Formale Grundlagen der Informatik I: Endliche Automaten und Formale Sprachen

Vorlesung Wintersemester 2006/07

M. Otto

Inhaltsverzeichnis

Einführung

Transitionssysteme

Idee: Man beschreibt ein System als eine Menge von Zuständen mit möglichen Übergängen zwischen Zuständen (Transitionen). Man erhält einen *Graph* mit beschrifteten Kanten: die Zustände bilden die Knoten des Graphen, die Transitionen die Kanten.

Anwendungen: vollständige Beschreibung von Maschinenzuständen auf der untersten Ebene; Analyse des Zustandsraums und der möglichen Berechnungsabläufe. [Allgemeiner gilt der gleiche Ansatz für Kontrollzustände oder abstrakte Zustände, bzw. für sogenannte Makro-Zustände, die Gruppen von Mikrozuständen eines Systems sinnvoll zusammenfassen.]

Ein Transitionssystem besteht aus einer (endlichen) Menge Q von Zuständen und gerichteten Kanten zwischen Paaren von Zuständen, die mit Zeichen aus einer endlichen Menge Σ gekennzeichnet sind. Zum Beispiel symbolisiert die Kante $q \xrightarrow{a} q'$, dass das System mit einer a-Transition vom Zustand q in den Zustand q' übergehen kann. Die Kantenbezeichnung a kann je nach Kontext z.B. eine äußere Aktion (oder Eingabe) bezeichnen, die den Übergang auslöst oder ermöglicht, oder auch eine innere Aktion des Systems beschreiben. Anhand des Transitionssystems kann man das Systemverhalten unter bestimmten Eingabesequenzen und mögliche Sequenzen von Zustandsänderungen analysieren. Wir behandeln informell zwei Beispiele.

Beispiel 0.0.1 Die Weckzeit-Kontrolle eines Weckers arbeitet mit einem Zeitformat (h, m) für $h \in \mathrm{H} = \{0, \dots, 23\}$ und $m \in \mathrm{M} = \{0, \dots, 59\}$. Äußere Aktionen werden durch Knöpfe *seth*, *setm*, $+$, $-$, *set*, *reset* kontrolliert. Als Zustände nehmen wir alle Kombinationen (h, m, q), wo $h \in \mathrm{H}$, $m \in \mathrm{M}$ und $q \in \{\mathrm{SETH}, \mathrm{SETM}, \mathrm{NIL}, \mathrm{ERROR}\}$ ist. Die Komponente q beschreibt den logischen Kontroll-Zustand:

SETH: in H-Setzen Modus;
SETM: in M-Setzen Modus;
NIL: nicht im Setzen Modus;
ERROR: Fehler.

Typische Transitionen wären z.B.:

$$
\begin{array}{rcll}
(h, m, \mathrm{NIL}) & \xrightarrow{\textit{seth}} & (h, m, \mathrm{SETH}) & \text{(Übergang in H-Setzen Modus)} \\
(h, m, \mathrm{SETH}) & \xrightarrow{\textit{set}} & (h, m, \mathrm{NIL}) & \text{(Beende H-Setzen Modus)} \\
(h, m, \mathrm{SETH}) & \xrightarrow{\textit{seth}} & (h, m, \mathrm{ERROR}) & \text{(da bereits in H-Setzen Modus)} \\
(h, m, \mathrm{NIL}) & \xrightarrow{+} & (h, m, \mathrm{ERROR}) & \text{(da nicht in Setzen Modus)} \\
(h, m, \mathrm{SETH}) & \xrightarrow{+} & ((h+1) \bmod 24, m, \mathrm{SETH}) & \text{(H vorstellen)} \\
(h, m, \mathrm{ERROR}) & \xrightarrow{\textit{reset}} & (0, 0, \mathrm{NIL}) & \text{(reset)}
\end{array}
$$

Diskussion: Man kann prinzipiell jedes reale hardware-System als endliches Transitionssystem modellieren; Grenzen der Anwendbarkeit/Nützlichkeit dieser (mikroskopisch vollständigen) Beschreibung ergeben sich vor allem aus der Größe der Zustandsmenge. Das folgende Beispiel ist überschaubarer.

Beispiel 0.0.2 Ein Mann muss mit seinen Schützlingen, einem Wolf, einem Hasen und einem Kohlkopf einen Fluss mit einem Boot überqueren, in dem er jeweils höchstens einen seiner Schützlinge mitnehmen kann. Beim mehrmaligem Übersetzen muss er beachten, dass weder der Hase mit dem Kohl, noch der Wolf mit dem Hase allein gelassen werden darf. Gesucht: ein optimaler Plan für die Bootsfahrten.

Als mögliche Zustände betrachten wir alle Verteilungen von $\{m, w, h, k\}$ auf rechtes/linkes Flussufer (zwischen den Bootsfahrten). Der Zustand $[m, w \| h, k]$ wäre also der (verbotene) Zustand, in dem der Hase rechts mit dem Kohl allein ist. Der Anfangszustand ist $[m, w, h, k \| \ ]$, der gewünschte Endzustand $[\ \| m, w, h, k]$. Zustandsänderungen geschehen durch Überfahrten von m alleine “$\Box$” oder mit einem Passagier, “w”, “h” oder “k”.

Man prüft nach, dass es 10 erlaubte Zustände gibt. Alle Übergänge zwischen diesen erlaubten Zuständen sind im Diagramm verzeichnet. Alle anderen Übergänge sind entweder unmöglich (weil der gewünschte Passagier nicht auf der Seite ist, auf der sich der der Mann befindet; z.B. steht ein Übergang mit “w” im Zustand $[m, h, k \| w]$ nicht zur Verfügung) oder führen aus dem Bereich der erlaubten Zustände heraus (z.B. würde ein Übergang mit “w” vom Anfangszustand $[m, w, h, k \| \ ]$ zum Zustand $[h, k \| m, w]$ führen, in dem der Hase mit dem Kohl alleine ist). Einen Plan für eine zulässige Bootstour ergibt sich aus den Markern längs eines Pfades, der vom Anfangs- zum Endzustand führt.

$$
\begin{tikzcd}
 & {[m,w,h,k\|\ ]} \arrow[d, bend left, "h"] & \\
 & {[w,k\|m,h]} \arrow[u, bend left, "h"] \arrow[d, bend left, "\Box"] & \\
 & {[m,w,k\|h]} \arrow[u, bend left, "\Box"] \arrow[dl, bend right, "w"'] \arrow[dr, bend left, "k"] & \\
{[k\|m,w,h]} \arrow[ur, bend right, "w"'] \arrow[d, bend left, "h"] & & {[w\|m,h,k]} \arrow[ul, bend left, "k"] \arrow[d, bend left, "h"] \\
{[m,h,k\|w]} \arrow[u, bend left, "h"] \arrow[dr, bend right, "k"'] & & {[m,w,h\|k]} \arrow[u, bend left, "h"] \arrow[dl, bend left, "w"] \\
 & {[h\|m,w,k]} \arrow[ul, bend right, "k"'] \arrow[ur, bend left, "w"] \arrow[d, bend left, "\Box"] & \\
 & {[m,h\|w,k]} \arrow[u, bend left, "\Box"] \arrow[d, bend left, "h"] & \\
 & {[\ \|m,w,h,k]} \arrow[u, bend left, "h"] &
\end{tikzcd}
$$

Es gibt also genau zwei optimale Strategien, beide benötigen 7 Bootsfahrten (zwei davon ohne Passagier).

Formaler könnte man die Zustände im obigen Beispiel durch Teilmengen der Menge $M = \{m, w, h, k\}$ darstellen, etwa durch Angabe der ‘rechten Seiten’ sodass z.B. der Startzustand durch die leere Menge $\emptyset$ und der Zielzustand durch die Menge M selbst beschrieben wird. Die erlaubten Zustände bilden dann eine Teilmenge der Potenzmenge $\mathcal{P}(M)$ der Menge M (siehe unten).

Definition 0.0.3 Ein *Alphabet* ist eine nicht-leere, endliche Menge Σ (deren Elemente wir Buchstaben oder Zeichen nennen). Ein Σ-*Wort* ist eine endliche Sequenz von Buchstaben aus Σ, $w = a_1 \dots a_n$ mit $a_i \in \Sigma$. Das *leere Wort* wird mit ε bezeichnet.

Die *Menge aller* Σ-*Wörter* wird mit Σ^* bezeichnet.

Eine Σ-Sprache ist eine Teilmenge $L \subseteq \Sigma^*$, d.h. eine Menge von Σ-Wörtern.

Σ^+ steht für die Menge der von ε verschiedenen (also nicht-leeren) Σ-Wörter.

Mit $|w|$ bezeichnen wir die *Länge* des Wortes w; für $w = a_1 \dots a_n$ ist $|w| = n$. Das leere Wort ε ist das einzige Wort der Länge 0. Σ^n steht für die Menge der Σ-Wörter der Länge $n \in \mathbb{N}$, $\Sigma^n = \{w \in \Sigma^* : |w| = n\}$. Beachte, dass $\Sigma^0 = \{\varepsilon\}$ ist.

Im letzten Beispiel waren wir an Wörtern über dem Alphabet $\Sigma = \{\square, w, h, k\}$ interessiert, anhand derer man von einem ausgezeichneten Anfangszustand zu einem bestimmten Endzustand gelangt. Aus dem Σ-Transitionssystem mit ausgezeichneten Anfangs- und Endzuständen gewinnen wir eine Sprache: die Menge derjenigen Wörter, die Transitionssequenzen vom Anfangs- zum Endzustand beschreiben. Im Beispiel waren wir Wörtern minimaler Länge in der betreffenden Sprache interessiert, und fanden als Lösungen $h\square khw\square h$ und $h\square whk\square h$.

Übung 0.0.4 Zu einer fest gewählten ganzen Zahl $n \geqslant 2$, einem Alphabet Σ und einem ausgewählten $a \in \Sigma$ betrachte folgende Aufgabe. Gesucht ist ein möglichst einfaches System, das auf einen (online fortlaufenden) Strom von Signalen aus Σ zu jedem Zeitpunkt die Information bereithält, ob die Anzahl der bisher eingetroffenen a durch n teilbar ist.

Ein a-Zähler mit arithmetischem Teilbarkeitstest braucht unbeschränkt viel Speicherplatz (Zustände).

(a) Kann man mit endlich vielen Zuständen auskommen? (Wenn nicht, wäre diese Aufgabe gar nicht in realen Systemen zu lösen!) Wieviele Zustände braucht man mindestens?

(b) Betrachte für $k = 0, \dots, n-1$ die Sprache $L_{n;k} \subseteq \Sigma^*$, die aus denjenigen Wörtern besteht, deren a-Zahl bei Division durch n den Rest k lässt. Wie verhält sich Zugehörigkeit zu den $L_{n;k}$ unter dem Aneinanderhängen von Wörtern?

Das Aneinanderfügen von zwei Σ-Wörtern u und v wie in (b) liefert das Σ-Wort $u \cdot v := uv$. Man bezeichnet die Operation $\cdot$ als *Konkatenation*:

$$u = a_1 \dots a_n \; ; \; v = b_1 \dots b_m \quad \longmapsto \quad u \cdot v := \underbrace{a_1 \dots a_n}_{u} \underbrace{b_1 \dots b_m}_{v}$$

Transitionssysteme werden uns in ähnlicher Rolle in der Form endlicher Automaten beschäftigen. Anhand derartiger Berechnungsmodelle werden wir uns hier vor allem mit der Klassifikation von Σ-Sprachen hinsichtlich ihrer ‘Kompliziertheit’, mit Methoden zu ihrer Generierung und Methoden zu ihrer Erkennung befassen.

1 Mengen, Relationen, Funktionen, ...

1.1 Mengen, Relationen, Funktionen, Strukturen

1.1.1 Mengen und Mengenoperationen

Mengen sind unstrukturierte Ansammlungen von Objekten, den *Elementen* der betreffenden Menge. Mengen können spezifiziert werden durch (naive) Aufzählung ihrer Elemente wie in $\mathbb{B} = \{0, 1\}$ für die Menge der Booleschen Werte, oder $\mathbb{N} = \{0, 1, 2, \dots\}$ für die Menge der natürlichen Zahlen.[1] Mengen sind wiederholungsfreie Sammlungen in dem Sinne, dass Mehrfachnennungen desselben Objekts ignoriert werden, z.B. $\{0, 0, 1\} = \{0, 1\}$. Standard-Mengen/Schreibweisen:

$\emptyset = \{\ \}$	die leere Menge
$\mathbb{B} = \{0, 1\}$	Menge der Booleschen Werte
$\mathbb{N} = \{0, 1, 2, 3, \dots\}$	Menge der natürlichen Zahlen (mit 0!)
$\mathbb{Z}$	Menge der ganzen Zahlen

sowie $\mathbb{Q}$ für die Menge der rationalen Zahlen, und $\mathbb{R}$ für die Menge der reellen Zahlen. Für endliche Mengen A bezeichnet $|A|$ die Anzahl der Elemente von A.

Elemente, Teilmengen, Mengengleichheit

Elementbeziehung: $a \in A$ (a ist Element der Menge A). [2] Z.B.: $0 \in \mathbb{N}$, $1/2 \notin \mathbb{N}$, $1/2 \in \mathbb{Q}$.

Teilmengenbeziehung (Inklusion): $B \subseteq A$ (B ist *Teilmenge* von A, oder B ist in A *enthalten*), wenn für alle $a \in B$ auch $a \in A$. Z.B. $\emptyset \subseteq \{0, 1\} \subseteq \mathbb{N} \subseteq \mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R}$.

Potenzmenge: Die Menge aller Teilmengen einer Menge A bildet eine Menge, die Potenzmenge der Menge A, $\mathcal{P}(A) = \{B \colon B \subseteq A\}$. Z.B. ist $\mathcal{P}(\mathbb{B}) = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$.

Mengengleichheit: Zwei Mengen sind gleich genau dann wenn sie genau dieselben Elemente haben: $A = B$ gdw. [3] sowohl $A \subseteq B$ als auch $B \subseteq A$.

Man schreibt auch $B \subsetneq A$ um zu sagen, dass $A \subseteq B$ und $A \neq B$ (strikte Inklusion). Man sagt dann auch, dass B eine *echte Teilmenge* von A ist. Z.B. ist $\Sigma^+ \subsetneq \Sigma^*$.

Teilmengen können durch eine definierende Eigenschaft spezifiziert werden nach dem Schema $B := \{a \in A \colon p(a)\}$ für eine Eigenschaft p. $B := \{a \in A \colon p(a)\}$ ist diejenige Teilmenge von A, die genau aus den Elementen von A besteht, die die Eigenschaft p erfüllen. Z.B. ist $\{n \in \mathbb{N} \colon 2 \text{ teilt } n\} = \{0, 2, 4, \dots\}$ die Menge der geraden natürlichen Zahlen.

Boolesche Mengenoperationen Die elementaren Mengenoperationen sind die Booleschen Operationen von *Durchschnitt*, *Vereinigung* und *Mengendifferenz*.

Der *Durchschnitt* $A \cap B = \{c \colon c \in A \text{ und } c \in B\} = \{a \in A \colon a \in B\} = \{b \in B \colon b \in A\}$. Zwei Mengen heißen *disjunkt*, falls ihr Durchschnitt die leere Menge ist.

Die *Vereinigung* $A \cup B = \{c \colon c \in A \text{ oder } c \in B\}$ (beachte, dass das logische "*oder*" nicht exklusiv ist: die Elemente von $A \cup B$ sind genau diejenigen, die Element von mindestens einer der Mengen A oder B sind).

---

[1] Wir fassen hier immer 0 als natürliche Zahl auf; es gibt hierzu unterschiedliche Standards in der Literatur.

[2] $a \notin A$ steht entsprechend als Abkürzung für "nicht $a \in A$", ebenso wie $a \neq b$ als Abkürzung für "nicht $a = b$" steht

[3] "gdw." und "$\Leftrightarrow$" stehen für "genau dann wenn".

Mengendifferenz: $A \setminus B = \{a \in A\colon a \notin B\}$. Beispiel: $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$.

Durchschnitte und Vereinigungen von mehr als zwei Mengen können analog definiert werden. Wir betrachten eine (endliche oder auch unendliche) Folge von Mengen A_i (wobei i irgendeine endliche oder unendliche Menge I durchläuft, die wir als Indexmenge für die Aufzählung der Mengen A_i verwenden). Dann kann man den Durchschnitt bzw. die Vereinigung aller Mengen A_i für $i \in I$ definieren:

$$\begin{aligned} \textstyle\bigcap_{i\in I} A_i &:= \big\{a\colon a \in A_i \text{ für alle } i \in I\ \big\}, \\ \textstyle\bigcup_{i\in I} A_i &:= \big\{a\colon a \in A_i \text{ für mindestens ein } i \in I\ \big\}. \end{aligned}$$

Beispiele: $\Sigma^* = \bigcup_{n\in\mathbb{N}} \Sigma^n$, wo Σ^n die Menge der Σ-Wörter der Länge n ist; und für die Menge Σ^+ der nicht-leeren Σ-Wörter ist $\Sigma^+ = \bigcup_{n\geqslant 1} \Sigma^n$.

Komplementbildung: Für Teilmengen einer festen Menge M, d.h. über dem Bereich $\mathcal{P}(M)$, heißt $\overline{B} := M \setminus B$ das *Komplement* von B (bezüglich M).

Bemerkung. $\mathcal{P}(M)$ bildet mit den Operationen Durchschnitt, Vereinigung und Komplement eine *Boolesche Algebra*, siehe Abschnitt 1.1.4.

Tupel und Mengenprodukte

Wir schreiben (a, b) für das *geordnete Paar* mit erster Komponente a und zweiter Komponente b. Beachte, dass $(a, b) = (a', b')$ gdw. $a = a'$ und $b = b'$.

Das *Kreuzprodukt* (kartesische Produkt, nach Descartes) der Mengen A und B ist die Menge aller geordneten Paare mit erster Komponente aus A und zweiter Komponente aus B: $A \times B := \{(a, b)\colon a \in A, b \in B\}$.

Analog zu geordneten Paaren (mit zwei Komponenten) betrachtet man geordnete Tripel (mit 3 Komponenten) usw. Allgemein spricht man von n-Tupeln (geordnete Tupel mit n Komponenten). Für n-Tupel schreiben wir z.B. $(a_1, \dots, a_n)$. Zwei n-Tupel $(a_1, \dots, a_n)$ und $(a'_1, \dots, a'_n)$ sind gleich, gdw. sie in allen Komponenten übereinstimmen, d.h., gdw. $a_i = a'_i$ für alle i von 1 bis n.

Kreuzprodukte von mehr als zwei Mengen sind als Mengen von Tupeln definiert:

$$A_1 \times A_2 \times \cdots \times A_n := \{(a_1, \dots, a_n)\colon a_i \in A_i \text{ für } i = 1, \dots n\}.$$

Für n-fache Produkte derselben Menge A schreibt man A^n:

$$A^n := \underbrace{A \times A \times \cdots \times A}_{n \text{ mal}} := \{(a_1, \dots, a_n)\colon a_i \in A \text{ für } i = 1, \dots n\}$$

ist die Menge aller n-Tupel über A.

Bemerkung. Man kann n-Tupel über Σ mit Wörtern der Länge n identifizieren (in der Notation verzichtet man bloß auf Klammern und Kommata): Σ^n kann also für beides stehen. Insbesondere wollen wir auch nicht zwischen Σ und Σ^1 unterscheiden, d.h. Wörter der Länge 1 (1-Tupel) "sind" die Elemente des Alphabets.

1.1.2 Relationen

Eine *n-stellige Relation* ($n \in \mathbb{N}$, $n \geqslant 1$) über einer Menge A ist eine Menge von n-Tupeln über A, d.h. eine Teilmenge $R \subseteq A^n$. $(a_1, \dots, a_n) \in R$ bedeutet, dass $a_1, \dots, a_n$ in der Relation R stehen. Speziell für zweistellige Relationen wird oft eine Schreibweise aRa' anstelle von $(a, a') \in R$ benutzt.

Z.B. ist die Gleichheitsrelation über einer Menge A die Relation $\{(a,a)\colon a \in A\}$. Ebenso werden etwa die üblichen Ordnungsrelation $\leqslant^{\mathbb{N}}$, $\leqslant^{\mathbb{Z}}$ über $\mathbb{N}$, $\mathbb{Z}$ usw. als zweistellige Relationen über der betreffenden Menge aufgefasst, z.B. $\leqslant^{\mathbb{N}} = \{(n,m) \in \mathbb{N}^2\colon n \leqslant m\}$. Die Kantenbeziehung in einem Graphen lässt sich als zweistellige Relation E über der Knotenmenge V beschreiben: $(u,v) \in E$ gdw. eine Kante von u nach v besteht.

Relationen können auch Elemente aus verschiedenen Grundmengen verbinden; man spricht auch von *mehrsortigen Relationen.*

Beispiel 1.1.1 Die Transitionen eines Transitionssystems über der Zustandsmenge Q mit Kantenbeschriftungen $a \in \Sigma$ kann als *Transitionsrelation* $\Delta \subseteq Q \times \Sigma \times Q$ formalisiert werden: $(q,a,q') \in \Delta$ bedeutet, dass es eine a-Transition von q nach q' gibt ($q \xrightarrow{a} q'$).

Beispiel 1.1.2 In *relationalen Datenbanken* werden Tabellen von Dateneinträge zu (mehrsortigen) Relationen zusammengefasst. In einer Personaldatenbank könnte z.B. eine 3-stellige Relation R für jeden Beschäftigten das Tripel (Pers-Nr, Alter, Eintrittsjahr) enthalten; oder eine 2-stellige Relation sämtliche Paare (Pers-Nr$_1$, Pers-Nr$_2$) derart, dass Beschäftigter 1 ein Vorgesetzter des Beschäftigten 2 ist.

Beispiel 1.1.3 Die *Präfixrelation* $\preccurlyeq$ auf Σ^* ist die 2-stellige Relation

$$\big\{(u, uw) \in \Sigma^* \times \Sigma^*\colon u, w \in \Sigma^*\big\}.$$

D.h., für Σ-Wörter u und v gilt $u \preccurlyeq v$ gdw. $v = uw$ für ein $w \in \Sigma^*$ ist, also wenn u ein Anfangsabschnitt (Präfix) von v ist.
Entsprechend schreiben wir $\prec$ für die strikte Präfixrelation, $u \prec v$ gdw. $v = uw$ für ein $w \in \Sigma^+$.

Einige wichtige Eigenschaften, die eine zweistellige Relation $R \subseteq A^2$ haben *kann*:

Reflexivität R heißt reflexiv gdw. für alle $a \in A$ gilt aRa.

Symmetrie R heißt symmetrisch gdw. für alle $a, b \in A$ gilt: $aRb \Leftrightarrow bRa$.

Transitivität R heißt transitiv gdw. für alle $a, b, c \in A$ gilt: (aRb und bRc) $\Rightarrow$ aRc.

Ordnungsrelationen

Partielle Ordnungsrelationen wie z.B. die Teilmengenrelation $\subseteq$ auf der Potenzmenge $\mathcal{P}(M)$ einer Menge M oder die Präfixrelation $\preccurlyeq$ auf Σ^* sind reflexiv, transitiv und *antisymmetrisch* i.d.S. dass (aRb und bRa) nur für $a = b$ gilt.

Die strikten Varianten, wie $\subsetneq$ oder $\prec$, sind ebenfalls transitiv, aber irreflexiv i.d.S., dass für kein Element aRa gilt. Diese sind demnach dann antisymmetrisch in dem strengeren Sinne, dass niemals aRb und bRa gleichzeitig gelten.

Lineare (oder totale) *Ordnungsrelationen* wie $\leqslant$ auf $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ oder $\mathbb{R}$ erfüllen zusätzlich eine Vergleichbarkeitsbedingung für je zwei Elemente: $a \neq b \Rightarrow (aRb$ oder $bRa)$.

Äquivalenzrelationen

Definition 1.1.4 Eine zweistellige Relation R über einer Menge A heißt *Äquivalenzrelation* falls R reflexiv, symmetrisch und transitiv ist.

Die Gleichheit ist eine Äquivalenzrelation. Äquivalenzrelationen allgemein sind wichtig als "verallgemeinerte Gleichheitsrelationen" i.d.S. dass man sie als "Gleichheit bis auf bestimmte ausgeblendete Unterschiede" deuten kann (Abstraktion). Interessiert uns z.B. bei Wörtern $w \in \Sigma^*$ gerade ausschließlich für ihre Länge (oder die Anzahl der "a"), so wollen wir Wörter derselben Länge (oder derselben "a"-Zahl) als äquivalent ansehen, selbst wenn sie nicht gleich sind.
Meist wählt man für Äquivalenzrelationen Symbole, die wie abgewandelte Gleichheitszeichen aussehen, z.B. $\equiv$, $\approx$, $\sim$ usw.

Beispiel 1.1.5 Sei $n \in \mathbb{N}, n \geqslant 2$. Zwei ganze Zahlen k, l heißen *kongruent modulo* n, in Symbolen: $k \equiv l \,(\mathrm{mod}\, n)$ oder $k \equiv_n l$, falls ihre Differenz $k - l$ ein ganzzahliges Vielfaches von n ist; d.h., falls $k = l + mn$ für ein geeignetes $m \in \mathbb{Z}$.
Die Relation $\equiv_n$ ist eine Äquivalenzrelation auf $\mathbb{Z}$.
Bedeutung von $k \equiv_n l$: k und l lassen denselben Rest bei Division durch n.

Übung 1.1.6 Sei Σ ein Alphabet, $a \in \Sigma$, $n \geqslant 2$. Für $w \in \Sigma^*$ sei $|w|_a$ die Anzahl der a in w. Betrachte die Relation $(u, v) \in R$ gdw. $|u|_a \equiv |v|_a \,(\mathrm{mod}\, n)$. Zeige, dass R eine Äquivalenzrelation auf Σ^* ist, und dass R mit Konkatenation (dem Aneinanderfügen von Wörtern) verträglich ist, in dem Sinne dass für uRv und $u'Rv'$ stets auch $uu'Rvv'$.

Ist R eine Äquivalenzrelation auf A, so zerfällt die Grundmenge A in disjunkte Teilmengen von jeweils untereinander äquivalenten Elementen. Diese Teilmengen bezeichnet man als die *Äquivalenzklassen* von R.

Definition 1.1.7 Sei R eine Äquivalenzrelation auf A. Wir bezeichnen mit $[a]_R$ die *Äquivalenzklasse von a bezüglich R*:

$$[a]_R := \{b \in A \colon aRb\}.$$

Die Elemente einer Äquivalenzklasse heißen auch *Repräsentanten* der Äquivalenzklasse.

Lemma 1.1.8 *Je zwei verschiedene Äquivalenzklassen sind disjunkt, und die Grundmenge ist die Vereinigung aller Äquivalenzklassen. D.h. die Äquivalenzklassen bilden eine disjunkte Zerlegung der Grundmenge.*

Beweis Sei $\approx$ eine Äquivalenzrelation auf A; wir schreiben $[a]$ für die Äquivalenzklasse von a.

Wenn $[a] \cap [b] \neq \emptyset$ so existiert ein $c \in [a] \cap [b]$. Es folgt, dass $a \approx c$ und $b \approx c$. Mit Symmetrie folgt $a \approx c$ und $c \approx b$; also mit Transitivität $a \approx b$. Dann ist aber $[a] = [b]$. Das zeigt, dass je zwei verschiedene Äquivalenzklassen disjunkt sind.

Dass jedes Element von A in (genau) einer Klasse enthalten ist, ergibt sich daraus, dass stets $a \in [a]$ (Reflexivität). $\square$

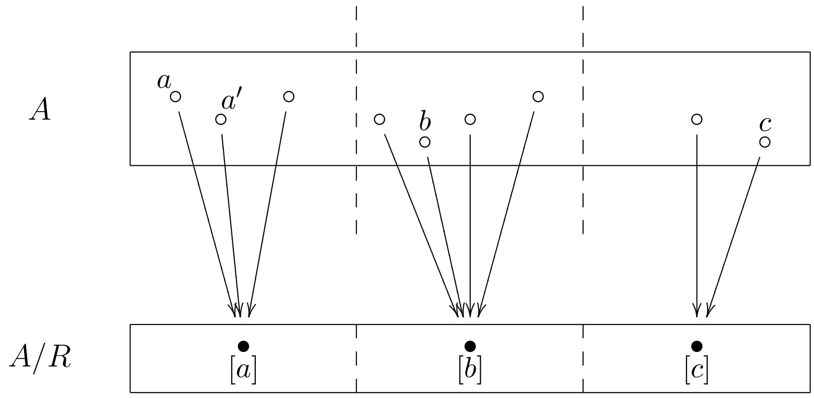
Definition 1.1.9 Die Menge aller Äquivalenzklassen von R auf A, der sogenannte *Quotient* bezüglich R, wird mit A/R bezeichnet:

$$A/R = \big\{[a]_R \colon a \in A\big\}.$$

Die Funktion $\pi_R \colon A \to A/R$, die jedem Element a seine Äquivalenzklasse $[a]_R$ zuordnet heißt *natürliche Projektion*.

Beachte, dass aRb gdw. $[a]_R = [b]_R$ gdw. $\pi_R(a) = \pi_R(b)$.

Das Diagramm illustriert den Fall einer Äquivalenzrelation R, die auf A gerade drei Äquivalenzklassen besitzt, die durch die untereinander nicht äquivalenten Elemente $a, b, c \in A$ repräsentiert werden (wir schreiben $[a]$ für $[a]_R$):



Beispiel 1.1.10 Im Falle von $\equiv_n$ zerfällt $\mathbb{Z}$ in genau n Äquivalenzklassen, die sogenannten Restklassen modulo n: für $k = 0, \ldots, n-1$ ist die k-te Restklasse modulo n die Teilmenge $[k]_n = \{nm + k \colon m \in \mathbb{Z}\}$ derjenigen ganzen Zahlen, die bei Division durch n Rest k haben.

Definition 1.1.11 Sei R eine Äquivalenzrelation auf A. Man sagt, dass R *endlichen Index* hat, wenn R endlich viele Äquivalenzklassen hat. Der *Index* von R ist dann die Anzahl der Klassen, $\mathrm{index}(R) := |A/R|$.

Übung 1.1.12 Bestimme die Äquivalenzklassen von R sowie den Index von R aus Übung 1.1.6.

Übung 1.1.13 Sei $f \colon A \to B$ irgendeine Funktion (s.u.). Dann wird durch $a \approx a' \;:\Leftrightarrow\; f(a) = f(a')$ eine Äquivalenzrelation auf A definiert. Jede Äquivalenzklasse entspricht genau einem Element des Bildes von f.
Jede beliebige Äquivalenzrelation R über A lässt sich auf diese Weise interpretieren, wenn man für B den Quotienten $B := A/R$ und für f die natürliche Projektion wählt.

Diskussion. Der Übergang von einer Menge zu ihrem Quotienten bezüglich einer Äquivalenzrelation entspricht i.d.R. einer Vereinfachung und Abstraktion, da Information, die im ursprünglichen Objektbereich verschiedene aber untereinander äquivalente Elemente unterscheiden ließ, dabei ausser acht gelassen wird. Man überlege sich dies an Beispielen wie der Verringerung der Farb- oder Pixelauflösung in einer Grafikdatei.

1.1.3 Funktionen und Operationen

Funktionen bilden die Elemente einer Ausgangsmenge (*Definitionsbereich*) auf Elemente einer Zielmenge (*Zielbereich*) ab. Das typische Format zur Spezifikation einer Funktion f von A nach B ist

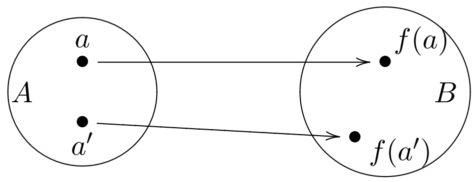
$$\begin{array}{rcl} f \colon A & \longrightarrow & B \\ a & \longmapsto & f(a), \end{array}$$

wo die erste Zeile die Mengen A und B als Definitions- und Zielbereich von f spezifiziert; die zweite Zeile enthält die konkrete Zuordnungsvorschrift, die z.B. durch eine Darstellung des Funktionswertes $f(a) \in B$ als Term oder durch irgend eine andere *eindeutige Definition* von $f(a)$ für alle $a \in A$ gegeben ist.

Ist $f(a) = b$, so ist b der *Funktionswert* oder das *Bild* von a unter f; a ein *Urbild* von b unter f. Das *Bild* von A unter f ist die Menge

$$f[A] := \{f(a)\colon a \in A\} \subseteq B.$$

Vorsicht: Das *Bild* $f[A]$ kann eine echte Teilmenge von B sein.



$f(a)$ ist das *Bild von* a unter f;
a ein *Urbild* von $b = f(a)$.

Injektivität und Surjektivität

Definition 1.1.14 Sei $f\colon A \to B$ eine Funktion.

(i) f heißt *surjektiv*, falls $f[A] = B$ (jedes Element des Bildbereichs wird mindestens einmal als Wert angenommen).

(ii) f heißt *injektiv*, falls für alle $a, a' \in A$ mit $a \neq a'$ gilt dass $f(a) \neq f(a')$ (jedes Element des Bildbereichs wird höchstens einmal als Wert angenommen).

(iii) f heißt *bijektiv*, falls f injektiv und surjektiv ist.

Verkettung von Funktionen Funktionen $g\colon A \to B$ und $f\colon B \to C$ können hintereinandergeschaltet werden. Ihre *Verkettung* ist die Funktion $f \circ g$ ("f nach g")

$$\begin{array}{rcl} f \circ g\colon A & \longrightarrow & C \\ a & \longmapsto & f(g(a)). \end{array}$$

Die Funktion $g\colon B \to A$ heißt *Umkehrfunktion* der Funktion $f\colon A \to B$ falls $g \circ f = \mathrm{id}_A$ und $f \circ g = \mathrm{id}_B$ ist. Dabei ist $\mathrm{id}_A\colon A \to A$ mit $\mathrm{id}_A(a) = a$ für alle $a \in A$ die *Identität* auf A, entsprechend für id_B.

Übung 1.1.15 f hat eine Umkehrfunktion gdw. f bijektiv ist.

Man schreibt auch f^{-1} für die Umkehrfunktion einer bijektiven Funktion f.
Bijektive Funktionen $f\colon A \to A$ werden als *Permutationen* bezeichnet.

Funktionen und Operationen Funktionen mit Definitionsbereich A^n heißen auch *n-stellige Funktionen auf A*. Eine Funktion des Typs $f\colon A^n \to A$ heißt auch *n-stellige Operation* über A. Eine n-stellige Operation ordnet jedem n-Tupel über A ein Element aus A zu. Die arithmetischen Operationen (Addition, Multiplikation) über $\mathbb{N}$ oder $\mathbb{Z}$ sind zum Beispiel zweistellige Operationen; ebenso Durchschnitt oder Vereinigung über $\mathcal{P}(M)$.

Speziell bei zweistelligen Operationen ist es oft üblich eine Notation wie $a * b$ anstelle von $f(a, b)$ zu wählen. Als Operationssymbole anstelle von $*$ werden dann auch Symbole wie $+$, $\cdot$ und dgl. verwendet (die dann oft nicht notwendigerweise für die vertrauten Operationen der Addition oder Multiplikation stehen).

Die *Identität* $\mathrm{id}_A\colon A \to A$ mit $\mathrm{id}_A(a) = a$ für alle $a \in A$, ist eine spezielle einstellige Operation auf A.

Beispiel 1.1.16 Auf Σ^* haben wir die zweistellige Operation der *Konkatenation*:

$$\begin{array}{rcl} \cdot\colon \Sigma^* \times \Sigma^* & \longrightarrow & \Sigma^* \\ (u,v) & \longrightarrow & u \cdot v := uv. \end{array}$$

Erinnerung: Für $u = a_1 \dots a_n$ und $v = b_1 \dots b_m$ ist $uv = \underbrace{a_1 \dots a_n}_{u}\underbrace{b_1 \dots b_m}_{v}$.

Ein- und zweistellige Operationen über endlichen Mengen A spezifiziert man oft durch eine Wertetafel. Für eine einstellige Operation $'\colon a \mapsto a'$ über $A = \{a_1, \dots, a_n\}$ bzw. für eine zweistellige Operation $*\colon (a,b) \mapsto a * b$ haben die Wertetafeln folgende Formate

	a_1	$\dots$	a_n
$'$	a_1'	$\dots$	a_n'

$*$	a_1	$\dots$	a_n
a_1	$a_1 * a_1$	$\dots$	$a_1 * a_n$
$\vdots$			
a_n	$a_n * a_1$	$\dots$	$a_n * a_n$

Zweistellige Operationen Einige wichtige Begriffe im Zusammenhang mit einer zweistellige Operation $*$ auf A (wir schreiben wie üblich $a * b$ anstelle von $*(a,b)$):

Assoziativität $*$ heißt assoziativ gdw. für alle $a,b,c \in A$ gilt: $(a * b) * c = a * (b * c)$.

Kommutativität $*$ heißt kommutativ gdw. für alle $a,b \in A$ gilt: $a * b = b * a$.

Neutrales Element $e \in A$ heißt neutrales Element für $*$ gdw. für alle $a \in A$ gilt: $a * e = e * a = a$.

Inverse Elemente Sofern $*$ ein neutrales Element e besitzt, heißt $a' \in A$ inverses Element zu $a \in A$ gdw. $a * a' = a' * a = e$. Die Operation $*$ hat inverse Elemente gdw. jedes Element $a \in A$ ein inverses Element besitzt.

Beispiel 1.1.17 Konkatenation auf Σ^* ist assoziativ und hat das leere Wort ε als neutrales Element. Kein Wort in $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ hat ein Inverses. Konkatenation ist nur über einelementigen Alphabeten kommutativ.

1.1.4 Algebraische Strukturen

Algebraische Strukturen (kurz: Strukturen) bestehen aus einer nicht-leeren Trägermenge und – je nach Typ der Struktur – einigen ausgezeichneten *Konstanten*, *Operationen* und *Relationen*. Konstanten sind ausgezeichnete Elemente der Trägermenge; (n-stellige) Operationen sind (n-stellige) Funktionen von der Trägermenge in die Trägermenge; n-stellige Relationen sind Mengen von n-Tupeln über der Trägermenge.

Beispiel 1.1.18 $(\mathbb{N}, +, 0)$ ist die Struktur mit der üblichen Addition (als zweistellige Operation) und der Null (als Konstante) über $\mathbb{N}$.

$(\mathbb{N}, +, \cdot, <, 0, 1)$ ist die Struktur mit der üblichen Addition und Multiplikation (als zweistelligen Operationen), der linearen Ordnung (als zweistelliger Relation), der Null und der Eins (als Konstanten) über $\mathbb{N}$.

$(\Sigma^*, \cdot, \varepsilon)$ ist die Struktur mit Konkatenation (als zweistelliger Operation) und dem leeren Wort ε (als Konstante) über Σ^*.

Halbgruppen, Monoide und Gruppen

Definition 1.1.19 Eine Struktur $(A, *)$ mit einer assoziativen zweistelligen Operation $*$ heißt *Halbgruppe*.

Eine Struktur $(A, *, e)$ mit zweistelliger Operation $*$ und Konstante e heißt *Monoid* falls $*$ assoziativ ist und e als neutrales Element hat. Wenn $*$ zusätzlich auch Inverse hat, so ist $(A, *, e)$ eine *Gruppe*.

Beispiel 1.1.20 $(\mathbb{N}, +, 0)$ ist ein Monoid, $(\mathbb{Z}, +, 0)$ eine Gruppe. Für jedes $n \in \mathbb{N}, n > 0$ bilden die Restklassen modulo n mit der Addition modulo n und der Restklasse der 0 eine Gruppe $\mathbb{Z}_n := (\mathbb{Z}/{\equiv_n}, +_n, [0]_n)$. Addition modulo n ist die Operation, die zwei Restklassen $[k]_n, [k']_n \in \mathbb{Z}/{\equiv_n}$ die Restklasse $[k + k']_n$ zuordnet (man muss sich vergewissern dass diese Definition unabhängig von der Wahl der Repräsentanten k und k' aus den betreffenden Restklassen ist!).

Beispiel 1.1.21 $(\Sigma^*, \cdot, \varepsilon)$ ist ein Monoid, das sogenannte *Wort-Monoid* zum Alphabet Σ (auch *freies Monoid über* Σ).

Boolesche Algebren

Wir betrachten Strukturen vom Typ der Potenzmengenalgebra $\big(\mathcal{P}(M), \cap, \cup, \bar{}\,, \emptyset, M\big)$ einer nicht-leeren Menge M (mit den Mengen-Operationen Durchschnitt, Vereinigung, Komplement, sowie der leeren und der vollen Menge als ausgezeichneten Elementen). Ein weiteres Standardbeispiel ist die zwei-elementige Boolesche Algebra mit Trägermenge $\mathbb{B} = \{0, 1\}$ und den aussagenlogischen Operationen (“*und*”, “*oder*”, “*nicht*”), und Konstanten 0 und 1 (vgl. Abschnitt 1.2.1).

Für die allgemeine Formulierung der Gesetzmäßigkeiten, die eine Struktur des genannten Typs erfüllen muss um eine Boolesche Algebra zu sein, verwenden wir die Symbole $+$ und $\cdot$ für die beiden zweistelligen Operationen, das Symbol $\;'$ für die einstellige Operation, und die Symbole 0 und 1 für die Konstanten.

Definition 1.1.22 Eine Struktur $\mathcal{B} = (B, \cdot, +, \;', 0, 1)$ heißt *Boolesche Algebra*, falls die folgenden Axiome erfüllt sind:

(1) $\cdot$ und $+$ sind assoziativ und kommutativ.

(2) *Idempotenz*; für alle $b \in B$ gilt: $\quad b \cdot b = b + b = b$

(3) *Distributivgesetze*:
- (i) für alle $a, b, c \in B$ gilt: $\quad a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
- (ii) für alle $a, b, c \in B$ gilt: $\quad a + (b \cdot c) = (a + b) \cdot (a + c)$

(4) *de Morgan Gesetze*:
- (i) für alle $a, b \in B$ gilt: $\quad (a \cdot b)' = a' + b'$
- (ii) für alle $a, b \in B$ gilt: $\quad (a + b)' = a' \cdot b'$

(5) *Absorption*; für alle $a, b \in B$ gilt: $\quad a \cdot (a + b) = a + (a \cdot b) = a$

(6) $\;'$ ist *involutiv*: für alle $b \in B$ gilt $\quad (b')' = b$

(7) für alle $b \in B$ gilt: $\quad b \cdot 0 = 0,\ b + 1 = 1$

(8) für alle $b \in B$ gilt: $\quad b \cdot 1 = b + 0 = b$

(9) $1 \neq 0$, und für alle $b \in B$ gilt: $\quad b \cdot b' = 0$ und $b + b' = 1$

Das angegebene Axiomensystem ist redundant. Es folgen sogar alle anderen Postulate auch bereits aus (1), (3), (8) und (9). [4] Wenn man sich hiervon überzeugt hat, reicht es, die Eigenschaften (1), (3), (8) und (9) nachzuprüfen, um festzustellen, ob eine gegebene Struktur eine Boolesche Algebra ist.

Beispiel 1.1.23 Für jede nicht-leere Menge M erfüllt die Struktur $\big(\mathcal{P}(M), \cap, \cup, ^-, \emptyset, M\big)$ die Axiome (1)–(9); jede derartige *Potenzmengenalgebra* ist also eine Boolesche Algebra.

Beispiel 1.1.24 Jede Boolesche Algebra muss mindestens zwei verschiedene Elemente für die Konstanten 0 und 1 haben (9). Tatsächlich bildet $\mathbb{B} = \{0, 1\}$ mit den folgenden Operationen eine (die kleinste) Boolesche Algebra, die Boolesche Algebra der klassischen Aussagenlogik (vgl. Abschnitt 1.2.1):

$\cdot$	0	1
0	0	0
1	0	1

$+$	0	1
0	0	1
1	1	1

	0	1
$'$	1	0

Übung 1.1.25

(a) Prüfen Sie die Axiome für die Potenzmengenalgebra einer Menge $M \neq \emptyset$ nach.

(b) Prüfen Sie die Axiome für die Boolesche Algebra der klassischen Aussagenlogik aus Beispiel 1.1.24 nach.

(c) Vergleichen Sie die Boolesche Algebra der klassischen Aussagenlogik mit der Potenzmengenalgebra einer einelementigen Menge M.

1.1.5 Homomorphismen und Isomorphismen

Homomorphismen sind *strukturerhaltende Abbildungen*. Sei

$$\begin{aligned} F\colon A &\longrightarrow B \\ a &\longmapsto F(a) \end{aligned}$$

eine Funktion. Seien A und B die Trägermengen von Strukturen $\mathcal{A}$ und $\mathcal{B}$ desselben Typs. D.h. zu jeder ausgezeichneten Konstanten $c^{\mathcal{A}} \in A$ der Struktur $\mathcal{A}$ gibt es die entsprechende Konstante $c^{\mathcal{B}}$ der Struktur $\mathcal{B}$; zu jeder Operation $f^{\mathcal{A}}$ der Struktur $\mathcal{A}$ die entsprechende Operation $f^{\mathcal{B}}$ der Struktur $\mathcal{B}$ von gleicher Stellenzahl; und zu jeder Relation $R^{\mathcal{A}}$ der Struktur $\mathcal{A}$ die entsprechende Relation $R^{\mathcal{B}}$ der Struktur $\mathcal{B}$ von gleicher Stellenzahl. Dann ist F ein *Homomorphismus* von $\mathcal{A}$ nach $\mathcal{B}$, falls F die gegebene Struktur respektiert i.d.S., dass:

(i) für jede Konstante c: $F(c^{\mathcal{A}}) = c^{\mathcal{B}}$.

(ii) für jede (n-stellige) Operation f: $F(f^{\mathcal{A}}(a_1, \dots, a_n)) = f^{\mathcal{B}}(F(a_1), \dots, F(a_n))$.

(iii) für jede (n-stellige) Relation R: $(a_1, \dots, a_n) \in R^{\mathcal{A}} \;\Rightarrow\; (F(a_1), \dots, F(a_n)) \in R^{\mathcal{B}}$.

Wir betrachten als Beispiele Homomorphismen für Strukturen des Typs $(A, *, e)$ mit einer zweistelligen Operation und einer Konstanten.

Beispiel 1.1.26 Die Längenfunktion auf Σ^*,

$$\begin{aligned} |\ |\colon \Sigma^* &\longrightarrow \mathbb{N} \\ w &\longmapsto |w|, \end{aligned}$$

[4] Für Spezialisten: Man zeigt, dass dass die Axiome (1), (3), (8) und (9) die Axiome (2), (7), (5) implizieren; und weiter, dass die Gleichungen in (9) das Element b' für jedes b eindeutig bestimmen, und dass daraus, zusammen mit (1), (6) folgt. Schließlich folgt (4) aus den übrigen.

ist ein Homomorphismus von $(\Sigma^*, \cdot, \varepsilon)$ nach $(\mathbb{N}, +, 0)$.

Ebenso ist, für $n \geqslant 1$ und $a \in \Sigma$, die Funktion $w \longmapsto |w|_a \bmod n$ ein Homomorphismus von $(\Sigma^*, \cdot, \varepsilon)$ nach $(\mathbb{Z}/\equiv_n, +_n, [0]_n)$ (vergleiche Beispiel 1.1.5 und Übung 1.1.6).

Beispiel 1.1.27 Betrachte eine Funktion $f \colon \Sigma_1 \to \Sigma_2$, die den Zeichen des Alphabets Σ_1 Zeichen in einem anderen Alphabet Σ_2 zuordnet. Die natürliche Fortsetzung zu einer Funktion

$$\begin{array}{rcl} \hat{f} \colon \Sigma^* & \longrightarrow & (\Sigma')^* \\ w = a_1 \dots a_n & \longmapsto & \hat{f}(w) := f(a_1) \dots f(a_n) \end{array}$$

ist ein Homomorphismus vom Wort-Monoid von Σ_1 in das Wort-Monoid von Σ_2. Tatsächlich ist $\hat{f}$ durch die Homomorphiebedingungen und die Bedingungen, dass $\hat{f}$ auf Wörtern der Länge 1 wie f wirkt, eindeutig charakterisiert.

Übung 1.1.28 Zeigen Sie analog zum letzten Beispiel, dass es zu jeder Vorgabe von Bildwörtern $h(a) = w_a \in \Sigma_2^*$ (für jedes $a \in \Sigma_1$) genau einen Homomorphismus $\hat{h} \colon \Sigma_1^* \to \Sigma_2^*$ zwischen den Wort-Monoiden zu Σ_1 und Σ_2 gibt mit $\hat{h}(a) = w_a$ für alle $a \in \Sigma_1$.

Definition 1.1.29 Ein Isomorphismus zwischen Strukturen $\mathcal{A}$ und $\mathcal{B}$ desselben Typs ist eine bijektive Abbildung zwischen den Trägermengen A und B, sodass sowohl die Abbildung selbst als auch ihre Umkehrung Homomorphismen sind. $\mathcal{A}$ und $\mathcal{B}$ heißen *isomorph*, $\mathcal{A} \simeq \mathcal{B}$, wenn es einen Isomorphismus zwischen ihnen gibt.

Isomorphismen erhalten alle Strukturmerkmale, und dies in umkehrbarer Weise. Die beteiligten Strukturen können als Realisierungen "derselben Struktur" über verschiedenen Trägermengen angesehen werden.

Beispiel 1.1.30 Betrachte die Potenzmengenalgebra $(\mathcal{P}(M), \cap, \cup, ^-, \emptyset, M)$ zu einer Menge $M \neq \emptyset$ und die Boolesche Algebra auf $\mathbb{B} = \{0, 1\}$. Die Abbildung $f \colon \mathbb{B} \to \mathcal{P}(M)$ mit $f(0) = \emptyset$ und $f(1) = M$ ist ein Homomorphismus zwischen Booleschen Algebren. Genau für einelementige M ist f surjektiv, und dann auch tatsächlich ein Isomorphismus.

Übung 1.1.31 Sei $M = \{1, \dots, n\}$ eine n-elementige Menge. Dann ist die Boolesche Algebra $(\mathcal{P}(M), \cap, \cup, ^-, \emptyset, M)$ isomorph zu einer Booleschen Algebra $(\mathbb{B}^n, \cdot, +, ', 0, 1)$ über der Trägermenge $\mathbb{B}^n$ (Binärwörter der Länge n). Wie müssen $\cdot, +, ', 0, 1$ über $\mathbb{B}^n$ definiert werden, damit die folgende Abbildung ein Isomorphismus ist? Eine Teilmenge $S \subseteq M$ wird abgebildet auf das Wort $b = b_1 \dots b_n$, wobei

$$b_i := \begin{cases} 1 & \text{wenn } i \in S \\ 0 & \text{wenn } i \notin S. \end{cases}$$

1.2 Elementare Beweistechniken

In diesem Abschnitt wollen wir informell einige wesentliche Konventionen und Schlussweisen der "mathematischen Alltagslogik" erläutern. Die mathematische Logik liefert hierfür (nächstes Semester) einen präzisen und formaleren Kontext.

1.2.1 Umgang mit aussagenlogischen Junktoren

Aussagen können wahr oder falsch sein; dies wird durch Boolesche *Wahrheitswerte* in $\mathbb{B} = \{0, 1\}$ wiedergegeben, wobei 0 für "falsch", 1 für "wahr" steht. Als Beispiel: die

Aussage $A \cap B = \emptyset$ (für zwei feste, vorgegebene Mengen A und B) hat den Wahrheitswert 0 (falsch), wenn A und B ein gemeinsames Element besitzen, andernfalls den Wahrheitswert 1 (wahr).

Die aussagenlogischen Operatoren (Junktoren) verknüpfen Aussagen. Semantisch weisen diese Operatoren den zusammengesetzten Aussagen Wahrheitswerte zu, die nur von den Wahrheitswerten der Bestandteile abhängen. Aussagenlogische Operatoren können in diesem Sinne anhand ihrer Operation auf den Wahrheitswerten definiert werden. Wir spezifizieren sie also als Operationen auf $\mathbb{B} = \{0, 1\}$.

Als grundlegende Operatoren verwenden wir die zweistelligen Operationen *Konjunktion* (aussagenlogisches "*und*", $\wedge$) und *Disjunktion* (aussagenlogisches "*oder*", $\vee$), sowie die einstellige Operation *Negation* ("*nicht*", $\neg$). Die Wertetabellen dieser Operationen:

$\wedge$	0	1
0	0	0
1	0	1

$\vee$	0	1
0	0	1
1	1	1

	0	1
$\neg$	1	0

Bemerkung: $\big(\mathbb{B}, \wedge, \vee, \neg, 0, 1\big)$ bildet eine Boolesche Algebra, wie man anhand der Axiome in Definition 1.1.22 nachprüft, vgl. Übung 1.1.24.

Die Disjunktion $\vee$ weist zwei Wahrheitswerten (p, q) genau dann den Wert 1 zu, wenn mindestens einer der beiden 1 ist. Entsprechend ist eine Disjunktion von zwei Aussagen A und B, "A oder B", wahr gdw. mindestens eine der Teilaussagen A, B wahr ist.

Man prüft nach, dass $p \wedge q$ stets denselben Wahrheitswert wie $\neg(\neg p \vee \neg q)$ ergibt. Man sagt, die aussagenlogischen Formeln $p \wedge q$ und $\neg(\neg p \vee \neg q)$ sind *logisch äquivalent*, da sie die gleiche Operation auf $\mathbb{B}$ beschreiben.

Für die bessere Lesbarkeit von zusammengesetzten Aussagen führt man noch einige weitere Junktoren ein, die sich aber durch die genannten zusammensetzen lassen. Die in der Tafel unten angegebenen Werte für *Implikation* ($\rightarrow$) und *Biimplikation* ("aussagenlogisches gdw.", $\leftrightarrow$) lassen sich z.B. darstellen als $p \rightarrow q := \neg p \vee q$ bzw. als $p \leftrightarrow q := (p \wedge q) \vee (\neg p \wedge \neg q)$. In der mathematischen Umgangssprache schreibt man meist "$\Rightarrow$" und "$\Leftrightarrow$" für diese logischen Verknüpfungen.

$\rightarrow$	0	1
0	1	1
1	0	1

$\leftrightarrow$	0	1
0	1	0
1	0	1

Achtung: Es ist wesentlich, dass in der mathematischen Normierung eine Implikation "A impliziert B" oder "aus A folgt B" insbesondere dann wahr ist, wenn A (die Voraussetzung) nicht erfüllt ist!

Übung 1.2.1 Zeige anhand einer Wahrheitswertetafel, dass die Negation von $p \wedge q$ äquivalent ist zu $\neg p \vee \neg q$; und die Negation von $p \rightarrow q$ äquivalent zu $p \wedge \neg q$.
Die Operation "*exklusives oder*", $p\dot{\vee}q$, soll wahr sein gdw. entweder p oder q wahr ist (und nicht beide). Drücke $p\dot{\vee}q$ mittels $\wedge$, $\vee$ und $\neg$ aus.

Aus allgemeinen logischen Beziehungen zwischen aussagenlogischen Formeln ergeben sich einige einfache Beweismuster für zusammengesetzte Aussagen. Ein paar nützliche Beispiele:

Kontraposition Die logische Äquivalenz von $(p \rightarrow q)$ und $(\neg q \rightarrow \neg p)$ erlaubt es, $A \Rightarrow B$ zu beweisen, indem man zeigt, dass aus "nicht B" folgt dass "nicht A" gilt.

Indirekter Beweis $\neg p \rightarrow 0$ ist logisch äquivalent zu p; daher kann man A beweisen, indem man aus "nicht A" irgendeine offensichtlich falsche Aussage (einen Widerspruch) herleitet.

Äquivalenzen $p \leftrightarrow q$ ist logisch äquivalent zu $(p \rightarrow q) \wedge (q \rightarrow p)$; man kann also eine Biimplikation (Äquivalenz) "A gdw. B" nachweisen, indem man zeigt, dass sowohl "A impliziert B" als auch "B impliziert A".

Implikationsketten Wenn $(p \rightarrow q)$ und $(q \rightarrow r)$ wahr sind, so auch $(p \rightarrow r)$; man kann also eine Implikation durch schrittweise Implikationen über geeignete Zwischenaussagen nachweisen. Insbesondere kann man die Äquivalenz von mehreren Aussagen wie z.B. A, B, C durch einen "Ringschluss" nachweisen: "A impliziert B", "B impliziert C" und "C impliziert A".

1.2.2 Quantoren

Wir verwenden gelegentlich die Quantoren-Schreibweisen $\forall$ ("*für alle*") und $\exists$ ("*es existiert*"), z.B. um eine Aussage $A(n)$ über natürliche Zahlen n zu quantifizieren: die *Existenzaussage* "$(\exists n \in \mathbb{N})A(n)$" besagt, dass $A(n)$ für mindestens eine natürliche Zahl n wahr ist; die *Allaussage* "$(\forall n \in \mathbb{N})A(n)$" besagt, dass $A(n)$ für alle natürlichen Zahlen n wahr ist.[5]

Bemerkung: Die Negation einer Allaussage "$(\forall n \in \mathbb{N})A(n)$" ist äquivalent zur Existenzaussage "$(\exists n \in \mathbb{N})$ nicht $A(n)$", und umgekehrt. Existenzaussagen kann man durch Angabe eines Existenzbeispiels nachweisen; eine Allaussage durch ein Gegenbeispiel widerlegen. Dagegen kann man eine Allaussage (über einem unendlichen Bereich) *nicht* durch Inspektion von Beispielen beweisen.

1.2.3 Beweise mittels Induktion

Beweise durch "*vollständige Induktion*" gehören zu den wichtigsten Methoden, Allaussagen über geeigneten Bereichen zu beweisen. Der Standardfall betrifft Allaussagen über den natürlichen Zahlen; wichtige andere Anwendungen (gerade auch in der Informatik) betreffen andere Bereiche von systematisch erzeugten Objekten ("induktive Datentypen").

Induktion über den natürlichen Zahlen

Das *Induktionsprinzip* für $\mathbb{N}$ besagt, dass man für eine Aussage $A(n)$ über natürliche Zahlen n, die Allaussage $(\forall n \in \mathbb{N})A(n)$ beweisen kann, indem man beweist:

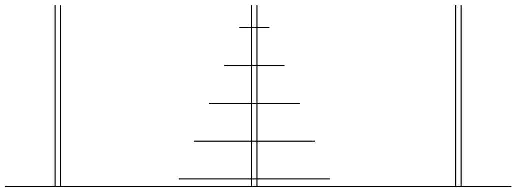
(i) (**Induktionsanfang**) $A(0)$.
Die Aussage $A(n)$ ist wahr für $n = 0$. (Einzelnachweis)

(ii) (**Induktionsschritt**) für alle $n \in \mathbb{N}$ gilt: $A(n) \Rightarrow A(n+1)$.
Wenn $A(n)$ wahr ist, muss auch $A(n+1)$ wahr sein.
(Nachweis von $A(n+1)$ unter der Annahme von $A(n)$)

Beispiel 1.2.2 Einer von drei Stäben trägt einen Stapel von n der Größe nach angeordneten Ringscheiben. Der Stapel soll auf einen der beiden anderen, anfangs leeren, Stäbe

[5] Die Klammern mit Angabe des Bereichs um die Quantoren wie in $(\forall n \in \mathbb{N})$ lässt man weg, wenn der Grundbereich klar ist.

umgeschichtet werden, ohne dass zwischendurch jemals eine größere Scheibe auf einer kleineren liegen darf. Behauptung: man braucht dazu minimal $f(n) := 2^n - 1$ Züge.



Beweis
Induktionsanfang: $n = 0$ (keine Scheiben), $f(0) = 0$ Schritte. [6]

Induktionsschritt: Wir nehmen an, dass man n Scheiben mit minimal $f(n) = 2^n - 1$ Zügen umschichten kann. Es ist zu zeigen, dass man dann $n + 1$ Scheiben in minimal $f(n + 1) = 2^{n+1} - 1$ Zügen umschichten kann. Das beinhaltet zwei Teilaussagen:

(1) nicht in weniger Zügen. Irgendwann muss die unterste, größte Scheibe bewegt werden. Dazu müssen alle n kleineren Scheiben auf dem dritten Stab sitzen, wozu mindestens $f(n)$ Züge nötig sind. Nach Bewegen der größten (1 Zug), müssen die n kleineren auf die größte umgesetzt werden, was wieder mindestens $f(n)$ Züge braucht. Man braucht also insgesamt mindestens $f(n) + 1 + f(n) = 2f(n) + 1$ Zügen. Mit der Induktionsvoraussetzung $f(n) = 2^n - 1$ also mindestens $f(n + 1) = 2^{n+1} - 1$ Zügen.

(2) die oben skizzierte Vorgehensweise lässt eine Umschichtung in $f(n) + 1 + f(n) = 2f(n) + 1$ Zügen zu. Nach Induktionsvoraussetzung $f(n) = 2^n - 1$ schafft man es also in $f(n + 1) = 2^{n+1} - 1$ Zügen. □

Das Induktionsprinzip lässt sich damit rechtfertigen, dass jede einzelne natürliche Zahl n durch eine Abfolge von n Nachfolgeschritten gemäß (ii) von 0 aus erreicht wird. Daraus ergibt sich also eine korrekte Implikationskette, die nach (i) mit der wahren Aussage $A(0)$ beginnt und also die Wahrheit von $A(n)$ liefert.

Bemerkung: Eine alternative Rechtfertigung beruht auf der Basis des "Minimalitätsprinzips", dass nämlich jede nicht-leere Teilmenge von $\mathbb{N}$ ein kleinstes Element hat. Damit argumentiert man wie folgt: wäre $A(n)$ nicht für alle $n \in \mathbb{N}$ wahr, so gäbe es ein kleinstes Gegenbeispiel, d.h. ein minimales $n_0 \in \mathbb{N}$, für das $A(n_0)$ falsch ist. Wegen (i) ist $n_0 \neq 0$ und hat also einen unmittelbaren Vorgänger $m_0 (= n_0 - 1)$, für den $A(m_0)$ wahr sein muss, da m_0 sonst ein kleineres Gegenbeispiel geliefert hätte. Aus der Wahrheit von $A(m_0)$ folgt mit dem Induktionsschritt (ii) aber, dass auch $A(m_0 + 1)$ wahr ist, also $A(n_0)$ da $m_0 + 1 = n_0$ ist – ein Widerspruch.

Induktion über anderen Bereichen

Das folgende Beispiel eines Induktionsprinzips über Σ^* lässt sich ganz ähnlich wie dasjenige über $\mathbb{N}$ rechtfertigen.

Beispiel 1.2.3 Sei $A(w)$ eine Aussage über Wörter $w \in \Sigma^*$. Dann lässt sich die Allaussage $(\forall w \in \Sigma^*)A(w)$ beweisen, indem man nachweist:

(i) (**Induktionsanfang**) $A(\varepsilon)$.
Die Aussage $A(w)$ ist wahr für $w = \varepsilon$. (Einzelnachweis)

(ii) (**Induktionsschritt**) für alle $w \in \Sigma^*$ und $a \in \Sigma$ gilt: $A(w) \Rightarrow A(wa)$.
(Nachweis von $A(wa)$ für jedes $a \in \Sigma$ unter der Annahme von $A(w)$)

[6] Man kann sich für ein paar weitere kleine Instanzen direkt vergewissern; also etwa, dass $f(1) = 1$ und $f(2) = 3$ stimmen.

Wieder besteht eine Rechtfertigung darin, dass sich jedes Σ-Wort in endlich vielen Schritten, in denen ein Buchstabe angehängt wird, aus dem leeren Wort generieren lässt.

Allgemeiner gibt es analoge Induktionsprinzipien über jedem Bereich, dessen Elemente systematisch aus gegebenen Anfangselementen durch vorgegebene Operationen erzeugt werden. Ist M die Menge derjenigen Objekte, die mittels (einer oder mehrerer, auch mehrstelliger) Operationen F aus der Anfangsmenge $M_0 \subseteq M$ erzeugt werden, so kann man die Allaussage $(\forall m \in M)A(m)$ anhand des folgenden Induktionsprinzips beweisen:

(i) (**Induktionsanfang**) $A(m)$ gilt für alle $m \in M_0$. (Einzelnachweise)

(ii) (**Induktionsschritt**) für *jede* der Operationen F gilt:
Für $m = F(m_1, \dots, m_n)$ ist $A(m)$ wahr, wenn $A(m_i)$ wahr ist für $i = 1, \dots, n$.
(Nachweis von $A(F(m))$ für $m = F(m_1, \dots, m_n)$ unter Annahme der $A(m_i)$)

Beispiel 1.2.4 Wir betrachten Terme aus einem zweistelligen Funktionssymbol $*$ und einer Konstanten c als Wörter über dem Alphabet $\Sigma = \{*, c, (,)\}$:

$$c, c * c, c * (c * c), \dots, (c * c) * (c * (c * c))), \dots$$

Die Menge T aller korrekt aufgebauten Terme wird als eine Σ-Sprache betrachtet. Wir wollen die folgende Behauptung durch Induktion über den Aufbau dieser Terme beweisen. Für jedes $t \in L$ gilt, dass das Symbol c genau einmal mehr als das Symbol $*$ vorkommt: $(\forall t \in M)(|t|_c = |t|_* + 1)$.

Beweis Induktion über den Aufbau der Terme. $M_0 = \{c\}$. Alle zulässigen Terme lassen sich, beginnend mit $c \in M_0$ erzeugen mittels der einen zweistelligen Operation F der "$*$-Anwendung", die aus Termen t_1 und t_2 eine korrekt geklammerte Termdarstellung für $t_1 * t_2$ produziert:

$$\begin{array}{rcl} F\colon M \times M & \longrightarrow & M \\ (t_1, t_2) & \longmapsto & F(t_1, t_2) := \begin{cases} (t_1) * (t_2) & \text{für } t_1, t_2 \neq c \\ c * (t_2) & \text{für } t_1 = c, t_2 \neq c \\ (t_1) * c & \text{für } t_1 \neq c, t_2 = c \\ c * c & \text{für } t_1 = t_2 = c \end{cases} \end{array} \qquad (\dagger)$$

Die Behauptung lässt sich damit per Induktion so beweisen:

Induktionsanfang: der einzige Term in M_0, c, erfüllt die Behauptung.

Induktionsschritt: unter der Annahme, dass in t_1 und t_2 jeweils c einmal mehr als $*$ auftritt, ist zu zeigen dass dies auch für $t = F(t_1, t_2)$ gilt. Dazu beachten wir dass in allen vier Fällen von $(\dagger)$ für $t = F(t_1, t_2)$ gilt:

$$|t|_c = |t_1|_c + |t_2|_c \quad \text{und} \quad |t|_* = 1 + |t_1|_* + |t_2|_*.$$

Mit der Induktionsvoraussetzung $|t_i|_c = |t_i|_* + 1$, für $i = 1, 2$, folgt die Behauptung. □

Beispiel 1.2.5 Betrachte den folgenden Erzeugungsprozess (eine Grammatik, wie wir sie in Kapitel 3 kennenlernen) für Wörter über dem Alphabet $\Sigma = \{X, (,)\}$:

$$\begin{array}{ll} (1) & X \to (\,) \\ (2) & X \to (X) \\ (3) & X \to XX \end{array}$$

Man deutet die drei Zeilen als Erzeugungsregeln, die jeweils aus bereits erzeugten Σ-Wörtern neue Wörter generieren, indem ein Vorkommen X im Ausgangswort durch eine der rechten Seiten ersetzt wird. Zum Beispiel erlaubt die gegebene Grammatik die Erzeugung der folgenden Erzeugungssequenzen, ausgehend von X (unser Startsymbol):

$$
\begin{aligned}
&X \xrightarrow{(1)} (\,) \\
&X \xrightarrow{(2)} (X) \xrightarrow{(1)} ((\,)) \\
&X \xrightarrow{(2)} (X) \xrightarrow{(3)} (XX) \xrightarrow{(1)} (X(\,)) \xrightarrow{(2)} ((X)(\,)) \xrightarrow{(1)} (((\,))(\,))
\end{aligned}
$$

Betrachtet man nur diejenigen erzeugbaren Wörter, in denen kein X mehr vorkommt, so erhält man gerade alle korrekt geschachtelten Klammerausdrücke.

Wir wollen durch Induktion zeigen, dass für alle aus X erzeugbaren Σ-Wörter jeweils bis zu jeder Stelle mindestens soviele "(" wie ")" aufgetreten sind, mit Gleichheit am Ende: Für jeden Präfix $u \preccurlyeq w$ ist $|u|_(\geqslant |u|_)$, und $|w|_(= |w|_)$.

Induktionsanfang: $w = X$ erfüllt die Behauptung.

Induktionsschritt: wenn w' aus w gewonnen wird, indem eine der drei Regeln auf ein Vorkommen von X in w angewandt wird, so gilt die Behauptung für w' wenn sie für w gilt.
Für Erzeugung mit Regel (1): Sei $w = w_1 X w_2$, $w' = w_1(\,)w_2$. Aus der Annahme für w folgt, dass $|w'|_(= |w'|_)$.

Alle Präfixe $u \preccurlyeq u'$ sind von der Form $u \preccurlyeq w_1$ oder $u = w_1($ oder $u = w_1(\,)$ oder $u = w_1(\,)u_2$ für ein $u \preccurlyeq w_2$. Für Präfixe der Form $u \preccurlyeq w_1$ folgt $|u|_(\geqslant |u|_)$ direkt aus der nach Annahme, da $u \preccurlyeq w$ ist. Damit folgt entsprechend für Präfixe

$$
\begin{array}{ll}
u = w_1(\,: & |u|_(= |w_1|_(+ 1 \geqslant |w_1|_) = |u|_); \\
u = w_1() \,: & |u|_(= |w_1|_(+ 1 \geqslant |w_1|_) + 1 = |u|_); \\
u = w_1()u_2 \text{ mit } u_2 \preccurlyeq w_2 \,: & |u|_(= |w_1u_2|_(+ 1 \geqslant |w_1u_2|_) + 1 = |u|_).
\end{array}
$$

Die Fälle für Regeln (2) und (3) lassen sich analog behandeln. □

Bemerkung: Die angegebene Bedingung zur Klammer-Bilanz charakterisiert genau die korrekt geschachtelten Klammerausdrücke, und alle solchen Ausdrücke werden von der angegebenen Grammatik erzeugt.

Bemerkung 1.2.6 *Analog zum Beweis durch Induktion gibt es Definitionen durch Rekursion, z.B. von Funktionen, deren Definitionsbereich in der beschriebenen Weise erzeugt wird; siehe z.B. Definition 2.1.3 im nächsten Abschnitt.*

Übung 1.2.7 Was ist faul am folgenden *falschen* Induktionsbeweis (über $\mathbb{N}$)?

Behauptung: Jede Gruppe von n Personen besteht aus gleichaltrigen Personen.

Induktionsanfang:
Für $n = 0$ (leere Gruppe) ist die Behauptung (leer aber) wahr. Ebenso für $n = 1$ (eine Person).

Induktionsschritt:
Unter der Annahme, dass jede Gruppe von $n > 0$ Personen aus gleichaltrigen Personen

besteht, wollen wir schließen, dass auch jede Gruppe von $n+1$ Personen aus gleichaltrigen Personen bestehen muss.

Sei also P eine Personengruppe der Größe $n + 1$. Greife zwei verschiedene Personen $p_1, p_2 \in P$ heraus und betrachte $P \setminus \{p_1\}$ und $P \setminus \{p_2\}$. Beide Gruppen haben die Größe n und bestehen daher aus untereinander gleichaltrigen Personen. Sei p eine Person im Durchschnitt von $P \setminus \{p_1\}$ und $P \setminus \{p_2\}$. Es folgt, dass in beiden Gruppen alle Personen dasselbe Alter wie p haben. Also sind alle Personen in P gleichaltrig.

2 Endliche Automaten – Reguläre Sprachen

2.1 Reguläre Sprachen

Erinnerung:

- $\Sigma \neq \emptyset$ endliches Alphabet; Σ^* Menge aller Σ-Wörter;
- Teilmengen $L \subseteq \Sigma^*$ heißen Σ-*Sprachen*.
- $\varepsilon \in \Sigma^*$ das leere Wort.
- $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ die Menge der nicht-leeren Σ-Wörter;
  für $n \in \mathbb{N}$: $\Sigma^n = \{w \in \Sigma^n : |w| = n\}$ Menge der Wörter der Länge n.
- $\begin{array}{rcl} \cdot : \Sigma^* \times \Sigma^* & \longrightarrow & \Sigma^* \\ (u,v) & \longmapsto & uv \end{array}$ Konkatenation von Wörtern;
- $(\Sigma^*, \cdot, \varepsilon)$ ist das zugehörige Wort-Monoid.

Beachte den Unterschied zwischen $\varepsilon \in \Sigma^*$ (dem leeren Wort), $\emptyset \subseteq \Sigma^*$ (der leeren Sprache) und $\Sigma^0 = \{\varepsilon\} \subseteq \Sigma^*$ (der Sprache, die aus genau dem leeren Wort besteht).

Für $a \in \Sigma$ und $n \in \mathbb{N}$ schreiben wir auch a^n für das Wort der Länge n, das aus n-maliger Wiederholung des Buchstabens a besteht; insbesondere ist $a^0 = \varepsilon$.

Operationen auf Sprachen Insbesondere hat man auf Σ-Sprachen die üblichen Booleschen Mengenoperationen:

Durchschnitt von zwei Σ-Sprachen, $L_1 \cap L_2$,

Vereinigung von zwei Σ-Sprachen, $L_1 \cup L_2$,

Komplement einer Σ-Sprache, $\overline{L} = \Sigma^* \setminus L$.

Daneben betrachten wir zwei weitere natürliche Operationen auf Σ-Sprachen: Konkatenation (von zwei Σ-Sprachen) und Stern-Operation oder Iteration (einer Σ-Sprache).

Konkatenation von Sprachen
Die Konkatenation der Σ-Sprachen L_1 und L_2 ist die Σ-Sprache

$$L_1 \cdot L_2 := \{v \cdot w : v \in L_1, w \in L_2\}.$$

Durch n-fach iterierte Konkatenation einer Sprache L mit sich selbst erhält man Sprachen L^n für $n \in \mathbb{N}$, rekursiv definiert als

$$\begin{aligned} L^0 &:= \{\varepsilon\} \\ L^{n+1} &:= L^n \cdot L \text{ für } n \in \mathbb{N}. \end{aligned}$$

Stern-Operation
Die Stern-Operation bildet aus der Σ-Sprache L die Σ-Sprache

$$L^* := L^0 \cup L^1 \cup L^2 \cup \ldots = \bigcup_{n \geqslant 0} L^n.$$

Beachte, dass $L^* = \{\varepsilon\} \cup \big\{w_1 \cdot \ldots \cdot w_n : n \geqslant 1, w_i \in L \text{ für } i = 1, \ldots, n\big\}$.

Wir lassen oft die Konkatenations-Punkte "·" weg, sowohl zwischen Wörtern wie auch zwischen Sprachen: also $w_1 w_2$ statt $w_1 \cdot w_2$ und auch $L_1 L_2$ statt $L_1 \cdot L_2$.

Übung 2.1.1 Weise die folgenden Gleichheiten für beliebige Σ-Sprachen L, L_1, L_2 nach:

(i) $L\big(L_1 \cup L_2\big) = (LL_1) \cup (LL_2)$ und $\big(L_1 \cup L_2\big)L = (L_1L) \cup (L_2L)$.

(ii) $L^* = \{\varepsilon\} \cup L \cdot L^*$.

(iii) $(L_1^* \cup L_2^*)^* = (L_1 \cup L_2)^*$.

Im Unterschied zu (i) ist im allgemeinen *nicht* $L\big(L_1 \cap L_2\big) = (LL_1) \cap (LL_2)$. Beispiel?

Reguläre Ausdrücke – reguläre Sprachen

Die regulären Σ-Sprachen werden durch Vereinigung, Konkatenation und Stern-Operation aus einfachen Ausgangssprachen erzeugt. Die Ausgangsprachen über dem Alphabet Σ sind: $\emptyset$, die leere Sprache; und, für jedes $a \in \Sigma$, die Sprache $\{a\}$, die nur aus dem Wort der Länge 1 (=Buchstabe) a besteht.

Wir benutzen die Syntax *regulärer Ausdrücke* als Notation für die Definition der regulären Sprachen.

Definition 2.1.2 [Syntax für REG] Die Menge REG(Σ) der *regulären Ausdrücke* über dem Alphabet Σ wird erzeugt wie folgt:

(i) $\emptyset$ ist ein regulärer Ausdruck.

(ii) für $a \in \Sigma$ ist $\mathbf{a}$ ein regulärer Ausdruck.

(iii) für $\alpha, \beta \in \mathrm{REG}(\Sigma)$ ist $(\alpha + \beta) \in \mathrm{REG}(\Sigma)$ [die "Summe" von α und β]. [7]

(iv) für $\alpha, \beta \in \mathrm{REG}(\Sigma)$ ist $(\alpha\beta) \in \mathrm{REG}(\Sigma)$ [das "Produkt" von α und β].

(v) für $\alpha \in \mathrm{REG}(\Sigma)$ ist $\alpha^* \in \mathrm{REG}(\Sigma)$ [der "Stern" von α, "α-Stern"].

Bemerkung: Man findet in der Literatur hierzu auch abweichende Syntax, z.B. $\alpha|\beta$ statt $\alpha + \beta$ und u.U. auch weitere (wie wir sehen werden, redundante) Terme (siehe z.B. die Konvention im Zusammenhang mit Beispiel 2.1.5 unten).

Die Semantik regulärer Ausdrücke besteht nun darin, dass jedem $\alpha \in \mathrm{REG}(\Sigma)$ eine Sprache $L(\alpha) \subseteq \Sigma^*$ zugeordnet wird – die *durch α beschriebene Σ-Sprache.*

Definition 2.1.3 [Semantik für REG] Rekursiv über $\alpha \in \mathrm{REG}(\Sigma)$ definiere die Sprache $L(\alpha) \subseteq \Sigma^*$ durch:

(i) $L(\emptyset) := \emptyset$.

(ii) $L(\mathbf{a}) := \{a\}$ für jedes $a \in \Sigma$.

(iii) $L(\alpha + \beta) := L(\alpha) \cup L(\beta)$.

(iv) $L(\alpha\beta) := L(\alpha) \cdot L(\beta)$.

(v) $L(\alpha^*) := \big(L(\alpha)\big)^*$.

Beispiel 2.1.4 Über $\Sigma = \{0, 1\}$ beschreibt der reguläre Ausdruck $\mathbf{1}^*\mathbf{0}\mathbf{1}^*$ die Sprache der Binärwörter mit genau einer 0. Das Komplement dieser Sprache, $\overline{L(\mathbf{1}^*\mathbf{0}\mathbf{1}^*)}$, wird z.B. durch diesen regulären Ausdruck beschrieben:

$$\underbrace{\mathbf{1}^*}_{\text{“keine 0”}} \quad + \quad \underbrace{(\mathbf{1}+\mathbf{0})^*\mathbf{0}(\mathbf{1}+\mathbf{0})^*\mathbf{0}(\mathbf{1}+\mathbf{0})^*}_{\text{“mindestens zwei 0”}}.$$

Beispiel 2.1.5 Für $\Sigma = \{a_1, \ldots, a_n\}$:

(i) $L(\emptyset^*) = \{\varepsilon\}$, unabhängig vom Alphabet.

[7] Überflüssige äußere Klammern lassen wir schließlich wieder weg. Wir vereinbaren auch, dass Produkte vor Summen Vorrang haben, um Klammern zu sparen.

(ii) $\Sigma = L(\mathbf{a}_1 + \cdots + \mathbf{a}_n)$.
(iii) $\Sigma^* = L((\mathbf{a}_1 + \cdots + \mathbf{a}_n)^*)$.
(iv) $\Sigma^+ = L((\mathbf{a}_1 + \cdots + \mathbf{a}_n)(\mathbf{a}_1 + \cdots + \mathbf{a}_n)^*)$.

Konvention. Wir wollen im folgenden auch die Ausdrücke ε, Σ, Σ^* und Σ^+ als reguläre Ausdrücke zulassen. Offiziell betrachten wir sie als Abkürzungen für die entsprechenden regulären Ausdrücke auf den rechten Seiten in Beispiel 2.1.5.

Übung 2.1.6 Finde reguläre Ausdrücke für die folgenden Sprachen über $\Sigma = \{0, 1\}$:
(a) Wörter einer Länge größer als 3, die mit 0 anfangen und mit 1 enden.
(b) Wörter ungerader Länge, in denen 0 und 1 alternieren.
(c) Wörter ohne 3 aufeinanderfolgende 1.
Beachte, dass in der Regel mehrere reguläre Ausdrücke dieselbe Sprache beschreiben.

Wir werden bald sehen, dass bei weitem nicht alle Sprachen durch reguläre Ausdrücke beschrieben werden, d.h., dass nicht alle Sprachen regulär sind im Sinne der folgenden Definition.

Definition 2.1.7 Eine Sprache $L \subseteq \Sigma^*$ heißt *regulär* falls $L = L(\alpha)$ für einen regulären Ausdruck $\alpha \in \mathrm{REG}(\Sigma)$.
Äquivalent: wenn sie mittels Vereinigungen, Konkatenationen und Stern-Operationen gewonnen werden kann, ausgehend von den Basissprachen $\emptyset$ und $\{a\}$ für $a \in \Sigma$.

Beobachtung 2.1.8 Jede endliche Σ-Sprache ist regulär.

Beweis Für jedes Wort $w \in \Sigma^*$ gibt es einen regulären Ausdruck α_w sodass $L(\alpha_w) = \{w\}$: Für $w = \varepsilon$ sei $\alpha_\varepsilon := \emptyset^*$; für $w = a_1 \ldots a_n$ mit $n \geqslant 1$ sei $\alpha_w := \mathbf{a}_1 \cdots \mathbf{a}_n$.
Für endliches $L \subseteq \Sigma^*$ ist $L = \emptyset = L(\emptyset)$ oder $L = \{w_1, \ldots, w_m\} = L(\alpha_{w_1} + \cdots + \alpha_{w_m})$. $\Box$

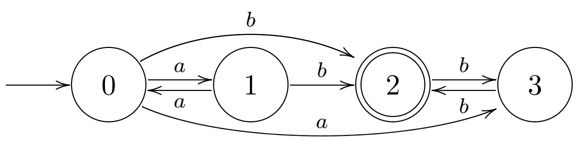
Bemerkung: Wir werden im Laufe der Vorlesung zwei wesentlich andere aber äquivalente Charakterisierungen der Klasse der regulären Sprachen kennen lernen. Nämlich als die von regulären Grammatiken erzeugten Sprachen (Kapitel 3) und als die von endlichen Automaten erkennbaren Sprachen (nächster Abschnitt).

2.2 Endliche Automaten

Endliche Automaten basieren auf Transitionssystemen, die verwendet werden, um Zugehörigkeit zu einer Σ-Sprache zu testen.

Automatentheorie ist eines der klassischen Gebiete der theoretischen Informatik, mit einer reichhaltigen Geschichte von ersten Anwendungen in der mathematischen Logik über die Theorie formaler Sprachen bis hin zu aktuellen Anwendungen im Model Checking für die Verifikation von Prozessen und Protokollen, und viele andere mehr.

Beispiel 2.2.1 Betrachte das Transitionssystem mit Zustandsmenge $Q = \{0, 1, 2, 3\}$ und Σ-Transitionen für $\Sigma = \{a, b\}$ wie im Diagramm:



Wir fassen den mit dem hereinkommenden Pfeil gekennzeichneten Zustand 0 als Anfangszustand auf (Start), den mit dem Doppelring gekennzeichneten Zustand 2 als akzeptierenden Endzustand (Ziel). In diesem Beispiel kann man sich überzeugen, dass genau die Wörter in

$$L = L\big((\mathbf{a}(\mathbf{aa})^*\mathbf{b}(\mathbf{bb})^*) + (\mathbf{aa})^*\mathbf{b}(\mathbf{bb})^* + \mathbf{b}(\mathbf{bb})^*\big) = L\big(\mathbf{a}^*\mathbf{b}(\mathbf{bb})^*\big)$$

einen Lauf haben, der vom Start zum Ziel führt. (In Worten: “beliebige Anzahl von a, gefolgt von ungerader Anzahl von b”.) In diesem Sinne werden wir sagen, dass dieses Transitionssystem die reguläre Sprache L *akzeptiert* oder *erkennt*.

Endliche Transitionssyteme

Sei Q eine endliche Zustandsmenge. Eine Transition zwischen Zuständen $q, q' \in Q$ ist eine gerichtete Kante $q \longrightarrow q'$, die anzeigt, dass der Übergang von q nach q' in einem Schritt möglich ist. Information über diesen Übergang steckt in einer Kantenbeschriftung mit Buchstaben eines Alphabets Σ:

$$q \xrightarrow{a} q'$$

besagt, dass der Übergang beim Lesen des Zeichens a erfolgen kann. Mehrfachbeschriftung wie in $q \xrightarrow{a,b} q'$ ist zulässig und besagt, dass der Übergang sowohl für a als auch für b möglich ist.

Formal ist ein *Transitionssystem* ein Tripel

$$\mathcal{S} = \big(\Sigma, Q, \Delta\big),$$

mit folgenden Komponenten:	Σ:	das Alphabet (für Kantenbeschriftungen)
	Q:	die endliche, nicht-leere Zustandsmenge
	$\Delta \subseteq Q \times \Sigma \times Q$:	die Transitionsrelation.

Die Transitionsrelation Δ enthält genau die Tripel (q, a, q') zu Transitionen $q \xrightarrow{a} q'$. [8]

Das Transitionssystem im Beispiel oben ist $\mathcal{S} = (\{a, b\}, \{0, 1, 2, 3\}, \Delta)$ wobei $\Delta = \big\{(0, a, 1), (1, a, 0), (0, b, 2), (0, a, 3), (1, b, 2), (2, b, 3), (3, b, 2)\big\}$. Das dort angegebene Diagramm ist der zugehörige *Transitionsgraph* mit Knoten für die Zustände und Σ-beschrifteten Kanten für die Transitionen.

2.2.1 Deterministische endliche Automaten

Ein Transitionssystem mit ausgezeichnetem Anfangszustand und ausgezeichneten akzeptierenden Zuständen ist ein endlicher Automat. Σ-Wörter werden als Eingaben aufgefasst, und die Kantenbeschriftungen legen fest, welche Übergangssequenzen ausgehend vom Anfangszustand ausgeführt werden bzw. ausgeführt werden können.

Unter prozeduralen Gesichtspunkten ist es natürlich, zunächst *deterministische* endliche Automaten (DFA: deterministic finite automata) zu betrachten, bei denen in *jedem* Zustand und für *jedes* $a \in \Sigma$ stets *genau ein Nachfolgezustand* gegeben ist. In diesem Fall ersetzt man die Übergangsrelation Δ durch eine Übergangsfunktion $\delta \colon Q \times \Sigma \to Q$, die für jede Kombination $(q, a) \in Q \times \Sigma$ gerade diesen eindeutig bestimmten Nachfolgezustand $q' = \delta(q, a)$ angibt.

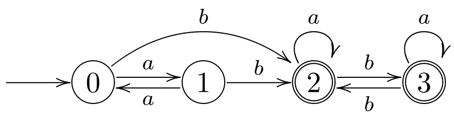
[8] Alternativ kann man auch eine Transitionsfunktion von $Q \times \Sigma$ nach $\mathcal{P}(Q)$ angeben, die jedem Paar $(q, a) \in Q \times \Sigma$ gerade die Menge derjenigen q' zuordnet, für die es eine Transition $q \xrightarrow{a} q'$ gibt.

Definition 2.2.2 [DFA] Ein *deterministischer endlicher* Σ-*Automat* ist spezifiziert als

$$\mathcal{A} = \big(\Sigma, Q, q_0, \delta, A\big).$$

Dabei ist	Q	die endliche, nicht-leere Zustandsmenge
	$q_0 \in Q$	der *Anfangszustand*
	$A \subseteq Q$	die Menge der *akzeptierenden Zustände*
	$\delta \colon Q \times \Sigma \to Q$	die *Übergangsfunktion.*

Zur anschaulichen Darstellung von endlichen Automaten durch Transitionsdiagramme markiert man den Anfangszustand (z.B. durch einen hereinkommenden Pfeil) und alle akzeptierenden Zustände durch Doppelkreise. Zum Beispiel:



Beachte, dass das *Transitionsdiagramm* eines DFA in jedem Zustand und für jedes $a \in \Sigma$ genau eine ausgehende a-Kante haben muss.

Berechnung eines DFA

Die *Berechnung eines DFA* $\mathcal{A}$ auf dem Eingabewort $w = a_1 \dots a_n \in \Sigma^*$ ist die Zustandsfolge $q_0, \dots, q_n$, wobei q_0 der Anfangszustand ist und $q_{i+1} = \delta(q_i, a_{i+1})$ für $i = 0, \dots, n-1$.

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \qquad \cdots \qquad \xrightarrow{a_n} q_n$$

Der Endzustand der Berechnung von $\mathcal{A}$ auf w ist q_n. Die Berechnung von $\mathcal{A}$ auf w ist *akzeptierend*, und $\mathcal{A}$ *akzeptiert* die Eingabe w genau dann wenn $q_n \in A$ (akzeptierender Endzustand). Ist der Endzustand $q_n \notin A$, so *verwirft* $\mathcal{A}$ das Eingabewort.

Der *Lauf eines DFA* $\mathcal{A}$ auf w vom Zustand q aus (nicht notwendig der Anfangszustand) ist analog definiert als die mit q beginnende Zustandsfolge mit Nachfolgezuständen gemäß δ.

Definition 2.2.3 Die *von* $\mathcal{A}$ *akzeptierte Sprache* oder *erkannte Sprache* ist

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

Definition 2.2.4 Man setzt die Transitionsfunktion $\delta \colon Q \times \Sigma \to Q$ eines DFA $\mathcal{A} = (\Sigma, Q, q_0, \delta, A)$ eindeutig fort zu einer Funktion

$$\hat{\delta} \colon Q \times \Sigma^* \quad \longrightarrow \quad Q,$$

die für jeden Zustand $q \in Q$ und jedes Wort $w \in \Sigma^*$ angibt, in welchem Zustand sich der Automat am Ende eines Laufes auf w vom Zustand q aus befindet.

Bemerkung: Die von $\mathcal{A}$ akzeptierte Sprache ist dann $L(\mathcal{A}) = \big\{w \in \Sigma \colon \hat{\delta}(q_0, w) \in A\big\}$.

$\hat{\delta}(q, w)$ ist rekursiv definiert über $w \in \Sigma^*$ durch:

$$\begin{aligned} \hat{\delta}(q, \varepsilon) &:= q \\ \hat{\delta}(q, wa) &:= \delta\big(\hat{\delta}(q, w), a\big). \end{aligned}$$

2.2.2 Nichtdeterministische endliche Automaten

Nichtdeterministische endliche Automaten (NFA: non-deterministic finite automata) arbeiten mit nicht notwendig deterministischen Transitionssystemen. Das bedeutet, dass in einzelnen Zuständen für manche $a \in \Sigma$ auch mehrere ausgehende a-Kanten oder garkeine a-Kante existieren können. Es können also sowohl sich verzweigende Berechnungspfade (Wahlmöglichkeiten, Nichtdeterminismus) als auch abbrechende Berechnungspfade auftreten (deadlock). Das Transitionssystem in Beispiel 2.2.1 ist nichtdeterministisch.

Definition 2.2.5 [NFA] Ein *nichtdeterministischer endlicher* Σ-*Automat* ist spezifiziert als

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

mit Q, $q_0 \in Q$ und $A \subseteq Q$ wie bei DFA, aber mit einer Transitions*relation*

$$\Delta \subseteq Q \times \Sigma \times Q.$$

Eine *Berechnung eines NFA* $\mathcal{A}$ auf $w = a_1 \dots a_n \in \Sigma^*$ ist jetzt eine Zustandsfolge $q_0, \dots, q_n$ mit $(q_i, a_{i+1}, q_{i+1}) \in \Delta$ für $i = 0, \dots, n-1$.
Eine Berechnung auf w ist *akzeptierend* wenn sie in einem akzeptierenden Zustand $q_n \in A$ endet.

Achtung: Zu gegebenem w kann es auch garkeine Berechnung auf w geben (wenn alle möglichen Berechnungen abbrechen), und etwaige Berechnungen sind in der Regel nicht eindeutig. Es kann insbesondere auch mehrere Berechnungen auf w geben, wovon auch manche akzeptierend und andere nicht akzeptierend sein können.

Definition 2.2.6 Die *vom NFA* $\mathcal{A}$ *akzeptierte Sprache* oder *erkannte Sprache* ist

$$L(\mathcal{A}) = \big\{ w \in \Sigma^* \colon \mathcal{A} \text{ hat mindestens eine akzeptierende Berechnung auf } w \big\}.$$

Diskussion: Beachte den Nichtdeterminismus und damit verbundener “Parallelismus” im NFA Modell. Die Definition der akzeptierten Sprache spricht über *alle möglichen Berechnungen* über dem Eingabewort.

Übung 2.2.7 Finde DFA und/oder NFA, die die folgenden $\{0,1\}$-Sprachen akzeptieren:

(i) $L = \{010, 0110\}^*$.
(ii) $L = \big\{ u010v \colon u, v \in \{0,1\}^* \big\}$.

Übung 2.2.8 Für einen gegebenen deterministischen endlichen Automaten $\mathcal{A}$ über Σ schreibe ein Programm (in Pseudocode), das für Eingabewörter w (Σ-wertige arrays) die Berechnung von $\mathcal{A}$ auf w simuliert und feststellt, ob w akzeptiert wird.
Wie könnte eine entsprechende Prozedur für einen nichtdeterministischen endlichen Automaten aussehen?
Welche Laufzeit haben diese Prozeduren in Abhängigkeit von der Länge der Eingabe w?

2.2.3 Von NFA zu DFA: der Potenzmengen-Trick

Satz 2.2.9 *Zu jedem nichtdeterministischen endlichen Automaten* $\mathcal{A}$ *lässt sich ein deterministischer endlicher Automat* $\mathcal{A}^{det}$ *konstruieren, der dieselbe Sprache erkennt, d.h. mit* $L(\mathcal{A}) = L(\mathcal{A}^{det})$.

Bemerkung: Tatsächlich sind mit NFA also nicht mehr Sprachen erkennbar als mit DFA. Die Zahl der Zustände von $\mathcal{A}^{\text{det}}$ kann aber exponentiell in der Zahl der Zustände von $\mathcal{A}$ sein. Ist $n = |Q|$ die Zahl der Zustände des gegebenen NFA $\mathcal{A}$, so hat der hier konstruierte äquivalente DFA $\mathcal{A}^{\text{det}}$ satt dessen $2^n = |\mathcal{P}(Q)|$ viele Zustände (vgl. auch Beispiel 2.4.17).

Beweis des Satzes. Sei $\mathcal{A} = (\Sigma, Q, q_0, \Delta, A)$ der gegebene NFA.
Idee: In der Berechnung auf $w = a_1 \dots a_n \in \Sigma^*$ sollen die Zustände des neuen DFA $\mathcal{A}^{\text{det}}$ in jedem Schritt gerade angeben, in welchen Zuständen der NFA $\mathcal{A}$ *sein kann* (in allen möglichen Berechnungen über dieser Eingabe). Dazu wählt man als neue Zustandsmenge $\hat{Q} := \mathcal{P}(Q)$ die Potenzmenge der alten Zustandsmenge – man spricht daher von der *Potenzmengen-Konstruktion* für die *Determinisierung eines NFA*.

$$\boxed{\begin{array}{lrl} \mathcal{A}^{\text{det}} := (\Sigma, \hat{Q}, \hat{q}_0, \delta, \hat{A}) \ \text{ mit} & \hat{Q} &:= \mathcal{P}(Q) = \big\{S \colon S \subseteq Q\big\} \\ & \hat{q}_0 &:= \{q_0\} \\ & \hat{A} &:= \big\{S \subseteq Q \colon S \cap A \neq \emptyset\big\} \\ & \delta(S,a) &:= \big\{q' \in Q \colon (q,a,q') \in \Delta \text{ für mindestens ein } q \in S\big\}. \end{array}}$$

Offensichtlich ist $\mathcal{A}^{\text{det}}$ ein DFA. Wir zeigen, dass $\mathcal{A}^{\text{det}}$ den NFA $\mathcal{A}$ wie gewünscht simuliert.

Sei $w = a_1 \dots a_n \in \Sigma^*$, $S_0, \dots, S_n$ die Zustandsfolge der Berechnung von $\mathcal{A}^{\text{det}}$ auf w. Für $0 \leqslant i \leqslant n$ sei $w(i) := a_1 \dots a_i$ der Präfix der Länge i von w. Also $w(0) = \varepsilon$, $w(1) = a_1$, $w(2) = a_1 a_2$, …, $w(n) = w$.

Per Induktion über i zeigen wir, dass

$$S_i = \left\{ q \in Q \colon \begin{array}{l} \text{mindestens eine Berechnung von } \mathcal{A} \text{ auf } w(i) \\ \text{ist nach dem } i\text{-ten Schritt in Zustand } q \end{array} \right\}$$

Induktionsanfang: Die Behauptung gilt für $i = 0$ und $S_0 = \{q_0\}$.

Induktionsschritt: Aus der Behauptung für i folgt die Behauptung für den $(i+1)$-ten Schritt $(i+1 \leqslant n)$ nach Definition von δ: $S_{i+1} = \delta(S_i, a_{i+1})$ ist die Menge der $q' \in Q$, die von einem der möglichen $q \in S_i$ über einen Übergang $q \xrightarrow{a_{i+1}} q'$ angenommen werden können.

Es bleibt zu zeigen, dass $L(\mathcal{A}^{\text{det}}) = L(\mathcal{A})$:

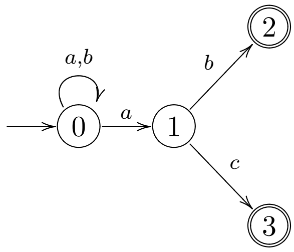
$$\begin{array}{ll} & w = a_1 \dots a_n \text{ wird von } \mathcal{A} \text{ akzeptiert} \\ \Leftrightarrow & \text{mindestens eine Berechnung von } \mathcal{A} \text{ auf } w \text{ führt zu einem Endzustand } q_n \in A \\ \Leftrightarrow & \text{mindestens ein akzeptierender Zustand ist Element von } S_n \\ \Leftrightarrow & S_n \in \hat{A} \\ \Leftrightarrow & \mathcal{A}^{\text{det}} \text{ akzeptiert } w. \end{array}$$

□

Bemerkung: Dass der NFA $\mathcal{A}$ auf Eingabe w ab einer Stelle in jeder Berechnung abbricht (deadlock), zeigt sich in der Berechnung von $\mathcal{A}^{\text{det}}$ darin, dass ab dieser Stelle nur noch der Zustand $\emptyset$ angenommen wird (beachte, dass der Zustand $\emptyset$ nie akzeptierend ist).

Man kann in der Potenzmengen-Konstruktion stets gleich alle Zustände in $\hat{Q}$ weglassen, die in keiner Berechnung vom Anfangszustand aus erreichbar sind (vgl. Bemerkung im folgenden Beispiel).

Beispiel 2.2.10 Zu $\Sigma = \{a, b, c\}$, betrachte den NFA



Die akzeptierte Sprache ist $L = L((a+b)^*a(b+c))$. Mit der Potenzmengen-Konstruktion bekommt man einen äquivalenten deterministischen Automaten mit δ gemäß der folgenden Wertetafel. Es empfiehlt sich allgemein, eine solche Tafel sukzessive ausgehend vom Startzustand $\{q_0\}$ aufzubauen und nur für Zustände $S \subseteq Q$ neue Zeilen aufzunehmen, die auch als δ-Werte auftreten: man eliminiert auf diese Weise gleich redundante, unerreichbare Zustände.

δ	a	b	c
$\{0\}$	$\{0,1\}$	$\{0\}$	$\emptyset$
$\{0,1\}$	$\{0,1\}$	$\{0,2\}$	$\{3\}$
$\{0,2\}$	$\{0,1\}$	$\{0\}$	$\emptyset$
$\{3\}$	$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Die akzeptierenden Zustände sind $\{0,2\}$ und $\{3\}$.

Es ist nützlich, in einem Diagramm für einige Eingaben parallel aufzuzeichnen, wie die *möglichen Berechnungen* des NFA und die *eindeutig bestimmte Berechnung* des DFA korrespondieren.

2.2.4 Abschlusseigenschaften

Ziel: die Klasse der von endlichen Automaten (DFA oder NFA) erkennbaren Sprachen ist abgeschlossen unter sämtlichen Booleschen Operationen (Durchschnitt, Vereinigung, Komplement) sowie unter Konkatenation und Stern. Wir werden daraus dann insbesondere folgern können, dass alle regulären Sprachen (i.S. von Definition 2.1.7) von endlichen Automaten erkannt werden.

Für manche der expliziten Automatenkonstruktionen arbeitet man leichter mit DFA, für andere leichter mit NFA – was der letzte Abschnitt rechtfertigt.

Lemma 2.2.11 *Zu DFA $\mathcal{A}_1$ und $\mathcal{A}_2$ über demselben Alphabet Σ gibt es DFA $\mathcal{A}$ für die folgenden Sprachen:*

(a) $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.

(b) $L(\mathcal{A}) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$.

(c) $L(\mathcal{A}) = \overline{L(\mathcal{A}_1)} = \Sigma^* \setminus L(\mathcal{A}_1)$.

Beweis Sei $\mathcal{A}_i = \big(\Sigma, Q^{(i)}, q_0^{(i)}, \delta^{(i)}, A^{(i)}\big)$, für $i = 1, 2$. Für jede der drei Aufgaben lässt sich der zugehörige DFA $\mathcal{A} = \big(\Sigma, Q, q_0, \delta, A\big)$ *effektiv* (d.h. mit einem Algorithmus) aus den gegebenen DFA konstruieren.

Für Durchschnitt (a) und Vereinigung (b) benutzt man einen *Produkt-Automaten* $\mathcal{A}$ mit Zustandsmenge $Q := Q^{(1)} \times Q^{(2)}$, der die Berechnung der einzelnen Automaten simultan simuliert, und wählt dann die akzeptierenden Zustände entsprechend. Sei dafür

$$\begin{aligned} Q &:= Q^{(1)} \times Q^{(2)}; \\ q_0 &:= (q_0^{(1)}, q_0^{(2)}); \\ \delta\big((q_1, q_2), a\big) &:= \big(\delta^{(1)}(q_1, a), \delta^{(2)}(q_2, a)\big). \end{aligned}$$

Für (a) wählen wir $A := A^{(1)} \times A^{(2)}$ (beide Einzelberechnungen akzeptieren).

Für (b): $A := (A^{(1)} \times Q^{(2)}) \cup (Q^{(1)} \times A^{(2)})$ (mindestens eine der Einzelberechnungen akzeptiert).

Den Automat für (c), Komplement, erhält man einfach indem man in $\mathcal{A}_1$ die akzeptierende Zustandsmenge durch ihr Komplement ersetzt: $A := Q^{(1)} \setminus A^{(1)}$.

□

Übung 2.2.12 Die Konstruktionen für $\cap$ und $\cup$ lassen sich auf NFA übertragen. Man überlege sich die Details. Für das Komplement sieht die Situation anders aus: Erkläre anhand eines Beispiels, welche Schwierigkeiten auftreten.

Übung 2.2.13 Gib eine bessere NFA-Konstruktion an, die aus zwei gegebenen NFA einen NFA für die Vereinigung der akzeptierten Sprachen liefert. "Besser" soll heißen, dass die Zustandszahl nur wie die Summe der Zustandszahlen der gegebenen Automaten anwachsen soll, anstatt mit deren Produkt.

Lemma 2.2.14 *Zu NFA $\mathcal{A}_1$ und $\mathcal{A}_2$ über demselben Alphabet Σ gibt es NFA $\mathcal{A}$ für die folgenden Sprachen:*
(a) $L(\mathcal{A}) = L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$.
(b) $L(\mathcal{A}) = \big(L(\mathcal{A}_1)\big)^*$.

Beweis Aus den gegebenen $\mathcal{A}_i = \big(\Sigma, Q^{(i)}, q_0^{(i)}, \Delta^{(i)}, A^{(i)}\big)$, $i = 1, 2$, erhalten wir die jeweiligen $\mathcal{A} = \big(\Sigma, Q, q_0, \Delta, A\big)$ wieder durch effektive Konstruktionen.

(a) Konkatenation. Wir nehmen der Einfachheit halber an, dass $Q^{(1)} \cap Q^{(2)} = \emptyset$, und betrachten den Fall, in dem $q_0^{(1)} \notin A^{(1)}$ (d.h. $\varepsilon \notin L(\mathcal{A}_1)$). Wir setzen

$$\begin{aligned} Q &:= Q^{(1)} \cup Q^{(2)} \quad \text{(disjunkt)}, \\ q_0 &:= q_0^{(1)}, \\ A &:= A^{(2)}, \\ \Delta &:= \Delta^{(1)} \cup \Delta^{(2)} \cup \big\{(q, a, q_0^{(2)}) \colon q \in Q^{(1)} \text{ und } (q, a, q') \in \Delta^{(1)} \text{ für ein } q' \in A^{(1)}\big\}. \end{aligned}$$

Die Definition von Δ ist so, dass die Transitionsgraphen der Einzelautomaten gerade durch Transitionen verbunden werden, die anstelle eines akzeptierenden Zustandes von $\mathcal{A}_1$ in den Anfangszustand von $\mathcal{A}_2$ überleiten. Es ist klar, dass jedes Wort in $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$ von $\mathcal{A}$ akzeptiert wird (man wählt die überleitende Transition an der Nahtstelle, um die zwei akzeptierenden Berechnungen zu verbinden.)

Umgekehrt muss jede akzeptierende Berechnung von $\mathcal{A}$ vom Anfangszustand in $Q^{(1)}$ zu einem Endzustand in $A \subseteq Q^{(2)}$ gelangen. An der Überleitungsstelle in der Berechnung hätte $\mathcal{A}_1$ den Anfangsabschnitt akzeptiert, und es schließt sich eine akzeptierende Berechnung von $\mathcal{A}_2$ an. Also werden nur Wörter in $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$ akzeptiert.

Frage: wie muss man diese Konstruktion modifizieren, falls $\varepsilon \in L(\mathcal{A}_1)$?

(b) Stern-Operation. Die Idee ist ganz analog; diesmal sollen akzeptierende Berechnungen von $\mathcal{A}_1$, in beliebiger Anzahl aneinandergehängt, zum Akzeptieren führen. Wir konstruieren zunächst einen Automaten $\mathcal{A}^+$ aus $\mathcal{A}_1$, indem wir Transitionen

$$(q, a, q_0^{(1)}) \qquad \text{für alle } (q, a, q') \in \Delta^{(1)} \text{ mit } q' \in A^{(1)}$$

hinzufügen. Die Zustandsmenge, Startzustand und akzeptierende Zustände bleiben unverändert.

Falls $\varepsilon \in L(\mathcal{A}_1)$ (d.h., wenn $q_0^{(1)} \in A^{(1)}$) erkennt der so gewonnene NFA $\mathcal{A}^+$ bereits wie gewünscht die Sprache $L(\mathcal{A}_1)^*$.

Im Fall $\varepsilon \notin L(\mathcal{A}_1)$ (d.h., wenn $q_0^{(1)} \notin A^{(1)}$) erkennt $\mathcal{A}^+$ lediglich die Sprache $L(\mathcal{A}_1)^* \setminus \{\varepsilon\}$. Dies lässt sich z.B. dadurch beheben, dass wir einen neuen, akzeptierenden Startzustand q_0 hinzufügen, der den Startzustand von $\mathcal{A}^+$ simuliert. □

Übung 2.2.15 Finde DFA/NFA, die die Ausgangssprachen $\emptyset$ und $\{a\}$ für $a \in \Sigma$ erkennen.

Die Beobachtung, dass

- die Ausgangssprachen von endlichen Automaten erkannt werden,
- die Vereinigung, Konkatenation und Stern von Sprachen, die von endlichen Automaten erkannt werden, wieder von endlichen Automaten erkannt werden,

zeigt (formal durch Induktion über die regulären Ausdrücke), dass alle regulären Sprachen von endlichen Automaten erkannt werden.

Korollar 2.2.16 *Jede reguläre Sprache wird von einem endlichen Automat erkannt; zu jedem regulären Ausdruck $\alpha \in \mathrm{REG}(\Sigma)$ kann man* effektiv *(d.h. mit einem Algorithmus) einen DFA konstruieren, der die Sprache $L(\alpha)$ erkennt.*

Die Umkehrung, dass nämlich endliche Automaten auch ausschließlich reguläre Sprachen erkennen können, liefert der Satz von Kleene (im nächsten Abschnitt).

Übung 2.2.17 Zeige, dass für jede reguläre Sprache $L \subseteq \Sigma^*$ auch die "Umkehrung" von L von einem NFA erkannt wird:

$$\mathrm{rev}(L) := \{w^{-1} \in \Sigma^* : w \in L\},$$

wo w^{-1} für "w rückwärts gelesen" steht: $w^{-1} = a_n \dots a_1$, wenn $w = a_1 \dots a_n$.
[Hinweis: man kann sich zuerst überlegen, dass man aus einem NFA, der nur einen akzeptierenden Zustand hat, durch "Umkehrung" der Transitionen einen geeigneten NFA bekommen kann. Andere Fälle lassen sich dann mit den übrigen Abschlusseigenschaften darauf zurückführen.]

Übung 2.2.18 Betrachte einen Homomorphismus vom Wort-Monoid von Σ_1 in das Wort-Monoid eines weiteren Alphabets Σ_2: $\hat{h} : \Sigma_1^* \to \Sigma_2^*$ wie in Beispiel 1.1.28.

Zeige, dass für jede reguläre Sprache $L \subseteq \Sigma_1^*$ auch $\hat{h}(L) := \{\hat{h}(w) : w \in L\} \subseteq \Sigma_2^*$ von einem NFA erkannt wird.

2.3 Der Satz von Kleene

Dieses zentrale Theorem der klassischen Automatentheorie besagt, dass die regulären Sprachen *genau* die von endlichen Automaten erkennbaren Sprachen sind. Man hat somit zwei äquivalente aber sehr verschiedene Charakterisierungen der Klasse der regulären Sprachen, *denotational* (reguläre Ausdrücke) bzw. *prozedural* (Automaten).

Satz 2.3.1 (Kleene's Theorem)
Für jede Sprache $L \subseteq \Sigma^*$ sind äquivalent:

(i) *L ist regulär: Es gibt einen regulären Ausdruck $\alpha \in \mathrm{REG}(\Sigma)$ mit $L = L(\alpha)$.*

(ii) *L ist Automaten-erkennbar: Es gibt NFA/DFA $\mathcal{A}$ mit $L = L(\mathcal{A})$.*

Als Korollar erhalten wir aus Lemma 2.2.11 und 2.2.14 Abschlusseigenschaften der Klasse der regulären Sprachen. Darunter sind der Abschluss unter Durchschnitt und Komplement (Lemma 2.2.11) nicht offensichtlich auf der Basis der Definition über reguläre Ausdrücke!

Korollar 2.3.2 *Die Klasse der regulären Sprachen ist abgeschlossen unter allen Booleschen Operationen sowie unter Konkatenation und Stern-Operation.*

Beweis des Satzes von Kleene. Korollar 2.2.16 behandelt die Richtung (i) $\Rightarrow$ (ii).

Für die umgekehrte Richtung, (ii) $\Rightarrow$ (i), wollen wir aus einem gegebenen endlichen Automaten einen regulären Ausdruck extrahieren.

Sei $\mathcal{A} = (\Sigma, Q, q_0, \delta, A)$ der gegebene DFA. Wir nehmen eine Zustandsmenge von der Form $Q = \{1, \dots, n\}$ an. Wie üblich sei $\hat{\delta}\colon Q \times \Sigma^* \to Q$ die Fortsetzung der Transitionsfunktion auf Σ^* (siehe Definition 2.2.4).

Für $0 \leqslant k \leqslant n$ und $1 \leqslant \ell, m \leqslant n$ betrachte die Sprache $L^k_{\ell,m}$, die aus denjenigen Σ-Wörtern besteht, für die $\mathcal{A}$ einen Lauf $\ell, q_1, \dots, q_s, m$ vom Zustand ℓ zum Zustand m hat, der zwischen ℓ und m nur Zustände q_i mit $q_i \leqslant k$ annimmt.

$$\ell \longrightarrow \underbrace{q_1 \longrightarrow q_2 \longrightarrow \cdots}_{q_i \in \{1,\dots,k\}} \longrightarrow m$$

Wir finden nun, induktiv über k, reguläre Ausdrücke $\alpha^k_{\ell,m}$ mit $L(\alpha^k_{\ell,m}) = L^k_{\ell,m}$.

Induktionsanfang. Für $k = 0$ ist

$$L^0_{\ell,m} = \begin{cases} \{a \in \Sigma\colon \delta(\ell,a) = m\} & \text{falls } \ell \neq m \\ \{\varepsilon\} \cup \{a \in \Sigma\colon \delta(\ell,a) = \ell\} & \text{falls } \ell = m \end{cases}$$

Diese endlichen Sprachen werden durch einfache reguläre Ausdrücke $\alpha^0_{\ell,m}$ beschrieben, vergleiche Beobachtung 2.1.8.

Induktionsschritt von k nach $k+1$ (wobei $k+1 \leqslant n$).

$$L^{k+1}_{\ell,m} = \underbrace{L^k_{\ell,m}}_{(1)} \cup \underbrace{L^k_{\ell,k+1}}_{(2)} \cdot \underbrace{(L^k_{k+1,k+1})^*}_{(3)} \cdot \underbrace{L^k_{k+1,m}}_{(4)},$$

mit Anteil (1) für Wörter, die einen entsprechenden Lauf sogar ohne Zustand $k+1$ zulassen; Anteil (2) für einen Lauf von Zustand ℓ zu einem ersten Auftreten von Zustand

$k+1$, Anteil (3) für (wiederholte) Schleifen durch Zustand $k+1$, und Anteil (4) für den Lauf vom letzten Auftreten von Zustand $k+1$ zum Zustand m.

Also können wir den gesuchten regulären Ausdruck für $L^{k+1}_{\ell,m}$ aus den bereits konstruierten regulären Ausdrücken für die $L^k_{..}$ mit Summe, Produkt und Stern zusammensetzten:

$$\alpha^{k+1}_{\ell,m} = \underbrace{\alpha^k_{\ell,m}}_{(1)} + \underbrace{\alpha^k_{\ell,k+1}}_{(2)}\, \underbrace{(\alpha^k_{k+1,k+1})^*}_{(3)}\, \underbrace{\alpha^k_{k+1,m}}_{(4)}.$$

Aus den regulären Ausdrücken für die $L^n_{\ell,m}$ gewinnen wir schließlich denjenigen für $L(\mathcal{A})$ als Summe über die Ausdrücke $\alpha^n_{q_0,q}$ für $q \in A$. □

2.4 Minimalautomaten und der Satz von Myhill-Nerode

Mit einer Sprache $L \subseteq \Sigma^*$ beziehungsweise mit einem endlichen deterministischen Automaten $\mathcal{A}$ über Σ assoziieren wir Äquivalenzrelationen $\sim_L$ bzw. $\sim_{\mathcal{A}}$ auf Σ^*. Aus der Untersuchung dieser Äquivalenzrelationen und ihrem Verhältnis im Falle $L = L(\mathcal{A})$ gewinnen wir

- ein algebraisches Kriterium dafür, dass eine Sprache L regulär ist.
- für reguläre Sprachen L einen DFA mit minimaler Zustandszahl, der L erkennt.

Erinnerung: (Definitionen 1.1.7 und 1.1.11) Ist R eine Äquivalenzrelation auf Σ^*, so schreiben wir $[w]_R$ für die Äquivalenzklasse von w bezüglich R; Σ^*/R bezeichnet die Menge aller Äquivalenzklassen von R. R hat *endlichen Index* wenn R endlich viele Äquivalenzklassen hat; $\mathrm{index}(R) = |\Sigma^*/R|$ heißt dann der *Index* von R.

Definition 2.4.1 Sei $L \subseteq \Sigma^*$. Die Äquivalenzrelation $\sim_L$ auf Σ^* ist definiert durch:

$$w \sim_L w' \quad \text{gdw.} \quad \text{für alle } x \in \Sigma^* \text{ ist } wx \in L \Leftrightarrow w'x \in L.$$

Wir prüfen nach, dass $\sim_L$ eine Äquivalenzrelation ist. $\sim_L$ ist offensichtlich reflexiv ($w \sim_L w$) und symmetrisch ($w \sim_L w' \;\Leftrightarrow\; w' \sim_L w$).

Transitivität: Sei $w_1 \sim_L w_2$ und $w_2 \sim_L w_3$. Dann gilt für alle $x \in \Sigma^*$: $w_1x \in L$ gdw. $w_2x \in L$ (da $w_1 \sim_L w_2$) gdw. $w_3x \in L$ (da $w_2 \sim_L w_3$). Also auch $w_1 \sim_L w_3$.

Weiter hat $\sim_L$ folgende interessante Eigenschaften:

Lemma 2.4.2 (i) *Die Äquivalenzrelation $\sim_L$ ist* rechts-invariant*: aus $w \sim_L w'$ folgt, dass für alle $u \in \Sigma^*$ auch $wu \sim w'u$.*

(ii) *L ist* abgeschlossen unter *$\sim_L$: $\big(w \in L$ und $w \sim_L w'\big) \;\Rightarrow\; w' \in L$.*

(iii) *L ist eine Vereinigung von Äquivalenzklassen von $\sim_L$.*

Beweis (i) folgt unmittelbar aus der Definition von $\sim_L$.

(ii) wenn $w \in L$ und $w \sim_L w'$, so besagt die Definition von $\sim_L$ für $x = \varepsilon$, dass $w = w\varepsilon \in L$ gdw. $w' = w'\varepsilon \in L$.

(iii) folgt direkt aus (ii): mit w gehört ganz $[w]_{\sim_L}$ zu L; also $L = \bigcup_{w \in L}[w]_{\sim_L}$. □

Auch ein Σ-DFA induziert eine natürliche und interessante Äquivalenzrelation auf Σ^*:

Definition 2.4.3 Sei $\mathcal{A} = (\Sigma, Q, q_0, \delta, A)$ ein DFA. $\hat{\delta}$ sei die Fortsetzung von δ auf $Q \times \Sigma^*$ gemäß Definition 2.2.4.[9] Die Äquivalenzrelation $\sim_{\mathcal{A}}$ auf Σ^* ist definiert durch:

$$w \sim_{\mathcal{A}} w' \quad \text{gdw.} \quad \hat{\delta}(q_0, w) = \hat{\delta}(q_0, w').$$

Man prüft nach, dass $\sim_{\mathcal{A}}$ eine Äquivalenzrelation ist.

Lemma 2.4.4 *Sei $\mathcal{A} = (\Sigma, Q, q_0, \delta, A)$ ein DFA, $L = L(\mathcal{A})$ die von $\mathcal{A}$ akzeptierte Sprache. Dann gilt für die Äquivalenzrelation $\sim_{\mathcal{A}}$ auf Σ^*:*

(i) *$\sim_{\mathcal{A}}$ hat endlichen Index; genauer:* $\text{index}(\sim_{\mathcal{A}}) \leqslant |Q|$.

(ii) *$\sim_{\mathcal{A}}$ ist rechts-invariant: Aus $w \sim_{\mathcal{A}} w'$ folgt, dass für alle $u \in \Sigma^*$ auch $wu \sim_{\mathcal{A}} w'u$.*

(iii) *$\sim_{\mathcal{A}}$ ist eine* Verfeinerung *von $\sim_L$: $w \sim_{\mathcal{A}} w'$ impliziert $w \sim_L w'$.*

Beweis (i) folgt unmittelbar aus der Definition von $\sim_{\mathcal{A}}$: den Äquivalenzklassen von $\sim_{\mathcal{A}}$ sind in injektiver Weise die Werte von $\hat{\delta}(q_0, \_)$ in Q zugeordnet: $[w]_{\mathcal{A}} = \{w' : \hat{\delta}(q_0, w') = \hat{\delta}(q_0, w')\}$. [Es kann sein, dass $\text{index}(\sim_{\mathcal{A}}) < |Q|$, nämlich wenn $\mathcal{A}$ *unerreichbare* Zustände hat, die in keiner Berechnung von angenommen werden.]

Zu (ii) benutzt man, dass $\hat{\delta}(q_0, wu) = \hat{\delta}\big(\hat{\delta}(q_0, w), u\big)$ ist.

Zu (iii) betrachte $w \sim_{\mathcal{A}} w'$. Dann ist $\hat{\delta}(q_0, w) = \hat{\delta}(q_0, w')$. Für jedes $u \in \Sigma^*$ gilt demnach $\hat{\delta}(q_0, wu) = \hat{\delta}\big(\hat{\delta}(q_0, w), u\big) = \hat{\delta}\big(\hat{\delta}(q_0, w'), u\big) = \hat{\delta}(q_0, w'u)$. Also für alle u:

$$\begin{aligned} wu \in L &\Leftrightarrow \hat{\delta}\big(\hat{\delta}(q_0, w), u\big) \in A \\ &\Leftrightarrow \hat{\delta}\big(\hat{\delta}(q_0, w'), u\big) \in A \\ &\Leftrightarrow w'u \in L. \end{aligned}$$

Das heißt also, dass $w \sim_L w'$. □

Korollar 2.4.5 *Sei $L = L(\mathcal{A})$ für einen DFA $\mathcal{A} = (\Sigma, Q, q_0, \delta, A)$. Dann ist*

$$\text{index}(\sim_L) \leqslant \text{index}(\sim_{\mathcal{A}}).$$

Beweis Direkt aus (iii) im letzten Lemma. Für jedes w ist nach (iii) $[w]_{\sim_{\mathcal{A}}} \subseteq [w]_{\sim_L}$. Also ist jede Äquivalenzklasse von $\sim_L$ eine Vereinigung von $\sim_{\mathcal{A}}$ Äquivalenzklassen. Daraus folgt dass $\text{index}(\sim_L) \leqslant \text{index}(\sim_{\mathcal{A}})$. □

Korollar 2.4.6 *Für jede reguläre Sprache L hat $\sim_L$ endlichen Index.*
Wird L von einem DFA mit n Zuständen erkannt, so ist $\text{index}(\sim_L) \leqslant n$.

Beweis Folgt aus dem vorangegangen Korollar zusammen mit (i) im Lemma. □

2.4.1 Der Satz von Myhill-Nerode

Zur Aussage des letzten Korollars gilt auch die umgekehrte Richtung, sodass sich eine Charakterisierung der regulären Sprachen ergibt.

Satz 2.4.7 (Myhill-Nerode) *Für jede Sprache $L \subseteq \Sigma^*$ sind äquivalent:*

(i) *$\sim_L$ hat endlichen Index.*

(ii) *L ist regulär.*

[9] Zur Erinnerung: $\hat{\delta}(q, w)$ gibt für $q \in Q$ und $w \in \Sigma^*$ an, in welchem Zustand der Lauf von $\mathcal{A}$ auf w endet, der in q startet.

Beweis Die Richtung (ii) $\Rightarrow$ (i) ist gerade der erste Teil von Korollar 2.4.6.

Für die interessantere Richtung (i) $\Rightarrow$ (ii) konstruieren wir einen Automaten, dessen Zustände gerade die Äquivalenzklassen von $\sim_L$ sind.

Der Äquivalenzklassen-Automat Sei $n :=$ index($\sim_L$). Wir schreiben $[w]$ für die Äquivalenzklasse von w unter $\sim_L$. Der folgende DFA $\mathcal{A} = (\Sigma, Q, q_0, \delta, A)$ benutzt die n-elementige Menge der Äquivalenzklassen, $\Sigma^*/{\sim_L}$, als Zustandsmenge (daher Äquivalenzklassen-Automat):

$Q := \Sigma^*/{\sim_L}$	alle Äq.-Klassen
$q_0 := [\varepsilon]$	die Äq.-Klasse von ε
$\delta([w], a) := [wa]$	die natürliche Operation von a auf Äq.-Klassen: ein a-Übergang führt von der Äq.-klasse von w zur Äq.-klasse von wa
$A := \{[w] \colon w \in L\}$	die Äq.-Klassen, aus denen L besteht.

Man muss zeigen, dass die Definition von δ konsistent ist, i.d.S., dass der Nachfolgezustand nur von der Äquivalenzklasse $[w]$, und nicht vom Repräsentanten w abhängt. Sei $w \sim_L w'$; dann ist für jedes $a \in \Sigma$ auch $wa \sim_L w'a$ (Rechts-Invarianz, siehe (i) in Lemma 2.4.2). Also wird durch $[wa] = [w'a]$ tatsächlich ein eindeutiger Zustand zugewiesen.

Zur Definition von A beachte, dass für $w \sim_L w'$ insbesondere auch $w \in L$ gdw. $w' \in L$; man könnte also äquivalent auch $A = \{[w] \colon [w] \subseteq L\}$ schreiben.

Es bleibt zu zeigen, dass der angegebene Automat gerade L erkennt: $L = L(\mathcal{A})$. Induktiv über $w \in \Sigma^*$ zeigt man dazu, dass für alle w gilt:

$$\hat{\delta}(q_0, w) = [w].$$

Induktionsanfang. Für $w = \varepsilon$ ist $\hat{\delta}(q_0, \varepsilon) = q_0 = [\varepsilon]$.

Induktionsschritt von w nach wa für $a \in \Sigma$. Sei $\hat{\delta}(q_0, w) = [w]$, $a \in \Sigma$. Dann ist $\hat{\delta}(q_0, wa) = \delta\big(\hat{\delta}(q_0, w), a\big) = \delta([w], a) = [wa]$.

Also endet die Berechnung von $\mathcal{A}$ auf w stets im Zustand $[w]$ und folglich wird w akzeptiert gdw. $[w] \in A$ gdw. $w \in L$. Damit ist die verbleibende Richtung im Satz von Myhill und Nerode bewiesen. Hat $\sim_L$ endlichen Index, so haben wir mit dem Äquivalenzklassen-Automat einen DFA gefunden, der L akzeptiert. Also ist L regulär. Zusätzlich haben wir erkannt, dass ein DFA mit gerade index($\sim_L$) vielen Zuständen die reguläre Sprache L erkennt. Aus Korollar 2.4.6 wissen wir, dass es mit weniger Zuständen nicht geht (dazu mehr in Abschnitt 2.4.3). □

Man kann den Satz von Myhill und Nerode benutzen, um nachzuweisen, dass eine Sprache nicht regulär ist. (Siehe dazu auch Abschnitt 2.5).

Beispiel 2.4.8 Die $\{0, 1\}$-Sprache $L = \{0^n 1^n \colon n \in \mathbb{N}\}$ ist nicht regulär. Es reicht, unendlich viele paarweise nicht $\sim_L$-äquivalente Wörter anzugeben, denn dann kann $\sim_L$ nicht endlichen Index haben.

Behauptung: die Wörter $w_n = 0^n$ für $n \in \mathbb{N}$ sind paarweise nicht äquivalent.

Begründung: für $k \neq \ell$ ist $w_k 1^k = 0^k 1^k \in L$ aber $w_\ell 1^k = 0^\ell 1^k \notin L$.

2.4.2 Exkurs: Das syntaktische Monoid

Sei $L \subseteq \Sigma^*$ eine Σ-Sprache. Zur Äquivalenzrelation $\sim_L$ auf Σ^* gibt es eine Verfeinerung $\approx_L$, die algebraisch an die Struktur des Wort-Monoids angepasst ist. Die *syntaktische Kongruenz* zu L, $\approx_L$ wird definiert durch

$$w \approx_L w' \quad \text{gdw.} \quad \text{für alle } x, y \in \Sigma^* \text{ ist } xwy \in L \Leftrightarrow xw'y \in L.$$

Übung 2.4.9 Zeige, dass $\approx_L$ eine Äquivalenzrelation auf Σ^* ist. $\approx_L$ ist eine Verfeinerung von $\sim_L$ i.d.S., dass stets $w \approx_L w' \;\Rightarrow\; w \sim_L w'$ gilt.

Lemma 2.4.10 *$\approx_L$ ist eine* Kongruenzrelation *auf Σ^*, d.h., $\approx_L$ ist verträglich mit der Konkatenationsoperation. Für alle $u, u', v, v' \in \Sigma^*$ gilt:*

$$\big(u \approx_L u' \text{ und } v \approx_L v'\big) \Rightarrow uv \approx_L u'v'. \qquad (*)$$

Beweis In der angegebenen Situation ist für $x, y \in \Sigma^*$ (die Definition von $\approx_L$ wird für den unterstrichenen Wortteil angewendet): $xu\underline{v}y \in L \;\Leftrightarrow\; xuv'y \in L \;\Leftrightarrow\; x\underline{u}v'y \in L$. □

Die Konkatenationsoperation lässt sich nun auf den Quotienten $\Sigma^*/{\approx_L}$ übertragen:

$$\begin{array}{rcl} \cdot : \Sigma^*/{\approx_L} \times \Sigma^*/{\approx_L} & \longrightarrow & \Sigma^*/{\approx_L} \\ ([u],[v]) & \longmapsto & [uv] \end{array}$$

ist unabhängig von den gewählten Repräsentanten der Äquivalenzklassen wegen $(*)$. Die algebraische Struktur $\big(\Sigma^*/{\approx_L}, \cdot, [\varepsilon]\big)$ ist ein Monoid, das *syntaktische Monoid* zur Sprache L.

Übung 2.4.11 Zeige, dass die natürliche Projektion, die einem Wort $w \in \Sigma^*$ seine Äquivalenzklasse $[w] \in \Sigma^*/{\approx_L}$ zuordnet, ein Homomorphismus vom Wort-Monoid $\big(\Sigma^*, \cdot, \varepsilon\big)$ auf das syntaktische Monoid $\big(\Sigma^*/{\approx_L}, \cdot, [\varepsilon]\big)$ ist.

In Analogie zum Fall von $\sim_L$ und mit dem Satz von Myhill-Nerode kann man (als Übung!) zeigen, dass $\approx_L$ endlichen Index hat gdw. $\sim_L$ endlichen Index hat (also gdw. L regulär ist).

2.4.3 Exkurs: Minimalautomat und Minimierung von DFA

Der Äquivalenzklassen-Automat aus dem Beweis des Satzes von Myhill und Nerode liefert für jede reguläre Sprache L einen DFA mit index($\sim_L$) vielen Zuständen, der L erkennt. Nach Korollar 2.4.6 kann es dafür keinen DFA mit weniger Zuständen geben.

Der im Beweis konstruierte Automat ist sogar als kleinster DFA für L bis auf Isomorphie eindeutig bestimmt, und heißt daher auch der *Minimalautomat* für die reguläre Sprache L.

Isomorphie von Automaten Um Isomorphie von DFA im Format (ein-sortiger) algebraischer Strukturen zu erfassen, kann man aus formalen Gründen zu einer Darstellung im Format $\big(Q, (\delta_a)_{a\in\Sigma}, A, q_0\big)$ übergehen, mit mehreren einstelligen Operationen

$$\begin{array}{rcl} \delta_a : Q & \longrightarrow & Q \\ q & \longmapsto & \delta_a(q) := \delta(q,a) \end{array}$$

anstelle der zwei-sortigen zweistelligen Transitionsfunktion δ. Es ist offensichtlich, dass dieses Format dieselbe Information erfasst. Ein Isomorphismus ist (im Sinne von Definition 1.1.29) genau was er sein sollte: eine Bijektion zwischen den Zustandsmengen, die die Operationen δ_a für jedes $a \in \Sigma$ ineinander überführt, und ebenso die Mengen der akzeptierenden Zustände und die Anfangszustände ineinander überführt. Das heißt also, dass ein Isomorphismus zwischen $\mathcal{A}^{(1)} = (\Sigma, Q^{(1)}, q_0^{(1)}, \delta^{(1)}, A^{(1)})$ und $\mathcal{A}^{(2)} = (\Sigma, Q^{(2)}, q_0^{(2)}, \delta^{(2)}, A^{(2)})$ durch eine bijektive Abbildung $f \colon Q^{(1)} \to Q^{(2)}$ gegeben ist, für die gilt:

(i) $f(q_0^{(1)}) = q_0^{(2)}$. [Anfangszustände]

(ii) für alle $a \in \Sigma$: $f\big(\delta^{(1)}(q,a)\big) = \delta^{(2)}\big(f(q),a\big)$. [Transitionen]

(iii) $f[A^{(1)}] = A^{(2)}$. [akzeptierende Zustände]

Die Verträglichkeitsbedingung (ii) lässt sich als Vertauschbarkeit der Wege im folgenden Diagramm deuten:

$$\begin{tikzcd} q \arrow[r, "\delta_a^{(1)}"] \arrow[d, "f"] & \delta^{(1)}(q,a) \arrow[d, "f"] \\ f(q) \arrow[r, "\delta_a^{(2)}"] & \delta^{(2)}(f(q),a) \end{tikzcd}$$

Wir gehen jetzt von einem gegebenen DFA $\mathcal{A}$ aus, der die reguläre Sprache $L = L(\mathcal{A})$ erkennt. Wir können annehmen, dass alle Zustände von $\mathcal{A}$ in Berechnungen erreichbar sind (unerreichbare können ohne Schaden entfernt werden).

Wir betrachten zur Vorbereitung die Konstruktion eines Äquivalenzklassen-Automaten auf den Äquivalenzklassen von $\sim_{\mathcal{A}}$. [10]

Beobachtung 2.4.12 Sei $\mathcal{A} = (\Sigma, Q, q_0, \delta, A)$ ein DFA ohne unerreichbare Zustände. Wir schreiben $[w]_{\mathcal{A}}$ für die Äquivalenzklasse von w bezüglich $\sim_{\mathcal{A}}$. Dann ist $\mathcal{A}$ isomorph zum Automaten $\mathcal{A}' = (\Sigma, Q', q_0', \delta', A')$ mit

$$\begin{aligned} Q' &:= \Sigma^*/{\sim_{\mathcal{A}}} = \big\{[w]_{\mathcal{A}} \colon w \in \Sigma^*\big\} \\ q_0' &:= [\varepsilon]_{\mathcal{A}}, \\ \delta'([w]_{\mathcal{A}}, a) &:= [wa]_{\mathcal{A}}, \\ A' &:= \big\{[w]_{\mathcal{A}} \colon \hat{\delta}(q_0, w) \in A\big\}. \end{aligned}$$

Man prüft nach, dass die Definition von δ' nicht von der Wahl des Repräsentanten w aus $[w]_{\mathcal{A}}$ abhängt. Die Abbildung, die jedem $q \in Q$ die Menge $\{w \in \Sigma^* \colon \hat{\delta}(q_0, w) = q\}$ zuordnet, ist eine Bijektion zwischen Q und $\Sigma^*/{\sim_{\mathcal{A}}}$ und ein Isomorphismus i.d.S., dass Anfangszustände, Transitionen und akzeptierende Zustände entsprechend ineinander überführt werden.

Lemma 2.4.13 *Jeder DFA $\mathcal{A}$ für L mit der minimalen Zustandszahl* index($\sim_L$) *ist isomorph zum Äquivalenzklassen-Automat zu $\sim_L$ aus dem Beweis des Satzes von Myhill und Nerode.*

[10] Die Vorgehensweise ist analog zur Konstruktion des Äquivalenzklassen-Automaten für $\sim_L$ im Beweis von Satz 2.4.7. Das Ergebnis ist diesmal ein zu $\mathcal{A}$ selbst isomorpher Automat.

Beweis Da $\mathcal{A}$ minimale Zustandszahl hat, müssen alle Zustände erreichbar sein, und $\mathcal{A}$ ist isomorph zum Äquivalenzklassenautomat zu $\sim_{\mathcal{A}}$ aus der Beobachtung. Aus Lemma 2.4.4 folgt, dass $\text{index}(\sim_{\mathcal{A}}) = \text{index}(\sim_L)$ und dass $\sim_{\mathcal{A}}$ gleich $\sim_L$ ist. Insbesondere ist also $\Sigma^*/\sim_{\mathcal{A}} = \Sigma^*/\sim_L$ und man überprüft, dass auch hinsichtlich der Anfangszustände, Transitionsfunktionen und akzeptierenden Zustände die beiden Äquivalenzklassenautomaten gleich sind. □

Minimierung eines DFA Sei ein DFA $\mathcal{A} = (\Sigma, Q, q_o, \delta, A)$ gegeben. Zunächst kann man alle Zustände eliminieren, die vom Anfangszustand nicht erreichbar sind. Wir nehmen also an, dass alle Zustände erreichbar sind: $\{\hat{\delta}(q_0, w) \colon w \in \Sigma^*\} = Q$.

Man kann dann einen DFA minimaler Größe aus $\mathcal{A}$ gewinnen, indem man Zustände von $\mathcal{A}$ geeignet miteinander identifiziert.

Durch sukzessive Verfeinerung einer Äquivalenzrelation $\sim$ auf der Zustandsmenge Q von $\mathcal{A}$ kann man die miteinander identifizierbaren Zustände in effizienter Weise finden.

Zwei Zustände $q \neq q'$ von $\mathcal{A}$ sind sicherlich wesentlich verschieden (stehen für unterschiedliche Information während möglicher Berechnungen) wenn

- $q \in A$ und $q' \notin A$, oder umgekehrt.
- wenn für ein $a \in \Sigma$ die Zustände $\delta(q, a)$ und $\delta(q', a)$ wesentlich verschieden sind.

Sei $q \not\sim_0 q'$ gdw. die erste Bedingung erfüllt ist. Dann ist $\sim_0$ eine Äquivalenzrelation auf Q mit zwei Klassen: A und $Q \setminus A$.

Sukzessive verfeinern wir nun diese Äquivalenzrelation durch Anwendung der zweiten Bedingung: $q \not\sim_{i+1} q'$ wenn $q \not\sim_i q'$ oder wenn für ein $a \in \Sigma$ gilt, dass $\delta(q, a) \not\sim_i \delta(q', a)$. Dann ist $\sim_{i+1}$ eine Äquivalenzrelation auf Q, die $\sim_i$ verfeinert.

Nach endlich vielen Iterationen erreichen wir eine Äquivalenzrelation $\sim = \sim_n = \sim_{n+1}$, die durch erneute Anwendung dieser Methode nicht mehr weiter verfeinerbar ist (dies geschieht nach höchstens $|Q|$ vielen Iterationen). Aus der Konstruktion ergibt sich dass

$$q \sim q' \quad \text{gdw.} \quad \text{für alle } u \in \Sigma^* \text{ gilt: } \hat{\delta}(q, u) \in A \Leftrightarrow \hat{\delta}(q', u) \in A.$$

Betrachte nun zwei (erreichbare) Zustände $q = \hat{\delta}(q_0, w)$ und $q' = \hat{\delta}(q_0, w')$. Wir behaupten, dass

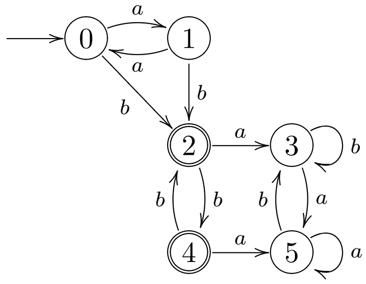
$$q \sim q' \quad \text{gdw.} \quad w \sim_L w'.$$

Wenn $\hat{\delta}(q_0, w) \sim \hat{\delta}(q_0, w')$, so gilt für alle $u \in \Sigma^*$, dass auch $\hat{\delta}(q_0, wu) \in A \Leftrightarrow \hat{\delta}(q_0, w'u) \in A$, und also $w \sim_L w'$.

Wenn $\hat{\delta}(q_0, w) \not\sim \hat{\delta}(q_0, w')$, so gibt es ein $u \in \Sigma^*$ mit $\hat{\delta}(q_0, wu) = \hat{\delta}\big(\hat{\delta}(q_0, w), u\big) \in A$ und $\hat{\delta}(q_0, w'u) = \hat{\delta}\big(\hat{\delta}(q_0, w'), u\big) \notin A$ oder umgekehrt. Also ist dann $w \not\sim_L w'$.

Die Äquivalenzklassen von Zuständen von $\mathcal{A}$ unter $\sim$ entsprechen daher genau den Äquivalenzklassen von $\sim_L$, und man erhält durch "Zusammenziehen dieser Klassen zu jeweils einem neuen Zustand" einen zum Minimalautomaten isomorphen DFA für L.

Übung 2.4.14 Finde einen äquivalenten DFA minimaler Größe für den folgenden DFA:



Übung 2.4.15 Finde den kleinstmöglichen DFA, der die $\{0,1\}$-Sprache der Wörter akzeptiert, die höchstens zwei voneinander getrennte Blöcke von mehr als 2 Einsen haben. Rechtfertige die Minimalitätsbehauptung.

Bemerkung 2.4.16 *Für NFA ist die Situation grundsätzlich anders als für DFA. Es gibt in der Regel keinen eindeutigen kleinsten NFA zur Erkennung einer regulären Sprache, sondern mehrere strukturell verschiedene.*

Generell kann ein minimaler NFA exponentiell viel kleiner sein als der kleinste äquivalente DFA. Hier ist ein Beispiel, in dem man dies gut sieht.

Beispiel 2.4.17 Für $n \geqslant 1$ sei $L_n \subseteq \{a,b\}^*$ die Sprache derjenigen $\{a,b\}$-Wörter, deren Länge mindestens n ist und in denen der n-te Buchstabe von hinten ein a ist. Als Übung gebe man NFA $\mathcal{A}_n$ an, die L_n mit nur $n+1$ Zuständen erkennen.

Demgegenüber muss der kleinste DFA, der L_n erkennt mindestens 2^n Zustände haben, da je zwei verschiedene Wörter $w, w' \in \{a,b\}^n$ nicht unter $\sim_{L_n}$ äquivalent sind.

Da $w \neq w'$ unterscheiden sich w und w' an einer Stelle i $(1 \leqslant i \leqslant n)$; dann ist genau eines der Wörter wa^{i-1} oder $w'a^{i-1}$ in L_n, da der unterscheidende Buchstabe in diesen Wörtern die n-te Stelle von hinten einnimmt.

Also ist $\mathrm{index}(\sim_{L_n}) \geqslant 2^n$ und die Behauptung folgt.

2.5 Nichtreguläre Sprachen: Das Pumping Lemma

Das "Pumping Lemma" erlaubt in vielen wichtigen Fällen einen einfachen Nachweis, dass eine Sprache *nicht regulär* ist. Formal gibt das Pumping Lemma eine restriktive notwendige Bedingung für Regularität. Intuitiv kommt die Beschränkung direkt von der *endlich beschränkten Speicher-Kapazität* eines endlichen Automaten. In jedem Punkt der Berechnung besteht die aktuelle Information über die Eingabe lediglich im aktuellen Zustand.

Beispiel 2.5.1 Betrachte die Sprache $\{0^n 1^n \colon n \in \mathbb{N}\}$ aus Beispiel 2.4.8. In einer Berechnung eines DFA, der die Eingabewörter von links nach rechts liest, müsste nach dem Lesen einer Serie von n Nullen der Zustand des Automaten die Information über die Länge n dieser Serie enthalten, denn es soll akzeptiert werden wenn sich genau n Einsen anschließen, nicht aber für irgend eine andere Anzahl. Schon um Wörter der Länge bis zu $2n$ korrekt zu bearbeiten, müsste ein Automat also mindestens n verschiedene Zustände haben. Demnach wird L nicht von einem endlichen Automat erkannt.

Lemma 2.5.2 (Pumping Lemma) *Für jede reguläre Sprache $L \subseteq \Sigma^*$ gibt es ein $n \in \mathbb{N}$, sodass sich jedes $x \in L$ mit $|x| \geqslant n$ zerlegen lässt als $x = u \cdot v \cdot w$, wobei $v \neq \varepsilon$, und für alle $m \in \mathbb{N}$ auch*

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

Man kann dabei u, v, w so wählen, dass $|u \cdot v| \leqslant n$.

Beweis Sei $\mathcal{A} = (\Sigma, Q, q_0, \delta, A)$ ein DFA, der L akzeptiert. Wähle $n := |Q|$.

Sei $x = a_1 \ldots a_\ell \in L$, $\ell \geqslant n$. Da $x \in L$, ist die Berechnung $q_0, \ldots, q_\ell$ von $\mathcal{A}$ auf x akzeptierend, d.h. $q_\ell \in A$. Da $\ell \geqslant n = |Q|$ müssen unter den ersten $n+1$ Zuständen dieser Berechnung Wiederholungen auftreten. Es gibt also i, j mit $0 \leqslant i < j \leqslant n$, für die $q_i = q_j$ ist. Wir wählen $v := a_{i+1} \dots a_j$ und entsprechend $u := a_1 \dots a_i$, $w := a_{j+1} \dots a_\ell$.

$$\underset{\substack{\uparrow\\ q_0}}{}\underbrace{a_1 \;\cdots\; a_i}_{u}\underset{\substack{\uparrow\\ q_i}}{}\underbrace{a_{i+1}\;\cdots a_j}_{v}\underset{\substack{\uparrow\\ q_j}}{}\underbrace{a_{j+1}\;\cdots a_\ell}_{w}\underset{\substack{\uparrow\\ q_\ell}}{}$$

Dann ist $x = uvw$, $|v| \geqslant 1$ und $|uv| \leqslant n$ wie gefordert.

Wir zeigen, dass mit diesen u, v, w auch $uv^m w \in L$ für jedes $m \in \mathbb{N}$.

Für $m = 0$ ist die Zustandsfolge $q_0, \dots, q_i, q_{j+1}, \dots, q_\ell$ ist eine akzeptierende Berechnung auf uw, da wegen $q_i = q_j$ der richtige Übergang an der Nahtstelle gesichert ist. Entsprechend wird die Zustandsfolge $q_{i+1}, \dots, q_j$ in der Berechnung über $uv^m w$ für $m > 1$ gerade m-fach wiederholt. Es ergibt sich also für jedes m eine akzeptierende Berechnung, die in q_ℓ endet. □

Beispiel 2.5.3 Es ist leicht zu sehen, dass die Sprache $\{0^n 1^n : n \in \mathbb{N}\}$ die Bedingung im Pumping Lemma verletzt. Vgl. auch Beispiel 2.4.8 und 2.5.1 oben.

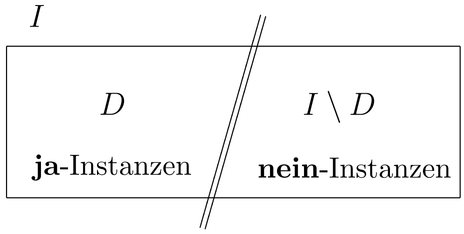
Übung 2.5.4 Betrachte die Sprache Palindrom $= \{w \in \{0,1\}^* : w = w^{-1}\}$, wo w^{-1} für "w rückwärts gelesen" steht (wenn $w = a_1 \dots a_n$, so ist $w^{-1} = a_n \dots a_1$). Ist Palindrom regulär?

Übung 2.5.5 Sei $\Sigma = \{(,)\}$. $L \subseteq \Sigma^*$ bestehe aus den korrekt geschachtelten Klammerausdrücken, vgl. Beispiel 1.2.5. Zeige, dass L nicht regulär ist.

2.6 Exkurs: Algorithmische Fragen

Entscheidungsprobleme

Ein *Entscheidungsproblem* ist ein Problem der folgenden Art. Gegeben eine Eingabe $x \in I$ aus einer Menge I von zulässigen Eingaben (Probleminstanzen), entscheide ob x die Eigenschaft D hat. Dabei wird D am einfachsten als eine Teilmenge $D \subseteq I$ aufgefasst, nämlich als die Teilmenge derjenigen Instanzen, die die gewünschte Eigenschaft haben.



Beispiel: Das Problem, von natürlichen Zahlen n zu entscheiden, ob sie Primzahlen sind, wird formalisiert als Entscheidungsproblem $D \subseteq I$, wo $I = \mathbb{N}$ und $D = \{p \in \mathbb{N} : p \text{ prim}\}$.

$D \subseteq I$ heißt *entscheidbar*, wenn es einen Algorithmus gibt, der für jede Eingabe $x \in I$ mit der korrekten Antwort "$x \in D$" bzw. "$x \notin D$" terminiert.

Im Zusammenhang mit regulären Sprachen und Automaten kann man zum Beispiel folgende Entscheidungsprobleme betrachten:

Zugehörigkeit zu einer regulären Sprache $L \subseteq \Sigma^*$. Auf Eingabe $w \in \Sigma^*$, entscheide ob $w \in L$. Entscheidbar durch Simulation eines DFA, der L erkennt. Ein DFA für L "ist" ein (linearer) Entscheidungsalgorithmus für dieses Problem. Vergleiche Übung 2.2.8.

Leerheitsproblem für reguläre Sprachen: Gegeben ein regulärer Ausdruck (oder ein DFA oder ein NFA), entscheide, ob die zugehörige Sprache leer ist. Entscheidbar durch Suchalgorithmus im Zustandsgraphen des DFA/NFA. Die Sprache ist nicht leer gdw. mindestens ein akzeptierender Zustand vom Anfangszustand aus erreichbar ist. Aus einer Eingabe in der Form eines regulären Ausdrucks kann man zunächst einen entsprechenden Automaten konstruieren (Kleene), und dann so verfahren.

Sprachgleichheit/Äquivalenzproblem gegeben zwei reguläre Ausdrücke (bzw. DFA, NFA), entscheide, ob die zugehörigen Sprachen L_1 und L_2 gleich sind (bzw., ob die gegebenen Automaten die gleiche Sprache erkennen). Entscheidbar durch Zurückführung auf die Leerheitsfrage für die der Automaten, die $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$ und $L_2 \setminus L_1 = L_2 \cap \overline{L_1}$ akzeptieren.

Ausdruckskomplexität

Wir haben im Zusammenhang mit dem Satz von Kleene gesehen, dass reguläre Ausdrücke und NFA (oder DFA) gleichermaßen als Beschreibungen regulärer Sprachen dienen können. Man kann sich nun fragen, wie sich die dabei die Größe des Automaten gegenüber der Länge des regulären Ausdrucks verhält. Bezüglich der regulären Ausdrücke (mit Summe, Konkatenation und Stern) sind die effekiven Konstruktionen von entsprechenden NFA effizient und die Größe des NFA polynomiell in der Länge des regulären Ausdrucks beschränkt. Für DFA tritt ein in der Regel exponentieller Größenzuwachs hinzu. Umgekehrt führt auch die Extraktion eines regulären Ausdrucks aus einem DFA (oder NFA) zu einem Ausdruck, dessen Länge exponentiell in der Größe des gegebenen Automaten sein kann.

Lässt man weitere Operationen und insbesondere Komplement-Bildung in "verallgemeinerten regulären Ausdrücken" zu, so kann der Komplexitätssprung zwischen der Beschreibung einer Sprache durch solche Ausdrücke und der Repräsentation durch Automaten dramatisch wachsen. Zwar wissen wir aus Lemma 2.2.11, dass jedes Komplement einer regulären Sprache selbst regulär und demnach durch einen regulären Ausdruck beschreibbar ist. Der Größenzuwachs für die Eliminierung eines Komplements ist jedoch exponentiell (Potenzmengen-Konstruktion), und bei geschachtelten Anwendungen von Konkatenation und Komplementbildung wird für die Konkatenation ein NFA konstruiert, der dann für die Komplementierung wieder in einen DFA verwandelt werden muss, usw.

Die wichtigsten Punkte aus Kapitel 2

reguläre Sprachen
REG **DFA** **NFA**
Abschlusseigenschaften / Automatenkonstruktionen
Satz von Kleene Satz von Myhill-Nerode
Pumping Lemma für reguläre Sprachen
Minimalautomat

3 Grammatiken und die Chomsky-Hierarchie

Grammatiken sind Formalismen zur endlichen Beschreibung von Sprachen durch einen *Erzeugungsprozess*. Man erfasst so auch allgemeinere Klassen von Sprachen als die regulären. Verglichen mit der Beschreibung der regulären Sprachen durch reguläre Ausdrücke, liegt die Betonung weniger auf einer statischen Beschreibung der Zielsprache als auf der Spezifikation eines dynamischen Prozesses zu ihrer Erzeugung.

3.1 Grammatiken

Die Grundidee ist, in einer *Grammatik* eine endliche Menge von *Produktionen* anzugeben, durch deren Anwendung sich jedes Wort der Zielsprache in endlich vielen Schritten erzeugen lässt. Anstelle von Produktionen spricht man auch von *Ableitungsregeln* und von den mit diesen Regeln (in endlich vielen Schritten) *ableitbaren* Wörtern.

Die einzelnen Regeln oder Produktionen erlauben *Ersetzungsprozesse*, in denen ein Abschnitt eines Wortes durch ein anderes Wort ersetzt werden darf. Für die Zwecke der Erzeugung einer Σ-Sprache stehen zwischendurch Hilfszeichen, die sogenannten *Variablen* der Grammatik, zur Verfügung, die im Laufe der Ableitung schließlich wieder eliminiert werden. Da in den schließlich erzeugten Wörtern nur noch Zeichen aus Σ vorkommen, spricht man von Σ als dem *Terminalalphabet*, im Unterschied zur *Variablenmenge* V. Diese beiden Alphabete sind disjunkt: $V \cap \Sigma = \emptyset$ und bilden zusammen das Alphabet der Grammatik.

Definition 3.1.1 [Grammatik] Eine *Grammatik* G ist spezifiziert als

$$G = \big(\Sigma, V, P, X_0\big)$$

Dabei ist		
	Σ	das endliche *Terminalalphabet*, $\Sigma \neq \emptyset$,
	V	die endliche Menge von *Variablen*, $V \cap \Sigma = \emptyset$
	$X_0 \in V$	die *Startvariable*/das *Startsymbol*
	$P \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$	die endliche Menge der *Produktionen/Regeln*

Für eine Produktion $p = (v, v') \in (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$ bezeichnet man v als die *linke Seite* und v' als die *rechte Seite* von p. Man schreibt auch $v \to v'$ für diese Produktion. Beachte, dass das leere Wort zwar als rechte Seite, nicht aber als linke Seite auftreten darf. Zur Angabe einer Grammatik G wird die Menge der Produktionen oft als Liste in der folgenden Form angegeben

$$\begin{array}{ccc} v_1 & \to & v_1' \\ \vdots & & \\ v_n & \to & v_n' \end{array}$$

wenn $P = \big\{(v_1, v_1'), \ldots, (v_n, v_n')\big\}$ aus diesen n Produktionen besteht.

Für den zugehörigen Erzeugungsprozess erlaubt eine Produktion $v \to v'$, in einem Wort uvw den Abschnitt v durch v' zu ersetzen und so in einem Ableitungsschritt aus dem Wort uvw das Wort $uv'w$ zu gewinnen. Als Variablen verwenden wir i.d.R. Großbuchstaben wie $X, X_1, X_2, \ldots, Y, Z$. Wenn z.B. $a, b, c \in \Sigma$ und $V = \{X, Y\}$, dann könnten z.B. die folgenden Produktionen vorkommen: $X \to \varepsilon$, $a \to \varepsilon$, $X \to ab$, $X \to aYb$, $aXbY \to aXbXc$, $aXbb \to Xc$, …

Ableitbarkeit Sei $G = (\Sigma, V, P, X_0)$ eine Grammatik, $x, x' \in (V \cup \Sigma)^*$. x' ist aus x *in einem Schritt ableitbar*, wenn $x = uvw$, $x' = uv'w$ für eine Produktion $p = v \to v'$ in P. Die 1-Schritt-Ableitbarkeit wird als 2-stellige Relation $\to_G$ formalisiert:

$$x \to_G x' \;:\Leftrightarrow\quad x = uvw,\ x' = uv'w \text{ für eine Produktion } v \to v' \text{ von } G.$$

Für die Erzeugung der Zielsprache werden beliebig lange endliche Ableitungssequenzen zugelassen. *Ableitbarkeit*, $x \to_G^* x'$, wird entsprechend definiert:

$$x \to_G^* x' \;:\Leftrightarrow\; \left\{ \begin{array}{l} \text{es existiert ein } \to_G\text{-Pfad von } x \text{ nach } x'\text{:} \\ x = x_1 \to_G x_2 \to_G \ldots \to_G x_n = x'. \end{array} \right.$$

Formal ist ein $\to_G$-Pfad von einem Wort x zu einem Wort x' eine endliche Folge von Wörtern $x_1, \ldots, x_n$ mit $x = x_1$, $x' = x_n$ und $x_i \to_G x_{i+1}$ für $i = 1, \ldots, n-1$. Dabei ist insbesondere auch $n = 0$ zugelassen. Also ist auch x aus x ableitbar, und $\to_G^*$ ist reflexiv und transitiv (vgl. Abschnitt 1.1.2). [11]

Definition 3.1.2 Ein Wort $w \in (V \cup \Sigma)^*$ heißt in der Grammatik $G = (\Sigma, V, P, X_0)$ *ableitbar*, gdw. w aus der Startvariablen X_0 ableitbar ist, d.h., falls $X_0 \to_G^* w$.

Die von G *erzeugte* Σ*-Sprache* ist die Menge aller Σ-Wörter (ohne Variablen!), die in G ableitbar sind:

$$L(G) := \big\{ w \in \Sigma^* \colon\ w \text{ ableitbar in } G \big\} = \big\{ w \in \Sigma^* \colon X_0 \to_G^* w \big\}.$$

Wir sagen dass zwei Grammatiken *äquivalent* sind, wenn sie dieselbe Sprache erzeugen.

Beispiel 3.1.3 Sei $G = (\Sigma, V, P, X_0)$ mit $\Sigma = \{0, 1\}$, $V = \{X\}$, $X_0 = X$ und Produktionen $P = \big\{ X \to \varepsilon, X \to 0, X \to 1, X \to 0X0, X \to 1X1 \big\}$. Dann ist $X \to_G^* w$ genau für alle Wörter $w \in \{0, 1, X\}^*$ der Gestalt

$$w = uXu^{-1} \quad \text{bzw.} \quad w = uu^{-1},\ w = u0u^{-1} \text{ oder } w = u1u^{-1}$$

wobei $u \in \{0, 1\}^*$ mit Umkehrung u^{-1}. Die Grammatik G erzeugt also die Sprache Palindrom aus Übung 2.5.4 (die nicht regulär ist).

Beispiel 3.1.4 (vgl. Beispiel 1.2.5 und Übung 2.5.5) Sei $\Sigma = \{(,)\}$, $V = \{X\}$, und G die Grammatik $G = (\Sigma, V, P, X)$ mit Produktionen

$$P: \quad \begin{array}{l} X \to (\,) \\ X \to (X) \\ X \to XX \end{array}$$

Beachte, dass hier für jede Produktion in G die rechte Seite strikt länger als die linke Seite ist. Entsprechend impliziert $v \to_G v'$, dass $|v'| > |v|$ und dass im Laufe einer Ableitung die Länge strikt zunimmt: $v \to_G^* v'$ für $v' \neq v$ impliziert $|v'| > |v|$.

In Beispiel 1.2.5 hatten wir uns schon überzeugt, dass $X \to_G^* w$ impliziert, dass

$$(*) \left\{ \begin{array}{l} |w|_(= |w|_) \\ \text{und für jeden Präfix } v \preccurlyeq w \text{ gilt: } |v|_(\geqslant |v|_). \end{array} \right.$$

[11] $\to_G^*$ ist die bezüglich $\subseteq$ minimale reflexive und transitive Relation, die $\to_G$ umfasst, der reflexive transitive Abschluss von $\to_G$.

Wir wollen jetzt zeigen, dass auch umgekehrt jedes nicht-leere $w \in \{X, (,)\}^*$ mit der Eigenschaft $(*)$ in G ableitbar ist. Dies zeigen wir durch Induktion über die Länge von w.

Behauptung: Für alle $n \geqslant 1$, und $w \in \{(,), X\}^*$ mit $1 \leqslant |w| \leqslant n$ gilt: $(*) \;\Rightarrow\; (X \rightarrow_G^* w)$.

Induktionsanfang, $n = 1$. Aus $|w| = 1$ und $(*)$ folgt, dass $w = X$, also $X \rightarrow_G^* w$.

Induktionsschritt. Sei $|w| = n + 1$ und die Behauptung gelte für n.
Wir unterscheiden zwei Fälle:

(a) Es gibt einen nicht-leeren echten Anfangsabschnitt u von w mit $|u|_(= |u|_)$. Dann ist $w = uv$ mit $u, v \neq \varepsilon$ und es folgt, dass u und v beide $(*)$ erfüllen. Da $|u|, |v| \leqslant n$, gilt $X \rightarrow_G^* u$ und $X \rightarrow_G^* v$. Daher kann man w ableiten, indem man zunächst $X \rightarrow XX$ verwendet und dann aus dem ersten X gemäß $X \rightarrow_G^* u$ das Teilwort u erzeugt, aus dem zweiten gemäß $X \rightarrow_G^* v$ das Teilwort v.

(b) Für alle nicht-leeren echten Anfangsabschnitte u von w ist $|u|_(> |u|_)$. Es folgt aus $(*)$, dass w mit "(" anfängt und mit ")" aufhört. Also ist $w = (v)$, und die Annahme (b) impliziert, dass $(*)$ auch für v gilt. Nun ist entweder $v = \varepsilon$ und w ableitbar gemäß $X \rightarrow ()$, oder $v \neq \varepsilon$ und demnach $X \rightarrow_G^* v$ nach Induktionsannahme, da $|v| < n$. Dann lässt sich w ableiten, indem man zunächst $X \rightarrow (X)$ anwendet und dann gemäß $X \rightarrow_G^* v$ fortfährt.

Also besteht $L(G)$ genau aus den nicht-leeren Klammerwörtern, die $(*)$ erfüllen. $\square$

Übung 3.1.5 Zeige mit analogen Argumenten, dass die folgende Grammatik G gerade die korrekt geformten arithmetischen Terme über $\Sigma = \{+, \cdot, (,), 0, 1\}$ erzeugt. Dabei benutzen wir die übliche infixe Notation für die 2-stelligen Operationen $+$ und $\cdot$ und klammern jeden Operationsterm außen ab: also z.B. "$(a + b)$" statt "$+(a, b)$".

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

$$
P: \quad \begin{array}{lll}
X & \rightarrow 0 & (1) \\
X & \rightarrow 1 & (2) \\
X & \rightarrow (X + X) & (3) \\
X & \rightarrow (X \cdot X). & (4)
\end{array}
$$

Beispielwörter in $L(G)$: $0, 1, (0 + 0), \dots, ((0 \cdot 1) + 1), \dots$

Man kann die Ableitung eines Wortes in $L(G)$ bzw. eines ableitbaren Wortes $w \in (\Sigma \cup V)^*$ durch eine endliche Kette von direkten Ableitungsschritten veranschaulichen. Z.B. für die Grammatik im letzten Beispiel und das Wort $((0 \cdot 1) + 1) \in L(G)$:

$$\mathbf{X} \rightarrow_G (\mathbf{X} + X) \rightarrow_G ((\mathbf{X} \cdot X) + X) \rightarrow_G ((0 \cdot \mathbf{X}) + X) \rightarrow_G ((0 \cdot 1) + \mathbf{X}) \rightarrow_G ((0 \cdot 1) + 1).$$

$$\mathbf{X} \rightarrow_G (X + \mathbf{X}) \rightarrow_G (\mathbf{X} + 1) \rightarrow_G ((\mathbf{X} \cdot X) + 1) \rightarrow_G ((0 \cdot \mathbf{X}) + 1) \rightarrow_G ((0 \cdot 1) + 1).$$

$$\mathbf{X} \rightarrow_G (X + \mathbf{X}) \rightarrow_G (\mathbf{X} + 1) \rightarrow_G ((X \cdot \mathbf{X}) + 1) \rightarrow_G ((\mathbf{X} \cdot 1) + 1) \rightarrow_G ((0 \cdot 1) + 1).$$

Hier ist jeweils dasjenige X, das als nächstes ersetzt wird, fett markiert. Beachte, dass die Reihenfolge und Auswahl der Ableitungschritte im Allgemeinen nicht eindeutig ist, auch nicht einmal die Länge der Ableitung eines gegebenen Wortes. Man kann die Ableitung eines Wortes auch baumartig darstellen, im Beispiel etwa (annotiert mit den

Produktionen (1) – (4):

$$\begin{tikzcd}
 & & X \arrow[d, "(3)"] & \\
 & & (X+X) \arrow[dl, "(4)"] \arrow[dr, "(2)"] & \\
 & (X \cdot X) \arrow[dl, "(1)"] \arrow[dr, "(2)"] & & 1 \\
0 & & 1 &
\end{tikzcd}$$

Übung 3.1.6 Betrachte $\Sigma := \Sigma_0 \cup \{(,), \emptyset, +, {}^*\}$. Gib eine Grammatik G an, die die regulären Σ_0-Ausdrücke erzeugt: $L(G) = \mathrm{REG}(\Sigma_0)$.

Beispiel 3.1.7 Sei $\mathcal{A} = (\Sigma, Q, q_0, \Delta, A)$ ein NFA. Betrachte die folgende Grammatik $G = G_{\mathcal{A}} := \big(\Sigma, V, P, X_0\big)$, die für jeden Zustand $q \in Q$ eine separate Variable X_q hat:

$$\begin{array}{rcll}
V & := & \big\{X_q \colon q \in Q\big\}, & \\
X_0 & := & X_{q_0}, & \\
P & : & X_q \;\to aX_{q'} & \text{für jedes } (q,a,q') \in \Delta \\
 & & X_q \;\to \varepsilon & \text{für jedes } q \in A.
\end{array}$$

Beachte, dass kein Ableitungsschritt dieser Grammatik die Anzahl der Variablen erhöhen kann. Ableitungsschritte zu Produktionen der Form $X_q \to \varepsilon$ eliminieren die einzig vorhandene Variable, und es kann kein weiterer Ableitungsschritt erfolgen. In jeder Ableitung in G kann demnach höchtens einmal, nämlich im letzten Schritt, eine Produktion dieser Art angewandt werden. Bis auf einen letzten Schritt dieser Art haben alle Ableitungsfolgen in G die Gestalt

$$X_0 \to_G a_1 X_{q_1} \to_G a_1 a_2 X_{q_2} \to_G \;\cdots\; \to_G a_1 \dots a_n X_{q_n},$$

wobei dann $q_0, \dots, q_n$ eine Berechnung von $\mathcal{A}$ auf $w = a_1 \dots a_n$ ist. Also sind in G genau die folgenden $w \in (\Sigma \cup V)^*$ ableitbar:

$$\begin{array}{rl}
w = w_0 X_q\colon & w_0 \in \Sigma^* \text{ derart, dass } \mathcal{A} \text{ eine Berechnung} \\
 & \text{auf } w_0 \text{ hat, die im Zustand } q \text{ endet,} \\[1ex]
w \in \Sigma^*\colon & \mathcal{A} \text{ hat eine akzeptierende Berechnung auf } w.
\end{array}$$

Es folgt, dass $L(G) = L(\mathcal{A})$.

Übung 3.1.8 Für einen NFA $\mathcal{A}$ zeige analog, dass die folgende Variante obiger Grammatik die Sprache $L(\mathcal{A}) \setminus \{\varepsilon\}$ erzeugt. Als Produktionen dienen hierbei

$$\begin{array}{rcll}
P & : & X_q \;\to aX_{q'} & \text{für jedes } (q,a,q') \in \Delta \\
 & & X_q \;\to a & \text{für jedes } (q,a,q') \in \Delta \text{ mit } q' \in A.
\end{array}$$

Korollar 3.1.9 *Jede reguläre Σ-Sprache L wird von einer Grammatik erzeugt. Dabei kann man mit Produktionen der Gestalt $X \to aY$ bzw. $X \to \varepsilon$ für Variablen X, Y und $a \in \Sigma$ auskommen.* [12].

[12]Man nennt solche Produktionen *rechtslinear* und eine Grammatik mit solchen Produktionen auch *reguläre Grammatik*, siehe Abschnitt 3.2.

Als wichtige Beispiel für nicht-reguläre Sprachen, die sich von Grammatiken erzeugen lassen, betrachten wir die Sprachen $\{a^n b^n : n \in \mathbb{N}\}$ (vgl. Beispiele 2.4.8/2.5.3) und $\{a^n b^n c^n : n \in \mathbb{N}\}$.

Übung 3.1.10 Betrachte $\Sigma = \{a, b\}$ und die Grammatik $G = \big(\Sigma, \{X\}, P, X\big)$ mit Produktionen $X \to \varepsilon$ und $X \to aXb$. Zeige, dass

$$L(G) = \{a^n b^n : n \in \mathbb{N}\}.$$

Beispiel 3.1.11 Zu $\Sigma = \{a, b, c\}$ betrachte die Grammatik $G = \big(\Sigma, \{X, Y, Z\}, P, X\big)$ mit folgenden Produktionen

$$\begin{array}{lrl} P: & X & \to \varepsilon \\ & X & \to aXYZ \\ & ZY & \to YZ \\ & aY & \to ab \\ & bY & \to bb \\ & bZ & \to bc \\ & cZ & \to cc \end{array}$$

Behauptung: $L(G) = \{a^n b^n c^n : n \in \mathbb{N}\}$.
Wir weisen dazu folgende Hilfsaussagen nach:

(i) $\{a^n b^n c^n : n \in \mathbb{N}\} \subseteq L(G)$.

(ii) $X \to_G^* w \quad \Rightarrow \quad |w|_a = |w|_b + |w|_Y = |w|_c + |w|_Z$.

(iii) für $w \in \Sigma^*$: $X \to_G^* w \quad \Rightarrow \quad w \in L(a^* b^* c^*)$.

Zu (i): Ableitbarkeit von ε ($n = 0$) ist klar. Für $n \geqslant 1$ wende zunächst n-mal die Produktion $X \to aXYZ$ an, um $a^n X (YZ)^n$ abzuleiten; dann die Produktion $X \to \varepsilon$, um daraus $a^n (YZ)^n$ zu erhalten; dann wiederholt (wie oft genau?) $ZY \to YZ$, um $a^n Y^n Z^n$ zu erhalten; schließlich lassen sich mittels $aY \to ab, bY \to bb, bZ \to bc, cZ \to cc$ sukzessive alle Y in b und alle Z in c verwandeln, und wir erhalten $a^n b^n c^n \in L(G)$.

(ii) folgt per Induktion (über die Anzahl der Ableitungsschritte), da diese Gleichungen für $w = X$ gelten und in jedem direkten Ableitungsschritt $\to_G$ erhalten bleiben.

Zu (iii): Beachte, dass alle a stets einen Anfangsabschnitt jedes ableitbaren Wortes bilden. Weiter können die Variablen Y und Z nur eliminiert werden, indem sie in b bzw. c verwandelt werden, und b und c können nur so gewonnen werden. Das erste Y, das in ein b verwandelt wird, muss ein rechter Nachbar eines a sein, alle weiteren rechte Nachbarn von bereits erzeugten b; ebenso muss das erste Z, das in ein c verwandelt wird, rechter Nachbar eines b sein, alle weiteren rechte Nachbarn von bereits erzeugten c. Es folgt, dass b und c nur von links nach rechts fortschreitend in dieser Ordnung erzeugt werden können.

Aus (ii) und (iii) folgt, dass $L(G) \subseteq \{a^n b^n c^n : n \in \mathbb{N}\}$, und mit (i) also Gleichheit.

Bemerkung 3.1.12 *Die im Beispiel betrachtete Grammatik ist* nicht kontextfrei *(vgl. Definition 3.2.3 unten) im anschaulichen Sinne, dass gewisse Produktionen, wie $bZ \to bc$, die Ersetzung einer Variablen nur in bestimmtem Kontext, hier von Z rechts neben einem b, zulassen.*

Übung 3.1.13 Gib eine Grammatik G an mit $L(G) = \{w \in \{a, b\}^* : |w|_a = |w|_b\}$.

Backus-Naur Form, BNF Zur kompakteren Notation fasst man i.d.R. die Liste der Produktionen einer Grammatik etwas strukturierter zusammen. Insbesondere ist es nützlich, Produktionen mit derselben linken Seite v durch Angabe der verschiedenen möglichen rechten Seiten zusammenzufassen. Dabei trennt man diese Alternativen durch einen senkrechten Strich | (man liest "oder"). Man schreibt gemäß dieser Backus-Naur Konvention dann z.B.

$$v \;\rightarrow\; v'_1 \mid v'_2 \mid \ldots \mid v'_n \qquad \text{anstelle von} \qquad \begin{array}{l} v \rightarrow v'_1 \\ \vdots \\ v \rightarrow v'_n \end{array}$$

wobei anstelle von $\rightarrow$ auch das Zeichen ::= üblich ist.

In erweiterter BNF Form (EBNF) (insbesondere für kontextfreie Grammatiken, s.u.) erlaubt man zusätzlich auch die Abkürzungen

$$X \;\rightarrow\; u[v]w \qquad \text{als Abkürzung für} \qquad X \rightarrow uw \mid uvw$$

mit dem Effekt, dass zwischen u und w in dieser Produktion v eingefügt werden *kann*. Und auch

$$X \;\rightarrow\; u\{v\}w \qquad \text{als Abkürzung für} \qquad \begin{array}{lcl} X & \rightarrow & uZw \mid uw \\ Z & \rightarrow & ZZ \mid v \end{array}$$

für eine *neue* Variable Z, mit dem Effekt, dass mit dieser Produktion $uv'w$ für ein beliebiges $v' \in \{v\}^*$ eingefügt werden darf.

3.2 Die Chomsky-Hierarchie

Zur Klassifikation von Grammatiken und der von ihnen erzeugten Sprachen betrachtet man verschiedene Niveaus, die durch Einschränkungen an die erlaubten Produktionen definiert werden.

Bemerkung: Für die Abgrenzung der Niveaus ist u.a. von Bedeutung, inwieweit Ableitungen die Länge des abgeleiteten Wortes anwachsen lassen. Um zu sehen, dass derartige Kriterien wichtige Auswirkungen haben, betrachte zu einer gegebenen Grammatik $G = (\Sigma, V, P, X_0)$ das zugehörige *Wortproblem* für $L = L(G)$ als Entscheidungsproblem:

Eingabe: $w \in \Sigma^*$ Entscheide, ob $w \in L$

bzw. die erweiterte Variante: Für $w \in (\Sigma \cup V)^*$, entscheide, ob $X_0 \rightarrow^*_G w$.

Wenn für $v \rightarrow_G v'$ immer $|v'| > |v|$ ist, so kann eine Ableitung eines Wortes w höchstens $|w|-1$ Schritte umfassen. Da P endlich ist, lassen sich zu gegebenem w also *alle* Ableitungen einer Länge bis zu $|w|-1$ systematisch erzeugen, und man kann nachprüfen, ob w dabei als Ergebnis einer Ableitung auftritt. Also ist das Wortproblem für eine solche Grammatik entscheidbar. Selbst die schwächere Bedingung, dass für $v \rightarrow_G v'$ immer $|v'| \geqslant |v|$ ist, reicht noch für eine endliche Beschränkung des Suchraums für mögliche Ableitungen von w mit $|w| = n$ aus. (Man überlege sich, warum das so ist.) [13]

[13] Wenn Ableitungsschritte in G dagegen die Länge auch verringern können, kann man den Suchraum für mögliche Ableitungen i.A. nicht a priori einschränken, und tatsächlich ist das Wortproblem für allgemeine Grammatiken i.d.R. unentscheidbar (siehe Abschnitt 3.4.3).

Ein Spezialfall von verkürzenden Produktionen sind ε-*Produktionen*, bei denen die rechte Seite das leere Wort ist (vergleichbar einer Löschoperation). Will man $\varepsilon \in L(G)$ nicht verbieten, so kann man nicht ganz auf ε-Produktionen verzichten (warum?).[14]

Definition 3.2.1 (i) In $G = (\Sigma, V, P, X_0)$ *tritt* X_0 *nur als Startsymbol* auf, wenn X_0 in keiner Produktion von G auf der rechten Seite vorkommt.

(ii) In einer Grammatik $G = (\Sigma, V, P, X_0)$, in der X_0 nur als Startsymbol auftritt, heißt die ε-Produktion $X_0 \to \varepsilon$ eine *harmlose* ε-Produktion.

Harmlose ε-Produktionen sind 'harmlos' i.d.S., dass sie zwar die Erzeugung von ε erlauben, aber nicht zu anderen verkürzenden Ableitungsschritten beitragen.

Beobachtung 3.2.2 Zu jeder Grammatik gibt es eine äquivalente Grammatik (also eine, die dieselbe Sprache erzeugt), in der X_0 nur als Startsymbol auftritt.[15]

Beweis Falls $G = (\Sigma, V, P, X_0)$ nicht wie gewünscht ist, sei X_0' eine neue Variable und $G' := (\Sigma, V \cup \{X_0'\}, P', X_0')$, wo $P' := P \cup \{X_0' \to X_0\}$. □

Definition 3.2.3 Eine Grammatik $G = (\Sigma, V, P, X_0)$ ist vom Typ 3, 2, 1 bzw. 0, wenn ihre Produktionen den folgenden Einschränkungen genügen; dabei sind $X, Y \in V$, $a \in \Sigma$, $v, v' \in (\Sigma \cup V)^*$:

Typ 3 **regulär**	alle Produktionen rechtslinear, d.h. von der Form $X \to \varepsilon$, $X \to a$ oder $X \to aY$.
Typ 2 **kontextfrei**	nur Produktionen von der Form $X \to v$.
Typ 1 **kontextsensitiv**	nur harmlose ε-Produktionen; alle anderen Produktionen nicht verkürzend: $v \to v'$ mit $\lvert v'\rvert \geqslant \lvert v\rvert$.
Typ 0 **allgemein**	keine Einschränkungen.

Siehe Bemerkung 3.1.12 zur Nomenklatur "kontextsensitiv" gegenüber "kontextfrei".

Lemma 3.2.4 *Jede kontextfreie Grammatik ist äquivalent zu einer kontextfreien Grammatik mit allenfalls einer harmlosen ε-Produktion.*

Beweis Sei $G = (\Sigma, V, P, X_0)$ kontextfrei. Falls G keine ε-Produktionen hat, ist nichts zu tun. Andernfalls können wir zunächst annehmen, dass X_0 nur als Startsymbol auftritt (vgl. Beobachtung 3.2.2; die dort angegebene Transformation führt von kontextfreien zu kontextfreien Grammatiken). $V_\varepsilon \subseteq V$ sei die Menge von Variablen Y, für die $Y \to_G^* \varepsilon$ ist. Betrachte eine Produktion der Form $X \to uYv$ mit $Y \in V_\varepsilon$. Wir können die Produktion $X \to uv$ zu G hinzunehmen, ohne dass sich die erzeugte Sprache oder V_ε ändern. Wir erreichen durch Abschluss unter diesen Erweiterungen, dass für jedes Vorkommen einer Variablen $Y \in V_\varepsilon$ auf der rechten Seite einer Produktion auch eine entsprechende Produktion mit Auslassung dieses Vorkommens zu G gehört.

[14] Das Buch von Schöning behandelt ε-Produktionen wie weisse Elefanten; das hat den Nachteil, dass man sich immer wieder auf Ausnahmeregelungen für $\varepsilon \in L$ berufen muss.

[15] Die Transformation erhält Zugehörigkeit zu den Niveaus der Chomsky-Hierarchie (s. unten).

Wenn $X_0 \notin V_\varepsilon$ (d.h. $\varepsilon \notin L(G)$), so können wir in der so entstandenen Grammatik alle ε-Produktionen streichen, ohne dass sich $L(G)$ ändert. Jede Anwendung einer ε-Produktion in einer Ableitungsequenz kann nur eine Variable $Y \in V_\varepsilon$ betreffen. Die Verwendung der ε-Produktion kann dadurch eliminiert werden, dass Y an der entsprechenden Stelle garnicht erst eingeführt wird.

Ist $X_0 \in V_\varepsilon$, so muss lediglich nach Weglassen aller ε-Produktionen die harmlose ε-Produktion $X_0 \to \varepsilon$ wieder hinzugefügt werden. □

Beobachtung 3.2.5 Jede reguläre Grammatik ist kontextfrei; jede kontextfreie Grammatik ist äquivalent zu einer kontextsensitiven Grammatik.

Übung 3.2.6 Zeige, dass jede reguläre Grammatik zu einer regulären Grammatik äquivalent ist, die allenfalls eine harmlose ε-Produktion hat. (Vgl. Übung 3.1.8.)

Satz 3.2.7 *Für Sprachen $L \subseteq \Sigma^*$ sind äquivalent:*

(i) *L regulär (also DFA/NFA erkennbar und durch regulären Ausdruck beschrieben).*

(ii) *L vom Typ 3, d.h. $L = L(G)$ für eine reguläre (Typ 3) Grammatik G.*

Beweis Für (i) $\Rightarrow$ (ii) vergleiche Beispiel 3.1.7 und Übung 3.1.8, wo aus einem NFA $\mathcal{A}$, der L erkennt, reguläre Grammatiken $G_{\mathcal{A}}$ gewonnen werden, die L erzeugen.

Für (ii) $\Rightarrow$ (i) wollen wir umgekehrt zu einer regulären Grammatik $G = (\Sigma, V, P, X_0)$ einen NFA konstruieren, der $L := L(G)$ erkennt. Wir modifizieren G durch Hinzunahme einer neuen Variablen Z wie folgt:

– Ersetze alle Produktionen der Form $X \to a$ durch Produktionen $X \to aZ$

– Füge die neue Produktion $Z \to \varepsilon$ ein.

Die so gewonnene neue reguläre Grammatik G erzeugt dieselbe Sprache, hat aber nur noch Produktionen der Form $X \to aY$ bzw. $X \to \varepsilon$. Wie in Beispiel 3.1.7 findet man, dass demnach jede Ableitung eines Wortes in $w = a_1 \dots a_n \in \Sigma^n$ von der folgenden Form sein muss:

$$X_0 \to_G a_1 X_1 \to_G a_1 a_2 X_2 \to_G \;\cdots\; \to_G a_1 \dots a_n X_n \to_G a_1 \dots a_n.$$

Als Zustandsmenge für $\mathcal{A}$ wählen wir V, X_0 als Anfangszustand. Dann erkennt der folgende NFA $\mathcal{A}$ die Sprache $L(G)$: $\mathcal{A} := \big(\Sigma, V, X_0, \Delta, A\big)$ mit $\Delta = \big\{(X, a, X') \colon X \to aX' \in P\big\}$ und $A = \big\{X \colon X \to \varepsilon \in P\big\}$. □

Wir können also reguläre Sprachen äquivalent auch durch reguläre Grammatiken charakterisieren. (Punkt (iv) in der folgenden Definition ist konsistent mit der bisherigen Definition!)

Definition 3.2.8 Eine Sprache $L \subseteq \Sigma^*$ heißt

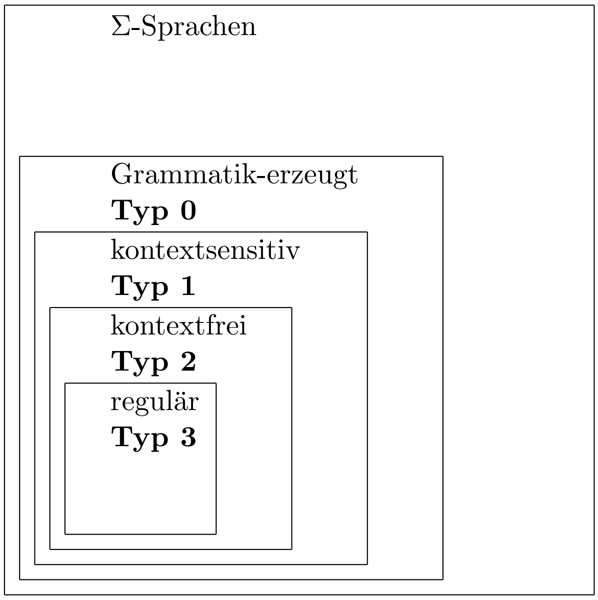
(i) vom Typ 0, wenn es eine Grammatik G gibt mit $L = L(G)$.

(ii) *kontextsensitiv* (Typ 1), wenn es eine Grammatik G vom Typ 1 gibt mit $L = L(G)$.

(iii) *kontextfrei* (Typ 2), wenn es eine Grammatik G vom Typ 2 gibt mit $L = L(G)$.

(iv) *regulär* (Typ 3), wenn es eine Grammatik G vom Typ 3 gibt mit $L = L(G)$.

Jede reguläre Sprache ist auch kontextfrei, jede kontextfreie auch kontextsensitiv, und jede kontextsensitive auch vom Typ 0 (vgl. Beobachtung 3.2.5).



Wir werden sehen, dass die Chomsky-Hierarchie strikt ist in dem Sinne, dass alle diese Inklusionen strikt sind. Bisher wissen wir das nur für (Typ 3) $\subsetneq$ (Typ 2).

3.3 Kontextfreie Sprachen

Wir haben in Beispielen bereits kontextfreie, nicht reguläre Sprachen gesehen, vgl. etwa Beispiele 2.4.8/2.5.3 mit Übung 3.1.10. Kontextfreie Sprachen treten an vielen Stellen in der Informatik auf: Syntax von Programmiersprachen, Funktionstermen, Logiken, usw.

3.3.1 Die Chomsky-Normalform für Typ 2 Grammatiken

Definition 3.3.1 Eine kontextfreie (Typ 2) Grammatik G ohne ε-Produktionen ist in *Chomsky-Normalform*, falls sie nur Produktionen von der Form $X \to YZ$ und $X \to a$ hat ($X, Y, Z \in V, a \in \Sigma$).

Satz 3.3.2 *Jede kontextfreie Grammatik ohne ε-Produktionen ist äquivalent zu einer Grammatik in Chomsky-Normalform. Also wird jede kontextfreie Sprache L mit $\varepsilon \notin L$ von einer Grammatik in Chomsky-Normalform erzeugt.*

Beweis Der Beweis liefert eine effektive Konstruktion der äquivalenten Grammatik in Normalform. Wir beschreiben die Umformung in mehreren Schritten. Sei die gegebene Grammatik $G = (\Sigma, V, P, X_0)$.

1. Schritt: Für jedes $a \in \Sigma$ sei Z_a eine neue Variable, $V' := V \cup \{Z_a \colon a \in \Sigma\}$.
Wir erhalten P' aus P wie folgt:

– In allen Produktionen von G, ersetze jedes Auftreten von $a \in \Sigma$ auf der rechten Seite durch Z_a,
– Füge die Produktionen $Z_a \to a$ hinzu.

Dieser Schritt verändert die erzeugte Sprache nicht, und die neue Grammatik $G' = (\Sigma, V', P', X_0)$ hat nur noch Produktionen der Form $X \to v$, wo v ein nicht-leeres Wort über V' ist, sowie Produktionen der Form $X \to a$ mit $a \in \Sigma$.

2. Schritt: Eliminierung von $X \to Y$ Produktionen.
Wir nehmen an, dass $G = (\Sigma, V, P, X_0)$ bereits die in Schritt 1 erreichte Form hat. Betrachte alle Paare von Variablen $(X, Y) \in V \times V$ mit $X \neq Y$ und $X \to_G^* Y$. Wenn $Y \to v_i$, $i = 1, \dots, k$ alle Produktionen mit linker Seite Y in P sind, für die v_i keine Variable ist, so können wir die Produktionen $X \to v_i$ für $i = 1, \dots, k$ hinzunehmen, ohne dass sich die erzeugte Sprache ändert. Wir tun dies für alle Paare $(X, Y) \in V \times V$ mit $X \neq Y$ und $X \to_G^* Y$. Nun können alle Produktionen der Form $X \to Y$ entfernt werden, ohne dass sich die erzeugte Sprache ändert. Tritt nämlich in einer Ableitungssequenz ein Abschnitt von Variablen-Ersetzungen $X = X_1 \to X_2 \to \dots \to X_n$ auf, die gefolgt ist von einer Ersetzung $X_n \to v$, wobei v keine Variable ist, so haben wir ja statt dessen nun die Produktion $X \to v$ zur Verfügung, die diesen Abschnitt der Ableitungssequenz in einem Schritt simuliert. Also haben wir nun die zu G äquivalente Grammatik $G' = (\Sigma, V', P', X_0)$, deren Produktionen alle von der Form $X \to a$ oder $X \to Y_1 \dots Y_k$ für $k \geqslant 2$ sind.
3. Schritt: Eliminierung von $X \to Y_1 \dots Y_k$ für $k \geqslant 3$.
Eine Produktion $X \to Y_1 \dots Y_k$ mit $k \geqslant 3$ kann äquivalent durch die folgenden beiden Produktionen ersetzt werden:

$$\begin{aligned} X &\to Y_1 \dots Y_{k-2} Z, \\ Z &\to Y_{k-1} Y_k, \end{aligned}$$

wo Z eine neue Variable ist. Durch Iteration dieses Prozesses lassen sich alle Produktionen mit zu langer rechter Seite eliminieren. Die so erzielte Grammatik ist in Chomsky-Normalform. □

Beobachtung 3.3.3 Für eine Chomsky-Normalform Grammatik G ist die Länge einer Ableitung von $w \in L(G)$ stets $2|w| - 1$.

Beweis Betrachte ein Wort $w \in (\Sigma \cup V)^*$ mit $X_0 \to_G^* w$, das in ℓ Ableitungsschritten aus X_0 hervorgeht. Zu w seien $|w|_V$ bzw. $|w|_\Sigma$ die Anzahl der Variablen bzw. Terminalsymbole in w, sodass $|w| = |w|_V + |w|_\Sigma$.

Wir zeigen durch Induktion über die Länge ℓ der Ableitung, dass stets

$$|w|_V + 2|w|_\Sigma = \ell + 1. \qquad (\dagger)$$

Induktionsanfang, $\ell = 0$. Die Behauptung gilt für $w = X_0$.

Induktionsschritt von ℓ nach $\ell + 1$. Entweder der letzte Ableitungsschritt basiert auf einer Ersetzung $X \to YZ$, oder auf einer Ersetzung $X \to a$. Im ersten Fall erhöht sich auf der linken Seite der Gleichung (†) die Variablenzahl um 1, während die Zahl der Terminalsymbole konstant bleibt; im anderen Fall verringert sich die Variablenzahl in (†) um 1, aber die Zahl der Terminalsymbole steigt um 1. In der Bilanz entsprechen beide Fälle dem Zuwachs von 1 auf der rechten Seite in (†).
Für ableitbare Σ-Wörter $w \in L(G)$ ist $|w|_V = 0$, $|w|_\Sigma = |w|$ und also $\ell = 2|w| - 1$. □

Für kontextfreie Grammatiken in Chomsky-Normalform betrachtet man den *Ableitungsbaum*, dessen Knoten die einzelnen Variablen des Ableitungsprozesses sind (die linken Seiten der benutzten Produktionen). In diesem Baum hat jeder *innere* Knoten X, der zur linken Seite einer Produktion $X \to YZ$ gehört, genau zwei Nachfolgerknoten (Y und Z). Ein Knoten X, der zur linken Seite einer Produktion $X \to a$ gehört, ist in diesem Baum ein Blatt (das dann noch durch das Terminalsymbol ersetzt wird). Ein Baum heißt *Binärbaum*, wenn jeder Knoten entweder ein Blatt ist (keine Nachfolger hat) oder genau zwei Nachfolger hat. Die Tiefe eines Baumes ist die Länge des längsten Pfades von der Wurzel zu einem Blatt.

Bemerkung 3.3.4 *Der oben beschriebene Ableitungsbaum zu einer kontextfreien Grammatik in Chomsky-Normalform ist stets ein Binärbaum. Der Ableitungsbaum eines Wortes w mit $|w| \geqslant 2^d$ muss demnach mindestens die Tiefe d haben (vgl. die folgende Übung).*

Übung 3.3.5 Zeige per Induktion, dass in jedem Binärbaum die Anzahl der Blätter durch 2^d beschränkt ist, wo d die Tiefe ist.

Übung 3.3.6 Gewinne aus der Grammatik in Übung 3.1.10 eine Chomsky-Normalform Grammatik für $L = \{a^n b^n : n \in \mathbb{N}\} \setminus \{\varepsilon\} = \{a^n b^n : n \geqslant 1\}$.

3.3.2 Abschlusseigenschaften

Wir haben durch Automaten-Konstruktionen gesehen, dass die Klasse der regulären Sprachen abgeschlossen ist unter sämtlichen Booleschen Operationen, unter Konkatenation und unter der Stern-Operation. Die größere Klasse der kontextfreien Sprachen ist zwar abgeschlossen unter Vereinigung, Konkatenation und Stern, jedoch *nicht* unter Komplement und Durchschnitt. Wir behandeln in diesem Abschnitt die positiven Fälle.

Satz 3.3.7 *Sind $L_1, L_2 \subseteq \Sigma^*$ kontextfrei, so auch $L_1 \cup L_2$ und $L_1 \cdot L_2$. Ist $L \subseteq \Sigma^*$ kontextfrei so auch L^*.*

Beweis Wir geben einfache Konstruktionen für Grammatiken an, die diesen Operationen entsprechen.

Vereinigung. Zu kontextfreien Grammatiken $G^{(i)} = (\Sigma, V^{(i)}, P^{(i)}, X_0^{(i)})$ mit $L(G^{(i)}) = L_i$ für $i = 1, 2$ erhalten wir eine kontextfreie Grammatik $G = (\Sigma, V, P, X_0)$ mit $L(G) = L(G^{(1)}) \cup L(G^{(2)})$ wie folgt. Ohne Beschränkung der Allgemeinheit seien die Variablenmengen disjunkt. Wir wählen eine neue Startvariable $X_0 \notin V^{(i)}$ für G.

$$\begin{array}{rcl} V & := & V^{(1)} \cup V^{(2)} \cup \{X_0\} \\ P & := & \{X_0 \rightarrow X_0^{(1)}, X_0 \rightarrow X_0^{(2)}\} \cup P^{(1)} \cup P^{(1)}. \end{array}$$

Offensichtlich ist G auch wieder kontextfrei, und $L(G) = L(G^{(1)}) \cup L(G^{(2)})$.

Konkatenation. Wir geben zu kontextfreien $G^{(i)} = (\Sigma, V^{(i)}, P^{(i)}, X_0^{(i)})$ mit $L(G^{(i)}) = L_i$ für $i = 1, 2$ eine neue kontextfreie Grammatik $G = (\Sigma, V, P, X_0)$ an mit $L(G) = L(G^{(1)}) \cdot L(G^{(2)})$. Wir nehmen wieder ohne Beschränkung an, dass die Variablenmengen disjunkt sind und wählen eine neue Startvariable $X_0 \notin V^{(i)}$.

$$\begin{array}{rcl} V & := & V^{(1)} \cup V^{(2)} \cup \{X_0\} \\ P & := & \{X_0 \rightarrow X_0^{(1)} X_0^{(2)}\} \ \cup \ P^{(1)} \ \cup \ P^{(2)}. \end{array}$$

Offensichtlich ist G auch wieder kontextfrei, und $L(G) = L(G^{(1)}) \cdot L(G^{(2)})$.

Stern-Operation. Wir geben zu einer kontextfreien Grammatik $G = (\Sigma, V, P, X_0)$ eine kontextfreie Grammatik $G' = (\Sigma, V', P', X_0')$ an mit $L(G') = L(G)^*$. Wir wählen eine neue Startvariable $X_0' \notin V$.

$$\begin{array}{rcl} V' & := & V \cup \{X_0'\} \\ P' & := & \{X_0' \rightarrow \varepsilon \ , \ X_0' \rightarrow X_0' X_0\} \ \cup \ P. \end{array}$$

G' ist auch wieder kontextfrei, und $L(G') = L(G)^*$. □

3.3.3 Nicht-kontextfreie Sprachen: Pumping Lemma

Um zu nachzuweisen, dass eine vorgelegte Sprache nicht kontextfrei ist, brauchen wir ein gutes notwendiges Kriterium für Kontextfreiheit. Analog zur Situation bei den regulären Sprachen gibt es eine Bedingung, die besagt, dass zu jeder kontextfreien Sprache ab einer gewissen Wortlänge bestimmte Abschnitte beliebig oft wiederholt werden können. In Analogie zum entsprechenden Sachverhalt für reguläre Sprachen und endliche Automaten spricht man auch hier von einem “pumping lemma”.

Satz 3.3.8 (Pumping Lemma) *Für jede kontextfreie Sprache $L \subseteq \Sigma^*$ gibt es ein $n \in \mathbb{N}$, sodass sich jedes $x \in L$ mit $|x| \geqslant n$ zerlegen lässt als $x = y \cdot u \cdot v \cdot w \cdot z$, wobei $uw \neq \varepsilon$, sodass für alle $m \in \mathbb{N}$*

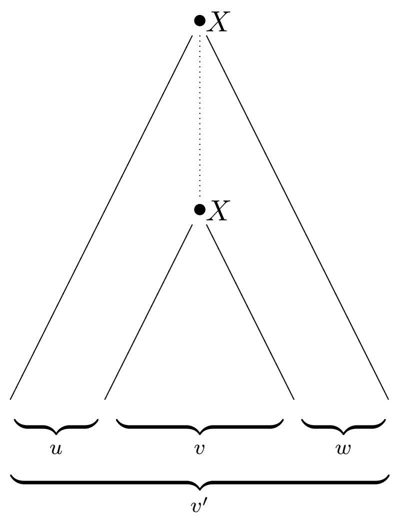
$$y \cdot u^m \cdot v \cdot w^m \cdot z = y \cdot \underbrace{u \cdots u}_{m\ mal} \cdot v \cdot \underbrace{w \cdots w}_{m\ mal} \cdot z \in L.$$

Man kann dabei u, v, w so wählen, dass $|uvw| \leqslant n$.

Beweis Sei $G = (\Sigma, V, P, X_0)$ eine kontextfreie Grammatik in Chomsky-Normalform für $L \setminus \{\varepsilon\}$. Wir setzen $n := 2^{|V|}$ (beachte: $|V|$ ist die Anzahl der Variablen in G).

Nach Bemerkung 3.3.4 hat der Ableitungsbaum eines Wortes der Länge $n = 2^{|V|}$ mindestens die Tiefe $|V|$. Demnach muss längs eines solchen Pfades dieselbe Variable X zweimal als linke Seite einer verwendeten Produktion $X \to \ldots$ auftreten.

Sei $X \in V$ eine Variable, die auf einem Pfad des Ableitungsbaumes von x zweimal vorkommt. Wir betrachten die Teilwörter von x, die von den Teilbäumen erzeugt werden, die von dem oberen bzw. unteren Vorkommen von X ausgehen (siehe Skizze).



Seien v' und v die Teilwörter von x, die vom oberen X bzw. vom unteren X aus abgeleitet werden. Da das untere X im Ableitungsbaum des oberen X enthalten ist, ist v in v' als echtes Teilwort enthalten und $v' = uvw$ für geeignete u und w, die nicht beide leer sein können. $v' = uvw$ ist seinerseits ein Teilwort von x, also $x = yuvwz$ für geeignete y, z.

Es bleibt zu zeigen, dass auch alle yu^mvw^mz für $m \neq 1$ ableitbar sind.

Für $m = 0$ ersetzt man den Teilbaum des oberen X durch denjenigen des unteren, und leitet so anstelle von $yv'z$ nun $yvz = yu^0vw^0z$ ab.

Für $m = 2$ ersetzt man den Teilbaum des unteren X durch eine Kopie des oberen, und leitet so anstelle von $yuvwz$ nun $yuv'wz = yu^2vw^2z$ ab. Man kann nun diese Ersetzung von v durch v' wiederholt anwenden und erhält so (induktiv) Ableitungsbäume für alle $yu^m vw^m z$ wie behauptet.

Um zu erreichen, dass auch noch $|v'| = |uvw| \leqslant n$, wählt man einen Teilbaum minimaler Höhe, derart dass die Variable an der Wurzel des Teilbaums im Inneren des Teilbaumes auftritt. Aus der Größenabschätzung oben folgt, dass dann das von diesem Teilbaum erzeugte Wort eine Länge $\leqslant n$ hat. $\square$

Mit Hilfe des Pumping Lemmas können wir nachweisen, dass bestimmte Sprachen nicht kontextfrei sind.

Beispiel 3.3.9 Die Sprache $L = \{a^n b^n c^n : n \in \mathbb{N}\}$ ist nicht kontextfrei.

Annahme: L ist kontextfrei. Sei dann n wie im Pumping Lemma zu L. Betrachte $x = a^n b^n c^n \in L$ für dieses n. Seien y, u, v, w, z zu x wie im Pumping Lemma: $x = a^n b^n c^n = yuvwz \in L$. Da $|uvw| \leqslant n$, kann das Teilwort uvw von $x = a^n b^n c^n$ höchstens zwei der drei Buchstaben a, b, c enthalten, mindestens einer, a oder c kommt in uvw nicht vor. Daraus folgt aber, dass sich in den $yu^m vw^m z$ für $m > 1$ die Anzahl dieses fehlenden Buchstabens nicht erhöht, während aber die Länge anwächst. Das ist in L aber unmöglich.

Also erhalten wir einen Widerspruch zur Annahme, dass L kontextfrei sei; L ist nicht kontextfrei.

Wir haben in Beispiel 3.1.11 gesehen, dass L von einer fast schon kontextsensitiven Grammatik G erzeugt wird. Man muss lediglich die nicht harmlose ε-Produktion $X \to \varepsilon$ eliminieren. Einführen eines neuen Startsymbols $X_0 \neq X, Y, Z$ und Ersetzung der ersten beiden Produktionen in G durch $X_0 \to \varepsilon \mid X$ und $X \to aXYZ \mid aYZ$ liefert eine kontextsensitive Grammatik für L.

Damit haben wir das folgende Korollar.

Korollar 3.3.10 *Die Klasse der kontextfreien Sprachen ist eine echte Teilklasse der Klasse der kontextsensitiven Sprachen.*

Wir können nun auch sehen, dass die Klasse der kontextfreien Sprachen nicht unter Durchschnitt (und demnach auch nicht unter Komplement) abgeschlossen ist. Beachte, dass Abschluss unter Komplement den Abschluss unter Durchschnitt implizieren würde, da wir nach Satz 3.3.7 Abschluss unter Vereinigung haben: $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.

Betrachte die kontextfreien Sprachen

$$L_1 = \{a^n b^n : n \in \mathbb{N}\} \cdot \{c\}^*,$$
$$L_2 = \{a\}^* \cdot \{b^n c^n : n \in \mathbb{N}\}.$$

(Vgl. Übungen 3.1.10 und 3.3.6 und Abschluss unter Konkatenation, Satz 3.3.7.) Wir haben gerade gesehen, dass $L_1 \cap L_2 = \{a^n b^n c^n : n \in \mathbb{N}\}$ nicht kontextfrei ist.

Korollar 3.3.11 *Die Klasse der kontextfreien Sprachen ist weder unter Komplement noch unter Durchschnitt abgeschlossen.*

Beispiel 3.3.12 Weitere Beispiele von nicht kontextfreien Sprachen über dem einelementigen Alphabet $\Sigma = \{|\}$ (eine Σ-Sprache L kann man als unäre Darstellung einer Teilmenge $\{n \in \mathbb{N} : |^n \in L\}$ von $\mathbb{N}$ auffassen):

(i) $\{|^n : n = m^2, m \in \mathbb{N}\}$ (Quadratzahlen).

(ii) $\{|^n : n = 2^m, m \in \mathbb{N}\}$ (Zweierpotenzen).

In beiden Fällen kann man das Pumping Lemma anwenden, um zu sehen, dass es sich nicht um kontextfreie Sprachen handelt.

3.3.4 Der CYK Algorithmus

Für eine kontextfreie Sprache, die von einer Grammatik in Chomsky-Normalform erzeugt wird, kann man aus dieser Grammatik einen Algorithmus gewinnen, der das Wortproblem (Zugehörigkeit von Wörtern zur Sprache L) effizient entscheidet. Nach seinen Autoren Cocke, Younger und Kasami wird dieser Algorithmus CYK Algorithmus genannt. [16]

Wir betrachten hier das zugrundeliegende rekursive Kriterium für Ableitbarkeit eines Wortes in einer Chomsky-Normalform Grammatik.

Lemma 3.3.13 *Sei $G = (\Sigma, V, P, X_0)$ eine Typ 2 Grammatik in Chomsky-Normalform. Für $w = a_1 \dots a_\ell \in \Sigma^\ell$ und $1 \leqslant i \leqslant j \leqslant \ell$ sei $u_{i,j}$ das Teilwort $u_{i,j} = a_i \dots a_j$. Dann gilt für alle $1 \leqslant i \leqslant j \leqslant \ell$ und $X \in V$:*

$$(*) \qquad X \to_G^* u_{i,j} \quad \text{gdw.} \quad \begin{array}{l} i = j \text{ und } (X \to a_i) \in P \text{ oder} \\[4pt] i < j \text{ und für ein } k \text{ mit } i \leqslant k < j \text{ und} \\ \text{eine Produktion } X \to YZ \text{ in } G \text{ ist} \\ Y \to_G^* u_{i,k} \text{ und } Z \to_G^* u_{k+1,j}. \end{array}$$

Beweis Sei $n = j - i + 1$. Ist $n = 1$, so ist $i = j$ und $u_{i,j} = a_i$ ein einzelner Buchstabe. $X \to_G^* a_i$ kann dann nur durch eine Produktion $X \to_G a_i$ sein (in einer Chomsky-Normalform Grammatik sind nur Produktionen der Form $X \to a$ nicht verlängernd).

Ist $n > 1$, so muss der erste Schritt in einer Ableitungssequenz $X \to_G^* u_{i,j}$ eine Produktion der Form $X \to YZ$ benutzen, und $u_{i,j}$ hat die Gestalt $u_{i,j} = vw$, wobei $Y \to_G^* v$ und $Z \to_G^* w$. $\square$

Anhand von $(*)$ kann man, systematisch von kürzeren zu längeren Teilworten aufsteigend, für alle $u_{i,j}$ die Menge derjenigen Variablen X bestimmen, für die $X \to_G^* u_{i,j}$ ist. Für $i = 1$ und $j = |u|$ erhält man so die Antwort auf die Frage, ob $X_0 \to_G^* u$, d.h. ob $u \in L(G)$ ist.

Beispiel 3.3.14 Wir wollen wissen, ob das Wort *bacb* von der Grammatik mit den folgenden Produktionen aus der Startvariablen X erzeugt wird:

$$\begin{array}{rcl@{\qquad}rcl} X & \to & XY \mid ZZ \mid a & Z & \to & XY \mid b \\ Y & \to & WZ \mid a & W & \to & YW \mid c \end{array}$$

Teilworte der Länge 1 sind jeweils von den folgenden Variablen erzeugbar:

$$\begin{array}{rl} a: & X, Y \\ b: & Z \\ c: & W \end{array}$$

Daraus findet man mit $(*)$ für Teilworte der Länge 2:

$$\begin{array}{rll} ac: & W & (W \to YW) \\ ba: & - & (\text{keine rechten Seiten } ZX, ZY) \\ cb: & Y & (Y \to WZ) \end{array}$$

Das Teilwort *bac* ist aus keiner Variablen erzeugbar, weder über die Zerlegung $b \cdot ac$ (keine rechte Seite ZW) noch über die Zerlegung $ba \cdot c$. Das Teilwort *acb* dagegen ist aus X, Y und Z erzeugbar (als $a \cdot cb$ über $X \to XY$ oder $Z \to XY$; als $ac \cdot b$ über $Y \to WZ$). Schließlich findet man, dass *bacb* als $b \cdot acb$ über $X \to ZZ$ aus X ableitbar ist.

[16] Das Verfahren folgt der Idee der “dynamischen Programmierung”, bei dem sukzessive Information zu Teilinstanzen tabelliert wird, um in rekursiver Weise zur Lösung der vorgelegten Instanz zu kommen.

3.4 Berechnungsmodelle

Die regulären Sprachen konnten wir algorithmisch durch das Berechnungsmodell der endlichen Automaten (DFA oder NFA) charakterisieren: Satz von Kleene, Satz 2.3.1. Das Entscheidungsproblem (*Wortproblem*) zu $L \subseteq \Sigma^*$,

Eingabe: $w \in \Sigma^*$ Entscheide, ob $w \in L$

ist für reguläre Sprachen L durch DFA (oder NFA) algorithmisch lösbar, nicht jedoch für nicht-reguläre Sprachen. Dabei war die entscheidende Einschränkung, dass endliche Automaten keinen 'dynamischen Speicher' zur Verfügung haben. In einer Berechnung eines DFA oder NFA über einem Eingabewort w ist die aktuelle Information allein in der aktuellen Position über dem Eingabewort und dem aktuellen Zustand enthalten. Schon für einfache kontextfreie Sprachen wie z.B. $L = \{a^n b^n : n \in \mathbb{N}\}$ reicht ein so eingeschränktes Berechnungsmodell nicht aus (vgl. Beispiel 2.5.3).

Für kontextfreie Sprachen (Typ 2) leisten nichtdeterministische *Kellerautomaten* (PDA: *pushdown automata*) das gewünschte (Abschnitt 3.4.1). Kellerautomaten verfügen über einen dynamischen Speicher, der jedoch als *Kellerspeicher* (englisch: *pushdown stack*) nur eingeschränkten Zugriff erlaubt.

Mit Turingmaschinen (Abschnitt 3.4.2) erreicht man ein allgemein akzeptiertes universelles Berechnungsmodell. Wir werden insbesondere sehen, dass für Typ 0 Sprachen das Wortproblem im allgemeinen nicht algorithmisch lösbar ist (Abschnitt 3.4.3).

3.4.1 Kellerautomaten und kontextfreie Sprachen

Wir betrachten nichtdeterministische Kellerautomaten und zeigen, dass sie genau die kontextfreien Sprachen erkennen. PDA verallgemeinern die nichtdeterministischen endlichen Automaten (NFA) dadurch, dass sie zusätzlich über einen Kellerspeicher verfügen. Ein Kellerspeicher hat als aktuellen Inhalt ein Wort über einem separaten Kelleralphabet (dieses Arbeitsalphabet Γ ist i.d.R. vom Eingabealphabet Σ verschieden), das in jedem Berechnungsschritt *vorne* dadurch modifiziert werden kann, dass das erste Zeichen gelesen und durch einen neuen Anfangsabschnitt ersetzt werden kann wird ('pop' und 'push' Operationen auf einem 'stack'). Dieser einseitige Zugriff macht die wesentliche Beschränkung des PDA Modells aus und passt gerade auf die Erfordernisse kontextfreier Sprachen.

Definition 3.4.1 [PDA] Ein *nichtdeterministischer Kellerautomat* ist spezifiziert als

$$\mathcal{P} = \big(\Sigma, Q, q_0, \Delta, A, \Gamma, \#\big).$$

Dabei ist	Σ	das Eingabealphabet
	Γ	das *Kelleralphabet*
	$\# \in \Gamma$	das *Anfangs-Kellersymbol*
	Q	die endliche Zustandsmenge
	$q_0 \in Q$	der Anfangszustand
	$A \subseteq Q$	die Menge der akzeptierenden Zustände
	$\Delta \subseteq Q \times \Gamma \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^* \times Q$	die endliche Übergangsrelation.

Eine *Berechnung eines PDA* $\mathcal{P}$ auf einer Eingabe $w = a_1 \dots a_n \in \Sigma^*$ wird als Folge von Konfigurationen beschrieben. Eine *Konfiguration* ist eine Beschreibung des aktuellen

Zustandes der Berechnung. Die Konfiguration $C = (q, v, \alpha) \in Q \times \Sigma^* \times \Gamma^*$ besteht aus folgenden Komponenten

$q \in Q$	aktueller Zustand,
$v \in \Sigma^*$	Restabschnitt des Eingabewortes,
$\alpha \in \Gamma^*$	aktueller Kellerinhalt.

Die *Startkonfiguration* auf Eingabe w ist

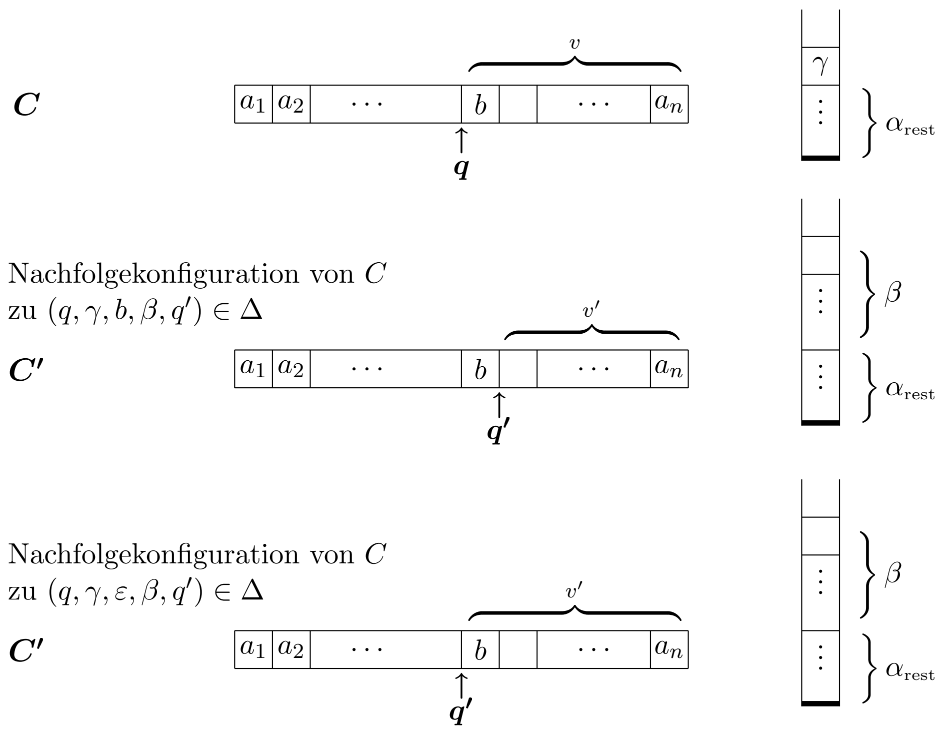
$$C_0[w] = (q_0, w, \#),$$

mit Startzustand, vollem Eingabewort und Anfangs-Kellersymbol #.

Zu einer Konfiguration $C = (q, v, \alpha)$ mit nicht-leerem Kellerinhalt $\alpha = \gamma\,\alpha_{\mathrm{rest}}$ ($\gamma \in \Gamma$ das oberste Kellersymbol) sind die möglichen *Nachfolgekonfigurationen* gerade die Konfigurationen $C' = (q', v', \alpha')$ mit

$$\left.\begin{array}{r} v = xv' \\ \alpha' = \beta\,\alpha_{\mathrm{rest}} \end{array}\right\} \text{ für ein } (q, \gamma, x, \beta, q') \in \Delta.$$

D.h., eine Transition $(q, \gamma, x, \beta, q') \in \Delta$ erlaubt es dem PDA $\mathcal{P}$, in einem Berechnungsschritt das oberste Kellersymbol γ durch das Wort β zu ersetzen, den Zustand von q nach q' zu wechseln und – im Falle $x \neq \varepsilon$ – auf der Eingabe um den Buchstaben x nach rechts weiterzurücken. Beachte, dass für $x = \varepsilon$ die Modifikation des Kellerinhaltes und der Zustandswechsel stattfinden, ohne dass der nächste Eingabebuchstabe verarbeitet wird; man spricht von einer *ε-Transition*. Eine Transition $(q, \gamma, x, \beta, q') \in \Delta$ ist nur dann anwendbar, wenn der Zustand q ist, das oberste Kellersymbol γ ist, und im Fall $x = a \in \Sigma$ auch der nächste Eingabebuchstabe ein a ist.



Eine Berechnung des PDA $\mathcal{P}$ auf dem Eingabewort $w \in \Sigma^*$ ist eine Konfigurationsfolge $C_0 \dots C_f$, beginnend mit $C_0 = C_0[w] = (q_0, w, \#)$, derart, dass jeweils C_{i+1} eine

Nachfolgekonfiguration von C_i ist, und dass in der *Endkonfiguration* C_f keine weitere Transition anwendbar ist. Eine Berechnung auf w ist *akzeptierend*, wenn sie das gesamte Eingabewort abarbeitet (leeres Restwort), mit *leerem Keller* endet (leeres Kellerwort), und in einem akzeptierenden Zustand $q \in A$ endet. D.h., dass die Endkonfiguration von der Form $C_f = (q, \varepsilon, \varepsilon)$ mit $q \in A$ ist. [17]

Wir schreiben $C_0[w] \xrightarrow{\mathcal{P}} C_f$, wenn es eine Berechnung von $\mathcal{P}$ auf w gibt, die in der Endkonfiguration C_f endet. Beachte, dass es verschiedene Berechnungen auf derselben Eingabe geben kann (Nichtdeterminismus), und dass es sowohl vorzeitig abbrechende Berechnungen (deadlock) geben kann als auch unendlich fortlaufende, nicht-abbrechende Berechnungen, die keine Endkonfiguration erreichen (Divergenz).

$\mathcal{P}$ hat also eine akzeptierende Berechnung auf w genau dann, wenn $C_0[w] \xrightarrow{\mathcal{P}} C_f$ für eine Endkonfiguration $C_f = (q, \varepsilon, \varepsilon)$ mit $q \in A$.

Definition 3.4.2 Die *vom PDA $\mathcal{P}$ akzeptierte Sprache* oder *erkannte Sprache* ist

$$L(\mathcal{P}) = \big\{ w \in \Sigma^* \colon C_0[w] \xrightarrow{\mathcal{P}} (q, \varepsilon, \varepsilon) \text{ für ein } q \in A \big\}.$$

Das folgende Beispiel zeigt, wie man den Keller als unbeschränkten Speicher benutzen kann – und damit auch Sprachen erkennen kann, die nicht regulär sind.

Beispiel 3.4.3 Der folgende Kellerautomat $\mathcal{P}$ akzeptiert die nicht-reguläre Sprache der korrekt geschachtelten Klammerausdrücke über $\Sigma = \{(,)\}$ (vgl. Beispiele 1.2.5 und Übung 2.5.5).

$$\mathcal{P} = \big(\{(,)\}, Q, q_0, \Delta, \{q_0\}, \Gamma, \#\big),$$

mit nur einem Zustand $q = q_0$, Kelleralphabet $\Gamma = \{|, \#\}$ und folgenden Transitionen:

$(q\,, \#\,, (\,, \|\#\,, q)$	verarbeitet "(" und addiert "|" im Keller
$(q\,, \|\,, (\,, \|\|\,, q)$	verarbeitet "(" und addiert "|" im Keller
$(q\,, \|\,,)\,, \varepsilon\,, q)$	verarbeitet ")" und eliminiert ein "|" im Keller
$(q\,, \#\,, \varepsilon\,, \varepsilon\,, q)$	ε-Transition, die # löscht

Hier fungiert ein Kellerinhalt $\alpha = |^k\#$ als Zähler für die noch zu schließenden Klammern: k ist der "("-Überschuss im bisher abgearbeiteten Wort. Man zeigt (durch Induktion über die Anzahl der Berechnungsschritte), dass genau diejenigen Konfigurationen $C = (q, v, |^k\#)$ in Berechnungen von $\mathcal{P}$ auf w vorkommen, für die $w = uv$ und $|u|_(- |u|_) = k$. Daraus folgt dann, dass $\mathcal{P}$ genau die richtigen Klammerwörter akzeptiert.

Übung 3.4.4 Konstruiere PDA für $\{a^n b^n \colon n \in \mathbb{N}\}$ und für PALINDROM (vgl. Beispiele. 2.5.3 und 2.4.8, bzw. Übung 2.5.4).

Satz 3.4.5 *Für $L \subseteq \Sigma^*$ sind äquivalent:*

(i) *L ist kontextfrei.*

(ii) *$L = L(\mathcal{P})$ für einen PDA $\mathcal{P}$.*

[17] Man spricht von "akzeptieren mit leerem Keller"; tatsächlich kann man auf die Auszeichnung akzeptierender Zustände verzichten, d.h. $A = Q$ vorschreiben, und erhält ein genauso starkes Berechnungsmodell.

Beweis Wir skizzieren die Beweisidee.

Für (i) $\Rightarrow$ (ii) sei $L = L(G)$ für eine kontextfreie Grammatik $G = (\Sigma, V, P, X_0)$. Wir nehmen an, dass $V \cap \Sigma = \emptyset$ ist. Betrachte den PDA $\mathcal{P} = (\Sigma, Q, q_0, \Delta, A, \Gamma, \#)$ mit einem einzigen Zustand q, $Q = A = \{q\}$, Kelleralphabet $\Gamma = V \cup \Sigma$ mit Anfangskellersymbol $\# = X_0$ und den folgenden Transitionen:

$$\begin{array}{ll} (q, X, \varepsilon, \alpha, q) & \text{für jede Produktion } X \to \alpha \text{ von } G, \\ (q, a, a, \varepsilon, q) & \text{für jedes } a \in \Sigma. \end{array}$$

Dann sind die Konfigurationen, die $\mathcal{P}$ in Berechnungen über $w \in \Sigma^*$ annehmen kann, gerade die $C = (q, v, \alpha)$ mit $w = uv$ und $X_0 \to_G^* u\alpha$ (Induktionsbeweis!). Es folgt, dass $\mathcal{P}$ gerade die w mit $X_0 \to_G^* w$, d.h. genau die $w \in L(G)$, akzeptiert.

Für (ii) $\Rightarrow$ (i) sei $L = L(\mathcal{P})$ für einen PDA $\mathcal{P} = (\Sigma, Q, q_0, \Delta, \Gamma, \#)$. Wir geben eine kontextfreie Grammatik $G = (\Sigma, V, P, X_0)$ an, deren Ableitungen zu den Berechnungen von $\mathcal{P}$ korrespondieren.

Sei $V := (Q \times \Gamma \times Q) \cup \{X_0\}$. Hierbei steht eine Variable (q, γ, q') für ein Kellersymbol γ zusammen mit "geratenen" Zuständen q und q'. Idee: q gibt zur Position γ in einem Kellerinhalt $\alpha_0\gamma\alpha_1$ an, in welchem Zustand $\mathcal{P}$ (nach Abarbeiten von α_0) γ bearbeitet; q' gibt an, in welchem Zustand $\mathcal{P}$ später, nach Abarbeiten des inzwischen neu generierten Kellerinhaltes, in den Abschnitt α_1 eintritt. (Äquivalent: $\mathcal{P}$ in Zustand q mit Kellerinhalt γ gestartet, kann eine Konfiguration mit leerem Keller im Zustand q' erreichen.)

Die Produktionen von G (jeweils für $q, q', q_i \in Q; \gamma, \gamma_i \in \Gamma; x \in \Sigma \cup \{\varepsilon\}$):

$$\begin{array}{rll} X_0 & \to (q_0, \#, q) & \text{für } q \in A \\ (q, \gamma, q') & \to x & \text{für } (q, \gamma, x, \varepsilon, q') \in \Delta, \\ (q, \gamma, q') & \to x(q_1, \gamma_1, q') & \text{für } (q, \gamma, x, \gamma_1, q_1) \in \Delta, \\ (q, \gamma, q') & \to x(q_1, \gamma_1, q_2)(q_2, \gamma_2, q_3) \dots (q_\ell, \gamma_\ell, q') & \text{für } (q, \gamma, x, \gamma_1 \dots \gamma_\ell, q_1) \in \Delta. \end{array}$$

Man kann nun (per Induktion) zeigen, dass für $u \in \Sigma^*$ gilt

$$(q, \gamma, q') \to_G^* u \quad \text{gdw.} \quad (q, u, \gamma) \xrightarrow{\mathcal{P}} (q', \varepsilon, \varepsilon).$$

Daraus ergibt sich, dass $L(G) = L(\mathcal{P})$. □

3.4.2 Turingmaschinen: ein universelles Berechnungsmodell

Turingmaschinen liefern ein *universelles Berechnungsmodell* und damit einen allgemeinen abstrakten Begriff der Berechenbarkeit und Entscheidbarkeit. Im Gegensatz zu den bisher behandelten Berechnungsmodellen (endliche Automaten und Kellerautomaten) sollen Turingmaschinen keiner prinzipiellen Einschränkung unterworfen sein ausser, dass ihre Berechnungen unserer (informellen) Vorstellung eines algorithmischen Verfahrens gehorchen sollen. Dabei werden in absichtlicher Idealisierung alle quantitativen Beschränkungen ausserachtgelassen. Das Ziel ist eine Charakterisierung dessen, was *prinzipiell berechenbar* ist. Dadurch erkennen wir z.B. die Grenze zwischen algorithmisch lösbaren und algorithmisch unlösbaren Entscheidungsproblemen. Turingmaschinen sollen also *jeden denkbaren Algorithmus* realisieren (implementieren). Sie erfassen daher insbesondere alles, was reale Computern realisieren. In diesem Sinne ist das auf Alan Turing [1912-1954] zurückgehende Konzept *die* anerkannte Charakterisierung der prinzipiellen Möglichkeiten und Grenzen von algorithmischen Problemlösungen. Die These, dass man mit Turingmaschinen tatsächlich alle erdenklichen Algorithmen erfasst (Churchsche

These oder Church-Turing These, s.u.) wird empirisch durch die Erfahrung mit allen akzeptierten algorithmischen Methoden bestätigt und theoretisch dadurch gestützt, dass eine ganze Reihe alternativer Modellbildungen trotz ganz unterschiedlicher Ansätze stets zu äquivalenten Charakterisierungen geführt haben.

Turingmaschinen realisieren inbesondere folgende Anforderungen als fundamentale Kriterien an algorithmische Verfahren: *Sequentialität* und *Definitheit* (schrittweises Abarbeiten von eindeutigen Einzelanweisungen), *endliche Kontrolle/Steuerung* (logische Abfolge der Einzelschritte muss endlich beschrieben sein) und *Lokalität* (lokaler Zugriff auf Daten in Einzelschritten).

Endlichkeit der Beschreibung wird bei Turingmaschinen, wie bei DFA, durch die Beschreibung der Einzelschritte mittels einer endlichen Zustandsmenge und einer Transitionsfunktion realisiert. Im Gegensatz zu DFA verfügen Turingmaschinen aber über einen unbeschränkten Arbeitsspeicher, den sie (im Gegensatz zu PDA) an jeder Stelle lesen und modifizieren können (allerdings nicht global, sondern nur lokal und in schrittweiser Ansteuerung).

Der Arbeitsspeicher wird durch ein unendliches *Band* modelliert, auf dessen Zellen ein *Lese/Schreib-Kopf* zugreift. Genauer besteht das Band aus einer beidseitig unendlichen Folge von Zellen, die jeweils ein Symbol des Alphabets Σ oder leer sind. Wir schreiben $\Box$ als Symbol für leeren Zelleninhalt. In jeder Konfiguration während einer Berechnung ist stets nur ein endlicher Teil des Bandes beschrieben. Die nach links und rechts unendliche Fortschreibung des Bandes (mit Inhalt $\Box$) vermeidet lediglich eine feste a priori Begrenzung des Speicherplatzes, d.h. der verwendete Platz kann im Laufe der Berechnung dynamisch erweitert werden. Der Lese/Schreib-Kopf befindet sich jeweils über genau einer Zelle, kann deren Inhalt lesen und überschreiben und sich schrittweise nach links oder rechts weiterbewegen. Die Übergangsfunktion legt fest, welche Operation der Lese/Schreibkopf in einem Berechnungsschritt ausführt – in Abhängigkeit vom Kontrollzustand und vom aktuell gelesenen Zelleninhalt.

Wir beschränken uns im folgenden auf *deterministische Turingmaschinen* (DTM), und auf ein Format für Entscheidungsprobleme ("ja"/"nein" Ausgabe, im Gegensatz zu Berechnungsproblemen mit komplexerer Ausgabe). Entsprechende Verallgemeinerungen sind leicht zu realisieren.

Wir fixieren ein Alphabet Σ und betrachten als Entscheidungsproblem das Wortproblem zu einer Sprache $L \subseteq \Sigma^*$. (Jedes andere Entscheidungsproblem lässt sich als Wortproblem zu einer geeigneten Sprache kodieren.) Auf Eingabe $w \in \Sigma^*$ soll also eine Antwort "ja" oder "nein" (akzeptieren oder verwerfen) erfolgen, je nachdem ob $w \in L$ ist. Dazu werden in der Zustandsmenge Q neben dem Anfangszustand $q_0 \in Q$ zwei voneinander verschiedene *Endzustände* $q^+, q^- \in Q$ ausgezeichnet.

Definition 3.4.6 [DTM]
Eine deterministische Turingmaschine über dem Alphabet Σ ist spezifiziert als

$$\mathcal{M} = \big(\Sigma, Q, q_0, \delta, q^+, q^-\big).$$

Dabei ist	Q	die endliche *Zustandsmenge,*
	$q_0 \in Q$	der *Anfangszustand,*
	$q^+ \in Q$	der *akzeptierende Endzustand,*
	$q^- \in Q$	der *verwerfende Endzustand,* $q^- \neq q^+$,
	δ	die *Übergangsfunktion.*

Die Übergangsfunktion ist von der Form

$$\delta\colon Q \times (\Sigma \cup \{\Box\}) \rightarrow (\Sigma \cup \{\Box\}) \times \{<, \circ, >\} \times Q$$

und beschreibt, in Abhängigkeit vom Kontrollzustand $q \in Q$ und vom aktuell gelesenen Zelleninhalt $x \in \Sigma \cup \{\Box\}$, als Tripel $\delta(q,x) = (x', d, q')$, welches Symbol $x' \in \Sigma \cup \{\Box\}$ anstelle von x in die aktuelle Zelle geschrieben wird, wie sich der Kopf bewegt ($<$: eine Zelle nach links; $>$: eine Zelle nach rechts; $\circ$: stehenbleiben) und den Nachfolgezustand $q' \in Q$.

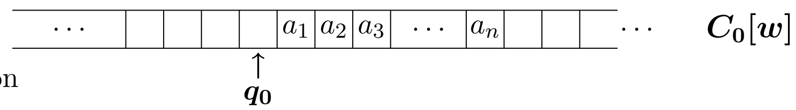
Bemerkung: Für eine Turingmaschine, die auf Eingabewörtern über dem Alphabet Σ arbeiten soll, kann man auch zusätzliche Zeichen aus einem erweiterten *Arbeitsalphabet* $\Gamma \supseteq \Sigma$ auf dem Band zulassen (z.B. Trennsymbole, Endmarker u. dgl.). Dadurch wird häufig eine übersichtlichere Darstellung der Berechnung möglich. Die Stärke des Berechnungsmodells ist in Wahrheit aber von solchen (und vielen anderen) Modifikationen unabhängig. Wir arbeiten im Folgenden offiziell daher mit einem Alphabet Σ, erlauben aber z.B. in allen angegebenen Übungen auch die Hinzunahme von Extra-Symbolen, wo das die Formalisierung vereinfacht.

Eine *Konfiguration* der TM $\mathcal{M}$ ist vollständig beschrieben durch Bandbeschriftung, Kontrollzustand und Kopfposition. Wir fassen diese Information zusammen als ein 4-Tupel

$$C = (\alpha, q, x, \beta) \in (\Sigma \cup \{\Box\})^* \times Q \times (\Sigma \cup \{\Box\}) \times (\Sigma \cup \{\Box\})^*,$$

wobei α den Bandinhalt links von der aktuellen Kopfposition beschreibt, x den Bandinhalt in der Kopfposition, β den Bandinhalt rechts von der aktuellen Kopfposition, und q der aktuelle Zustand ist. Dabei ist $\alpha x \beta$ der (relevante) Bandinhalt in dem Sinne dass links von α und rechts von β sich nur leere Bandzellen anschließen; da α und β selbst $\Box$ (auch am Anfang von α bzw. am Ende von β) enthalten können, unterscheiden wir nicht zwischen Konfigurationen (α, q, x, β), $(\Box\alpha, q, x, \beta)$, $(\alpha, q, x, \beta\Box)$ usw., und verlängern insbesondere jeweils nach Bedarf mit weiteren $\Box$, wenn der Kopf sich über die bisherige Grenze der Beschriftung hinausbewegt. Für die Berechnung auf Eingabe $w \in \Sigma^*$ initialisieren wir $\mathcal{M}$ in der *Startkonfiguration*

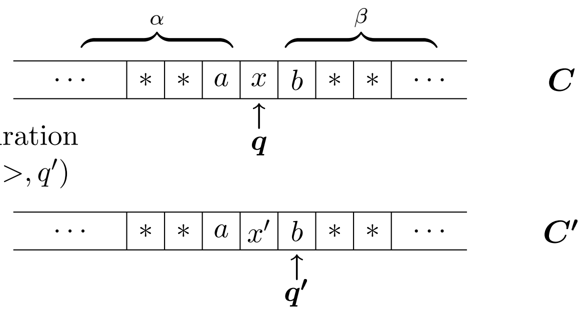
$$C_0[w] := (\varepsilon, q_0, \Box, w).$$



Startkonfiguration
auf $w = a_1 \dots a_n$

In Konfiguration $C = (\alpha, q, x, \beta)$ mit $q \neq q^+, q^-$ ist die *Nachfolgekonfiguration* C' von $\mathcal{M}$ eindeutig festgelegt durch δ wie folgt. Sei $a \in \Sigma \cup \{\Box\}$ das letzte Zeichen von α, $\alpha = \alpha_0 a$ und $b \in \Sigma \cup \{\Box\}$ das erste Zeichen von β, $\beta = b\beta_0$. Dann liest $\mathcal{M}$ in Konfiguration $C = (\alpha, q, x, \beta) = (\alpha_0 a, q, x, b\beta_0)$ im Zustand q das Zeichen x. Sei $\delta(q,x) = (x', d, q')$. Dann ist

$$C' = \begin{cases} (\ \alpha\ , q', x',\ \beta\ ) & \text{falls } d = \circ, \\ (\ \alpha_0\ , q',\ a\ , x'\beta) & \text{falls } d = <, \\ (\alpha x', q',\ b\ ,\ \beta_0\ ) & \text{falls } d = >. \end{cases}$$



Nachfolgekonfiguration
bei $\delta(q, x) = (x', >, q')$

Konfigurationen mit $q = q^-$ oder $q = q^+$ sind *Endkonfigurationen* und haben keine Nachfolgekonfiguration. Die *Berechnung* von $\mathcal{M}$ auf Eingabe w ist die maximale Folge von Konfigurationen $C_0, C_1, \ldots$, wo $C_0 = C_0[w]$ die Anfangskonfiguration zu w ist, und $C_{i+1} = C_i'$ jeweils die Nachfolgekonfiguration, sofern nicht C_i bereits eine Endkonfiguration ist.

Eine Endkonfiguration ist akzeptierend bzw. verwerfend, je nachdem, ob der Kontrollzustand q^+ oder q^- ist. Entsprechend heißt eine Berechnung, die in einer akzeptierenden bzw. verwerfenden Endkonfiguration endet, akzeptierend oder verwerfend, und wir sagen, dass $\mathcal{M}$ die Eingabe akzeptiert oder verwirft.

Eine *unendliche* Berechnung, die keine Endkonfiguration erreicht, wird als *divergent* bezeichnet und liefert kein Ergebnis. Im Allgemeinen kann also eine gegebene Turingmaschine $\mathcal{M}$ auf einer gegebenen Eingabe w divergieren, oder die Eingabe akzeptieren oder verwerfen. In den beiden letzten Fälle, in denen nach endlicher Berechnung ein Ergebnis vorliegt, spricht man auch von terminierender Berechnung (im Gegensatz zu divergenter Berechnung). Wir benutzen folgende Kurznotation:

$w \xrightarrow{\mathcal{M}} \infty$	die Berechnung von $\mathcal{M}$ auf w divergiert.
$w \xrightarrow{\mathcal{M}} \mathrm{STOP}$	die Berechnung von $\mathcal{M}$ auf w terminiert.
$w \xrightarrow{\mathcal{M}} q^+$	$\mathcal{M}$ akzeptiert Eingabe w.
$w \xrightarrow{\mathcal{M}} q^-$	$\mathcal{M}$ verwirft Eingabe w.

Wir geben ein einfaches konkretes Beispiel einer Turingmaschine an.

Beispiel 3.4.7 Wir geben eine (partielle) Spezifikation einer DTM $\mathcal{M}$ über dem Alphabet $\Sigma = \{0, 1\}$ an, die das Wortproblem zur Sprache $\textsc{Palindrom} = \{w \in \{0,1\}^* \colon w = w^{-1}\}$ entscheidet: $\mathcal{M}$ terminiert auf allen Eingaben $w \in \Sigma^*$ und akzeptiert/verwirft je nachdem ob w ein Palindrom ist oder nicht.

Der implementierte Algorithmus besteht darin, das Eingabewort wiederholt im Zick-Zack zu durchlaufen und schrittweise zu verkürzen, sofern erster und letzter Buchstabe passen. Dabei soll stets der erste verbleibende Buchstabe gelöscht und dann (im Zustand gespeichert und) mit dem letzten verglichen werden. Wir benutzen folgende Zustände, um die Phasen dieses Verfahrens zu organisieren:

q_0	Startzustand
$q^?$	ersten Buchstaben abfragen
$q^{\to 0}$	Bewegung zum Ende (erstes $\Box$ rechts), merke 0
$q^{\to 1}$	Bewegung zum Ende (erstes $\Box$ rechts), merke 1
$q^{\leftarrow 0}$	Vergleich des letzten Buchstaben mit 0
$q^{\leftarrow 1}$	Vergleich des letzten Buchstaben mit 1
$q^{\leftarrow}$	Bewegung zurück zum Anfang (erstes $\Box$ links)
q^+	akzeptierender Endzustand
q^-	verwerfender Endzustand

Die wesentlichen Werte der Übergangsfunktion δ:

δ	$\Box$	0	1
q_0	$(\Box, >, q^?)$		
$q^?$	$(\Box, \circ, q^+)$	$(\Box, >, q^{\to 0})$	$(\Box, >, q^{\to 1})$
$q^{\to 0}$	$(\Box, <, q^{\leftarrow 0})$	$(0, >, q^{\to 0})$	$(1, >, q^{\to 0})$
$q^{\to 1}$	$(\Box, <, q^{\leftarrow 1})$	$(0, >, q^{\to 1})$	$(1, >, q^{\to 1})$
$q^{\leftarrow 0}$	$(\Box, \circ, q^+)$	$(\Box, <, q^{\leftarrow})$	$(\Box, \circ, q^-)$
$q^{\leftarrow 1}$	$(\Box, \circ, q^+)$	$(\Box, \circ, q^-)$	$(\Box, <, q^{\leftarrow})$
$q^{\leftarrow}$	$(\Box, >, q^?)$	$(0, <, q^{\leftarrow})$	$(1, <, q^{\leftarrow})$

Übung 3.4.8 Analysiere die obige DTM und überprüfe, dass sie auf jeder Eingabe $w \in \{0,1\}^*$ terminiert und genau die Wörter in Palindrom akzeptiert.
Man kann eine Turingmaschine mit einem etwas vereinfachten Berechnungsablauf programmieren, wenn man ein erweitertes Arbeitsalphabet wie $\Sigma^+ := \{0, 1, {\mid}0, {\mid}1, 0{\mid}, 1{\mid}\}$ benutzt. Dabei sollen die zusätzlichen Zeichen die Funktion von Endmarkierungen übernehmen, sodass die TM z.B. beim Lesen von $1{\mid}$ weiß, dass es sich um eine 1 in der derzeit letzten beschrifteten Bandposition handelt. Organisiere eine entsprechend modifizierte Übergangsfunktion für eine Variante der obigen TM. (Man nehme zunächst der Einfachheit halber an, dass die Eingabe schon mit Endmarkern versehen ist.)

Übung 3.4.9 Skizziere zu einer gegebenen DTM $\mathcal{M} = (\Sigma, Q, q_0, \delta, q^+, q^-)$ ein Programm (in Pseudocode), das M simuliert und auf jeder Eingabe $w \in \Sigma^*$ genau wie $\mathcal{M}$ terminiert und "ja"/"nein" ausgibt, je nachdem, ob $\mathcal{M}$ akzeptiert oder verwirft.

Übung 3.4.10 Man überzeuge sich davon, dass DTM eine Verallgemeinerung von DFA sind, indem man zu einem gegebenen DFA $\mathcal{A}$ eine DTM $\mathcal{M}_\mathcal{A}$ angibt, die $\mathcal{A}$ simuliert: $\mathcal{M}_\mathcal{A}$ soll auf jeder Eingabe $w \in \Sigma^*$ terminieren und genau wie $\mathcal{A}$ akzeptieren bzw. verwerfen.

3.4.3 Entscheidbarkeit und Aufzählbarkeit

Definition 3.4.11 (i) Die DTM $\mathcal{M}$ akzeptiert die Eingabe $w \in \Sigma^*$ wenn die Berechnung von $\mathcal{M}$ auf w akzeptierend ist (in q^+ terminiert).

(ii) Die von $\mathcal{M}$ akzeptierte (erkannte) Sprache ist $L(\mathcal{M}) = \{w \in \Sigma^* \colon \text{ M akzeptiert } w\ \}$.

(iii) $\mathcal{M}$ *entscheidet die Sprache* $L \subseteq \Sigma^*$ (löst das Wortproblem der Sprache L) falls $\mathcal{M}$ auf jeder Eingabe $w \in \Sigma^*$ terminiert *und* $L = L(\mathcal{M})$ ist (auf $w \in L$ in q^+, auf $w \in \Sigma^* \setminus L$ in q^- terminiert).

Definition 3.4.12 (i) $L \subseteq \Sigma^*$ heißt *aufzählbar* (rekursiv aufzählbar) oder *semi-entscheidbar* wenn L von einer Turingmaschine akzeptiert wird.

(ii) $L \subseteq \Sigma^*$ heißt *entscheidbar* (rekursiv) falls L von einer Turingmaschine entschieden wird.

Es ist wichtig, sich den Unterschied zwischen Entscheiden und Aufzählen klarzumachen. Eine Turingmaschine, die L entscheidet, akzeptiert insbesondere auch L. Eine TM $\mathcal{M}$, die die Sprache L (oder ihr Wortproblem) entscheidet, implementiert ein *Entscheidungsverfahren* und muss auf jeder zulässigen Eingabe terminieren. Ein Entscheidungsverfahren garantiert also stets eine definitive Antwort.

Wird dagegen L lediglich von $\mathcal{M}$ akzeptiert, so braucht $\mathcal{M}$ auf negativen Eingabeinstanzen ($w \notin L$) nicht zu terminieren. In diesem Sinne garantiert ein *Aufzählungsverfahren* nur im positiven Falle ($w \in L$) eine definitive Antwort. Im Fall, dass das Verfahren auf einem $w \notin L$ nicht terminiert, weiss man zu keinem Zeitpunkt solange das Verfahren noch läuft, ob es noch terminieren wird; man bekommt also zu keinem Zeitpunkt eine definitive Antwort. Diese Asymmetrie erklärt die Bezeichnung *semi-entscheidbar*, vergleiche hierzu auch Beobachtung 3.4.13.

Entscheidbarkeit impliziert *Aufzählbarkeit*, sowohl von L also auch von $\overline{L} = \Sigma^* \setminus L$. Aufzählbarkeit von L ist schwächer als Entscheidbarkeit (siehe das unentscheidbare aber aufzählbare Halteproblem unten). Sind aber sowohl $L \subseteq \Sigma^*$ als auch $\overline{L} = \Sigma^* \setminus L$ aufzählbar, so folgt, dass L entscheidbar ist. Man gewinnt ein Entscheidungsverfahren durch parallele Simulation beider Aufzählungsverfahren, wobei man akzeptiert sobald eines der simulierten Verfahren akzeptieren würde. Da $\Sigma^* = L \cup \overline{L}$ muss stets entweder das Aufzählungsverfahren für L oder dasjenige für $\overline{L}$ akzeptieren, und man bekommt eine definitive Antwort. Demnach gilt:

Beobachtung 3.4.13 Für $L \subseteq \Sigma^*$ sind äquivalent:

(i) L ist entscheidbar.

(ii) Sowohl L als auch $\overline{L}$ sind aufzählbar.

Die Church-Turing These Trotz ihrer sehr elementaren Natur erweisen sich Turingmaschinen als ausreichend, alle bekannten algorithmischen Verfahren nachzubilden. Diese empirische Beobachtung und die Tatsache, dass sehr verschiedene alternative Ansätze sämtlich zu äquivalenten Ergebnissen führen, münden in die These, dass man hiermit eine adäquate Präzisierung des informellen Konzepts der algorithmischen Lösbarkeit (bzw. Berechenbarkeit) gefunden hat.

Church-Turing These

Ein (Entscheidungs-)Problem ist genau dann algorithmisch lösbar, wenn es eine Turingmaschine zu seiner Lösung gibt. Das Wortproblem zu einer Sprache $L \subseteq \Sigma^*$ ist entscheidbar (algorithmisch lösbar) genau dann, wenn es eine Turingmaschine gibt, die es löst. [18]

Man kann sich konstruktiv davon überzeugen, dass jeder Entscheidungsalgorithmus, der sich in einer herkömmlichen Programmiersprache programmieren lässt, auch mit einer Turingmaschine realisiert werden kann. Die obige These geht aber viel weiter. Es handelt sich um eine These und nicht um ein Theorem, da man über das informelle oder intuitive Konzept dessen, *was ein Algorithmus ist*, unabhängig von einer solchen Präzisierung keine präzise Aussagen machen kann.

[18] Entsprechendes gilt für die Berechenbarkeit von Funktionen; wir beschränken uns aber hier der Einfachheit halber auf Entscheidungsprobleme.

Die erste große Leistung, die mit der so gewonnen Präzisierung verbunden ist, ist die Angabe eines expliziten Entscheidungsproblems, zu dessen Lösung es nachweisbar keine Turingmaschine gibt. Im Lichte der Church-Turing These also ein konkretes algorithmisch nicht lösbares Problem.

Unentscheidbarkeit des Halteproblems Fixiere ein geeignetes Alphabet Σ, das eine natürliche Kodierung der endlichen Beschreibungen von Turingmaschinen $\mathcal{M} = (\Sigma, \ldots)$ durch Σ-Wörter erlaubt (z.B. Binärkodierungen in $\Sigma = \{0, 1\}$, aber jedes andere Alphabet kann ebenso benutzt werden). Die Kodierung von $\mathcal{M}$ werde mit $\langle\mathcal{M}\rangle \in \Sigma^*$ bezeichnet. Die Details der Kodierung $\mathcal{M} \mapsto \langle\mathcal{M}\rangle$ interessieren uns hier nicht.

Damit kann man nun Entscheidungsprobleme betrachten, die das Verhalten von Turingmaschinen betreffen. Das *Halteproblem* ist von dieser Art. Wir sehen uns hier eine Variante an, für die der Beweis der Unentscheidbarkeit besonders einfach ist.

Halteproblem:	Auf Eingabe $\langle\mathcal{M}\rangle$ entscheide, ob $\mathcal{M}$ auf der Eingabe $\langle\mathcal{M}\rangle$ terminiert.

Formal ist das Halteproblem also das Entscheidungsproblem mit zulässigen Eingabeinstanzen $I = \{\langle\mathcal{M}\rangle\colon$ M DTM über Σ $\}$ und positiven Instanzen

$$H = \big\{\langle\mathcal{M}\rangle\colon \langle\mathcal{M}\rangle \xrightarrow{\mathcal{M}} \text{STOP}\big\} \subseteq I.$$

Man macht sich klar, dass das Halteproblem bzw. die zugehörige Sprache $H \subseteq \Sigma^*$ (bezüglich einer vernünftigen Kodierungsfunktion[19] $\mathcal{M} \longmapsto \langle\mathcal{M}\rangle$) immerhin aufzählbar ist. Aus der Unentscheidbarkeit folgt daher mit Beobachtung 3.4.13, dass $\overline{H} = I \setminus H$ nicht aufzählbar ist.

Satz 3.4.14 *Das Halteproblem ist unentscheidbar: es gibt keine DTM über Σ, die das Halteproblem entscheidet.*

Beweis Wir beweisen die Behauptung indirekt, indem wir aus der Annahme, es gebe eine TM $\mathcal{M}_0$, die das Halteproblem entscheidet, einen Widerspruch folgern. Die Idee des Halteproblems und des Beweises bezeichnet man auch als *Diagonalisierung*; als informelles Vorbild können Paradoxien durch *Selbstbezug* wie etwa die Aussage "dieser Satz ist falsch" dienen.

Annahme: $\mathcal{M}_0$ entscheidet das Halteproblem.

Demnach gilt für jede Eingabe $w = \langle\mathcal{M}\rangle$:

$$\begin{aligned} \langle\mathcal{M}\rangle \xrightarrow{\mathcal{M}_0} q^+ \quad &\Leftrightarrow \quad \langle\mathcal{M}\rangle \xrightarrow{\mathcal{M}} \text{STOP} \\ \langle\mathcal{M}\rangle \xrightarrow{\mathcal{M}_0} q^- \quad &\Leftrightarrow \quad \langle\mathcal{M}\rangle \xrightarrow{\mathcal{M}} \infty \end{aligned}$$

$\mathcal{M}_0$ können wir zunächst so umbauen, dass wir eine neue TM $\mathcal{M}_1$ erhalten, die auf einer Eingabe

– divergiert, genau wenn $\mathcal{M}_0$ akzeptiert
– ebenfalls verwirft, wenn $\mathcal{M}_0$ verwirft.

[19] Man braucht hierfür, dass es eine Turingmaschine gibt, die auf Eingabe $\langle\mathcal{M}\rangle$ (also den Code einer beliebigen anderen Maschine $\mathcal{M}$) diese andere Turingmaschine $\mathcal{M}$ simuliert; man kann diese Supermaschine als einen TM-Compiler verstehen. Dass es das für geeignete Kodierungen von Turingmaschinen gibt, lässt sich zeigen.

Dazu muss man lediglich, wenn $\mathcal{M}_0$ einen Übergang in den akzeptierenden Endzustand hätte, $\mathcal{M}_1$ in einen neuen Zustand schicken, der die Berechnung in eine unendliche Schleife führt.

Die Existenz der Maschine $\mathcal{M}_1$ ist aber widersprüchlich.

Aus der Bedingung an $\mathcal{M}_0$ folgt für $\mathcal{M}_1$:

$$\langle \mathcal{M} \rangle \xrightarrow{\mathcal{M}_1} \infty \quad \Leftrightarrow \quad \langle \mathcal{M} \rangle \xrightarrow{\mathcal{M}} \text{STOP}$$

und für den Spezialfall der Eingabe $w := \langle \mathcal{M}_1 \rangle$ also

$$\langle \mathcal{M}_1 \rangle \xrightarrow{\mathcal{M}_1} \infty \quad \Leftrightarrow \quad \langle \mathcal{M}_1 \rangle \xrightarrow{\mathcal{M}_1} \text{STOP}.$$

□

Korollar 3.4.15 *Es gibt aufzählbare, aber nicht entscheidbare Sprachen, sowie Sprachen, die nicht aufzählbar sind.*

Viele interessante Entscheidungsprobleme sind unentscheidbar, d.h. *prinzipiell nicht algorithmisch lösbar*. Der Nachweis beruht auf *Reduktion* des Halteproblems, durch die man nachweist, dass auch das Halteproblem entscheidbar wäre, wenn das gegebene neue Problem entscheidbar wäre. Daraus folgt die Unentscheidbarkeit des neuen Problems.

Beispiel 3.4.16 Das folgende Problem Terminierung ist unentscheidbar: Für $\langle \mathcal{M} \rangle$ entscheide, ob $\mathcal{M}$ auf allen Eingaben $w \in \Sigma^*$ terminiert.

Begründung (durch Reduktion auf die Unentscheidbarkeit des Halteproblems):

Aus einem Entscheidungsverfahren für Terminierung könnten wir folgendes Entscheidungsverfahren für das Halteproblem gewinnen. Berechne aus der Eingabe $\langle \mathcal{M} \rangle$ zunächst die Kodierung der Maschine $\mathcal{M}'$, die ihre Eingabe löscht, dann $\langle \mathcal{M} \rangle$ auf das Band schreibt und dann auf $\langle \mathcal{M} \rangle$ wie $\mathcal{M}$ weiterrechnet. Offenbar ist $\langle \mathcal{M} \rangle \in H$ genau dann, wenn $\langle \mathcal{M}' \rangle$ eine positive Instanz von Terminierung ist.

Weitere Beispiele sehr verschiedener unentscheidbarer Probleme, neben vielen anderen:

- *Äquivalenzproblem*: Für zwei vorgelegte Programme (oder Turingmaschinen), entscheide ob sie äquivalent sind (dieselbe Sprache erkennen bzw. auf allen Eingaben dasselbe Ergebnis haben).
  [Für DFA ist dieses Äquivalenzproblem noch entscheidbar; nicht aber für PDA.]

- *Parkettierungsproblem*: Für eine endliche Menge von quadratischen Kacheln mit gefärbten Kanten, entscheide, ob man beliebig große Quadrate mit Kacheln dieser Typen (jeweils unbegrenzter Vorrat) so pflastern kann, dass die Farben angrenzender Seiten übereinstimmen.
  [Entscheidbar in der Variante für beliebig lange Rechtecke vorgegebener Breite.]

- *Arithmetik*: Für arithmetische Aussagen (in der Logik erster Stufe), wie etwa Behauptungen über die Existenz von ganzzahligen Nullstellen für Polynome über $\mathbb{Z}$, entscheide, ob diese Aussage für die Arithmetik der ganzen Zahlen wahr ist. Analoges gilt für $\mathbb{N}$ oder $\mathbb{Q}$.
  [Entscheidbar für die Arithmetik der reellen Zahlen.]

3.4.4 Aufzählbarkeit und Entscheidbarkeit in der Chomsky-Hierarchie

Wir fassen einige zentrale Aussagen zur Entscheidbarkeit bzw. Aufzählbarkeit von Sprachen auf verschiedenen Niveaus der Chomsky-Hierarchie ohne detaillierte Beweise zusammen.

Satz 3.4.17 *Für jede Sprache $L \subseteq \Sigma^*$ sind äquivalent:*

(i) *L wird von einer Grammatik erzeugt ($L = L(G)$ für Grammatik G, Typ 0).*

(ii) *L wird von einer DTM akzeptiert ($L = L(\mathcal{M})$ für DTM $\mathcal{M}$), d.h., L ist aufzählbar.*

Für den Beweis von (ii) $\Rightarrow$ (i) konstruiert man zu gegebener Turingmaschine $\mathcal{M}$ eine Grammatik, die die Konfigurationenfolge einer akzeptiertenden Berechnung von $\mathcal{M}$ rückwärts simuliert. In der Grundidee haben wir ähnliche Konstruktionen im Zusammenhang mit PDA in Satz 3.4.5 kennengelernt.

Umgekehrt wird für (i) $\Rightarrow$ (ii) eine TM konstruiert, die systematisch (immer längere) Ableitungsketten der gegebenen Grammatik G erzeugt und jeweils das Ergebnis mit dem Eingabewort vergleicht. Wenn so eine Ableitung in G gefunden wird, wird die Eingabe akzeptiert, andernfalls divergiert die Berechnung.

Satz 3.4.18 *Jede kontextsensitive Sprache $L \subseteq \Sigma^*$ ist entscheidbar (Das Wortproblem zu einer Typ 1 Grammatik G wird von einer DTM $\mathcal{M}$ gelöst).*

Hier nutzt man aus, dass Produktionen und Ableitungen in einer kontextsensitiven Grammatik nicht längenverkürzend sind. Deshalb kann man – im Gegensatz zum allgemeinen Fall einer Typ 0 Grammatik – hier den Suchraum für mögliche Ableitungen eines gegebenen $w \in \Sigma^*$ endlich beschränken. [20] Es reicht bei Eingabe $w \in \Sigma^*$ alle Ableitungssequenzen von G zu betrachten, deren Zwischenergebnisse aus Wörtern $v \in (\Sigma \cup V)^*$ mit $|v| \leqslant |w|$ bestehen. Wenn man Zykel von $\rightarrow_G^*$, die ja keine neuen Ableitungergebnisse liefern, eliminiert, so kommen also nur endlich viele Ableitungen in Frage. Für eine Typ 1 Grammatik G lassen sich deshalb die endlichen Mengen

$$L_n(G) = \big\{ v \in (\Sigma \cup V)^* \colon X_0 \rightarrow_G^* v,\ |v| \leqslant n \big\}$$

sukzessive, induktiv über n, bestimmen. Man gewinnt daraus ein Entscheidungsverfahren für $L(G)$, da für $w \in \Sigma^*$ gilt: $w \in L(G)$ gdw. $w \in L_n(G)$ für $n = |w|$.

Bemerkung: Auch zum Niveau der kontextsensitiven Sprachen (Type 1) gibt es eine äquivalente Charakterisierung durch ein Berechnungsmodell. Die Kontextsensitiven Sprachen werden gerade von linear platzbeschränkten nichtdeterministischen Turingmaschinen erkannt. Oder, an Automaten orientiert, von nichtdeterministischen Automaten, die auf dem Eingabewort vorwärts und rückwärts laufen und dabei auch dessen Buchstaben überschreiben können.

Aus den Betrachtungen zum Halteproblem, den beiden letzten Sätzen und Korollar 3.4.15 können wir auch erkennen, dass alle verbleibenden Stufen der Chomsky-Hierarchie ebenfalls echt sind, d.h. dass Typ 3 $\subsetneq$ Typ 2 $\subsetneq$ Typ 1 $\subsetneq$ Typ 0 und dass auch Typ 0 nicht alle Sprachen erfasst. [Zwischen Typ 3, Typ 2 und Typ 1 hatten wir das mit entsprechenden Pumping Lemmas eingesehen.]

[20] Für strikt verlängernde Produktionen bräuchte man nur alle Ableitungsketten einer Länge bis zu $|w|$ durchzuprobieren. Da aber auch längenerhaltende Produktionen erlaubt sind, benötigt man eine zusätzliche Einsicht.

Korollar 3.4.19 *Auch die Inklusionen zwischen Typ 1, Typ 0, und allgemeinen Sprachen sind strikt:*

(i) *Es gibt nicht aufzählbare Sprachen, die demnach nicht von Grammatiken erzeugt werden können (nicht Typ 0).*

(ii) *Es gibt aufzählbare Sprachen (Typ 0), die nicht entscheidbar sind (nicht Typ 1).*

3.5 Übersicht zu den Abschlusseigenschaften

Auch das Niveau 1 der Chomsky-Hierarchie, die Klasse der kontextsensitiven Sprachen, besizt eine präzise Charakterisierung mittels eines passenden Berechnungsmodells. Hierzu betrachtet man *nichtdeterministische, linear platzbeschränkte* Turingmaschinen. Nichtdeterministische Turingmaschinen NTM haben anstelle der Übergangsfunktion eine Übergangs*relation*, was Verzweigungen der Berechnungspfade oder nichtdeterministische Auswahlen erlaubt (vergleiche die Verallgemeinerung von DFA zu NFA); akzeptierende Berechnungen (Läufe, die in q^+ enden) und die akzeptierte Sprache sind entsprechend definiert. Linear platzbeschränkt heißt eine Turingmaschine $\mathcal{M}$ dann, wenn es eine Konstante c gibt, sodass sich der Kopf in Läufen von M auf Eingaben $w \in \Sigma^*$ nie weiter als $c|w|$ Schritte von seiner Startposition entfernt. Man kann zeigen, dass eine Sprache genau dann kontextsensitiv ist, wenn sie von einer linear platzbeschränkten NTM akzeptiert wird.

Im Rahmen der Komplexitätstheorie zeigt man auch, dass die Klasse der so erkennbaren Sprachen abgeschlossen unter Komplement ist (ein Spezialfall des Satzes von Immerman und Szelepcsenyi). Alle übrigen Abschlusseigenschaften in der folgenden Tabelle sind aus unseren Betrachtungen unmittelbar zu gewinnen.

	abgeschlossen unter				
Typ	$\cup$	$\cap$	$^-$	$\cdot$	$*$
3	+	+	+	+	+
2	+	−	−	+	+
1	+	+	+	+	+
0	+	+	−	+	+
bel. Σ-Sprachen	+	+	+	+	+

Die wichtigsten Punkte aus Kapitel 3

Grammatiken und Erzeugungsprozesse

Niveaus der **Chomsky-Hierarchie**

Normalform und Pumping Lemma für
kontextfreie Sprachen

Satz von Myhill-Nerode

Berechnungmodelle
PDA **TM**
Aufzählbarkeit / Entscheidbarkeit

4 Exkurs: Beispiele algorithmischer Anwendungen

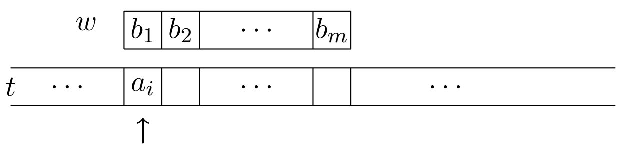
4.1 Automatentheoretisches string matching

Der einfachste Fall des "string matching" über einem Alphabet Σ ist das folgende Problem:

> zu gegeben Text $t \in \Sigma^*$ und Suchwort $w \in \Sigma^*$,
> finde alle Positionen, in denen t das Teilwort w enthält.

Offensichtlich löst jeder Editor im Hintergrund dieses Problem, wannimmer man eine Textsuche startet. Wir wollen sehen, dass ein automatentheoretisches Verständnis dieses Suchproblems zu einer algorithmischen Idee führt, die der Implementierung eines naiven Suchalgorithmus überlegen ist.

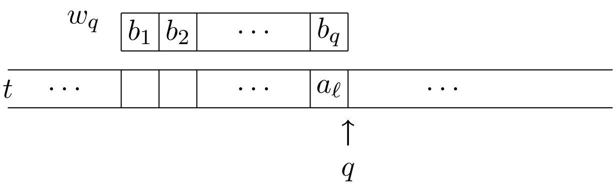
Formalisierung des Problems: Fixiere Σ. Die Eingabe bestehe aus dem *Suchwort* $w = b_1 \dots b_m$ und dem *Text* $t = a_1 \dots a_n$. Die Länge des Suchwortes ist m, die Länge des Textes n; dabei ist typischerweise n viel größer als m. Wir sagen, dass w in Position i in t vorkommt, wenn $t_{i,i+m-1} = w$ [wir schreiben $t_{i,j} = a_i \dots a_j$ für das Teilwort der Länge $j - i + 1$, das in Position i beginnt].



Naiver Algorithmus: Die naive Suche würde Schritt für Schritt für jede Position i ($1 \leqslant i \leqslant n - m$) vorwärts testen, ob die folgenden m Zeichen auf w passen: $a_{i+j} = b_{j+1}$ für $j = 0, \dots, m - 1$? Für jede Position benötigt man so bis zu m Vergleiche, und insgesamt also eine wie $m(n-m)$ oder grob wie das Produkt von n und m anwachsende Zahl. Beachte, dass in diesem Verfahren jeder Textbuchstabe für bis zu m verschiedene Positionen i untersucht wird (einmal als mögliche Passung für die letzte Position in w, einmal für die vorletzte, etc).

Verbesserung: DFA und dynamische Programmierung. Anstelle der geschachtelten Schleifen soll der Text nur ein einziges Mal von vorne nach hinten durchgegangen werden. Wir wollen einen DFA durch den Text schicken, der Buchstabe für Buchstabe des Textes liest und alle relevante Information in seinem Zustand speichert. *Was ist die relevante Information?* Beachte, dass ein solcher Automat natürlich erst in Position $i+m-1$ feststellen kann, ob w in Position i als Teilwort passt. Dazu muss in Position $i+m-2$ festgehalten worden sein, ob $w_{m-1} := w_{1,m-1}$ (der Präfix der Länge $m-1$) in Position i passt; dazu in Position $i + m - 3$, ob $w_{m-2} := w_{1,m-2}$ (der Präfix der Länge $m - 2$) in Position ℓ passt, usw.

Für $0 \leqslant m$ sei $w_q := w_{1,q} = b_1 \dots b_q$ der Präfix der Länge q von w. Der DFA $\mathcal{A}_w$ habe Zuständsmenge $Q = \{0, \dots, m\}$. Im Lauf über t soll der Zustand q von $\mathcal{A}_w$ nach Bearbeitung der Position i gerade das maximale q angeben, für das w_q bis zu dieser Position passt.

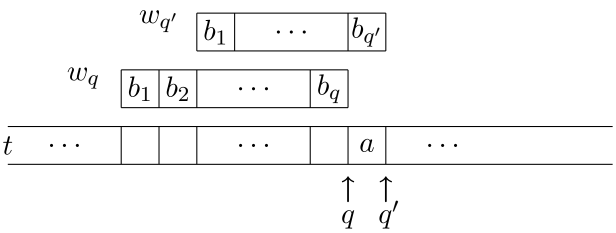


Ein vollständiges Passen von w (in Position i) erkennt man nun daran, dass $\mathcal{A}_w$ in Position $\ell = i + m - 1$ den Zustand m annimmt.

Wie muss die Zustandsinformation beim Vorrücken um einen Buchstaben in t angepasst werden? Die korrekte Übergangsfunktion von $\mathcal{A}_w$ ist gegeben durch

$$\delta(q, a) := \max\{k \colon w_k \text{ Suffix von } w_q a\}$$ [21]

Die Korrektheit dieses Übergangs lässt sich an folgendem Diagramm ablesen:



Weitere Verbesserung: Knuth-Morris-Pratt. Durch trickreiche Vorabberechnung und Tabellierung der partiellen Selbstüberlappungen von w, kann man die δ-Werte "on the fly" berechnen und die Simulation von $\mathcal{A}_w$ in einen Linearzeitalgorithmus, d.h. linear in $n + m$, durchführen.

4.2 Automatentheoretisches model checking

Eine der wichtigsten aktuellen Anwendungen automatentheoretischer Methoden ergibt sich in der *Verifikation.* Ziel ist die (automatisierte oder teilautomatisierte) Überprüfung, ob Systementwürfe bestimmte in der Spezifikation geforderte Eigenschaften tatsächlich haben. Neben dem (stichprobenartigen) Testen geforderter Eigenschaften gibt es formale Methoden zur Spezifikation (in geeigneten logischen Sprachen), die sich für eine algorithmische Überprüfung eignen. Automatentheoretische Ideen spielen bei vielen effizienten Verfahren zur Lösung solcher Entscheidungsprobleme eine große Rolle.

Die grobe Idee ist (vereinfacht) wie folgt. Sei $\mathcal{S}$ ein System(entwurf), vorgelegt etwa in Form eines Transitionssystems. E sei eine Eigenschaft, die in der Spezifikation von allen Läufen von $\mathcal{S}$ gefordert wird. Läufe von $\mathcal{S}$ werden als Zustandsfolgen beschrieben; zu $\mathcal{S}$ gehört so die Sprache $L_{\mathcal{S}}$ aller Zustandsfolgen, die in Läufen von $\mathcal{S}$ realisiert werden. Andererseits entspricht E einer Sprache L_E all derjenigen Zustandsfolgen, die die Eigenschaft E haben. Dass $\mathcal{S}$ die geforderte Eigenschaft hat, bedeutet nun gerade, dass $L_{\mathcal{S}} \subseteq L_E$ ist. Dies ist äquivalent dazu, dass $L_{\mathcal{S}} \setminus L_E = L_{\mathcal{S}} \cap \overline{L_E} = \emptyset$ ist: ein Leerheitsproblem für eine einfache Boolesche Kombination von formalen Sprachen. In

[21] Ein Suffix ist ein Endabschnitt; vgl. Präfix für Anfangsabschnitt.

vielen relevanten Fällen sind die beteiligten Sprachen regulär, und dieses Leerheitsproblem lässt sich algorithmisch lösen. Dabei spielen dann effiziente automatentheoretische Methoden und Konstruktionen eine entscheidende Rolle für den Entwurf guter Entscheidungsalgorithmen, die auch in der Praxis angewandt werden können.

Eine besonders nützliche Charakteristik solcher Methoden, die auf Automaten basieren, ist, dass im ungünstigen Fall, dass also $\mathcal{S}$ nicht der Eigenschaft E gehorcht, aus einem Automaten, der die Sprache $L_{\mathcal{S}} \cap \overline{L_E}$ erkennt (die also nicht leer ist), ein Gegenbeispiel gefunden werden kann. Man findet z.B. einen Lauf minimaler Länge von $\mathcal{S}$, der E verletzt. Dieses schwarze Schaf kann man zusammen mit dem negativen Verifikationsergebnis ausgeben, um dem Systemdesigner einen Hinweis darauf zu geben, *warum* etwas schiefgeht.

Index