

Burchard v. Braunmühl

GTI-Skript

Grundlagen der Theoretischen Informatik

Informatik III

1 Finite Automaten**1.1 Grundbegriffe****1.1.1 Mengen**

Wir verstehen unter einer Menge erst einmal intuitiv eine Zusammenfassung von bestimmten wohlunterschiedenen Objekten. Jede Menge von Mengen ist wieder eine Menge. Eine Menge kann leer sein, d.h. keine Elemente besitzen. In der Mathematik werden Mengen mit geschweiften Klammern $\{, \}$ angedeutet. So schreibt man die Menge der Zahlen 0, 1, 2, 3 und 4 als $\{0, 1, 2, 3, 4\}$. Hierbei kommt es *nicht* auf die Reihenfolge an, in der die Elemente aufgelistet sind. Die Menge der Telefonbesitzer bleibt dieselbe, ob ich das Telefonbuch alphabetisch oder nach Telefonnummern ordne. Und auch, wenn der Name eines Telefonteilnehmers zweimal auftaucht, ändert sich diese Menge nicht. Merke: $\{1, 2\} = \{2, 1\}$, $\{1, 1\} = \{1\}$.

Auch unendliche Mengen kann man so durch Aufzählung der Elemente definieren, wobei für die Aufzählung allerdings ein Gesetz angegeben werden muß. So kann man die Menge der natürlichen Zahlen \mathbb{N} (mit der Null) etwa beschreiben durch

$$\mathbb{N} = \{0, 1, 2, \dots\}, \text{ wobei } 0 = \emptyset \text{ und } i = \{0, \dots, i-1\}.$$

Mit \mathbb{N}_+ bezeichnen wir die Menge der natürlichen Zahlen ohne die Null, also $\mathbb{N}_+ = \mathbb{N} - \{0\}$.

Gewöhnlich beschreiben wir Mengen durch Eigenschaften, die ihre Elemente haben. Formal schreiben wir dann:

$$\{x \mid E(x)\},$$

wobei $E(x)$ der Name einer Eigenschaft für x sei. Wir schreiben also eine Variable x , die für ein Objekt steht, dann kommt ein senkrechter Trennstrich und dahinter werden die Eigenschaften formuliert, die die Objekte haben sollen, wenn wir sie in die Menge aufnehmen wollen. Statt $\{x \mid x \in A \wedge E(x)\}$ schreiben wir auch $\{x \in A \mid E(x)\}$.

Zum Beispiel:

- $\{x \in \mathbb{N} \mid x = 2 \vee x = 3\}$
- $\{x \in \mathbb{N} \mid x^2 - 5x + 6 = 0\}$
- $\{x \in \mathbb{N} \mid x \text{ ist Primzahl} \wedge x < 5 \wedge x > 1\}$
- $\{x \in \mathbb{N} \mid x \neq x\}$ (das ist die leere Menge)
- $\{i \in \mathbb{N} \mid 1 \leq i \leq n\} = \{i \in \mathbb{N} \mid i = 1, \dots, n\}$

Die wichtigsten Mengenbeziehungen sind durch Zeichen abgekürzt:

- $a \in A$ das Objekt a ist ein Element der Menge A , gehört also zu A .
- $a \notin A$ das Objekt a ist nicht Element der Menge A , gehört nicht zu A .
- $A \cup B$ dies bezeichnet die Menge der Elemente, die in A oder in B liegen.
- $A \cap B$ dies bezeichnet die Menge der Elemente, die in A und in B liegen.
- $A - B$ dies bezeichnet die Menge der Elemente, die in A , aber nicht in B liegen.
- $A \triangle B := (A - B) \cup (B - A)$ die Menge der Elemente, die in entweder in A oder in B liegen. .
Diese Menge wird symmetrische Differenz von A und B genannt.
- $A \subseteq B$ die Menge A enthält nur Elemente aus B . A ist eine Teilmenge von B .
- $A \subset B$ die Menge A enthält nur Elemente aus B , aber nicht alle. A ist echte Teilmenge von B .
- $A \not\subseteq B$ die Menge A enthält auch Elemente, die nicht in B liegen. A ist keine Teilmenge von B .
- \emptyset dies bezeichnet die Menge, die kein Element enthält, die leere Menge.
- $\{A\}$ dies bezeichnet die Menge, die die Menge A als einziges Element enthält.
- $\mathcal{P}(A)$ dies bezeichnet die Menge, die alle Mengen B enthält, für die gilt $B \subseteq A$.
Diese Menge aller Teilmengen von A nennen wir die Potenzmenge von A .
Sie wird oft auch 2^A geschrieben.
- $\bigcup \mathfrak{A} := \bigcup \{A \mid A \in \mathfrak{A}\} := \bigcup_{A \in \mathfrak{A}} A := \{x \mid \exists A \in \mathfrak{A} : x \in A\}$
ist die Vereinigung aller Mengen des Mengensystems \mathfrak{A} .
 $\bigcup \emptyset := \emptyset$.
- $\bigcap \mathfrak{A} := \bigcap \{A \mid A \in \mathfrak{A}\} := \bigcap_{A \in \mathfrak{A}} A := \{x \mid \forall A \in \mathfrak{A} : x \in A\}$
ist der Durchschnitt aller Mengen des Mengensystems \mathfrak{A} .
 $\bigcap \emptyset$ ist nicht definiert (führte zu einem Widerspruch).

Für die Operationen \cup und \cap gelten folgende Gesetze:

$(A \cup B) \cup C = A \cup (B \cup C)$	wie $(x + y) + z = x + (y + z)$	(assoziativ)
$(A \cap B) \cap C = A \cap (B \cap C)$	wie $(x \cdot y) \cdot z = x \cdot (y \cdot z)$	(assoziativ)
$A \cup B = B \cup A$	wie $x + y = y + x$	(kommutativ)
$A \cap B = B \cap A$	wie $x \cdot y = y \cdot x$	(kommutativ)
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	hier stimmt der Vergleich nicht	(distributiv)
$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$	wie $x \cdot (y + z) = xy + xz$	(distributiv)
$A \cup \emptyset = A$	wie $x + 0 = x$	(neutrales Element)
$A \cap \emptyset = \emptyset$	wie $x \cdot 0 = 0$	(absorbierendes Element)

Für die Teilmengenbeziehung \subseteq gilt:

$A \subseteq A$	(reflexiv),
$A \subseteq B$ und $B \subseteq C$, so auch $A \subseteq C$	(transitiv),
$A \subseteq B$ und $B \subseteq A$, so $A = B$	(antisymmetrisch).

Haben wir eine Grundmenge G und Mengen $A, B \subseteq G$, so bezeichnet $\bar{A} = G - A$ das Komplement von A bezüglich der Grundmenge G . Es gelten die Regeln von de Morgan: $\overline{(A \cap B)} = \bar{A} \cup \bar{B}$ und $\overline{(A \cup B)} = \bar{A} \cap \bar{B}$.

1.1.2 Relation

Ein Grundbegriff in der Mathematik ist das *Paar*. Haben wir zwei Elemente a und b , so können wir das Paar (a, b) bilden (streng mengentheoretisch wird auch das Paar, wie alle mathematischen Objekte als Menge aufgefaßt: $(a, b) = \{\{a, b\}, \{a\}\}$).

Definition 1.1.1 (Kreuzprodukt, Relation)

Sind A und B zwei Mengen, so ist das *Kreuzprodukt* (oder *Kartesisches Produkt*) $A \times B := \{(a, b) \mid a \in A \wedge b \in B\}$ die Menge aller Paare mit erster Komponente aus A und zweiter Komponente aus B . Eine (binäre) *Relation* ist eine Teilmenge $R \subseteq A \times B$ zusammen mit den Mengen A und B .

Wir beschränken uns hier auf Relationen $R \subseteq A \times A$. Wir sprechen von einer Relation auf A .

Notation 1.1.2

Statt $(a, b) \in R$ schreiben wir auch aRb .

Beispiel 1.1.3

Betrachte die Menge \mathbb{N} der natürlichen Zahlen. Die Gleichheit $a = b$ ist eine Relation auf \mathbb{N} ebenso wie die Kleinerrelation $a < b$.

1.1.3 Gerichtete Graphen (Digraphen)

Eine andere Sichtweise von Relationen auf A führt uns zu Graphen. Sei $R \subseteq A \times A$ und $(a, b) \in R$. Um zu signalisieren, daß a zu b in der Beziehung R steht, verbindet man auch manchmal a und b durch einen Pfeil, der von a nach b führt. Wir stellen uns vor, daß die Elemente von A Punkte sind, die gemäß R durch Pfeile verbunden sind. Mathematisch nennen wir das einen "gerichteten Graphen" oder "Digraphen", die Elemente von A "Knoten" und die Pfeile "gerichtete Kanten". Im Unterschied zum Digraphen, können in einem Multigraphen von einem Knoten zum anderen mehrere Kanten verlaufen. Hier wird einer Kante durch eine Funktion das Paar der Knoten zugeordnet, zwischen denen die Kante verläuft.

Definition 1.1.4 (Digraph)

Ein Digraph (engl: directed graph) $G = (A, R)$, besteht aus einer Menge A von Knoten und einer Relation $R \subseteq A \times A$ auf der Knotenmenge. Ein Paar dieser Relation nennen wir eine (gerichtete) Kante (Pfeil) des Graphen.

Definition 1.1.5 (Multigraph)

Ein gerichteter Multigraph $G = (V, E, r)$ besteht aus einer Menge V von Knoten, eine Menge E von Kanten und einer totalen Funktion $r : E \rightarrow V \times V$, die jeder Kante Start- und Zielknoten zuordnet.

Definition 1.1.6 (Pfad, Schleife, Schlinge)

Sei $G = (V, R)$ ein Digraph. Eine Knotenfolge a_0, \dots, a_n über V heißt *Pfad der Länge n* des Graphen G von a_0 nach a_n , wenn $(a_i, a_{i+1}) \in R$ für alle $i \in \{0, \dots, n-1\}$. Diesem Pfad entspricht die Kantenfolge (oder Kantenzug) $(a_0, a_1)(a_1, a_2), \dots (a_{n-1}, a_n)$. Ist $n = 0$, so ist der Kantenzug leer, er enthält keine einzige Kante. Immer führt der leere Kantenzug von einem Knoten in ihn selbst. Ist $a_0 = a_n$, so heißt dieser Kantenzug *Schleife*. Eine Schleife der Länge 1 heißt *Schlinge*. G heißt *schleifenfrei*, wenn G keine Schleife hat.

Bei Multigraphen werden diese Begriffe analog definiert.

1.1.4 Funktionen

Die meist behandelten und wichtigsten Relationen sind die Funktionen, Beziehungen, die rechts-eindeutig sind, die also die Eigenschaft haben, daß ein Objekt nie zu mehreren anderen, sondern zu höchstens einem in dieser Beziehung steht. Umgekehrt aber kann ein Objekt sehr wohl mehrere Partner haben, die zu ihm in dieser Beziehung stehen. So kann ein Kind nur einen natürlichen Vater haben, ein Vater aber mehrere Kinder. Die Kind-Vater-Relation ist also eine Funktion: wir können jedem Kind seinen einen Vater zuordnen.

Definition 1.1.7 (total, eindeutig)

Eine Relation R aus $A \times B$ heißt

linkstotal, gdw. es für alle $a \in A$ mindestens ein $b \in B$ gibt mit $(a, b) \in R$,

rechtstotal, gdw. es für alle $b \in B$ mindestens ein $a \in A$ gibt mit $(a, b) \in R$,

linkseindeutig, gdw. es für alle $b \in B$ höchstens ein $a \in A$ gibt mit $(a, b) \in R$,
rechtseindeutig, gdw. es für alle $a \in A$ höchstens ein $b \in B$ gibt mit $(a, b) \in R$.

Definition 1.1.8 (Funktion, total, partiell, injektiv, surjektiv, bijektiv)

Eine rechtseindeutige Relation heißt auch *Funktion*. Ist sie linkstotal, so spricht man auch von einer *totalen Funktion*. Will man betonen, daß man auch Funktionen betrachten will, die u.U. nicht total sind, spricht man *partiellen Funktionen*. In der Mathematik betrachtet man i. Allg. totale Funktionen, in der Informatik i. Allg. partielle Funktionen. Als Informatiker wollen wir daher unter einer Funktion immer eine partielle Funktion verstehen (ein Begriff, der ja auch die totalen Funktionen einschließt). Die rechtseindeutige Relation f aus $A \times B$ nennt man dann Funktion *aus A nach B*. Ist f total, so sagt man auch, f sei eine Funktion *von A nach B*. Ist f rechtstotal, so sagt man f ist eine Funktion aus (von) A *auf* B . Man sagt, f ist *surjektiv* (oder salopp: eine Funktion *auf*). Ist f linkseindeutig, so nennt man f *eineindeutig* oder *injektiv*. Eine linkstotale, injektive und surjektive Funktion heißt *bijektiv*.

Notation 1.1.9

Es ist gut, sich an folgende Schreibweisen zu halten:

Ist f eine Funktion aus A nach B , so schreiben wir statt $f \subseteq A \times B$ auch $f : A \rightarrow B$. Wollen wir betonen, daß f total ist, so schreiben wir $f : A \mapsto B$. Ist $a \in A$ und $b \in B$ mit $(a, b) \in f$, so schreibt man statt $(a, b) \in f$ auch $f : a \mapsto b$ (Pfeil mit Endstrich) oder $f(a) = b$, was nur geht, weil es nur ein $f(a)$ gibt. Statt $f(a)$ wiederum schreibt man manchmal f_a . Gesprochen wird das: "f von a ist b" oder "f angewandt auf a ist b" oder "Das Bild von a unter f ist b".

Definition 1.1.10 (Inverse einer Funktion, Produktfunktion)

Gegeben seien die Funktionen $f : A \rightarrow B$ und $g : B \rightarrow C$. Dann ist f^{-1} eine Relation über $B \times A$ (das Inverse von f), und zwar die Relation $\{(b, a) \in B \times A \mid f(a) = b\}$. Die Produktfunktion $g \circ f$ von f und g ist die Funktion aus A nach C mit $g \circ f := \{(a, c) \in A \times C \mid \exists b \in B : (a, b) \in f \wedge (b, c) \in g\}$, d.h. $(g \circ f)(a) = g(f(a))$, falls $f(a)$ und $g(f(a))$ definiert sind, andernfalls ist die Produktfunktion für a nicht definiert.

Anmerkung 1.1.11

Ist f eineindeutig, so ist die zu f inverse Funktion f^{-1} eine (partielle), ebenfalls eineindeutige Funktion. Ist f bijektiv, so ist f^{-1} ebenfalls eine Bijektion.

Definition 1.1.12 (Bild, Urbild, Faser, Faserung)

Ist $f : A \rightarrow B$ eine Funktion und $b \in B$, so nennt man die Menge $\{a \in A \mid f(a) = b\}$ das *Urbild* oder die *Faser* von b unter f und schreibt dafür $f^{-1}(b)$. Die Relation $a_1 \sim a_2$ gdw. $f(a_1) = f(a_2)$ ist eine Äquivalenzrelation. $\{f^{-1}(b) \mid b \in f(A)\}$ ist die dazugehörige Klasseneinteilung (Partition). Die Partition ist also gerade die Menge der Fasern bzgl. f . Man nennt sie daher auch die *Faserung von A bzgl. f*.

Ist $X \subseteq A$, so ist nennt man die Menge $f(X) = \{b \in B \mid \exists a \in X : b = f(a)\} = \{f(a) \mid a \in X \wedge f(a) \text{ definiert}\}$ das *Bild von X unter f*. Ist $Y \subseteq B$, so ist $f^{-1}(Y) = \{a \in A \mid \exists b \in Y : b = f(a)\} = \{a \in A \mid f(a) \in Y\}$ das *Urbild von Y unter f*.

1.1.5 Folgen

Eine spezielle Art von Funktionen sind die Folgen. Eine unendliche Folge aus A ist eine totale Funktion a von $\mathbb{N}_+ = \mathbb{N} - \{0\}$ nach A . Man nennt sie Folgen, weil man die Bilder $a(1), a(2), a(3), \dots$ in derselben Reihenfolge schreiben kann, wie die Zahlen $1, 2, 3, \dots$ selbst. Statt $a(1), a(2), a(3), \dots$ schreibt man meistens a_1, a_2, a_3, \dots oder $(a_i)_{i \in \mathbb{N}_+}$. Endliche Folgen der Länge $n \in \mathbb{N}$ aus A sind Funktionen vom Abschnitt $[1, n] = \{1, \dots, n\} \subset \mathbb{N}$ nach $A : a_1, \dots, a_n$, wobei wir hier gleich verabreden, daß für $n = 0$ diese Darstellung die leere Folge der Länge 0 meint. Für die leere Folge schreiben wir also: a_1, \dots, a_0 oder $(a_i)_{i \in \emptyset}$ oder einfach $()$ und manchmal auch ε . Analog bedeute a_1, \dots, a_1 das Wort a_1 der Länge 1.

1.1.6 Ordnungsrelation / Äquivalenzrelation

Es gibt Relationen mit besonderen Eigenschaften.

Eine Relation $R \subseteq A \times A$ heißt *Ordnungsrelation*, gdw.:

$$\begin{aligned} \forall a \in A & : aRa && \text{(reflexiv),} \\ \forall a, b \in A & : aRb \wedge bRa \implies a = b && \text{(antisymmetrisch),} \\ \forall a, b, c \in A & : aRb \wedge bRc \implies aRc && \text{(transitiv).} \end{aligned}$$

So ist etwa die Kleingleich-Relation \leq eine Ordnungsrelation auf \mathbb{N} , aber auch die Teilmengen-Relation \subseteq für Mengen aus \mathbb{N} . (Im ersteren Fall reden wir von einer Ordnung, weil alle Elemente miteinander vergleichbar sind, wodurch sie eine strenge Ordnung, eine Kette bilden, im anderen Fall von einer Halbordnung, weil es auch unvergleichbare Elemente gibt (Elemente A und B mit weder $A \subseteq B$ noch $B \subseteq A$).

Eine Relation $R \subseteq A \times A$ heißt *Äquivalenzrelation* (kurz *Äquivalenz*) gdw.:

$$\begin{aligned} \forall a \in A & : aRa && \text{(reflexiv),} \\ \forall a, b \in A & : aRb \implies bRa && \text{(symmetrisch),} \\ \forall a, b, c \in A & : aRb \wedge bRc \implies aRc && \text{(transitiv).} \end{aligned}$$

So ist die Gleichheit $=$ für natürliche Zahlen eine Äquivalenzrelation, aber auch die Gleichmächtigkeit ($A \equiv B$ gdw. $|A| = |B|$) für Teilmengen von \mathbb{N} .

1.1.7 Partition

Wenn wir eine Menge A in Teilmengen A_1, A_2, A_3, \dots zerlegen, sodaß

1. die Teilmengen A_i nicht leer sind ($\forall i : A_i \neq \emptyset$),
2. die Teilmengen A_i paarweise disjunkt sind ($\forall i, j : i \neq j \implies A_i \cap A_j = \emptyset$),
3. alle Teilmengen A_i zusammen wieder A ergeben, also A überdecken ($\bigcup_i A_i = A$),

dann reden wir von einer *Partition* oder *Klasseneinteilung* von A .

Kurz: eine Partition von A ist eine disjunkte Überdeckung von A durch nichtleere Teilmengen. Diese Teilmengen nennen wir Klassen. 2 Elemente a, b heißen äquivalent ($a \sim b$), wenn sie zur selben Klasse gehören. Wie man sich leicht klarmacht, ist dies wirklich eine Äquivalenzrelation.

Haben wir eine Äquivalenzrelation R auf A , und bilden wir Klassen in A , indem wir alle einander äquivalenten Elemente zusammenfassen, so bekommen wir eine Klasseneinteilung:

1. Jede Klasse enthält wenigstens ein Element, da $\forall a : aRa$.
2. Die Klassen sind disjunkt, denn gäbe es zwei unterschiedliche Klassen K, K' , die ein Element x gemeinsam haben, so wäre x zu allen Elementen aus K äquivalent, aber auch zu allen aus K' . Wegen Transitivität und Symmetrie, wären dann alle Elemente von K zu allen von K' äquivalent, d.h. K und K' bildeten zusammen eine Klasse oder lägen in einer Klasse. Wir haben aber angenommen, daß K und K' für sich schon Klassen bilden. W!
3. Die Klassen überdecken A , d.h. jedes Element $a \in A$ liegt in einer Klasse, wieder weil jedes Element a zu sich selbst äquivalent ist (aRa). In der Klasse aller zu a äquivalenten Elemente liegt also wenigstens a selbst.

Wir nennen sie die von R induzierte *Partition* oder auch kurz die *Partition von R* . Wir werden eine Klasse von R , die das Element a enthält, später auch mit $[a]_R$ bezeichnen oder auch mit $[a]$, wenn R klar ist. Mit A/R bezeichnen wir die Menge aller Klassen der Partition von R .

1.1.8 Worte

Für Informatiker existiert nur eine Welt von Zeichen. Ein Computer kann nur Zeichenreihen behandeln. Was sich nicht durch eine Zeichenreihe darstellen läßt, ist für einen Computer überhaupt nicht faßbar. Wir sprechen von Informationsverarbeitung, aber korrekter sollten wir von Zeichenverarbeitung reden. Daß Zeichen und Zeichenreihen für uns Information tragen, eine Semantik haben, ist für den Computer nicht relevant. Genau genommen gibt es für den Computer sogar nur zwei Zeichen, 0 und 1 oder ja und nein oder richtig und falsch oder wie immer man diese beiden einzigen vom Rechner unterscheidbaren Zeichen nennen möchte. Wenn wir von mehr als zwei Zeichen reden, so in dem Bewußtsein, daß jedes Zeichen wieder durch diese beiden Grundzeichen dargestellt werden kann (jedenfalls, wenn wir uns auf abzählbar viele Zeichen beschränken). Aber was ist ein Zeichen? Etwas, dem wir eine Information zuordnen können? Buchstaben sind Zeichen, aber physikalisch gesehen sind Buchstaben Anhäufungen von Druckerschwärze oder Kreidepartikel oder Graphitplättchen, die man unterscheiden kann. So sind alle unterscheidbaren Objekte geeignet, die Rolle von Zeichen zu spielen. Der Begriff "Alphabet" (Menge von Zeichen) ist eine Abstraktion und meint eine Menge von Gegenständen, die unterscheidbar sind (wobei "unterscheidbar" wieder vom Subjekt abhängt: Menschen unterscheiden anders als Computer).

Wir wollen unter einem *Alphabet* immer eine endliche Menge von Zeichen (solch unterscheidbaren Gegenständen) verstehen. Ein *Wort* ist eine Folge von Zeichen. Wir betrachten in diesem Skript nur endliche Worte, also Zeichenfolgen von endlicher Länge. Da wir hier in der Informatik sind, schreiben wir alles ein bißchen anders als in der Mathematik üblich. So schreiben wir eine Zeichenfolge ohne Kommata zwischen den einzelnen Zeichen, den Gliedern der Folge.

Definition 1.1.13 (Wort der Länge n , Konkatenation)

Sei Σ ein Alphabet. Die Folge $w = a_1 \cdots a_n$ heißt *Wort über Σ der Länge n* , wenn $a_i \in \Sigma$ für alle $i \in [1, n]$. Für die Länge n von w schreiben wir $|w|$. Das Wort der Länge 0 nennen wir das *leere Wort* und schreiben dafür ε . Wir definieren:

$$\begin{aligned} \Sigma^n &:= \text{Menge der Worte der Länge } n, \text{ insbesondere} \\ \Sigma^0 &= \{\varepsilon\}. \text{ Schließlich ist} \\ \Sigma^* &:= \bigcup_{n \in \mathbb{N}} \Sigma^n \text{ die Menge aller Worte über } \Sigma. \end{aligned}$$

Auf dieser Menge Σ^* definieren wir eine Operation \cdot (die *Konkatenation*), die zwei Worte u und v zu einem Wort $u \cdot v$ verknüpft, das durch Hintereinanderschreiben entsteht: sind $u = a_1 \cdots a_n$ und $v = b_1 \cdots b_m$ zwei Worte über Σ ($a_i, b_j \in \Sigma$, $i = 1, \dots, n$, $j = 1, \dots, m$) so ist ihre Konkatenation $u \cdot v = c_1 \cdots c_{n+m}$ mit $c_1 \cdots c_n = a_1 \cdots a_n$ und $c_{n+1} \cdots c_{n+m} = b_1 \cdots b_m$ ($c_i \in \Sigma$, $i = 1, \dots, n+m$). Insbesondere gilt für jedes Wort $w \in \Sigma^*$: $w \cdot \varepsilon = \varepsilon \cdot w = w$. Gewöhnlich schreiben wir für $u \cdot v$ auch kurz uv .

Definition 1.1.14 ($A \cdot B$, A^n , A^*)

Jede Menge von Worten $A \subseteq \Sigma^*$ nennen wir eine *Sprache über Σ* (ungeachtet der Tatsache, daß diese Worte noch sinnlos sind, und keinen semantischen oder syntaktischen Zusammenhang besitzen). Für die leere Sprache schreiben wir \emptyset (dasselbe Zeichen wie für die leere Menge, da die leere Sprache eben gerade die Menge ist). Wir verallgemeinern die Konkatenation von Worten auf Sprachen und definieren für Sprachen A und B über Σ :

$$A \cdot B := \{w \mid \exists u \in A, v \in B : w = uv\}$$

Insbesondere gilt: $A \cdot \emptyset = \emptyset \cdot A = \emptyset$ (die leere Sprache ist das absorbierende Element in der Struktur

$(\mathcal{P}(\Sigma^*), \cdot)$, wo $\mathcal{P}(\Sigma^*)$ die Potenzmenge von Σ^* ist). Weiter definieren wir die iterierte Konkatenation:

$$\begin{aligned} A^0 &:= \{\varepsilon\}, \\ A^{n+1} &:= A^n \cdot A, \\ A^* &:= \bigcup_{n \in \mathbb{N}} A^n \text{ und} \\ A^+ &:= \bigcup_{n \in \mathbb{N}_+} A^n. \end{aligned}$$

Auf Sprachen können wir - wie auf jede Menge - die Booleschen Operationen Vereinigung, Durchschnitt und Komplement anwenden: Seien A und B Sprachen über Σ , so gelten auch hier die Gesetze von de Morgan ($\overline{\overline{A}} = A = \Sigma^* - A$):

$$\overline{\overline{A}} = A, \quad \overline{A \cup B} = \overline{A} \cap \overline{B}, \quad \overline{A \cap B} = \overline{A} \cup \overline{B}.$$

1.1.9 Rechtskongruenzen auf Σ^*

Haben wir eine Menge A , auf der eine 2-stellige Operation \bullet definiert ist, so schreiben wir diese Struktur (A, \bullet) - z.B. ist $(\mathbb{N}, +)$ die Menge \mathbb{N} mit der Addition. Wir können Äquivalenzrelationen auf A betrachten, die mit dieser Operation verträglich sind. Verträglich heißt hier: Sind 2 Elemente a und b äquivalent, so auch noch, die Elemente $a \cdot x$ und $b \cdot x$, die man erhält, wenn man a und b mit demselben Element x verknüpft. Statt verträglich sagen wir auch invariant. Da nicht immer $a \bullet x = x \bullet a$ sein muß, unterscheiden wir noch zwischen einer rechtsinvarianten Äquivalenzrelation (d.h. verträglich gegenüber Verknüpfung von rechts) und einer linksinvarianten.

Definition 1.1.15 (rechtsinvariant, finiter Index)

Sei (A, \bullet) eine Menge mit der 2-stelligen Operationen \bullet und R eine Äquivalenzrelation auf A . Wir nennen R *rechtsinvariant*, gdw. gilt:

$$\forall a, b \in A : a R b \implies \forall x \in A : (a \bullet x) R (b \bullet x)$$

Eine rechtsinvariante Äquivalenzrelation nennen wir Rechtskongruenzrelation (kurz Rechtskongruenz). Wir sagen, die Rechtskongruenz hat *finiten Index*, wenn ihre Partition nur endlich viele Klassen hat.

Definition 1.1.16 (saturieren)

Sei $L \subseteq \Sigma^*$. Eine Rechtskongruenz R auf Σ^* *saturiert* L , wenn gilt:

$$x R y \implies (x \in L \iff y \in L).$$

Anmerkung 1.1.17

Eine Rechtskongruenz R auf Σ^* saturiert (lat. befriedigt, sättigt) eine Sprache $L \subseteq \Sigma^*$, wenn L keine Klasse von R schneidet, d.h. alle Klassen liegen entweder ganz in L oder ganz außerhalb von L . L ist also die Vereinigung von ganzen Klassen von R . Formaler: ist $\{K_i \mid i \in I\}$ die Partition von R ($I = \{1, \dots, n\}$ oder $I = \mathbb{N}$), so gilt: $\exists J \subseteq I : L = \bigcup \{K_j \mid j \in J\}$.

Beispiel 1.1.18

Betrachte $(\mathbb{N}, +)$ die natürlichen Zahlen mit Addition. Definiere $a R_5 b$, gdw. $|a - b|$ teilbar durch 5 (ohne Rest). R_5 ist eine Rechtskongruenz: ist $|a - b|$ teilbar durch 5, so auch $|(a + c) - (b + c)|$.

Anmerkung 1.1.19

Da auf Σ^* eine Operation (die Konkatenation \cdot) definiert ist, können wir auch auf (Σ^*, \cdot) Rechtskongruenzen betrachten. Statt (Σ^*, \cdot) schreiben wir in der Regel nur Σ^* , da wir auf Σ^* immer nur diese eine Operation, die Konkatenation, betrachten.

Sei R eine Rechtskongruenz auf Σ^* mit den Klassen K_1, K_2, K_3, \dots . Die Rechtsinvarianz besagt:

$$\forall i \in \mathbb{N} \quad \forall z \in \Sigma^* \quad \exists j \in \mathbb{N} : K_i \cdot \{z\} \subseteq K_j.$$

Um die Rechtsinvarianz zu zeigen, genügt es allerdings, dies nur für alle Symbole aus Σ zu zeigen:

$$\forall i \in \mathbb{N} \quad \forall a \in \Sigma \quad \exists j \in \mathbb{N} : K_i \cdot \{a\} \subseteq K_j.$$

Denn dann gilt:

$$\begin{aligned} (\forall i \in \mathbb{N}) \quad (\forall z = a_1 \cdots a_n \in \Sigma^*) \quad (\exists i_1, \dots, i_n = j \in \mathbb{N}) : \\ K_i \cdot \{a_1\} \subseteq K_{i_1}, \quad K_{i_1} \cdot \{a_2\} \subseteq K_{i_2}, \dots, K_{i_{n-1}} \cdot \{a_n\} \subseteq K_{i_n} = K_j, \\ \text{d.h. } K_i \cdot \{a_1 \cdots a_n\} \subseteq K_j. \end{aligned}$$

1.2 Finite Automaten

Formal wird ein erkennender, finiter Automat definiert durch Angabe seiner (endlich vielen) Zustände, seines Inputalphabets, von Anfangszustand und akzeptierenden Zuständen und durch Angabe seiner Befehle der Form (p, a, q) , die angeben, wie die Zustände bei bestimmten Eingaben wechseln. Also

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

wo Q die endliche Zustandsmenge, Σ das endliche Inputalphabet, q_0 der *Startzustand*, F die Menge der *akzeptierenden Zustände* und δ eine Relation aus $(Q \times \Sigma) \times Q$ ist (Sprachgebrauch: Transitionsrelation, Menge der Befehle).

Wir nennen M *deterministisch*, wenn δ eine rechtseindeutige Relation, also eine partielle Funktion $Q \times \Sigma \rightarrow Q$ ist (wir sprechen dann von der *Transitionsfunktion*). Andernfalls nennen wir M *nichtdeterministisch*. M ist *vollständig*, wenn für jedes Paar (p, a) aus $Q \times \Sigma$ ein $q \in Q$ existiert, mit $((p, a), q) \in \delta$, also wenn δ linksstotal ist.

Wir definieren die Relation $\delta^* : (Q \times \Sigma^*) \times Q$ induktiv über die Länge der Worte vermöge: für alle $q \in Q$ ist $((q, \varepsilon), q) \in \delta^*$ und für alle $w \in \Sigma^*$, $a \in \Sigma$ und $p, q, r \in Q$ gilt: ist $((p, w), q) \in \delta^*$ und $((q, a), r) \in \delta$, so ist auch $((p, wa), r) \in \delta^*$.

In Zukunft werden wir aber einen Befehl $((q, a), r) \in \delta$ salopper einfach als (q, a, r) notieren, $((p, w), q) \in \delta^*$ als (p, w, q) und analog $(Q \times \Sigma) \times Q$ als $Q \times \Sigma \times Q$ bzw. $(Q \times \Sigma^*) \times Q$ als $Q \times \Sigma^* \times Q$.

Die von M erkannte (akzeptierte) Sprache ist $L(M) = \{w \in \Sigma^* \mid \exists p \in F : (q_0, w, p) \in \delta^*\}$. Ein Zustand p ist *erreichbar*, wenn es ein Wort $w \in \Sigma^*$ gibt mit $(q_0, w, p) \in \delta^*$.

Ist M deterministisch, so schreiben wir statt $(p, a, q) \in \delta$ auch $\delta(p, a) = q$ und statt $(p, w, q) \in \delta^*$ auch $\delta^*(p, w) = q$ ($w \in \Sigma^*$).

1.2.1 Finite Automaten und Digraphen

Wir können die Relation $\delta \subseteq (Q \times \Sigma) \times Q$ auch auffassen als eine Menge von Relationen $\{\delta_a \mid a \in \Sigma\}$, wobei jedes δ_a eine Relation aus $Q \times Q$ ist. Oben haben wir gesehen, daß wir eine Relation R auf Q wiederum auffassen können als Digraphen (Q, R) . Ist $|\Sigma| = k$, so bekommen wir aus den k Relationen δ_a eben k Graphen (Q, δ_a) .

Da die Knotenmenge aller dieser Graphen dieselbe ist, so können wir diese wieder auffassen als einen Multigraphen mit der Knotenmenge Q , aber verschiedenen durch $a \in \Sigma$ gekennzeichneten

(quasi eingefärbten) Kanten. Diesen Graphen mit bewerteten (gelabelten) Kanten können wir als eine adäquate Sicht eines finiten Automaten betrachten, wenn wir noch den Anfangsknoten (Anfangszustand) und die Endknoten (akzeptierenden Zustände) auszeichnen (d.h. ebenfalls bewerten oder einfärben).

Formal ist also ein endlicher Automat auch eindeutig gegeben durch einen gerichteten Multigraphen mit bewerteten Knoten und Kanten und einem ausgezeichneten Knoten:

$$\Gamma = (Q, \Delta, r, e, \nu, s), \quad \text{mit } r : \Delta \rightarrow Q \times Q, \quad e : \Delta \rightarrow \Sigma, \quad \nu : Q \rightarrow \{a, v\}, \quad s \in Q$$

(Q ist die Knotenmenge, Δ die Kantenmenge, r die Start-Ziel-Funktion, e die Kantenbewertung (Symbol), ν die Knotenbewertung (a für akzeptierend und v für verwerfend), s steht für den Startknoten). Wir sagen: ein Wort $w = a_1 \cdots a_n$ führt vom Zustand p in den Zustand q , wenn es einen Pfad der Länge n gibt, der von p nach q führt, und dessen Kanten der Reihe nach mit den Symbolen a_1, \dots, a_n bewertet sind. Die Sprache des Automaten besteht gerade aus den Worten, die vom Startzustand in einen Endzustand führen.

1.2.2 DFA M und zugehörige Rechtskongruenz R_M

Notation 1.2.1 ($K_q(M)$)

Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein DFA. Dann ist:

1. $K_q(M) := \{w \in \Sigma^* \mid \delta^*(q_0, w) = q\}$, d.h. die Menge aller Worte, die vom Startzustand in den Zustand q führen ($q \in Q$).
2. $R_M := \{(u, v) \in \Sigma^* \times \Sigma^* \mid \delta^*(q_0, u) = \delta^*(q_0, v)\}$, d.h. die Menge der Paare von Worten, die vom Startzustand in denselben Zustand führen

Satz 1.2.2

Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein vollständiger DFA, dessen Zustände alle erreichbar sind. Dann ist $\{K_q(M) \mid q \in Q\}$ eine Partition auf Σ^* und R_M die zugehörige Äquivalenz. R_M ist eine Rechtskongruenz auf (Σ^*, \cdot) .

Beweis:

- $\forall q \in Q : K_q(M) \neq \emptyset$ (erreichbar).
- $\forall w \in \Sigma^* \exists q \in Q : w \in K_q(M)$ (vollständig).
- $\forall p, q \in Q$ mit $p \neq q : K_p(M) \cap K_q(M) = \emptyset$ (deterministisch).
- $\exists q \in Q : u, v \in K_q(M) \iff \delta^*(q_0, u) = \delta^*(q_0, v) \iff u R_M v$.
- $u R_M v \implies \delta^*(q_0, u) = \delta^*(q_0, v) \implies \forall z \in \Sigma^* : \delta^*(q_0, uz) = \delta^*(q_0, vz) \implies \forall z \in \Sigma^* : uz R_M vz$. ■

Definition 1.2.3 (die zu M gehörige Rechtskongruenz R_M)

Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein DFA. $R_M = \{(u, v) \in \Sigma^* \times \Sigma^* \mid \delta^*(q_0, u) = \delta^*(q_0, v)\}$ nennen wir *die zu M gehörige Rechtskongruenz*.

1.2.3 Rechtskongruenz mit endlichem Index und zugehöriger isomorpher DFA

Definition 1.2.4 (der zu R und L gehörige DFA $M_{R,L}$)

Sei R eine Rechtskongruenz von endlichem Index, die eine Sprache L saturiert. Die Klassen von R seien K_0, \dots, K_n , o.B.d.A. $\varepsilon \in K_0$. Dann ist *der zu R und L gehörenden DFA* $M_{R,L} = (\{q_0, \dots, q_n\}, \Sigma, \delta, q_0, F)$ mit: $F := \{q_i \mid K_i \subseteq L\}$ und $(q_i, z) = q_j \iff K_i \cdot \{z\} \subseteq K_j$ für alle $i, j \in \{0, \dots, n\}$ und $z \in \Sigma$.

Satz 1.2.5

Sei R eine Rechtskongruenz von endlichem Index, die eine Sprache L saturiert. Dann erkennt der zu R und L gehörenden DFA $M_{R,L}$ die Sprache L und es gilt $R = R_{M_{R,L}}$.

Beweis: Seien K_0, \dots, K_n die Klassen von R , o.B.d.A. $\varepsilon \in K_0$.

Wir zeigen zunächst durch Induktion über die Wortlänge: $\forall w \in \Sigma^* : w \in K_i \iff w \in K_{q_i}$.

Induktionsanfang ($n = 0$): $\varepsilon \in K_0 \wedge \varepsilon \in K_{q_0}$.

Induktionsannahme: die Behauptung gelte für alle Worte der Länge n .

Induktionsschritt: Sei w ein Wort der Länge $n + 1$, d.h. $w = va$ für ein Wort v mit $|v| = n$ und ein Zeichen $a \in \Sigma$. Das Wort v liegt in irgendeiner Klasse K_i . Nach Induktionsvoraussetzung gilt damit $\delta^*(q_0, v) = q_i$. Dann gilt aber:

$$va \in K_j \iff K_i \cdot \{a\} \subseteq K_j \iff \delta(q_i, a) = q_j \iff \delta^*(q_0, va) = q_j \iff va \in K_{q_j}.$$

Es bleibt noch zu zeigen, daß M die Sprache L erkennt:

- Sei u aus L , und u liege in der Klasse K_i . Dann $K_i \subseteq L$ (da R die Sprache L saturiert) und damit $q_i \in F$. Wie oben bewiesen gilt $u \in K_{q_i}$ und damit $\delta^*(q_0, u) = q_i$. Somit wird u von M erkannt.
- Sei u ein von M erkanntes Wort, und sei $q_i = \delta^*(q_0, u)$. Dann ist $q_i \in F$, nach Konstruktion damit $K_i \subseteq L$. Wieder liegt u in der Klasse K_i und damit ist $u \in L$. ■

Korollar 1.2.6

Es gilt $R = R_{M_{R,L(M)}}$ und $M \approx M_{R_M,L(M)}$.

Beispiel 1.2.7

Gegeben der DFA $M = \{\{0, 1, 2\}, \{0, 1\}, \delta, 0, \{2\}\}$ mit

$$\delta = \{(0, 0, 1), (0, 1, 0), (1, 0, 2), (1, 1, 0), (2, 0, 2), (2, 1, 0)\}$$

Dann sind die Klassen

$$\begin{aligned} K_0 &= \{w \in \{0, 1\}^* \mid w \text{ endet auf } 1\} \cup \{\varepsilon\}, \\ K_1 &= \{w \in \{0, 1\}^* \mid w \text{ endet auf } 10\} \cup \{0\}, \\ K_2 &= \{w \in \{0, 1\}^* \mid w \text{ endet auf } 00\}. \end{aligned}$$

Beispiel 1.2.8

Wir definieren eine Rechtskongruenz R auf Σ^* vermöge $uRv \iff |u| = |v|$. Diese hat die unendlich vielen Klassen $K_i = \{w \in \Sigma^* \mid |w| = i\}$ ($i \in \mathbb{N}$). Dafür werden wir also keinen Automaten mit endlich vielen Zuständen finden.

Beispiel 1.2.9

Wir definieren eine Rechtskongruenz R auf $\{a, b\}^*$ vermöge uRv gdw. $|u| - |v|$ durch 3 teilbar ist. Diese Rechtskongruenz hat die Klassen K_0, K_1, K_2 , wobei $K_i = \{w \mid \exists k : |w| = 3k + i\}$. Wir wählen eine Sprache L , die von R saturiert wird, mit ε und a aus L und aa nicht aus L . Der zugehörige DFA ist $M_{R,L} = (\{0, 1, 2\}, \{a, b\}, \delta, 0, F)$ mit $\delta = \{(0, a, 1), (0, b, 1), (1, a, 2), (1, b, 2), (2, a, 0), (2, b, 0)\}$ und $F = \{0, 1\}$.

1.2.4 *Beispiel Stringmatching

Gegeben sei eine endliche Menge SW von Schlüsselworten aus Σ^* und ein Text T aus Σ^* . Wir wollen einen DFA bauen, der bei Eingabe des Texts T immer dann ein Signal gibt (akzeptiert), wenn der bis dahin eingegebene Text mit einem Schlüsselwort endet, d.h. der jedes Vorkommen eines Schlüsselworts im Text signalisiert.

Sei SWA die Menge der Schlüsselwortanfänge. Mit SW ist auch SWA endlich. Außerdem kommt es vor, daß ein Schlüsselwortanfang auf einen anderen Schlüsselwortanfang (oder ein Schlüsselwort) endet, d.h. daß es x, y aus SWA gibt, sodaß y ein Endstück von x ist.

Sei $x \in \text{SWA}$. Wir bezeichnen mit $[x]$ die Menge der Worte, die auf x enden, aber auf keinen längeren Schlüsselwortanfang (deren längstes Schlüsselwortanfangsende x ist), d.h.:

$$[x] = \{u \in \Sigma^* \mid u \text{ endet auf } x \text{ und endet } u \text{ auch auf ein } y \in \text{SWA}, \text{ so } |x| \geq |y|\}.$$

Die Menge $\mathcal{K} = \{[x] \mid x \in \text{SWA}\}$ ist eine Partition von endlichem Index auf Σ^* , denn

1. $\forall x \in \text{SWA} : [x] \neq \emptyset$, da $x \in [x]$,
2. $\forall u \in \Sigma^* \exists x \in \text{SWA} : u \in [x]$ (jedes Wort endet auf ein irgendeinen Schlüsselwortanfang und wenn es der leere Schlüsselwortanfang ε ist),
3. $w \in [x] \implies (\forall y \in \text{SWA} : x \neq y \implies w \notin [y])$ (jedes Wort hat genau ein längstes Schlüsselwortanfangsende).

Die durch \mathcal{K} induzierte Äquivalenz \sim ist auch Rechtskongruenz: Ist $x \sim y$, so haben x und y dasselbe längste Schlüsselwortanfangsende u . Verlängert man beide um das Symbol $a \in \Sigma$, so haben auch xa und ya wieder dasselbe längste Schlüsselwortanfangsende (das längste Schlüsselwortanfangsende von xa kann nicht länger sein als ua , sonst wäre u nicht das längste Schlüsselwortanfangsende von x gewesen. Damit ist das längste Schlüsselwortanfangsende von xa gerade das längste Schlüsselwortanfangsende von ua , und somit gleich dem von ya). Damit sind aber auch xa und ya äquivalent ($xa \sim ya$). Die Rechtskongruenz \sim aber saturiert $L = \{uv \in \Sigma^* \mid v \in \text{SW}\} = \bigcup\{[ux] \mid ux \in \text{SWA} \text{ und } x \in \text{SW}\} = \bigcup\{[y] \mid y \text{ endet auf ein Wort aus SW}\}$. Also gibt es einen DFA der w akzeptiert, wenn w auf ein Schlüsselwort endet.

1.2.5 Sprache L und zugehörige Rechtskongruenz R_L

Definition 1.2.10 (Die zur Sprache L gehörige Rechtskongruenz R_L , trennbar)

Die zu $L \subseteq \Sigma^*$ gehörige Rechtskongruenz R_L ist definiert vermöge:

$$(x, y) \in R_L \iff \forall z \in \Sigma^* : xz \in L \iff yz \in L$$

Umgekehrt ist

$$(x, y) \notin R_L \iff \exists z \in \Sigma^* : xz \in L \iff yz \notin L$$

Wir sagen: x und y sind z -trennbar gdw. $xz \in L \iff yz \notin L$. Wir sagen: x und y sind trennbar, wenn es ein z gibt, das sie trennt. Damit sind x und y trennbar gdw. x und y nicht in der Relation R_L stehen.

Anmerkung 1.2.11

R_L saturiert L : denn, wenn $u \in L$ und $v \notin L$, so sind u und v durch ε trennbar.

Lemma 1.2.12

Seien $L \subseteq \Sigma^*$, R_L die zugehörige Rechtskongruenz und $u, v \in \Sigma^*$. Dann gilt: wenn u und v nicht ε -trennbar sind und $\forall a \in \Sigma : uaR_Lva$, dann gilt uR_Lv .

Beweis: Wären u und v z -trennbar für ein $z \neq \varepsilon$ und beginnt das z mit dem Zeichen a , so wäre auch ua und va trennbar W! ■

Aufgabe 1.2.13

Geben Sie eine Sprache L an und eine Rechtskongruenz R von finitem Index, die L saturiert, so daß die in Lemma 1.2.12 für R_L formulierte Aussage für R nicht gilt.

Definition 1.2.14 (feiner, gröber bei Äquivalenzrelationen)

Seien R_1, R_2 zwei Äquivalenzrelationen auf Σ^* . Wir sagen: R_1 ist *feiner als* R_2 , bzw. R_2 ist *gröber als* R_1 , wenn $R_1 \subseteq R_2$ (d.h. $\forall x, y \in \Sigma^* : xR_1y \implies xR_2y$).

Man erhält die Partition von R_1 aus der von R_2 , indem man die Klassen von R_2 weiter unterteilt (für jede Klasse von R_2 eine disjunkte Überdeckung nichtleerer Unterklassen definiert).

Satz 1.2.15

Ist R eine Rechtskongruenz auf Σ^* , die eine Sprache L über Σ saturiert, so ist R feiner als R_L . Somit R_L ist die gröbste Rechtskongruenz auf Σ^* , die L saturiert.

Beweis: $xRy \implies \forall z \in \Sigma^* : xzRyz$ (R rechtsinvariant)
 $\implies \forall z \in \Sigma^* : (xz \in L \iff yz \in L)$ (R saturiert L)
 $\implies xR_Ly$. (per Definition)

Beispiel 1.2.16

Sei $L = \{w \in \{0, 1\}^* \mid w \text{ enthält genau 2 Einsen}\}$. Wir definieren folgende Partition:

$$\begin{aligned} [\varepsilon] &= \{w \mid w \text{ hat keine Eins}\} \\ [1] &= \{w \mid w \text{ hat genau 1 Eins}\} \\ [11] &= \{w \mid w \text{ hat genau 2 Einsen}\} \\ [111] &= \{w \mid w \text{ hat mehr als 2 Einsen}\} \end{aligned}$$

Alle 4 Klassen sind nichtleer und disjunkt. Außerdem kommt jedes Wort vor, d.h. die Klassen überdecken $\{0, 1\}^*$. Damit haben wir eine Partition. Sie saturiert zudem L , weil L gerade der Klasse $[11]$ entspricht. Nun überzeugen wir uns noch, daß es sich um eine Rechtskongruenz handelt:

$$\begin{array}{ll} [\varepsilon] \cdot \{0\} \subseteq [\varepsilon] & [\varepsilon] \cdot \{1\} \subseteq [1] \\ [1] \cdot \{0\} \subseteq [1] & [1] \cdot \{1\} \subseteq [11] \\ [11] \cdot \{0\} \subseteq [11] & [11] \cdot \{1\} \subseteq [111] \\ [111] \cdot \{0\} \subseteq [111] & [111] \cdot \{1\} \subseteq [111] \end{array}$$

Und zuletzt stellen wir fest, daß die durch diese Partition definierte Rechtskongruenz – nennen wir sie einmal R – gerade die zu L gehörige Rechtskongruenz R_L ist.

1. $R \subseteq R_L$ wegen Satz 1.2.15.
2. ε und 1 sind 1 -trennbar, ε und 11 sind ε -trennbar, ε und 111 sind 11 -trennbar, 1 und 11 sind ε -trennbar, 1 und 111 sind 1 -trennbar und 11 und 111 sind ε -trennbar, d.h. die Repräsentanten der Klassen von R sind paarweise trennbar, also nicht rechtskongruent unter R_L . Also enthält keine R_L -Klasse zwei R -Klassen. Damit kann R nicht echt feiner als R_L sein. Also ist $R = R_L$.

Der zugehörige Automat hat 4 Zustände, die gerade den 4 Klassen entsprechen. Also nehmen wir doch die Klassen selbst als Zustände: $M = (\{[\varepsilon], [1], [11], [111]\}, \{0, 1\}, \delta, [\varepsilon], \{[11]\})$ mit:

δ	$[\varepsilon]$	$[1]$	$[11]$	$[111]$
0	$[\varepsilon]$	$[1]$	$[11]$	$[111]$
1	$[1]$	$[11]$	$[111]$	$[111]$

Beispiel 1.2.17

$L = \{0^n 1^n \mid n \in \mathbb{N}, n > 0\}$. Wir definieren folgende Klassen:

- $K_i := [0^i] = \{0^i\}$ ($i \in \mathbb{N}$) (Achtung: $\{\varepsilon\}$ bildet eine eigene Klasse, die Klasse K_0)
- $\widetilde{K}_i := [0^{i+1}1] = \{0^{n+i}1^n \mid n \in \mathbb{N}, n > 0\}$ ($i \in \mathbb{N}$)
- $K_R := [1] = \{0, 1\}^* - \bigcup \{K_i, \widetilde{K}_i \mid i \in \mathbb{N}\}$

Wieder haben wir eine disjunkte Überdeckung nichtleerer Mengen von $\{0, 1\}^*$. Die dazuhörige Äquivalenz R ist eine Rechtskongruenz, die L saturiert (L ist gerade die Klasse $\widetilde{K_0}$). Die Worte einer R -Klasse sind nicht trennbar. Damit ist $R = R_L$.

Beispiel 1.2.18

Sei $L = \{ab, aba, aab\}^*$. Wir wollen die zu L gehörige Relation R_L bestimmen. Sei $[x]$ die Klasse, die x enthält. x ist ein Repräsentant seiner Klasse. Wir gehen die Worte ihrer Länge nach durch, und schauen, welche zu früheren äquivalent sind oder von allen früheren trennbar sind, und daher Anlaß zu einer neuen Klasse geben:

Worte der Länge 0:

Die **1. Klasse** ist $[\varepsilon]$.

Worte der Länge 1:

$a \notin [\varepsilon]$, da $a \in \overline{L}$, $\varepsilon \in L$.

Die **2. Klasse** ist $[a]$.

$b \notin [\varepsilon]$, da $b \in \overline{L}$, $\varepsilon \in L$.

$b \notin [a]$, da $bb \in \overline{L}$, $ab \in L$, d.h. b trennt.

Die **3. Klasse** ist $[b]$.

Worte der Länge 2:

$aa \notin [\varepsilon]$ (ε trennt).

$aa \notin [a]$ (ab trennt).

$aa \notin [b]$ (b trennt).

Die **4. Klasse** ist $[aa]$.

$ab \notin [\varepsilon]$ (a trennt).

$ab \notin [a]$ (a trennt).

$ab \notin [b]$ (a trennt).

$ab \notin [aa]$ (b trennt).

Die **5. Klasse** ist $[ab]$.

$bb \in [b]$ ebenso wie bw für alle $w \in \{a, b\}^*$ (alle Worte, die mit b beginnen, liegen in \overline{L}).

Worte der Länge 3:

aaa und $aaaw \in [b]$ für alle $w \in \{a, b\}^*$ (alle Worte, die mit aaa beginnen, liegen in \overline{L}).

$aab \in [\varepsilon]$, denn für alle $z \in L$ gilt: $\varepsilon z \in L$ und $aabz \in L$ und für alle $z \in \overline{L}$ gilt:

$\varepsilon z \notin L$ und $aabz \notin L$, da $aabz \in L \implies z \in L$, denn:

$(w \in L \wedge |w| \geq 3) \implies (\exists v \in L : w = avv \vee w = abav \vee w = aabv)$.

$aba \notin [\varepsilon]$ (b trennt).

$aba \notin [a]$ (ε trennt).

$aba \notin [b]$ (ε trennt).

$aba \notin [aa]$ (ε trennt).

$aba \notin [ab]$ (a trennt).

Die **6. Klasse** ist $[aba]$.

$abb \in [b]$ (keine Fortsetzung führt nach L).

$baa, \dots, bbb \in [b]$.

Worte der Länge 4:

$aaaa \in [b]$ (siehe oben).

$aaab \in [b]$ (siehe oben).

$aaba \in [a]$ ($aab \sim \varepsilon \implies aaba \sim a$).

$aabb \in [b]$ ($aab \sim \varepsilon \implies aabb \sim b$).

$abaa \in [a]$, denn: $abaa \in L$ gdw. $\exists x \in L : z = bx \vee z = abx \vee z = bax$ gdw. $az \in L$.

$abab \in [ab]$, denn: $abab \in L$ gdw. ($z = \varepsilon \vee \exists x \in L : z = ax$) gdw. $abz \in L$.

$abba \in [b]$ (kein Wort aus L beginnt mit abb).

$abbb \in [b]$ (kein Wort aus L beginnt mit abb).

Die Worte, die mit b beginnen haben wir schon gehabt.

Also: alle Worte der Länge 4 sind zu kürzeren äquivalent. Damit sind wegen der Rechtsinvarianz von R_L auch alle Worte einer Länge größer 4 zu einem Wort einer Länge kleiner 4 äquivalent: Ist $w = uv$ mit $|u| = 4$, so existiert u mit $|u'| < 4$ und uR_Lu' . Dann ist auch $uvR_Lu'v$. Also ist jedes Wort einer Länge ≥ 4 zu einem kürzeren äquivalent. Per Induktion ist es dann sogar zu einem Wort < 4 äquivalent. Damit haben wir aber auch schon alle Klassen von R_L gefunden.

1.2.6 Äquivalenzrelation auf Q

Notation 1.2.19 ($L_p(M)$)

Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein DFA und $p \in Q$. Dann ist $L_p(M) = \{w \in \Sigma^* \mid \delta^*(p, w) \in F\}$ die Menge aller Worte, die vom Zustand p in einen Endzustand führen, die also von M akzeptiert würden, wenn p der Startzustand wäre.

Definition 1.2.20 (äquivalente, trennbare Zustände)

Gegeben sei DFA $M = (Q, \Sigma, \delta, q_0, F)$. Für Zustände $p, q \in Q$ definieren wir:

p und q sind *äquivalent* (i.Z. $p \sim q$) gdw. $L_p(M) = L_q(M)$
 gdw. $\forall z \in \Sigma^* : (\delta^*(p, z) \in F \iff \delta^*(q, z) \in F)$.

p und q sind *z -trennbar* gdw. $[\delta^*(p, z) \in F \iff \delta^*(q, z) \notin F] \quad (z \in \Sigma^*)$.

p und q sind *trennbar* (i.Z. $p \approx q$) gdw. $\exists z \in \Sigma^* : p$ und q sind z -trennbar.

Anmerkung 1.2.21

1. Die Zustände p und q eines DFA $M = (Q, \Sigma, \delta, q_0, F)$ sind trennbar gdw. p und q nicht äquivalent sind.
2. \sim ist eine Äquivalenzrelation auf Q .

Lemma 1.2.22

Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein vollständiger DFA, dessen Zustände alle vom Startzustand aus erreichbar sind, und $u \in K_p(M)$, $v \in K_q(M)$. Dann: $p \sim q \iff uR_{L(M)}v$.

Beweis: $p \sim q \iff \forall z \in \Sigma^* : (\delta^*(p, z) \in F \iff \delta^*(q, z) \in F)$
 $\iff \forall z \in \Sigma^* : (\delta^*(q_0, uz) \in F \iff \delta^*(q_0, vz) \in F)$
 $\iff \forall z \in \Sigma^* : (uz \in L(M) \iff vz \in L(M))$
 $\iff uR_{L(M)}v$. ■

Satz 1.2.23

Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein vollständiger DFA, dessen Zustände alle erreichbar sind und der keine zwei äquivalente Zustände hat. Dann und nur dann ist $R_M = R_{L(M)}$.

Beweis: \implies : Sei also M ein DFA wie oben beschrieben, der die Sprache L erkennt. Aus xR_My folgt xR_Ly , da R_M feiner als R_L . Wenn xR_Ly , dann ist $\delta^*(q_0, x) = \delta^*(q_0, y)$, da nach obigem Lemma $\delta^*(q_0, x) \sim \delta^*(q_0, y)$, also ist xR_My .

\impliedby : Sei $R_M = R_L$. Sind p und q zwei verschiedene Zustände von M , dann gibt es zwei verschiedene Worte x und y mit $x \in K_p$ und $y \in K_q$ (M ist deterministisch und in jeden Zustand führt wenigstens ein Wort). Also $(x, y) \notin R_M$. Damit ist auch $(x, y) \notin R_L$ und nach Lemma wieder: $p \approx q$. ■

Korollar 1.2.24

Ist M ein vollständiger DFA, so daß alle Zustände erreichbar und keine zwei Zustände äquivalent sind, dann ist M (bzgl. der Anzahl der Zuständen) ein minimaler DFA für $L(M)$.

Beweis: Wegen vorigem Satz ist $R_M = R_{L(M)}$. Gäbe es einen kleineren DFA M' , so wäre $R_{M'}$ eine Rechtskongruenz auf Σ^* , die $L(M)$ saturiert mit weniger Klassen als R_L . W! ■

Anmerkung 1.2.25

Der minimale DFA für L ist eindeutig (bis auf Isomorphie).

Beweis: Ist M ein minimaler DFA für L , so ist $R_M = R_L$. Nach Korollar 1.2.6 gibt es zu R_L und L aber (bis auf Isomorphie) nur den DFA $M_{R_L,L} \approx M$. ■

Notation 1.2.26 (regulär, REG)

Die von finiten Automaten erkannten Sprachen wollen wir *regulär* nennen. REG sei die Klasse aller regulären Sprachen.

Jetzt können wir folgenden Satz formulieren:

Satz 1.2.27 (Satz von Nerode)

Eine Sprache L ist regulär, gdw. R_L finiten Index hat, gdw. es eine Rechtskongruenz R von finitem Index auf Σ^* gibt, die L saturiert.

1.3 Die Minimisierung von finiten Automaten

Beispiel 1.3.1

Sei $L = \{ab, aba, aab\}^*$.

	δ	a	b
SF	1	2	—
	2	4	13
F	13	12	—
	4	—	1
F	12	24	13
	24	4	13
	—	—	—

	1	2	—	4	13	12	24
1	-	ε	ε	ε	a	b	ε
2		-	b	ab	ε	ε	?
—			-	b	ε	ε	b
4				-	ε	ε	ab
13					-	a	ε
12						-	ε
24							-

	δ	a	b
SF	1	2	—
	2	4	13
F	13	12	—
	4	—	1
F	12	2	13
	—	—	—

Die erste Tabelle gibt einen DFA für die Sprache L an. Die mittlere Tabelle gibt an, durch welche Worte sich die Zustandspaare trennen lassen: Die Zustände 2 und 24 (hier mit ? markiert) sind äquivalent, weil sie sowohl mit dem Symbol a als auch mit dem Symbol b in denselben Zustand überführt werden: $\delta(2, x) = \delta(24, x)$ ($x \in \{a, b\}$). Die rechte Tabelle gibt den minimalen DFA für L an.

Wenn $Q = F$ oder $F = \emptyset$, so besteht der minimale DFA aus einem Zustand. Also nehmen wir an, daß $Q \neq F$ und $F \neq \emptyset$. Um zu einem solchen vollständigen DFA den minimalen DFA zu finden, suchen wir nun aber nicht direkt die Paare äquivalenter Zustände, sondern umgekehrt, wir fassen alle Zustände zu einer großen Klasse zusammen und versuchen diese Klasse immer weiter zu unterteilen, indem wir Zustände, die sich trennen lassen, in verschiedene Unterklassen sortieren.

Wir bilden zunächst die Menge Q aller erreichbaren Zustände. Nun teilen wir Q , indem wir die Zustände, die sich durch das leere Wort ε trennen lassen, in die beiden Unterklassen F und $Q - F$ sortieren. Dann bilden wir Klassen, indem wir Zustände, die sich durch Worte bis zur Länge 1 trennen lassen, einsortieren. Das sind die Klassen $\{q \in K \mid \delta(q, a) \in K'\}$ für alle $K, K' \in \{F, Q - F\}$ und $a \in \Sigma$, insgesamt $4|\Sigma|$ Klassen, von denen allerdings einige leer sein können. Allgemein: haben wir Klassen K_1, \dots, K_m gefunden, für die gilt, daß sich Zustände derselben Klasse nicht durch Worte einer Länge kleinergleich k trennen lassen, wohl aber Zustände verschiedener Klassen, dann bilden wir durch weitere Unterteilung dieser Klassen Unterklassen von Zuständen, so daß sich Zustände verschiedener Unterklassen durch Worte bis zur Länge $k + 1$ trennen lassen, Zustände derselben Unterklasse aber nicht.

Betrachten wir die Klassen $\{q \in K \mid \delta(q, a) \in K'\}$ für alle $K, K' \in \{K_1, \dots, K_m\}$ und $a \in \Sigma$. Sind etwa p und q Zustände aus K_i und $\delta(p, a) \in K_j$ und $\delta(q, a) \in K_l$ ($j \neq l$), so lassen sich die Zustände $\delta(p, a)$ und $\delta(q, a)$ durch ein Wort z einer Länge kleinergleich k trennen, p und q also durch az , ein Wort einer Länge kleinergleich $k + 1$. Die Klasse K_i muß also weiter unterteilt werden in Unterklassen K_{ij} und K_{il} , die jeweils p und q enthalten.

Lassen sich die Klassen K_1, \dots, K_m aber durch kein Symbol aus Σ weiter unterteilen, so sind die Zustände einer Klasse überhaupt nicht mehr trennbar. Sind nämlich p und q in einer Klasse und ließen sich p und q doch durch ein Wort $w = a_1 \cdots a_n$ trennen, so ließen sich die Zustände $\delta^*(p, w)$ und $\delta^*(q, w)$ durch das leere Wort trennen. Da aber mit p und q auch $\delta(p, a_1)$ und $\delta(q, a_1)$ in einer Klasse liegen und ebenso $\delta^*(p, a_1 a_2)$ und $\delta^*(q, a_1 a_2)$ usw. und schließlich auch $\delta^*(p, w)$ und $\delta^*(q, w)$, so kann dies nicht sein.

Damit sind die Zustände einer Klasse äquivalent und wir sind fertig, da wir die gröbste Klasseneinteilung gefunden haben, deren Klassen nur äquivalente Zustände enthalten, d.h. die Klasseneinteilung für die auf den Zuständen definierten Äquivalenzrelation, die aber gemäß Lemma 1.2.22 der zur Sprache L gehörigen Rechkongruenz R_L entspricht.

Algorithmus 1.3.2 (Erreichbarkeitsalgorithmus)

Input: ein vollständiger DFA $M = (Q, \Sigma, \delta, q_0, F)$.

Output: ein vollständiger DFA M' , der äquivalent ist zu M und nur vom Startzustand aus erreichbare Zustände besitzt.

Idee: Das Alphabet sei $\{a, b\}$. Wir durchlaufen den Graphen vom Startzustand q_0 aus immer die a -Kante entlang bis wir auf einen Zustand stoßen, den wir schon einmal besucht haben (beachte, daß M vollständig ist). Dann gehen wir die a -Kante zurück und gehen mit der b -Kante weiter. Haben wir den jetzt erreichten Zustand noch nicht besucht, so verfahren wir von diesem neuen Ausgangspunkt aus wieder genauso. Haben wir ihn aber schon besucht, so gehen wir die b -Kante zurück und landen in einem Zustand, von dem aus wir schon beide Kanten durchlaufen haben. Zu diesem Zustand sind wir schon 2-mal zurückgekehrt, d.h. alles was wir von diesem Zustand aus erreichen können haben wir schon besucht.

Sind wir zum Zustand q_0 zum zweiten Mal zurückgekehrt, so haben wir alle Zustände besucht, die man von q_0 aus erreichen kann.

Algorithmus: (Tiefensuche, engl. depth-first search)

```

procedure ERR(p)
begin
  if p ∉ Z then begin
    Z := Z ∪ {p};
    forall a ∈ Σ do ERR(δ(p, a));
  end {if};
end;

begin
  Z := ∅ ; ERR(q0); Q' := Z; F' := F ∩ Q'; δ' := δ ∩ Q' × Σ × Q';
  M' := (Q', Σ, δ', q0, F');
end.
    
```

Der Zeitaufwand ist proportional zu der Zahl n der Zustände von M . ■

Algorithmus 1.3.3 (Minimalisierungsalgorithmus straightforward)

Input: ein vollständiger DFA $M = (Q, \Sigma, \delta, q_0, F)$, dessen Zustände alle vom Startzustand aus erreichbar und geordnet (\leq) sind, mit $|Q| = n$ und $|\Sigma| = k$.

Output: der minimale DFA M' zu M .

Algorithmus:

```

begin
   $\Pi' := \{Q\}; \Pi := \{F, Q - F\};$ 
  while  $\Pi' \neq \Pi$  do begin
     $\Pi' := \Pi;$ 
    for all  $a \in \Sigma$  do  $\Pi := \{\{q \in K \mid \delta(q, a) \in K'\} \mid K \in \Pi, K' \in \Pi'\} - \{\emptyset\}$ 
  end {while};
   $Q_0 :=$  die Klasse von  $\Pi$ , in der  $q_0$  liegt;
   $F' := \{K \in \Pi \mid K \subseteq F\};$ 
  for all  $(K, a) \in \Pi \times \Sigma$  do
     $\delta'(K, a) :=$  die Klasse von  $\Pi$ , die  $\delta(K, a)$  enthält;
   $M' := \Pi, \Sigma, \delta', Q_0, F'$ ;
end.

```

Der Zeitaufwand ist (bei geeigneter Implementierung) quadratisch in der Zahl n der Zustände: In jedem der möglicherweise n Durchläufe der while-Schleife wird zu jedem Zustand und jedem Symbol der Nachfolger in den Klassen gesucht. Wir legen einen $n \times (k + 1)$ -Array an und schreiben in die oberste Zeile in Feld i den Namen der Klasse, in der sich der Zustand i gerade befindet, in die nächste Zeile den Namen der Klasse, in die der Zustand nach Anwendung des ersten Symbols gelangt, in die übernächste Zeile den Namen der Klasse, in die der Zustand nach Anwendung des zweiten Symbols gelangt, usw., bis die letzte Zeile gefüllt ist. Dann zählen wir die verschiedenen Klassennamen in der ersten und letzten Zeile. Ist diese Zahl in beiden Zeilen gleich, so hat sich die Partition nicht verändert. Wenn nicht, so kopieren wir die unterste Zeile in die oberste und beginnen den nächsten Durchlauf. So kostet ein Durchlauf also nur ckn Schritte für eine feste Konstante c . ■

Wenn p und q trennbar sind, so sind natürlich auch q und p trennbar. Also genügt es statt aller n^2 Paare (p, q) nur die Hälfte davon zu untersuchen. Wir beschränken uns daher auf die Paare (p, q) mit $p \leq q$. Der folgende Algorithmus markiert Paare von Zuständen, die als trennbar gefunden wurden. Er beginnt damit, daß alle Paare aus $F \times (Q - F)$ markiert werden, da ihre Zustände ja bereits durch das leere Wort getrennt werden. Dann überprüft er für jedes noch nicht markiertes Paar, ob eines seiner Nachfolgerpaare (Paar der Nachfolgezustände unter einem Symbol aus Σ) markiert ist. Ist dies so, so wird das untersuchte Paar markiert. Wäre dies alles, so wäre der Zeitaufwand proportional zu n^4 , da in einem Durchgang für jedes der $\frac{1}{2}n^2$ Paare die Nachfolgerpaare unter jedem Symbol überprüft werden müßten, und man unter Umständen $\frac{1}{2}n^2$ Durchläufe hätte, wenn nämlich in jedem Durchlauf nur ein weiteres Paar markiert würde. Daher wird für jedes Paar (p, q) eine Liste $LIST(p, q)$ angelegt, in der wieder Paare aufgelistet werden können. Sind die Nachfolgerpaare eines Paares (p, q) alle nicht markiert, so wird (p, q) auf die Listen aller seiner Nachfolgerpaare gesetzt. Wird ein Nachfolgerpaar später einmal markiert, so werden in einem Aufwasch auch gleich alle Paare auf dessen Liste mit markiert, also auch (p, q) . So werden mit Markierung eines Paares (p, q) also alle Paare auf $LIST(p, q)$ markiert und wiederum alle Paare auf der Liste der Paare in $LIST(p, q)$, usw..Dadurch vermeidet man, daß die Nachfolgerpaare von (p, q) immer wieder überprüft werden. Man überprüft sie einmal und meldet bei ihnen den Anspruch von (p, q) auf Markierung an, der eingelöst wird, wenn diese selbst markiert werden.

Algorithmus 1.3.4 (Minimalisierungsalgorithmus nach Hopcroft-Ullman)

Input: ein vollständiger DFA $M = (Q, \Sigma, \delta, q_0, F)$, dessen Zustände alle vom Startzustand aus er-

reichbar sind, mit $|Q| = n$ und $|\Sigma| = k$.

Output: der minimale DFA M' zu M .

Algorithmus:

```

procedure MARK( $p, q$ );
begin
  if ( $p, q$ ) nicht markiert then begin
    markiere ( $p, q$ );
    for all ( $r, s$ )  $\in$  LIST( $p, q$ ) do MARK( $r, s$ )
  end {if};
end;

begin
  for all ( $p, q$ )  $\in$   $Q \times Q$  with  $p \leq q$  do if  $p \in F \iff q \notin F$  then markiere( $p, q$ );
  for all ( $p, q$ )  $\in$   $Q \times Q$  with ( $p \leq q$  and nicht markiert) do
    if ( $\delta(p, a), \delta(q, a)$ ) markiert für ein  $a \in \Sigma$ 
      then MARK( $p, q$ )
      else for all  $a \in \Sigma$  do
        put ( $p, q$ ) in LIST( $\delta(p, a), \delta(q, a)$ ) bzw. LIST( $\delta(q, a), \delta(p, a)$ );
      {die jetzt nicht markierten Paare enthalten äquivalente Zustände}
       $Q' :=$  eine maximale Menge nicht äquivalenter Zustände aus  $Q$ ;
       $F' := Q' \cap F$ ;
      for all ( $p, a, q$ )  $\in \delta$  do begin
        find  $p', q' \in Q'$  with  $p \sim p'$  and  $q \sim q'$ ;  $\delta' := \delta' \cup \{(p', a, q')\}$ ;
      end {for}
       $M' := (Q', \Sigma, \delta, q_0, F)$ ;
end.

```

Der Zeitaufwand ist quadratisch:

1. Aufwand ohne die Prozedur MARK: Für jedes der $\frac{1}{2}n^2$ Paare werden höchstens $k + 1$ Paare aufgesucht, wenn k die Mächtigkeit von Σ ist (für jedes Paar wird das Paar selbst aufgesucht, dann die k Nachfolgepaare angesprungen).
2. Aufwand der Prozedur MARK: In allen Listen zusammen stehen höchstens $\frac{1}{2}kn^2$ Paare (ein Paar steht in höchstens k Listen), d.h. höchstens so viele Paare müssen in Folge einer Markierung noch einmal aufgesucht werden, um nachträglich markiert zu werden.

Damit werden insgesamt höchstens $(k + 1)n^2$ Paare aufgesucht. ■

Satz 1.3.5

Sei $\Sigma = \{a_1, \dots, a_n\}$. Die Sprache $L = \{wa \mid w \in (\Sigma - \{a\})^*, a \in \Sigma\}$, (letztes Symbol von w kommt vorher nicht vor), wird von einem DFA nur erkannt, wenn er wenigstens $2^{n+1} - 1$ Zustände hat.

Beweis: Die zu L gehörige Rechtskongruenz R_L ist:

$$\begin{aligned}
 uR_Lv & \text{ gdw. } uz \in L \iff vz \in L \text{ für alle } z \in \Sigma^* \\
 & \text{ gdw. } 1) u, v \text{ haben dieselben Symbole und} \\
 & \quad 2) u \in L \iff v \in L \text{ (} ab \text{ trennbar von } abb \text{)}
 \end{aligned}$$

R_L hat $2^n + 2^n - 1 = 2^{n+1} - 1$ Klassen. Der zugehörige minimale Automat hat die Zustände $\{(A, -) \mid A \subseteq \Sigma\} \cup \{(A, +) \mid \emptyset \subset A \subseteq \Sigma\}$. In $(A, -)$ laufen die Worte mit genau den Zeichen aus A , die nicht aus L sind. $(\emptyset, -)$ ist der Startzustand. In $(A, +)$ laufen alle Worte mit genau den Zeichen aus A , die aus L sind ($(\emptyset, +)$ ist nicht nötig, da $\varepsilon \notin L$). ■

Satz 1.3.6

Die Sprache $L = \{w \in \{0,1\}^* \mid |w| \geq n, \text{ das } n\text{-te Symbol von hinten ist } 1\}$ wird von einem DFA nur erkannt, wenn er wenigstens 2^n Zustände hat.

Beweis: Die zu L gehörige Rechtskongruenz erhalten wir, wenn wir uns überlegen, daß 2 Worte einer Länge $\geq n$ trennbar sind, gdw. sie sich in einer der letzten n Stellen unterscheiden. Um auch die kürzeren Worte mit einzubeziehen, wählen wir folgende Formalisierung:

Zu jedem der 2^n Worte $t = t_1 \cdots t_n \in \{0,1\}^n$ bilden wir die Klasse

$$K_t = \{w \in \{0,1\}^* \mid t_i = 1 \implies |w| \geq i \text{ und das } i\text{-te Symbol von hinten ist } 1 \ (i = 1, \dots, n)\} \\ = \{w \in \{0,1\}^* \mid \text{das } i\text{-te Symbol von } 0^n w \text{ von hinten ist } t_i \ (i = 1, \dots, n)\}$$

Dies sind die Klassen von R_L . Wenn wir t als eine n -stellige Binärzahl (mod 2^n) auffassen, können wir den zugehörigen minimalen DFA wie folgt skizzieren: $M = ([0, 2^n - 1], \Sigma, \delta, 0, \{t \mid t \geq 2^{n-1}\})$ mit

$$\delta = \{(t, 0, 2t \bmod 2^n), (t, 1, 2t + 1 \bmod 2^n) \mid t \in [0, 2^n - 1]\}$$

■

1.4 Das reguläre Pumpinglemma

Definition 1.4.1 (reguläre Pumpingeigenschaft)

Eine Sprache L aus Σ^* hat die *reguläre Pumpingeigenschaft* gdw.

$$(\exists k \in \mathbb{N}) (\forall w \in L \text{ mit } |w| \geq k) (\exists u, v, x \in \Sigma^* \text{ mit } w = uvx, v \neq \varepsilon, |v| \leq k) (\forall i \in \mathbb{N}) : uv^i x \in L$$

Oder anders geschrieben:

$$\begin{array}{ccccccc} \exists & \forall & \exists & \forall & : & uv^i x \in L \\ k \in \mathbb{N} & w \in L & u, v, x \in \Sigma^* & i \in \mathbb{N} & & & \\ & |w| \geq k & w = uvx & & & & \\ & & v \neq \varepsilon & & & & \\ & & |v| \leq k & & & & \end{array}$$

Anschaulich heißt dies: Alle genügend großen Worte w aus L lassen sich in drei Teile u, v, x zerlegen, wobei der mittlere Teil v (die Pumpstelle) nicht leer und nicht zu groß sein darf, so daß das Wort auch dann noch zu L gehört, wenn der mittlere Teil herausgenommen (herausgepumpt) wird oder beliebig oft eingefügt (hineingepumpt) wird. Salopp: in allen genügend großen Worten aus L finden wir eine kleine Pumpstelle.

Etwas schärfer ist die Aussage, wenn wir auch noch verlangen, daß die Pumpstelle schon unter den ersten k Zeichen des Wortes w aus L zu finden ist:

Definition 1.4.2 (schärfere reguläre Pumpingeigenschaft)

$$\begin{array}{ccccccc} \exists & \forall & \exists & \forall & : & uv^i x \in L \\ k \in \mathbb{N} & w \in L & u, v, x \in \Sigma^* & i \in \mathbb{N} & & & \\ & |w| \geq k & w = uvx & & & & \\ & & v \neq \varepsilon & & & & \\ & & |uv| \leq k & & & & \end{array}$$

Lemma 1.4.3 (reguläres Pumping Lemma)

Jede reguläre Sprache hat die schärfere reguläre Pumpingeigenschaft.

Beweis: Ist k die Zahl der Zustände des DFA für die reguläre Sprache, w ein Wort aus L mit einer Länge größergleich k , und z der Präfix bestehend aus den ersten k Zeichen von w , so durchläuft z einen Zustand des DFA wenigstens zweimal, d.h. es gibt einen Teil v von z , dessen Länge kleinergleich k ist und der von einem Zustand q wieder zu q führt. Dieser kann herausgenommen oder wiederholt eingefügt werden, ohne das Akzeptierungsverhalten des DFA zu ändern. ■

Üblicherweise wird dieser Satz dazu benützt, zu zeigen, daß eine Sprache nicht regulär ist, weil sie nicht einmal die reguläre Pumpingeigenschaft hat, nicht pumpbar ist. Will man zeigen, daß L nicht pumpbar ist, gilt es also zu zeigen:

$$\forall k \in \mathbb{N} \quad \exists w \in L \quad \forall u, v, x \in \Sigma^* \quad \exists i \in \mathbb{N} : uv^i x \notin L$$

$$|w| \geq k \quad w = uvx$$

$$v \neq \varepsilon$$

$$|v| \leq k$$

Aufgabe 1.4.4

$\{a^n b^n \mid n \in \mathbb{N}\}$ hat nicht die reguläre Pumpingeigenschaft.

Satz 1.4.5

$L = \{a^{n^2} \mid n \in \mathbb{N}\}$ hat nicht die regulären Pumpingeigenschaft.

Beweis: $(n + 1)^2 - n^2 = 2n + 1$, d.h. der Abstand benachbarter Quadratzahlen steigt mit n . Die reguläre Pumpingeigenschaft fordert eine unendliche Reihe von Worten, deren Längen sich nur um eine feste Konstante c unterscheiden.

Hätte L die reguläre Pumpingeigenschaft und sei k die Pump-Konstante, so wäre für jedes $n > k$ auch $n^2 + r$ eine Quadratzahl für ein $r \leq k$, was nicht sein kann, da $n^2 + r < n^2 + n < (n + 1)^2$. ■

Satz 1.4.6

$\{a^p \mid p \text{ Primzahl}\}$ hat nicht die reguläre Pumpingeigenschaft.

Beweis: Es gibt unendlich viele Primzahlen. Denn wäre p die größte, so wäre $p! + 1$ Primzahl oder teilbar durch eine Primzahl größer p . W!

Es gibt beliebig große Abstände zwischen benachbarten Primzahlen: die Zahlen $n! + 2, \dots, n! + n$ sind alle teilbar, d.h. an der Stelle $n!$ haben wir wenigstens $n - 1$ Nichtprimzahlen zwischen zwei benachbarten Primzahlen. Die reguläre Pumpingeigenschaft läßt aber wachsende Längenabstände zwischen benachbarten Worten nicht zu.

Angenommen wir können in a^p ein Teilwort der Länge $r \leq p$ p -mal hineinpumpen, so müßte auch $a^{p+rp} = a^{(r+1)p}$ in der Sprache liegen. W! da $(r + 1)p$ offenbar eine zusammengesetzte Zahl, also keine Primzahl ist. ■

Satz 1.4.7

Es gibt nichtreguläre Sprachen mit der regulären Pumpingeigenschaft.

Beweis: Sei $L \subseteq \Sigma^*$ eine nichtreguläre Sprache und $c \notin \Sigma$. Die Sprache $\{c\}^* \cdot L$ hat die einfache reguläre Pumpingeigenschaft nur wenn L sie schon hat (betrachte die Wörter aus $\{c\} \cdot L$). Darum betrachten wir $\hat{L} := \{c\}^* \cdot L \cup \Sigma^*$. Dann hat \hat{L} die reguläre Pumpingeigenschaft, denn enthält $w \in \hat{L}$ ein c , so kann man dieses pumpen (raus und rein), und enthält es kein c , so kann man überall pumpen. \hat{L} ist nicht regulär; denn wäre \hat{L} regulär, so auch $\{c\} \cdot L = \hat{L} \cap \{c\} \cdot \Sigma^*$ und damit auch L selbst. W! (siehe Abschlußigenschaften von regulären Mengen) ■

Definition 1.4.8 (ultimativ periodisch (u.p.))

Eine Menge $U \subseteq \mathbb{N}$ von natürlichen Zahlen heißt *ultimativ periodisch* gdw.

$$\exists k \in \mathbb{N} \exists p \in \mathbb{N}_+ (\forall m \in \mathbb{N} \text{ mit } m \geq k) : m \in U \iff m + p \in U.$$

(mit jeder Zahl $m \in U$, $m \geq k$, ab einer Größe k , gehört die ganze p -Reihe von m zu U , d.h. $\{m + ip \mid i \in \mathbb{N}\} \subseteq U$)

Notation 1.4.9 (#L)

Sei $L \subseteq \Sigma^*$. Dann ist $\#L := \{n \mid \exists w \in L : |w| = n\}$.

Notation 1.4.10 (1-Symbol-Sprache)

Wir nennen eine Sprache über einem einelementigen Alphabet eine 1-Symbol-Sprache.

Satz 1.4.11

Für 1-Symbol-Sprachen L gilt: L regulär gdw. $\#L$ ultimativ periodisch.

Beweis: Ist $L \subseteq \{1\}^*$ regulär, so existiert ein DFA M für L . M habe k Zustände. Der Graph von M hat folgende Form: ein Anfangsstück (Länge a), das in einen Kreis (Länge p) mündet ($a + p = k$, a oder p können 0 sein). Dann gilt für $m \geq a$: $1^m \in L$ gdw. $1^{m+p} \in L$.

Sei $\#L$ ultimativ periodisch, d.h. $1^m \in L$ gdw. $1^{m+p} \in L$ für $m \geq k$. Dann definieren wir einen DFA mit den Zuständen q_0, \dots, q_{k+p} dessen Graph ein Anfangsstück der Länge k hat, das in einen Kreis der Länge p mündet, d.h. $(q_i, 1, q_{i+1}) \in \delta$ für $i = 0, \dots, k + p$ und $(q_{k+p}, 1, q_k) \in \delta$. Startzustand ist q_0 und die Endzustandsmenge ist $F = \{q_i \mid i \in \#L\}$. ■

Satz 1.4.12

Sei $L \subseteq \{1\}^*$. Dann ist L^* regulär.

Beweis: $\#L^* = \{n \in \mathbb{N} \mid \exists k \in \mathbb{N} \exists n_1, \dots, n_k \in \#L : n = n_1 + \dots + n_k\}$. Sei $p = \min(\#L - \{0\})$ und $E = \{m \in \#L^* \mid \forall n \in \#L^* : n < m \implies n \not\equiv m \pmod{p}\}$, d.h. E ist die Menge der Zahlen, die eine neue p -Reihe begründen. Es ist $|E| \leq p$. Sei $M = \max E$. Dann gilt für jedes $m > M$: $m \in \#L^* \iff m + p \in \#L^*$, d.h. $\#L^*$ ist ultimativ periodisch und L^* regulär. ■

1.5 Nichtdeterminismus

Beziehungen sind in der Regel nicht eindeutig. Eine Mutter kann mehrere Kinder haben, d.h. die Mutter-Kind-Beziehung ist nicht eindeutig. Nur sehr spezielle Relationen sind (rechts-)eindeutig, so daß ein Element zu höchstens einem anderen in dieser Relation stehen kann. Diese speziellen Relationen nennen wir auch Funktionen oder Abbildungen. Umgekehrt können wir die Relationen als nichtdeterministische Funktionen auffassen, nämlich Funktionen, die zu einem gegebenen Argument mehrere Werte liefern.

Unsere Computer können nur deterministisch arbeiten, d.h. die Programme müssen so geschrieben sein, daß für jeden Prozessor in jedem Moment klar ist, welches der nächste Befehl ist, der ausgeführt werden soll. Damit können die Computer auch nur Funktionen berechnen: für jede Eingabe gibt es höchstens ein Ergebnis.

Allerdings ist die Welt voll interessanter Relationen, voll nichtdeterministischer Phänomene. Ein gutes Beispiel ist unser Immunsystem. Dringt ein fremdes Protein (ein Antigen) in unseren Körper ein, so setzt sich ein von uns erzeugter Antikörper darauf und signalisiert so den Fresszellen, daß dieses Antigen zu vernichten ist. Gebildet werden diese Antikörper von bestimmten weißen Blutkörperchen, den

Plasmazellen, die sich aus den B-Lymphozyten entwickeln. Es zirkuliert ständig eine Mischung zahlreicher verschiedener Plasmazellen im Blut, von denen jede ihren eigenen Antikörper herstellt, der jeweils ausschließlich auf die Erkennung eines bestimmten Merkmals spezialisiert ist. Die Antikörpermoleküle werden ins Blut abgegeben. Wir haben etwa 100 000 Gene auf 46 Chromosomen, jedes zuständig für eine Art Protein. Obwohl diese Zahl nicht gering sind, so wären wir dennoch völlig außerstande die ungeheure Anzahl an Antikörper zu produzieren, die nötig wäre, um alle vorhandenen oder zukünftig in uns eindringenden Antigene zu erkennen.

Wie kann unser Immunsystem dennoch adäquat auf die Vielzahl der Eindringlinge reagieren? Die Lösung ist, daß das Immunsystem zufällig Antikörper produziert und mit diesem nichtdeterministischen Prozeß eine viel größere Spannbreite an möglichen Antigenen abdeckt. Sobald einer der zufällig erzeugten Antikörper auf ein eingedrungenes Antigen paßt, geht die Nachricht davon zu den Plasmazellen, und die Plasmazelle, die die Konstruktionsanleitung für den erfolgreichen Antikörper in ihrem Erbgut hat, beginnt sich zu teilen. Jede der gebildeten Tochterzellen produziert den entsprechenden Antikörper in großen Mengen.

Immer wenn es nicht nur eine Möglichkeit des Weitergehens gibt, haben wir ein nichtdeterministisches System. Jede Mutation ist nichtdeterministisch. Je größer die Vielzahl an Entwicklungsmöglichkeiten ist, d.h. je höher der inhärente Nichtdeterminismus ist, desto leichter kann ein System auf veränderte Bedingungen reagieren. Man kann wohl sagen, daß der Nichtdeterminismus eine der Eigenschaften des Lebens ist, die es am besten charakterisieren.

Auch viele mathematische Verfahren sind nichtdeterministisch, d.h. sie schreiben nicht jeden Schritt vor, sondern überlassen es dem Ausprobieren, dem Raten, der Intuition und Erfahrung des Anwenders, den richtigen Schritt selber zu finden. So ist der Schulalgorithmus für das Dividieren nichtdeterministisch, weil man raten muß, wie oft der Divisor in den jetzt vorliegenden erweiterten Rest eingeht. Auch das klassische Verfahren für die Integration, ist ein nichtdeterministisches Verfahren, weil man die geeignete Substitution raten muß. (Daß man diese Verfahren deterministisch machen kann, indem man angibt, wie der richtige Schritt gefunden werden kann, ändert daran nichts. Eine der großen Fragen der Informatik ist, ob es eine effizientere Weise gibt ist, ein nichtdeterministisches Verfahren deterministisch zu machen, als einfach alle möglichen Schrittfolgen durchzuprobieren).

Alle mathematischen Kalküle sind nichtdeterministische Verfahren. Der Kalkül des mathematischen Beweisens ist nichtdeterministisch: man hat einen Satz von Axiomen und von Regeln, die aus wahren Aussagen, wieder wahre Aussagen ableiten. Ein (formaler) Beweis für eine Aussage ist eine Folge von Aussagen, die durch die Regeln aus früheren Aussagen oder Axiomen abgeleitet werden, und an deren Ende die zu beweisende Aussage steht. Wie man zu dem Beweis kommt ist gänzlich nichtdeterministisch, da es viele Arten gibt, neue Aussagen abzuleiten und sicher auch viele Beweise für ein und dieselbe Aussage. Das Durchsuchen aller möglichen Beweisketten, bis zum Finden einer, die in einer gegebenen Aussage endet, ist ein unzumutbar aufwendiges Verfahren: es ist nicht durchführbar, weil die Zeit rasch die Lebensdauer unseres Alls überschreitet. Dennoch werden Beweise geführt, aber eben nichtdeterministisch.

Kalküle der Informatik sind Termersetzungssysteme, wie unsere formalen Grammatiken oder die Lindenmayersysteme, mit denen wir unter anderem Fraktale erzeugen können, alles nichtdeterministische Verfahren.

Einen Beweis auf seine Richtigkeit zu überprüfen ist ein deterministisches und viel einfacheres Problem als einen richtigen Beweis zu finden. Nichtdeterminismus bedeutet so etwas wie: den richtigen Weg raten und ihn anschließend zu prüfen (verifizieren): Den richtigen Beweis, den richtigen Antikörper, die erfolgreiche Mutation, die richtige Substitution beim Integrieren.

Wir glauben, wir haben genügend Gründe uns mit nichtdeterministischen Automatenmodellen zu beschäftigen, mit denen wir die zahlreichen nichtdeterministischen Phänomene besser beschreiben

können, auch wenn noch offen ist, wie wir diese in deterministische Programme wandeln können, die in zumutbarer Zeit rechnen.

1.5.1 Nichtdeterministischer finiter Automat

Bisher haben wir uns vor allem mit deterministischen Automaten befaßt. Es ist oft einfacher, zu einer Aufgabe zunächst nichtdeterministische Automaten zu konstruieren und diese dann in deterministische umzuwandeln. Der Entwurf eines Automaten muß erdacht und von Hand gemacht werden, das Wandeln in einen deterministischen aber geht algorithmisch, als kann uns ein Rechner abnehmen.

Zur Erinnerung: ein nichtdeterministischer finiter Automat (NFA) ist ein Automat, dessen Transitionrelation nicht rechtseindeutig ist (vgl. Seite 8). In einem Zustand kann man mit demselben Zeichen unter Umständen in mehrere Folgezustände gelangen. Für den Graphen heißt das, daß es Knoten gibt, von denen mehrere, mit demselben Zeichen belegte Pfeile abgehen.

Satz 1.5.1 (NFA = DFA)

Zu jedem nichtdeterministischen finiten Automaten gibt es einen deterministischen, der dieselbe Sprache erkennt.

Beweis: Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein NFA. Wir definieren den DFA $M' = (\mathcal{P}(Q), \Sigma, \delta', \{q_0\}, F')$ ($\mathcal{P}(Q)$ = Potenzmenge von Q) mit

$$\begin{aligned} \delta' &= \{(P, a, P') \in \mathcal{P}(Q) \times \Sigma \times \mathcal{P}(Q) \mid P' = \{q \mid \exists p \in P : (p, a, q) \in \delta\}, \\ &F' = \{P \in \mathcal{P}(Q) \mid P \cap F \neq \emptyset\}. \end{aligned}$$

δ' ist eine rechtseindeutige Relation, da P' durch P und a eindeutig definiert ist. ■

Diesen so gewonnenen Automaten M' nennen wir auch den Potenzautomaten von M , da seine Zustandsmenge gerade die Potenzmenge der alten Zustandsmenge ist. Intuitiv kann man sich vorstellen, daß M' die verschiedenen Möglichkeiten von M parallel mithält: ein Zustand von M' umfaßt alle Zustände, in denen M nach der bisherigen Geschichte (Eingabefolge) jetzt sein könnte.

1.5.2 Finiter ε -Automat

Aus Gründen der Bequemlichkeit wollen wir finite Automaten betrachten, die auch spontane, autonome Zustandsübergänge haben können, die von keinem Inputsymbol ausgelöst worden sind. Das entspricht im Digraphen des Automaten Pfeilen, die keine Beschriftung haben. Da sie aber eine neue Art von Pfeilen darstellen, sollen sie doch einen Namen (eine Bewertung) bekommen: wir belegen diese Pfeile mit dem Symbol ε und reden daher auch von ε -Transitionen oder ε -Zustandübergängen.

Wir sollten uns freilich klarmachen, daß dies eine sehr gefährliche Art von Doppelbedeutung des Zeichens ε mit sich führt: einmal ist ε ein Metazeichen für das leere Wort, das andere Mal ist ε ein Zeichen (ein Label) einer Kante im Digraphen, beides Mal aber kein Zeichen unseres Inputalphabets. Die gewollte und nahegelegte Verwechslung dieser beiden Bedeutungen ist bequem, aber nicht pädagogisch: führt die Zeichenfolge $a_1 \cdots a_n$, wobei einige der a_i das Zeichen ε sein können, von einem Startzustand in einen Endzustand, so wird nicht das Wort $a_1 \cdots a_n$ erkannt, sondern das Wort, daß wir erhalten, wenn wir das Zeichen ε umdeuten in das Metazeichen für das leere Wort. Dann blendet die Konkatenation das Zeichen ε aus und übrig bleibt die Folge der nichtautonomen Transitionen mit ihren Labeln.

Definition 1.5.2 (ε -FA)

Seien Q und Σ Alphabete, $q_0 \in Q$, $F \subseteq Q$, und $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$. Dann heißt $M = (Q, \Sigma, \delta, q_0, F)$ finiter ε -Automat (ε -NFA). Sei $\delta_\varepsilon = \{(p, q) \mid (p, \varepsilon, q) \in \delta\}$ und die reflexive, transitive Hülle δ_ε^* :

$$(p, p) \in \delta_\varepsilon^* \text{ für alle } p \in Q$$

$$((p, q) \in \delta_\varepsilon^* \wedge (q, r) \in \delta_\varepsilon) \implies (p, r) \in \delta_\varepsilon^*,$$

d.h. $\delta_\varepsilon^* = \{(p, q) \mid \text{es existiert ein reiner } \varepsilon\text{-Pfad von } p \text{ nach } q\}$. Dann definieren wir δ^* vermöge

$$(p, q) \in \delta_\varepsilon^* \implies (p, \varepsilon, q) \in \delta^*$$

$$((p, w, r) \in \delta^* \wedge (r, s) \in \delta_\varepsilon^* \wedge (s, a, t) \in \delta \wedge (t, q) \in \delta_\varepsilon^*) \implies (p, wa, q) \in \delta^*,$$

und keine anderen Tripel sind aus δ^* .

Satz 1.5.3

Zu jedem ε -NFA existiert ein äquivalenter NFA.

Beweis: Sei $M_\varepsilon = (Q, \Sigma, \delta, q_0, F)$ ein ε -NFA. Wir definieren δ' vermöge: $(p, a, q) \in \delta'$ gdw. $\exists r, s \in Q : (p, r) \in \delta_\varepsilon^* \wedge (r, a, s) \in \delta \wedge (s, q) \in \delta_\varepsilon^*$. Dann ist $M = (Q, \Sigma, \delta', q_0, F')$ und $F' = \{p \mid \exists q \in F : (p, q) \in \delta_\varepsilon^*\}$ ein zu M_ε äquivalenter NFA ohne ε -Transitionen. ■

Um einen ε -NFA $M_\varepsilon = (Q, \Sigma, \delta, q_0, F)$ in einen DFA M zu verwandeln, benutzen wir die folgende Potenzautomatenkonstruktion. $M := (\mathcal{P}(Q), \Sigma, \delta', Q_0, F')$, wobei $Q_0 := \{p \in Q \mid (q_0, p) \in \delta_\varepsilon^*\}$, $F := \{P \mid P \cap F \neq \emptyset\}$, und (P, a, P') gdw. $P' = \{q \in Q \mid \exists p \in P \exists r \in Q : (p, a, r) \in \delta \wedge (r, q) \in \delta_\varepsilon^*\}$.

Beispiel 1.5.4

Die linke Tabelle zeigt einen ε -NFA und die rechte sein deterministisches Analogon, wieder eine Potenzautomatenkonstruktion:

	δ	a	b	ε
S	0	-	2	1
	1	0,4	-	2,3
	2	-	4	-
	3	4	-	-
F	4	-	-	3

	δ	a	b
SF	0123	01234	234
F	01234	01234	234
F	234	34	34
F	34	34	-
	-	-	-

Beispiel 1.5.5

Seien die Sprachen $A = \{w \in \{a\}^* \mid |w| \text{ teilbar durch } 2\}$ und $B = \{w \in \{a\}^* \mid |w| \text{ teilbar durch } 3\}$ gegeben. Die zugehörigen DFA's sind durch die ersten beiden Tabellen definiert. Beide durch ε -Pfeile verbunden ergeben einen ε -NFA, der $A \cup B$ erkennt. Machen wir ihn deterministisch, so erhalten wir die vierte Tabelle.

	δ_A	a
SF	1	2
	2	1

	δ_B	a
SF	3	4
	4	5
	5	3

	$\delta_{A \cup B}$	ε	a
S	0	1, 3	-
F	1	-	2
	2	-	1
F	3	-	4
	4	-	5
	5	-	3

	$\delta'_{A \cup B}$	a
SF	013	24
	24	15
F	15	23
F	23	14
F	14	25
	25	13
F	13	24

Beispiel 1.5.6

Gegeben seien die Sprachen $A = \{ab, aba\}^*$ und $B = \{w \in \{a, b\}^* \mid w \text{ enthält } aa\}$. Die zugehörigen NFA's sind durch die beiden ersten Tabellen gegeben. Beide zusammengelegt ergeben einen ε -NFA mit Startzustand 0, der $A \cup B$ erkennt. Machen wir ihn deterministisch, so erhalten wir die dritte

Tabelle.

δ_A	a	b
SF	1	2
	2	1, 3
	3	1

δ_B	a	b
S	p, q	p
	q	r
F	r	r

$\delta_{A \cup B}$	a	b
SF	$01p$	$2pq$
	$2pq$	pqr
	p	pq
	pq	pqr
F	pqr	pqr
F	pr	pqr
F	$13p$	$12pq$
F	$12pq$	$2pqr$
F	$2pqr$	pqr
F	$13pr$	$12pqr$
F	$12pqr$	$2pqr$

1.6 Reguläre Ausdrücke

Wir definieren hier einen weiteren Kalkül, der ebenfalls gerade die regulären (von finiten Automaten erkannten) Sprachen beschreibt. Es ist ein Termkalkül, d.h. ein Kalkül in dem gewisse erlaubte Zeichenreihen - Terme - mit einer Bedeutung unterlegt werden. Ein Term bekommt eine Bedeutung, wenn man die Zeichen interpretiert, die einen als Elemente einer vorgegebenen Menge, andere als Operationen auf dieser Menge. Eine Menge, auf der Operationen definiert sind, nennt man eine algebraische Struktur. Wir können verschiedene Terme gleichsetzen, d.h. Termgleichungen (Axiome) formulieren. Dann ist eine algebraische Struktur ein Modell, wenn die Interpretation den Termen einer Termgleichung gleiche Elemente der Modellmenge zuordnet, wenn also die interpretierte Termgleichung in der algebraischen Struktur zu einer wahren Gleichung wird.

Wir kennen schon einige Termkalküle, etwa die arithmetischen Ausdrücke mit den Zeichen $0, \dots, 9, +, -, \cdot$ und $/$, mit denen wir Gesetze des Körpers der rationalen oder reellen Zahlen modellieren oder die Booleschen Formeln mit den Zeichen $0, 1, \wedge, \vee, \neg$, die Boolesche Algebren modellieren.

Die Terme des unten definierten Kalküls nennen wir reguläre Ausdrücke und das Modell, das wir betrachten ist $\{\mathcal{P}(\Sigma^*), \cdot, \cup, *\}$, also die Menge der Sprachen über Σ mit den Mengenoperationen Konkatenation, Vereinigung und Sternbildung. Jeder Term steht für eine Sprache über Σ und - wie wir sehen werden - für eine reguläre.

Definition 1.6.1 (regulärer Ausdruck (RA) über Σ)

Ist R ein regulärer Ausdruck, so bezeichnen wir mit $L(R)$ die von ihm repräsentierte Sprache. Wir definieren den regulären Ausdruck induktiv über den Aufbau:

1. (\emptyset) ist ein regulärer Ausdruck und $L((\emptyset)) = \emptyset$ seine Sprache.
2. Ist $a \in \Sigma$, so ist (a) ein regulärer Ausdruck und $L((a)) = \{a\}$.
3. Sind R und S reguläre Ausdrücke, so auch

$(R \cdot S)$	wo	$L((R \cdot S)) = L(R) \cdot L(S)$,
$(R + S)$	wo	$L((R + S)) = L(R) \cup L(S)$ und
(R^*)	wo	$L((R^*)) = (L(R))^*$.
4. Keine anderen Ausdrücke sind regulär.

Der leichten Notation halber schreiben wir abkürzend

$$\begin{aligned}
 (\varepsilon) & \text{ für } ((\emptyset)^*) \quad \text{und} \\
 (R^+) & \text{ für } (R \cdot (R)^*).
 \end{aligned}$$

Um bei der Notation von regulären Ausdrücken Klammern zu sparen, bedienen wir uns Prioritätsregeln

- wie wir das auch bei den arithmetischen Ausdrücken kennen: es steht $*$ vor \cdot und \cdot vor $+$. Außerdem lassen wir den Punkt für die Konkatenation weg. Dann können wir $(1^*01)^*00(0+1)^*$ schreiben statt

$$\langle \{ [[[(1)^* \cdot ((0) \cdot (1))]^* \cdot ((0) \cdot (0)) \cdot [((0) + (1))^*]] \} \rangle .$$

Definition 1.6.2 ($R \equiv S$)

Wir nennen zwei reguläre Ausdrücke R und S äquivalent ($R \equiv S$), wenn $L(R) = L(S)$.

Beispiel 1.6.3

Der reguläre Ausdruck $(ab + aba)^*$ steht für die Sprache $\{ab, aba\}^*$

Der reguläre Ausdruck $(0 + 1)^*111(0 + 1)^*$ steht für die Sprache $\{w \in \{0, 1\} \mid w \text{ enthält } 111\}$

Die beiden regulären Ausdrücke $(1 + 01)^*00(0 + 1)^*$ und $(0 + 1)^*00(0 + 1)^*$ sind äquivalent!!

Ein regulärer Ausdruck für die Sprache $\{\varepsilon\}$ ist $((\emptyset)^*)$ oder in unserer Kurzschreibweise \emptyset^* .

Wir wollen natürlich sehen, ob die regulären Ausdrücke tatsächlich dieselben Sprachen beschreiben wie die finiten Automaten. Das kann man zeigen, wenn man einen Weg findet die beiden Kalküle ineinander zu überführen. Dieser Weg hätte zudem den Vorteil, daß man ein Verfahren hat, den Kalkül zu wechseln, wenn es sich aus praktischen Gründen als geboten erweist. Also zeigen wir:

Satz 1.6.4 (Kleenesche Synthese))

Zu jedem regulären Ausdruck R gibt es einen (nichtdeterministischen) finiten Automaten, der $L(R)$ erkennt (Wir können zu R einen äquivalenten NFA synthetisieren).

Beweis: Wir beweisen dies durch Induktion über den Aufbau des regulären Ausdrucks. Zu jedem regulären Ausdruck konstruieren wir einen finiten Automaten, der genau einen Startzustand, genau einen Endzustand und genausoviele Zustände hat wie der (korrekt geklammerte) reguläre Ausdruck Klammern:

- NFA für (\emptyset) : Der NFA hat einen Start- und einen Endzustand, aber keine Transition.
- NFA für (a) : Der NFA hat einen Start- und einen Endzustand und dazwischen eine a -Transition.
- NFA für $(R \cdot S)$: Ist M_R der NFA für R und M_S der für S , so definieren wir einen neuen Startzustand und einen neuen Endzustand und verbinden - jeweils durch eine ε -Transition - den neuen Startzustand mit dem von M_R , den Endzustand von M_R mit dem Startzustand von M_S und den Endzustand von M_S mit dem neuen Endzustand.
- NFA für $(R + S)$: Ist M_R der NFA für R und M_S der für S , so definieren wir einen neuen Startzustand und einen neuen Endzustand und verbinden - jeweils eine ε -Transition - den neuen Startzustand mit den Startzuständen von M_R und M_S und die Endzustände von M_R und M_S mit dem neuen Endzustand.
- NFA für (R^*) : Sei M_R der NFA für R , so definieren wir einen neuen Startzustand und einen neuen Endzustand und verbinden - jeweils durch eine ε -Transition - den neuen Startzustand mit dem von M_R und dem neuen Endzustand sowie den Endzustand von M_R mit dem neuen Startzustand.

■

Satz 1.6.5 (Kleenesche Analyse)

Zu jedem ε -NFA M existiert ein regulärer Ausdruck R mit $L(R) = L(M)$.

Beweis: Gegeben sei der ε -NFA $M = (\{1, \dots, q\}, \Sigma, \delta, 1, \{q\})$ (Zu jedem ε -NFA kann man einen äquivalenten dieser Form finden). Für alle $1 \leq i, j \leq q$ und $0 \leq k \leq q$ definieren wir die folgenden Sprachen:

$$L_{i,j}^k := \{w \in \Sigma^* \mid \text{es existiert ein } w\text{-Pfad von } i \text{ nach } j, \text{ der keine Knoten größer als } k \text{ durchläuft}\}.$$

Das sind also gerade die Worte, die vom Knoten i zum Knoten j führen und dabei höchstens die Knoten 1 bis k durchlaufen. Wenn ein Wort von i nach j durch den Zustand k läuft, läßt es sich zerlegen in ein Anfangsstück, das von i nach k führt, ohne k zu durchlaufen, mehrere k -freie Stücke von k nach k - jenachdem, wie oft es den Zustand k durchläuft - und in ein wiederum k -freies Endstück von k nach j . D.h. es gilt:

$$L_{i,j}^k = L_{i,j}^{k-1} \cup L_{i,k}^{k-1} (L_{k,k}^{k-1})^* L_{k,j}^{k-1}$$

Durch Induktion über k zeigen wir jetzt, daß es für jede dieser Sprachen $L_{i,j}^k$ einen regulären Ausdruck $R_{i,j}^k$ gibt, der sie beschreibt.

$$k = 0 : \quad \begin{aligned} L_{ii}^0 &= \{a \in \Sigma \mid (i, a, i) \in \delta\} \cup \{\varepsilon\} \\ L_{ij}^0 &= \{a \in \Sigma \cup \{\varepsilon\} \mid (i, a, j) \in \delta\} \quad (\text{für } i \neq j) \end{aligned}$$

Dann ist

$$\begin{aligned} R_{ij}^0 &= \emptyset && \text{falls } L_{ij}^0 = \emptyset \\ R_{ij}^0 &= a_1 + \dots + a_n && \text{falls } L_{ij}^0 = \{a_1, \dots, a_n\} \\ R_{ij}^0 &= \varepsilon && \text{falls } L_{ij}^0 = \{\varepsilon\} \\ R_{ij}^0 &= a_1 + \dots + a_n + \varepsilon && \text{falls } L_{ij}^0 = \{a_1, \dots, a_n\} \cup \{\varepsilon\}. \end{aligned}$$

Die Behauptung sei richtig für $k - 1$. Dann ist

$$R_{ij}^k = R_{i,j}^{k-1} + R_{i,k}^{k-1} \cdot (R_{k,k}^{k-1})^* \cdot R_{k,j}^{k-1}$$

der reguläre Ausdruck für $L_{i,j}^k$. Da $L(M) = L_{1,q}^q$ folgt der Satz. ■

Beispiel 1.6.6

Zeige $((10)^*0)^* \equiv \varepsilon + (0 + 10)^*0$.

Wir setzen $a = 10$ und $b = 0$ und abstrahieren das Problem zu $(a^*b)^* = \varepsilon + (a + b)^*b$

1. $L((a^*b)^*) = L(\varepsilon) \cup L((a^*b)^*a^*b) \subseteq L(\varepsilon) \cup L((a + b)^*b)$, da $L((a^*b)^*a^*) \subseteq L((a + b)^*)$.
2. $L(\varepsilon) \subseteq L((a^*b)^*)$ und $L((a + b)^*b) \subseteq L((a^*b)^*)$ (ein Wort $w \in L((a + b)^*b)$ kann zerlegt werden in Worte $u_1 \cdots u_k$ mit $u_i \in L(a^*b)$).

Beispiel 1.6.7

Gegeben sei der DFA

	0	1
SF	1	2
	3	-
	1	2

Der zugehörige RA ist $(0 + 10(10)^*0)^* = ((10)^*0)^* = \varepsilon + (0 + 10)^*0$.

1.7 Abschlußeigenschaften regulärer Sprachen

1.7.1 Produktautomat

Notation 1.7.1

Sei A eine Menge auf der eine Operation op definiert ist. Wir sagen: eine Teilmenge $B \subseteq A$ ist abgeschlossen unter der Operation op , wenn das Ergebnis der Anwendung von op auf Elemente von B wieder in B liegt. Z.B. ist die Menge \mathbb{N} der natürlichen Zahlen abgeschlossen unter der Addition und der Multiplikation, nicht aber unter Subtraktion und Division, während die Menge \mathbb{Q} der rationalen Zahlen abgeschlossen ist unter allen diesen vier Operationen, nicht aber unter der Quadratwurzel.

Nach dem Abschnitt über die regulären Ausdrücke wissen wir, daß die Menge REG der regulären Sprachen abgeschlossen ist unter Vereinigung, Konkatenation und Kleeneschem Abschluß (Sternbildung, Iteration). Wir wollen den Abschluß von REG unter weiteren Mengenoperationen untersuchen.

Satz 1.7.2

REG ist abgeschlossen unter Komplementbildung.

Beweis: Ist L regulär und M ein vollständiger DFA für L , so erhalten wir aus M einen DFA \overline{M} für das Komplement $\overline{L} = \Sigma^* - L$ von L , indem wir die Endzustände von M zu nichtakzeptierenden Zuständen erklären, und umgekehrt die nichtakzeptierenden zu Endzuständen. ■

Definition 1.7.3 (Produktautomat)

Seien $M_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$ ($i = 1, 2$) zwei NFA's. Den NFA $M_1 \times M_2 = (Q_1 \times Q_2, \Sigma, \delta, (q_1, q_2), F)$ mit $((p, q), a, (p', q')) \in \delta \iff (p, a, p') \in \delta_1 \wedge (q, a, q') \in \delta_2$ nennen wir einen *Produktautomaten* von M_1 und M_2 . Einem w -Pfad in $M_1 \times M_2$ entsprechen je ein w -Pfad in M_1 und in M_2 , die gleichsam parallel durchlaufen werden. Es gibt einen w -Pfad von (p, q) nach (p', q') in $M_1 \times M_2$ gdw. es einen in M_1 einen w -Pfad von p nach p' und in M_2 einen w -Pfad von q nach q' gibt.

Da wir über die Endzustandsmenge F nichts gesagt haben, kann es verschiedene Produktautomaten von M_1 und M_2 geben. Legen wir F fest auf $F = F_1 \times F_2$, so ist $L(M_1 \times M_2) = L(M_1) \cap L(M_2)$. Ist $F = F_1 \times Q_2 \cup Q_1 \times F_2$ und sind die NFAs M_1 und M_2 vollständig, so ist $L(M_1 \times M_2) = L(M_1) \cup L(M_2)$.

Hier fällt nochmal der Satz ab, den wir schon kennen:

Satz 1.7.4

REG ist abgeschlossen unter Schnitt und Vereinigung

Den Abschluß unter Vereinigung kennen wir von den regulären Ausdrücken und den unter Schnitt als Konsequenz aus den de Morgan'schen Regeln: $L \cap L' = \overline{\overline{L} \cup \overline{L'}}$. Der Produktautomat liefert aber eine direkte Konstruktionsanweisung und zwingt uns nicht zu der teureren Potenzautomatenkonstruktion, die für die Komplementbildung von NFA's nötig ist. Zudem bringt uns diese Konstruktion auch noch später Nutzen.

Anmerkung 1.7.5

Die symmetrische Differenz $A \triangle B$ zweier Mengen A und B sei definiert vermöge $A \triangle B := (A - B) \cup (B - A)$. Seien M_1 und M_2 zwei vollständige DFAs und sei $F = F_1 \times (Q_2 - F_2) \cup (Q_1 - F_1) \times Q_2$. Dann ist $L(M_1 \times M_2) = L(M_1) \triangle L(M_2)$.

Definition 1.7.6 (Spiegelung)

Sei $w = a_1 \cdots a_n$ ein Wort über Σ ($a_i \in \Sigma, i = 1 \dots, n$). Dann nennen wir $\overleftarrow{w} = a_n \cdots a_1$ die Spiegelung von w . Entsprechend ist $\overleftarrow{L} = \{\overleftarrow{w} \mid w \in L\}$ die Spiegelung von L .

Satz 1.7.7

REG ist abgeschlossen unter Spiegelung.

Aufgabe 1.7.8

Beweisen Sie den Satz 1.7.7.

1.7.2 Homomorphismus, GSM-Abbildung

Betrachten wir ein kombinatorisches Schaltwerk S mit k Eingängen und l Ausgängen. S realisiert eine Funktion $f : \{0, 1\}^k \longrightarrow \{0, 1\}^l$ oder $f : \Sigma \longrightarrow \Delta$, wenn wir $\{0, 1\}^k = \Sigma$ und $\{0, 1\}^l = \Delta$ setzen.

Erlauben wir dem Schaltwerk S als Bild des Inputsymbols nicht nur ein Symbol aus Δ , sondern ein ganzes Wort aus Δ^* auszugeben, so sprechen wir von einem generalisierten kombinatorischen Schaltwerk. Wir können die Funktion $f : \Sigma \rightarrow \Delta^*$ wie folgt auf Σ^* fortsetzen: Wir geben S eine Folge $a_1 \cdots a_n$ von Inputsymbolen ein und erhalten eine Folge $v_1 \cdots v_n$ von Outputworten, d.h. S realisiert eine Funktion $f^* : \Sigma^* \rightarrow \Delta^*$ mit $f^*(a_1 \cdots a_n) = v_1 \cdots v_n$.

Eine solche Abbildung $h : \Sigma^* \rightarrow \Delta^*$, bei der man das Bild $h(w)$ eines Wortes w erhält, indem man h auf jedes einzelne Symbol anwendet, nennt man einen Homomorphismus auf Σ^* . Solche Worthomomorphismen sind besonders einfache und bequeme Abbildungen. Kennt man ihre Bilder auf den Symbolen (genauer auf den Worten der Länge 1), so kennt man sie ganz

Definition 1.7.9 (Homomorphismus)

Eine totale Abbildung $h : \Sigma^* \rightarrow \Delta^*$ heißt (Wort-)Homomorphismus gdw.

$$\forall x, y \in \Sigma^* : h(x \cdot y) = h(x) \cdot h(y)$$

Anmerkung 1.7.10

Zu jeder Struktur - nicht nur für das freie Monoid (Σ^*, \cdot) - gibt es einen Homomorphismus-Begriff. Eine Abbildung heißt Homomorphismus, wenn sie verträglich ist mit den Operationen der Struktur, d.h. es spielt keine Rolle, ob man erst die Operation ausführt und dann Homomorphismus anwendet oder umgekehrt. Ein Beispiel ist der Logarithmus $\log : \mathbb{R}_+ \rightarrow \mathbb{R}$, wenn wir ihn als eine Abbildung zwischen der multiplikativen Gruppe (\mathbb{R}_+, \cdot) und der additiven Gruppe $(\mathbb{R}, +)$ auffassen. Ein anderes Beispiel sind die linearen Abbildungen auf einem Vektorraum X mit der Skalarmultiplikation und der Vektoraddition.

Lemma 1.7.11

Ist $w = a_1 \cdots a_n$ ($a_i \in \Sigma^1$) ein Wort aus Σ^* und $h : \Sigma^* \rightarrow \Sigma^*$ ein Homomorphismus auf Σ^* , dann gilt:

$$h(w) = h(a_1) \cdots h(a_n).$$

Aufgabe 1.7.12

Was ist $h(\varepsilon)$ (h beliebiger Homomorphismus)?

Lemma 1.7.13

Ein Homomorphismus auf Σ^* ist bereits durch seine Bilder auf Σ^1 festgelegt, d.h. jede totale Abbildung $h : \Sigma^1 \rightarrow \Sigma^*$ läßt sich auf genau eine Weise zu einem Homomorphismus $h^* : \Sigma^* \rightarrow \Sigma^*$ fortsetzen.

Anmerkung 1.7.14

Vorsicht: Aus $B = h(A)$ folgt *nicht* notwendig $A = h^{-1}(B)$.

Satz 1.7.15

Ist L regulär und $h : \Sigma^* \rightarrow \Sigma^*$ ein Homomorphismus auf Σ^* , so sind auch $h(L)$ und $h^{-1}(L)$ regulär.

Beweis: Sei M ein DFA für L .

1. $h(L)$: Wir ersetzen jeden Befehl (Transition) (p, a, q) zunächst durch $(p, h(a), q)$. Da ein Pfeil aber nicht mit einem Wort belegt werden darf, ersetzen wir die Transition $(p, h(a), q)$ durch eine Folge von Transitionen, und zwar durch $(p, b_1, q_{b_1})(q_{b_1}, b_2, q_{b_2}) \cdots (q_{b_{k-1}}, b_k, q)$, wenn $h(a) = b_1 \cdots b_k$ ($b_1, \dots, b_k \in \Sigma$), wobei q_{b_1}, \dots, q_{b_k} neue Zustände sind.
2. $h^{-1}(L)$: Wir erstellen eine Kopie des Graphen von M und entfernen alle Kanten. Für jedes $a \in \Sigma$ definieren wir eine Transition (p, a, q) in der Kopie, wenn $h(a) = w$ und in M ein w -Pfad von p nach q führt.

■

Betrachten wir jetzt sequentielle Schaltwerke. Sie realisieren nicht mehr Funktionen $f : \{0, 1\}^k \rightarrow \{0, 1\}^l$, die ein Symbol immer in dasselbe Outputsymbol abbilden, sondern DFA's mit Output, die man in Analogie sequentielle Maschinen nennt. Erlauben wir wieder, daß der Output aus einem Wort, statt einem Symbol besteht, so bekommt man eine generalisierte sequentielle Maschine (GSM).

Ein Homomorphismus h bildet ein Symbol (Signal) a immer auf denselben Wert $h(a)$ ab, egal wo im Inputstrom dieses Symbol auftaucht. Eine GSM-Abbildung g hingegen hat ein (endliches) Gedächtnis (realisiert durch die Zustände). Sie bildet ein Symbol a in einen Wert ab, der von der Vergangenheit des Inputstroms abhängt. g kann bei Input a also (endlich viele) verschiedene Werte liefern. Dennoch kann man bei einer GSM-Abbildung immer noch ein einfaches (endliches) Gesetz erkennen, wie das Bild $g(w)$ eines Wortes w symbolweise entsteht.

Definition 1.7.16 (generalisierte sequentielle Maschine, GSM)

Eine generalisierte sequentielle Maschine, kurz GSM, ist ein DFA, der pro Transition ein Wort ausgeben kann: Seien Q, Σ, Δ endliche Mengen, $q_0 \in Q$, $F \subseteq Q$ und $\gamma : Q \times \Sigma \rightarrow \Delta^* \times Q$. Dann ist $M = (Q, \Sigma, \Delta, \gamma, q_0)$ eine GSM. Die von M berechnete GSM-Abbildung $g : \Sigma^* \rightarrow \Delta^*$ ist definiert vermöge: $g(w) = v \iff (q_0, w, v, q) \in \gamma^*$, für ein q , wobei γ^* wie folgt definiert ist :

1. $(q, \varepsilon, \varepsilon, q) \in \gamma^*$ für alle $q \in Q$.
2. $(p, u, v, r) \in \gamma^*$, $(r, a, w, q) \in \gamma \implies (p, ua, vw, q) \in \gamma^*$ ($p, r, q \in Q$; $u \in \Sigma^*$; $a \in \Sigma$; $v, w \in \Delta^*$).

Wir können uns eine GSM also wie einen DFA als Digraph vorstellen, dessen Kanten allerdings zweifach bewertet sind, einmal durch das Inputsymbol, das diese Transition auslöst, zum anderen durch das Outputwort, das der Automat bei dieser Transition ausgibt. Wir können uns die Pfeile des Graphen durch ein Paar (Inputsymbol, Outputwort) beschriftet denken.

Satz 1.7.17

Sei g eine GSM-Abbildung. Dann gilt: Ist L regulär, so auch $g(L)$ und $g^{-1}(L)$.

Beweis: Sei $M = (P, \Sigma, \delta, p_0, F)$ der DFA für L und $G = (Q, \Sigma, \Delta, \gamma, q_0)$ die GSM für g . Einen Pfad in einem FA-Graphen, der mit $w \in \Sigma^*$ belegt ist und vom Startzustand in einen Endzustand führt, wollen wir einen akzeptierenden w -Pfad nennen. Einen Pfad in einem GSM-Graphen, der bei Betrachten der ersten Komponenten mit u und der zweiten mit v belegt ist und in einem Startzustand beginnt, wollen wir einen (u, v) -Pfad nennen. Wir können auch von M und G einen Produktautomaten bilden, allerdings auf zweierlei Weise: einmal berücksichtigen wir die ersten Komponenten eines Pfades von G , das andere Mal die zweiten Komponenten. Wir können uns auch G als DFA vorstellen, wobei der Input einmal in der ersten Komponente besteht, das andere Mal in der zweiten (mit der kleinen Variante, daß hier ein Wort zu einer Transition führt statt eines Symbols). Wir definieren also den Produktautomaten $(M \times G)_1$ nach der ersten Komponente wie folgt: $[(p, q), a, v, (p', q')] \in \delta_{M \times G}^1$ gdw. $(p, a, p') \in \delta$ und $(q, a, v, q') \in \gamma$ und $F^1 = \{(p, q) \mid p \in F_M\}$. Den Produktautomaten $(M \times G)_2$ nach der zweiten Komponente definieren wir: $[(p, q), a, v, (p', q')] \in \delta_{M \times G}^2$ gdw. $(p, v, p') \in \delta^*$ und $(q, a, v, q') \in \gamma$ und $F^2 = \{(p, q) \mid p \in F_M\}$.

a) $g(L)$: das Wort w ist aus $g(L)$, wenn es ein Wort u gibt, mit $u \in L$ und $g(u) = w$, wenn es also einen akzeptierenden u -Pfad in M , und wenn es einen (u, w) -Pfad in G gibt. Dann muß es im Produktautomaten nach der ersten Komponente einen akzeptierenden (u, w) -Pfad geben. Wir bilden zunächst den Produktautomaten von M und G nach der ersten Komponente und ersetzen dann jede Kante im Graphen, wenn sie mit (a, v) beschriftet ist ($a \in \Sigma$, $v \in \Sigma^*$), durch einen v -Pfad (mit neuen Zuständen). Der endgültige Automat M_G wird dann wie folgt definiert: ist $(P, a, v, Q) \in \delta_{M \times G}^1$ und $v = a_1 \cdots a_k$, so seien die folgenden Befehle (P, a_1, P_{a_1}) , (P_{a_1}, a_2, P_{a_2}) , \dots , $(P_{a_{k-1}}, a_k, Q)$ aus δ_G .

b) $g^{-1}(L)$: das Wort u ist aus $g^{-1}(L)$, wenn es ein Wort $w \in L$ gibt mit $g(u) = w$, wenn es also einen akzeptierenden w -Pfad in M gibt und einen (u, w) -Pfad in G , d.h. wenn es in dem Produktautomaten nach der zweiten Komponente einen akzeptierenden (u, w) -Pfad gibt. Wir bilden wieder zunächst den

Produktautomaten von M und G nach der zweiten Komponente und streichen dann an jeder Kante die zweite Komponente, d.h. aus einem akzeptierenden (u, w) -Pfad, wird ein akzeptierender u -Pfad.

■

Ein zweite Verallgemeinerung des Homomorphismus ist die reguläre Substitution. Betrachten wir eine Abbildung $\sigma : \Sigma \rightarrow \text{REG}$. Jedem Symbol wird hier eine ganze reguläre Sprache zugeordnet. Auch diese Abbildung setzen wir fort auf Σ^* zu $\sigma^* : \Sigma^* \rightarrow \Sigma^*$ mit $\sigma(w) = \sigma(a_1) \cdots \sigma(a_n)$, wenn $w = a_1 \cdots a_n$. (Eine reguläre Substitution ist ein Homomorphismus von $(\Sigma^*, \cdot) \rightarrow (\text{REG}, \cdot)$.)

Definition 1.7.18 (reguläre Substitution)

Eine totale Abbildung $\sigma : \Sigma^* \rightarrow \text{REG}$ heißt *reguläre Substitution* gdw.

$$\forall x, y \in \Sigma^* : \sigma(x \cdot y) = \sigma(x) \cdot \sigma(y)$$

Lemma 1.7.19

Ist $w = a_1 \cdots a_n$ ($a_i \in \Sigma$), ein Wort aus Σ^* und $\sigma : \Sigma^* \rightarrow \text{REG}$ eine reguläre Substitution, dann gilt: $\sigma(w) = \sigma(a_1) \cdots \sigma(a_n)$.

Wie bei Homomorphismen gilt:

Lemma 1.7.20

Eine reguläre Substitution $\sigma : \Sigma^* \rightarrow \text{REG}$ ist bereits durch ihre Bilder auf Σ^1 festgelegt, d.h. jede totale Abbildung $\sigma : \Sigma^1 \rightarrow \text{REG}$ läßt sich auf genau eine Weise zu einer regulären Substitution $\sigma : \Sigma^* \rightarrow \text{REG}$ fortsetzen.

Satz 1.7.21

Ist L regulär und $\sigma : \Sigma^* \rightarrow \text{REG}$ eine reguläre Substitution, so ist auch $\sigma(L) := \bigcup \{ \sigma(w) \mid w \in L \}$ regulär.

Beweis: Sei M ein DFA für L und M_a ein NFA für die reguläre Sprache $\sigma(a)$ ($a \in \Sigma$). O.B.d.A habe M_a genau einen Endzustand. Ist (p, a, q) eine Transition von M , so löschen wir sie, führen einen ε -Pfeil von p in den Startzustand einer Kopie von M_a und einen ε -Pfeil vom Endzustand dieser Kopie M_a in q . ■

2 Symbolverzeichnis

Symbol	Seite	Erklärung
\mathbb{N}	1	Menge der natürlichen Zahlen
\mathbb{N}_+	1	Menge der natürlichen Zahlen ohne die Null
$f : A \longrightarrow B$	4	(partielle) Funktion aus A nach B
$f : A \rightarrow B$	4	totale Funktion von A nach B
$f : x \mapsto y$	4	f überführt Argument x in sein Bild y
K_q	9	Menge aller in den Zustand q führenden Worte
L_q	14	Menge aller vom Zustand q aus akzeptierten Worte
\notin	2	nicht Element aus
\in	2	Element aus
\cup	2	Vereinigung
\cap	2	Schnitt
$-$	2	Mengendifferenz
\subseteq	2	Teilmenge von
$\not\subseteq$	2	nicht Teilmenge von
\emptyset	2	leere Menge
\times	2	Kreuzprodukt
$\{A\}$	2	
$\bigcup \mathfrak{A}$	2	Vereinigung über ein Mengesystem
$\bigcap \mathfrak{A}$	2	Schnitt über ein Mengesystem
$\{A \mid \dots\}$	1	
$\mathcal{P}(A)$	2	Potenzmenge von A
A/R	5	Die Menge der R -Klassen in A
$\#L$	21	Menge der Wortlängen der Worte aus L
$R \equiv S$	26	Äquivalenz von regulären Ausdrücken
$L(R)$	25	Die vom RA R beschriebene Sprache
REG	15	Menge der regulären Sprachen
GSM	30	generalized sequential machine
\overleftarrow{w}	28	Spiegelung