua]_L$.

3.10 Algorithmen für Reguläre Sprachen

3.11 Minimierung von DFAs

folgt.

3.12 Anwendung: First-Longest-Match-Analyse

Wir schauen uns hier einmal eine praktische Anwendung von regulären Ausdrücken und endlichen Automaten an. Bisher haben wir stets das *einfache Matching-Problem* für reguläre Ausdrücke betrachtet. Dabei ging es darum zu prüfen, ob ein gegebenes Wort $w \in \Sigma^*$ zur Sprache eines regulären Ausdrucks $r \in \mathsf{RE}_\Sigma$ gehört. Dies ließ sich mit Hilfe der Thompson-Konstruktion einfach als das Wortproblem eines ε-NFA betrachten. In der Praxis ist das einfach Matching-Problem aber zu primitiv um echte Probleme lösen zu können. Vorallem im Compilerbau benötigen wir etwas mehr, wenn wir Programmcode in seine Bestandteile zerlegen wollen um später Syntax-Checks darauf durchführen zu können und schließlich eine Semantik für das Programm festzulegen. In der Fachsprache nennt man diesen Teil eines Compilers auch *Scanner* oder *Lexer* (siehe auch die Programme `lex`, `flex`). In diesem Dokument verwenden wir die Variablen h, i, j, k, ℓ stets als natürliche Zahlen aus einer Menge $[m] \coloneqq \{1, \dots, m\}$. Manchmal ist die Notation etwas sloppy, aber hoffentlich verständlich genug.
Das *erweiterte (extended) Matching-Problem* kommt direkt aus der Anwendung des Compilerbaus. Gegeben ist ein Wort $w \in \Sigma^*$ und eine Reihe von regulären Ausdrücken $r_1, \dots, r_n \in \mathsf{RE}_\Sigma$. (*Bemerkung:* In der Praxis ist die Annahme $\varepsilon \notin L(r_i) \neq \varnothing$ für alle i sinnvoll). Gesucht ist nun eine Zerlegung des Wortes $w = u_1 \dots u_k$, wobei für jedes u_j ein r_{i_j} existieren muss, sodass $u_j \in L(r_{i_j})$. Man nennt die Zerlegung in $(u_1, \dots, u_k)$ auch *Dekomposition* und die zugehörigen Indizes der regulären Ausdrücke $(i_1, \dots, i_k)$ *Analyse* von w bezüglich $r_1, \dots, r_n$.
Wir sehen schnell, dass weder Dekomposition, noch Analyse eindeutig sein müssen (insbesondere dann, wenn wir $\varepsilon \in L(r_i)$ zulassen).

Beispiel 3.64. (i) $r_1 = a^+, w = aa$. Ergibt Dekompositionen (aa) und (a, a) mit zugehöriger (eindeutiger) Analyse (1) bzw. $(1, 1)$.

(ii) $r_1 = a + b, r_2 = a + c, w = a$. Ergibt eindeutige Dekomposition (a) mit zwei verschiedenen Analysen (1) und (2).

Im zweiten Fall stellen wir uns vor, dass r_1 mögliche Schlüsselwörter einer Programmiersprache (`while`, `true`, `if`) beschreibt und r_2 mögliche Variablennanem (Identifier), dann sieht man warum eine eindeutige Analyse wichtig ist. In diesem Anwendungsfall ist es natürlich zu sagen, dass der Ausdruck r_1 "wichtiger" ist als r_2 (deshalb verbieten die mesiten Programmiersprachen auch Schlüsselwörter als Identifier). Ähnliche Beispiele lassen sich auch für die Dekomposition finden.

Wir wollen nun sowohl, die Dekomposition, als auch die Analyse eindeutig machen. Ein erster Ansatz wäre zunächst das leere Wort zu verbieten und sicherzustellen, dass $L(r_i) \cap L(r_j) = \varnothing$ für alle $i \neq j$, indem wir den gemeinsamen Schnitt von zwei Ausdrücken aus dem "unwichtigeren" Ausdruck entfernen. Dadurch würde zumindest die Analyse eindeutig werden. Diese Idee ist jedoch nicht sinnvoll, u.a. deshalb, da das Entfernen des Schnittes – also das Erkennen von $L(r'_j) \coloneqq L(r_j) \setminus L(r_i)$ – eine Produktkonstruktion erfordert und somit tendenziell teuer ist.
Der am meisten verbreitete Weg ist es folgende zwei Prinzipien zu implementieren:

Longest Match Mache jedes u_i der Zerlegung so lang wie möglich.

First Match Wähle aus den matchenden regulären Ausdrücken den mit dem kleinesten Index.

Definition 3.65. Eine Dekomposition $(u_1, \dots, u_k)$ von $w \in \Sigma^*$ bezüglich der regulären Ausdrücke $r_1, \dots, r_n \in \mathsf{RE}_\Sigma$ heißt *Longest-Match-Dekomposition* (LMD), wenn für jedes $i \in [k]$, $x \in \Sigma^+$, $y \in \Sigma^*$ mit $w = u_1 \dots u_i x y$ gilt, dass kein $j \in [n]$ existiert, sodass $u_i x \in L(r_j)$.

Umgangssprachlich bedeutet das, dass wir an einen Teil der Komposition u_i kein nicht-leeres Wort mehr anhängen können, was noch nicht verarbeitet wurde. Es ist klar, dass eine LMD eindeutig ist, falls sie existiert. Sie muss jedoch nicht immer existieren:

Beispiel 3.66. $r_1 = a^+, r_2 = ab, w = aab$ hat Dekomposition (a, ab) aber keine LMD.

Definition 3.67. Sei $(u_1, \dots, u_k)$ eine LMD von $w \in \Sigma^*$ bezüglich der regulären Ausdrücke $r_1, \dots, r_n \in \mathsf{RE}_\Sigma$. Die zugehörige *First-Longest-Match-Analyse* (FLM-Analyse) $(i_1, \dots, i_k)$ ist gegeben durch

$$i_j \coloneqq \min\{\ell \mid u_j \in L(r_\ell)\}$$

für jedes $j \in [k]$.

Auch hier ist klar, dass es höchstens eine FLM-Analyse gibt und sie existiert, wenn die LMD existiert.
Es gibt viele Möglichkeiten eine FLM-Analyse durchzuführen. Die von mir hier vorgestellte Art ist nicht die von mir favorisierte, aber sie ist insofern intuitiv, als dass sie nur Konzepte aus der FoSAP-Vorlesung verwendet und ein wenig über Arrays.
Als Eingabe erhalten wir also $w \in \Sigma^*$ und die regulären Ausdrücke $r_1, \dots, r_n \in \mathsf{RE}_\Sigma$. Gesucht ist eine FLM-Analyse inklusive zugehöriger LMD oder ein Error, falls diese nicht existieren.
In Abbildung 16 seht ihr wie der ε-NFA der nach Zeile 3 entsteht aussieht.

Algorithmus 4 : First-Longest-Match-Analyse

<u>FLM</u>$(w, (r_i)_{i=1}^n)$

Output : FLM-Analyse $(i_j)_{j=1}^k$ und LMD $(u_j)_{j=1}^k$

`// Vorbereitung`

Wandle alle r_i mit Thompson-Konstruktion in ε-NFAs $\mathcal{A}_i = (Q_i, \Sigma, \delta_i, q_0^i, F_i)$ um.

Baue neuen ε-NFA $\mathcal{A} = (\biguplus_i Q_i \uplus \{q_0, q_f\}, \Sigma \uplus [n], \delta, q_0, \{q_f\})$ mit $\delta(p, a) = \delta_i(p, a)$ falls $p \in Q_i$ und $\delta(q_0, \varepsilon) = \{q_0^i \mid i \in [n]\}$ und $\delta(q, i) = \{q_f\}$, falls $q \in F_i$.

Eliminiere ε-Transitionen.

Determinisiere mit Potenzmengenkonstruktion.

Minimiere mit Markierungsalgorithmus.

`/* Wir haben jetzt einen DFA mit folgender Eigenschaft: Vom Zustand` $\hat{\delta}(q_0, u)$ `ist genau dann eine Transition mit` $i \in [n]$ `in einen Endzustand möglich, wenn` $u \in L(r_i)$`. */`

`// Löse nun das erweiterte Matching-Problem`

Setze $\ell = 1, w_\ell = w$.

while $w_\ell \neq \varepsilon$ **do**

- `A` = leeres Array der Länge $|w_\ell|$.
- Simuliere $\mathcal{A}$ auf w_ℓ, prüfe in jedem Schritt j, ob eine i-Transition in einen akzeptierenden Zustand möglich ist. Falls ja, setze `A[`j`]` auf das kleinste mögliche i.
- Nach der Simulation laufe `A` von hinten nach vorne durch bis wir den ersten nicht-leeren Eintrag an Stelle h finden. Sollte es keinen geben gebe einen Error aus, ansonsten ist $u_\ell = w_{\ell_1} \dots w_{\ell_h}$ und $i_\ell =$ `A[`h`]`.
- Setze $w_{\ell+1} = w_{\ell_h} \dots w_{\ell_{|w_\ell|}}, \ell = \ell + 1$.

`// Jetzt ist` $w_\ell = \varepsilon$`.`

Gebe $(u_1, \dots, u_{\ell-1})$ und $(i_1, \dots, i_{\ell-1})$ aus.

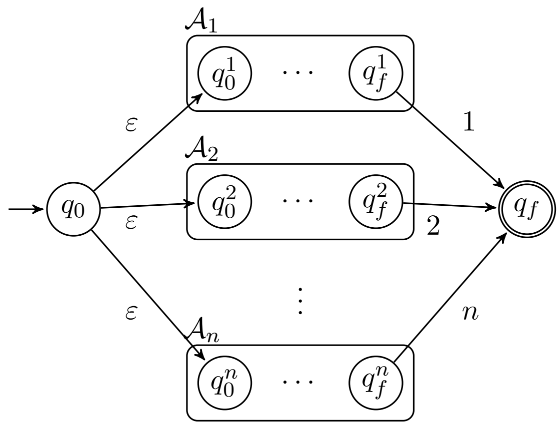


Abbildung 16: Skizze des NFAs aus Algorithmus 4.

4 Kellerautomaten und Kontextfreie Sprachen

5 Kontextsensitive Sprachen

6 Prozesskalküle und Petri-Netze