Rheinisch-Westfälische Technische Hochschule Aachen

Skript zur Vorlesung

Formale Systeme, Automaten, Prozesse

Letzte Änderung:
1. Juni 2017

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 **4**
1.1 Mengen 4
1.2 Operationen auf Mengen 6
1.3 Relationen 7
1.4 Gesetze für Mengen 9
1.5 Abbildungen 10
1.6 Strukturen 11
1.7 Graphen 11
1.8 Beweismethoden 11

2 Alphabete, Wörter, Sprachen **12**
2.1 Grundlegende Definitionen 12
2.2 Operationen und Relationen auf Wörtern und Sprachen . . . 13
2.3 Gesetze für Wörter und Sprachen 14

3 Endliche Automaten und Reguläre Sprachen **16**
3.1 Deterministische Endliche Automaten 16
3.2 Abschlusseigenschaften DFA-erkennbarer Sprachen 20
3.3 Nichtdeterministische Endliche Automaten 23
3.4 Äquivalenz von NFAs und DFAs 27
3.5 Reguläre Ausdrücke 29
3.6 Algorithmen für Reguläre Sprachen 31
3.7 Weitere Abschlusseigenschaften 31
3.8 Nicht-reguläre Sprachen 31
3.9 Myhill-Nerode-Äquivalenz 31

4 Kellerautomaten und Kontextfreie Sprachen **32**

5 Kontextsensitive Sprachen **33**

6 Prozesskalküle und Petri-Netze **34**

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 sämtliche Begriffe und Notationen aus dem ersten Semester bereits bekannt sein. Deshalb ist dieses Kapitel eher nur für Sommersemesteranfänger bestimmt.

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

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

die Menge aller Elemente aus M, die die Aussage A 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:

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

oder auch

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

Eine wichtige Menge haben wir bisher außen vor gelassen: die *leere Menge*. Wir schreiben $\emptyset := \{\}$. 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*.

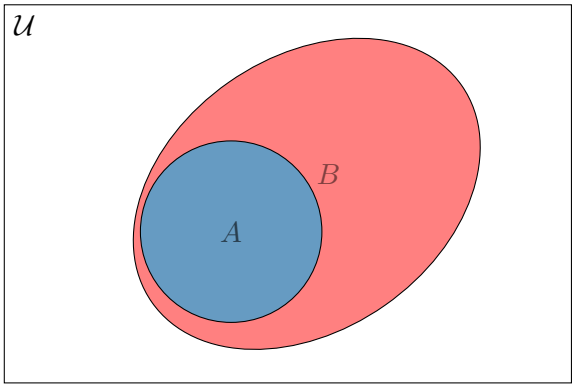


Abbildung 1: Venn-Diagramm für $A \subseteq B$.

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 $\mathbb{R}$ (solange man die komplexen Zahlen eben weglässt), 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

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

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

$$A \cap B := \{a \in A \mid a \in B\}.$$

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

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

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

$$A \setminus B := A \cap \overline{B}.$$

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

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

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

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

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

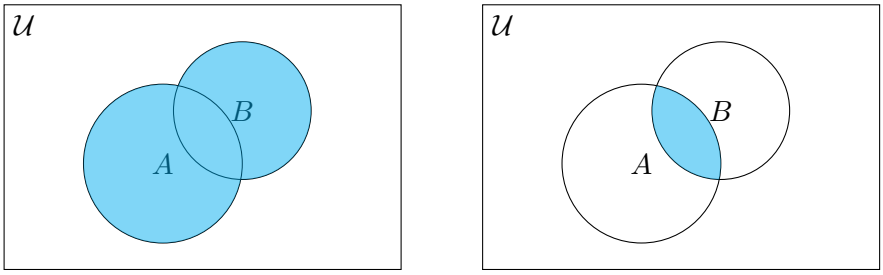
$$2^A := \{M \subseteq \mathcal{U} \mid M \subseteq A\}.$$

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.

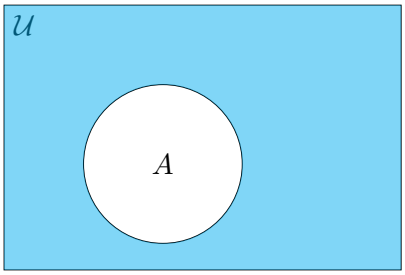
Definition 1.5. Es seien $M_1, \ldots, M_k$ nicht-leere Mengen. Eine Teilmenge $R \subseteq M_1 \times \ldots \times M_k$ heißt *Relation* zwischen $M_1, \ldots, M_k$ (oder auf M, falls $M = M_1 = \ldots = M_k$).

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

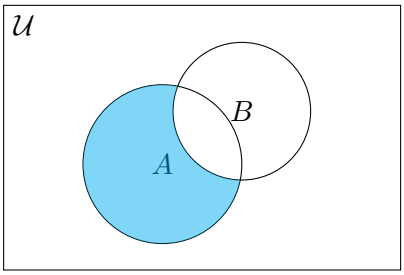


(a) $A \cup B$

(b) $A \cap B$



(c) $\overline{A}$



(d) $A \setminus B$

Abbildung 2: Venn-Diagramme für Mengen-Operationen.

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

- *reflexiv*, falls $x \sim x$ für alle $x \in M$,
- *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 $\{(a, b) \subseteq \mathbb{R}^2 \mid a^2 = b^2\}$ ist eine Äquivalenzrelation auf $\mathbb{R}$.

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

$$[x] := [x]_\sim := \{y \in M \mid x \sim y\}$$

die *Äquivalenzklasse* von x. Die Menge aller Äquivalenzklassen von $\sim$ wird notiert mit $M/_\sim := \{[x]_\sim : x \in M\}$.

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

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

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. □

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 Abbildungen

Definition 1.9. Seien M, N Mengen. Eine *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

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

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 Abbildungen von M nach N wird mit N^M bezeichnet.

Definition 1.10. Eine Abbildung $f : M \to N$ 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 Abbildung aufgefasst werden:

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

$+$ 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 Abbildungen $f : [k] \to M$ 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 $i \in [k]$.

1.6 Strukturen

1.7 Graphen

1.8 Beweismethoden

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 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

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

Für $n = 0$ ist $w : \emptyset \to \Sigma$ das *leere Wort*, was wir als ε bezeichnen.
Die *Länge* des Wortes w ist bezeichnet mit $|w| = n$.
$|w|_a := |\{i \in [n] \mid w(i) = a\}|$ ist 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 $\emptyset$.
- Die Sprache, die nur das leere Wort enthält $\{\varepsilon\}$.
- Die Sprache aller Binärdarstellungen von Primzahlen $\{bin(n) : n \in \mathbb{P}\}$.
- 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

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

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.

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 $A = \emptyset$) 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: $\{\varepsilon\} \neq \emptyset$.

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 $\emptyset L = L\emptyset = \emptyset$.

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

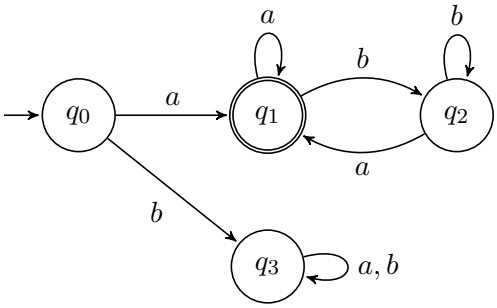


Abbildung 3: DFA für die Sprache aus Beispiel 3.5.

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, 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

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

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 $(r_0, a_0, r_1, a_1, \ldots, a_{n-1}, r_n)$ kürzen wir gelegentlich durch $(r_0, r_1, \ldots, r_n)$ 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

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

(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:

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

Beweis per vollständige Induktion über Wortlänge $n \in \mathbb{N}$. Ist $r = (r_0, \ldots, r_n)$ der Lauf auf dem Wort $w = a_0 \ldots a_{n-1}$, 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 $w = a_0 \ldots a_{n-1}$ sei $r = (r_0, \ldots, r_n)$ der Lauf auf $\mathcal{A}$ mit r_n wie in der Behauptung.
Induktionsschritt: Betrachte nun das Wort $w = a_0 \ldots a_n$.

- $a_0 \ldots a_{n-1} = \varepsilon$. 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:

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

□

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.

Für alle $w \in \Sigma^*$ gilt: $\mathcal{A}$ akzeptiert w g.d.w. $\overline{\mathcal{A}}$ 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 $L_1 \cup L_2$ 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*

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

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 $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$.
Wir zeigen nun, dass $L(\mathcal{A}) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$ ist. Sei dazu $w = a_0 \dots a_{n-1}$ gegeben. Wir zeigen, dass $\mathcal{A}$ akzeptiert w g.d.w. $\mathcal{A}_1$ oder $\mathcal{A}_2$ das Wort w akzeptiert.
"Wenn $\mathcal{A}_1$ oder $\mathcal{A}_2$ akzeptiert, dann akzeptiert $\mathcal{A}$": O.B.d.A. sei $\mathcal{A}_1$ der Automat, der w akzeptiert. Es gibt also einen Lauf $(r_0, \dots, r_n)$ mit $r_n \in F_1$ von $\mathcal{A}_1$ auf w. Sei $(p_0, \dots, p_n)$ der Lauf auf $\mathcal{A}_2$ mit p_n beliebig. 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 $r_n \in F_1$ ist auch $(r_n, p_n) \in F_1 \times Q_2 \subseteq F$. Also akzeptiert $\mathcal{A}$.
"Wenn $\mathcal{A}$ akzeptiert, dann akzeptiert $\mathcal{A}_1$ oder $\mathcal{A}_2$": Wir zeigen die Kontraposition dieser Aussage. $\mathcal{A}_1$ und $\mathcal{A}_2$ verwerfen also beide. Die Läufe von $\mathcal{A}_1, \mathcal{A}_2$ sind also $(r_0, \dots, r_n)$ und $(p_0, \dots, p_n)$ mit $r_n \notin F_1, p_n \notin F_2$. Wie oben ist $((r_0, p_0), \dots, (r_n, p_n))$ der Lauf von $\mathcal{A}$ und es gilt $(r_n, p_n) \notin F$. Somit verwirft auch $\mathcal{A}$ das Wort. □

Eine Beispielkonstruktion für einen Produktautomaten für die Vereinigung 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.

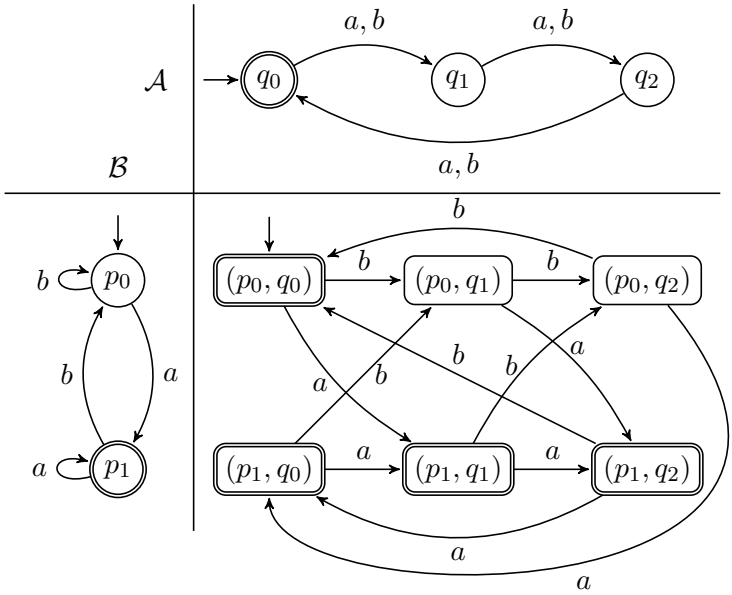


Abbildung 4: Produktkonstruktion: Der Automat $\mathcal{A}$ erkennt die Sprache $\{w \in \{a,b\}^* \mid |w|_a + |w|_b \text{ teilbar durch } 3\}$, $\mathcal{B}$ erkennt die Sprache $\{ua : u \in \{a,b\}^*\}$. Der Produktautomat erkennt die Vereinigung.

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

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

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.

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

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

Satz 3.10. *Seien $L_1, L_2 \subseteq \Sigma^*$ DFA-erkennbare Sprachen. Dann ist auch $L_1 \Cap L_2$ 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.
"$L(\mathcal{A}) \subseteq L_1 \lll L_2$": 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 $c_0 c_2 \dots c_{n-2}$. Umgekehrt sind die ungeraden Positionen induziert durch den Lauf von $\mathcal{A}_2$ auf $c_1 c_3 \dots c_{n-1}$. 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 $c_0 c_2 \dots c_{n-2}$. Analog für $\mathcal{A}_2$. Also ist $w \in L(\mathcal{A}_1) \lll L(\mathcal{A}_2)$.
"$L(\mathcal{A}) \supseteq L_1 \lll L_2$": Sei $w = a_0 b_0 \dots a_{n-1} b_{n-1} \in L_1 \lll L_2$. Dann exitieren Läufe $\varrho_1 = (p_0, \dots, p_n), \varrho_2 = (q_0, \dots, q_n)$ 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

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

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. 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.

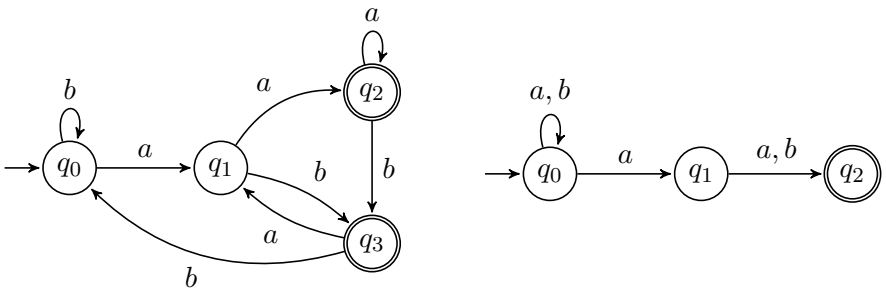
Bisher ist noch nicht klar, wie das Akzeptanzverhalten von NFAs sein soll.

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) $r_0 = q_0$,



(a) DFA (b) NFA

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

(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.

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

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

(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. $E(\mathcal{A}, w) \cap F \neq \emptyset$.*

Beweis. "wenn, dann": Sei $w \in L(\mathcal{A})$. Dann exitiert ein Lauf $(r_0, \dots, r_n)$ 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 $\emptyset \neq \{q\} \subseteq E(\mathcal{A}, w) \cap F$.
"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

```
1 NFA-Akzeptanz(A, w)
  Input  : NFA A, Wort w
  Output : Ja g.d.w. A akzeptiert w = a_0 ... a_{n-1}.
2 u := ε
3 E(A, u) := {q_0}
4 for i = 0, ..., n − 1 do
5 |   Bestimme E(A, ua_i) aus E(A, u) (Lemma 3.18)
6 |   u := ua_i
7 Prüfe, ob E(A, w) ∩ F ≠ ∅.
```

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 $E(\mathcal{A}, w) \cap F \neq \emptyset$ 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. □

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*:

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

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. $E(\mathcal{A}, w) \cap F \neq \emptyset$ (Lemma 3.17). Nach der oben gezeigten Behauptung gilt dies g.d.w. $P \cap F \neq \emptyset$ 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 $\mathcal{A} : q_0 \overset{w}{\to} q$.

(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.

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. □

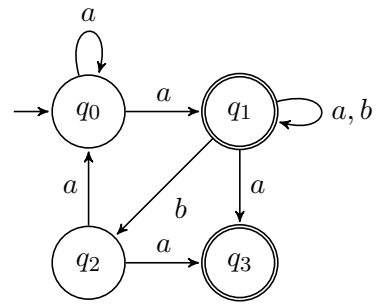
In diesem Sinne ist es sinnvoll von nun an nur noch von FA-erkennbaren Sprachen zu sprechen statt zwischen DFA- und NFA-erkennbaren Sprachen zu unterscheiden.

3.5 Reguläre Ausdrücke

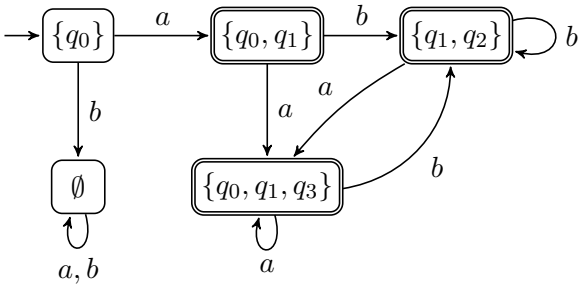
Bisher haben wir Sprachen betrachtet, die sich von endlichen Automaten erkennen lassen. Diese Art von Sprachen liefern uns eine eigene Klasse von Sprachen, die unter verschiedenen Operationen abgeschlossen ist. Auch wenn bisher noch nicht alles gezeigt wurde (Konkatenation, Kleene'sche Hülle) lässt sich 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.

Algorithmus 2 : NFA-Determinisierung

```
1  Potenzmengenkonstruktion(A)
   Input  : NFA A = (Q, Σ, Δ, q_0, F)
   Output : äquivalenter, reduzierter DFA A' = (Q', Σ, δ, q'_0, F')
2  Q' := {{q_0}}
3  q'_0 := {q_0}
4  F' := ∅
5  Queue P := Q'
6  while P ≠ ∅ do
7      S := P.dequeue()
8      for a ∈ Σ do
9          R := {⋃_{p∈S} {r | (p, a, r) ∈ Δ}
10         P.enqueue(R)
11         Q' := Q' ∪ R
12         δ(S, a) = R
13 for P ∈ Q' do
14     if P ∩ F ≠ ∅ then
15         F' := F' ∪ {P}
```



(a) NFA



(b) Potenzmengenautomat

Abbildung 6: Beispiel zur Potenzmengenkonstruktion.

Definition 3.26. Sei Σ ein endliches Alphabet. Ein *regulärer Ausdruck* ist induktiv definiert mit:

- $\boldsymbol{\emptyset}$ 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 $r\boldsymbol{*}$ ein regulärer Ausdruck.

Die Menge aller regulären Ausdrücke über Σ bezeichnen mit REG_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 3.27. Sei Σ ein endliches Alphabet. Die *Interpretation eines regulären Ausdrucks* ist die Abbildung

$$[\![\cdot]\!] : \mathsf{REG}_\Sigma \to 2^{\Sigma^*}$$

mit

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

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

Wir erlauben in der Regel auch als Abkürzung die Ausdrücke $\boldsymbol{\varepsilon}$ für $\boldsymbol{\emptyset *}$ und r^+ für $r \cdot r\boldsymbol{*}$. Außerdem lassen wir den Punkt und Klammern $((,),\cdot)$ weg, es sei denn die dienen der Lesbarkeit. Statt $[\![r]\!]$ ist auch die Schreibweise $L(r)$ gebräuchlich.

3.6 Algorithmen für Reguläre Sprachen

3.7 Weitere Abschlusseigenschaften

3.8 Nicht-reguläre Sprachen

3.9 Myhill-Nerode-Äquivalenz

4 Kellerautomaten und Kontextfreie Sprachen

5 Kontextsensitive Sprachen

6 Prozesskalküle und Petri-Netze