Skript zur Lehrveranstaltung

Theoretische Informatik 1

Prof. Dr. Sebastian Siebertz

AG Theoretische Informatik

Stand: 25. September 2023

Universität
Bremen

Inhaltsverzeichnis

Acknowledgement

Dieses Skript ist eine erweiterte und modifizierte Version eines Vorlesungsskriptes von Prof. Dr. Thomas Schneider und Prof. Dr. Carsten Lutz, welches wiederum auf ein Skript von Prof. Dr. Franz Baader zurückgeht.

Einführung

Die theoretische Informatik beschäftigt sich mit zentralen Fragestellungen der Informatik, wie etwa den prinzipiellen Grenzen der Berechenbarkeit. Zentrale Methoden sind die Abstraktion und die Modellbildung, d. h. es werden wichtige Konzepte und Methoden der Informatik identifiziert und in abstrakter Form beschrieben und studiert. Daraus ergibt sich eine Sammlung mathematischer Theorien, die die Grundlage für zahlreiche andere Teilgebiete der Informatik bildet.

Die theoretische Informatik ist in zahlreiche Teilgebiete untergliedert, wie etwa die Komplexitätstheorie, die Algorithmentheorie, die Kryptographie und die Datenbanktheorie. Die Lehrveranstaltungen *Theoretische Informatik 1 & 2* geben eine Einführung in folgende zwei Bereiche der theoretischen Informatik:

Automatentheorie und formale Sprachen

Behandelt in *Theoretische Informatik 1*

Im Mittelpunkt stehen *Wörter* und *formale Sprachen* (Mengen von Wörtern). Diese sind ein nützliches Abstraktionsmittel in der Informatik. Es kann z. B. die Eingabe oder Ausgabe eines Programms als Wort betrachten werden und die Menge der syntaktisch korrekten Eingaben als Sprache. Wichtige Fragestellungen sind z. B.:

- Was sind geeignete Beschreibungsmittel für (meist unendliche) formale Sprachen? (z. B. Automaten, Grammatiken und reguläre Ausdrücke)
- Welche Typen von Sprachen lassen sich unterscheiden?
- Welche Eigenschaften haben die verschiedenen Sprachtypen?

Berechenbarkeit und Komplexität

Behandelt in *Theoretische Informatik 2*

Hier geht es darum, welche Probleme und Funktionen prinzipiell berechenbar sind und welche nicht. Außerdem wird untersucht, welcher zeitliche Aufwand zur Berechnung eines Problems/einer Funktion notwendig ist (unabhängig vom konkreten Algorithmus).

Wichtige Fragestellungen sind z. B.:

- Welche Berechnungsmodelle gibt es und wie verhalten sich diese bzgl. ihrer Mächtigkeit zueinander?
- Gibt es Funktionen oder Probleme, die prinzipiell nicht berechenbar, also nicht von einer Maschine lösbar, sind?
- Kann jede berechenbare Funktion mit akzeptablem Zeit- und Speicherplatzaufwand berechnet werden?
- Für in der Informatik häufig auftretende Probleme/Funktionen: wie viel Zeit und Speicherplatz benötigt ein optimaler Algorithmus mindestens um sie zu lösen?

Teil I + II: Automatentheorie und formale Sprachen

Formale Sprachen, also (endliche oder unendliche) Mengen von Wörtern, sind ein wichtiger Abstraktionsmechanismus der Informatik. Hier einige Anwendungsbeispiele:

- Die Menge aller wohlgeformten Programme in einer gegebenen Programmiersprache wie Java, C++ oder Haskell ist eine formale Sprache.
- Die Menge aller wohlgeformten Eingaben für ein Programm oder für ein Formular auf einer Webseite (z. B. Menge aller Kontonummern / Menge aller Geburtsdaten) ist eine formale Sprache.
- Jeder Suchausdruck (z. B. Linux Regular Expression) definiert eine formale Sprache: die Menge der Dokumente, in der der Ausdruck zu finden ist.
- Kommunikationsprotokolle: z. B. die Menge aller wohlgeformten TCP-Pakete ist eine formale Sprache.
- Das „erlaubte“ Verhalten von Soft- und Hardwaresystemen kann in sehr natürlicher Weise als formale Sprache modelliert werden.

Wir beginnen mit einem kurzen Überblick über die zentralen Betrachtungsgegenstände und Fragestellungen.

1. **Charakterisierung**:

   Nützliche und interessante formale Sprachen sind in der Regel unendlich, wie in allen obigen Beispielen (es gibt zum Beispiel unendlich viele wohlgeformte Java-Programme). Wie können wir derartige Sprachen mit einem endlichem Formalismus beschreiben?

   - *Automaten* oder *Maschinen*, die genau die Wörter der Sprache akzeptieren. Wir werden mehrere Automaten- bzw. Maschinenmodelle kennen lernen, z. B. endliche Automaten, Kellerautomaten und Turingmaschinen.
   - *Grammatiken*, die genau die Wörter der Sprache generieren; auch hier gibt es viele verschiedene Typen, z. B. rechtslineare Grammatiken und kontextfreie Grammatiken (vgl. auch VL „Praktische Informatik“: kontextfreie Grammatiken (EBNF) zur Beschreibung der Syntax von Programmiersprachen).
   - *Ausdrücke*, die beschreiben, wie die Sprache aus Basissprachen mit Hilfe gewisser Operationen (z. B. Vereinigung) erzeugt wird.

Abhängig von dem verwendeten Automaten-/Grammatiktyp erhalten wir verschiedene Klassen von Sprachen. Wir werden hier die vier wichtigsten Klassen betrachten; diese sind in der **Chomsky-Hierarchie** zusammengefasst:

Klasse	Automatentyp	Grammatiktyp
Typ 0	Turingmaschine (TM)	allgemeine Grammatik
Typ 1	TM mit linearer Bandbeschrünkung	monotone Grammatik
Typ 2	Kellerautomat	kontextfreie Grammatik
Typ 3	endlicher Automat	einseitig lineare Grammatik

Für Typ 3 existiert auch eine Beschreibung durch reguläre Ausdrücke. Typ 2 kann beispielsweise weitgehend die Syntax von Programmiersprachen beschreiben.

2. Welche Fragen sind für eine Sprachklasse entscheidbar und mit welchem Aufwand? Die folgenden Probleme werden eine zentrale Rolle spielen:

   - *Wortproblem*: gegeben eine Beschreibung der Sprache L (z. B. durch einen Automaten, eine Grammatik, einen Ausdruck, ...) und ein Wort w. Gehört w zu L?

     Anwendungsbeispiele:

     – Programmiersprache, deren Syntax durch eine kontextfreie Grammatik beschrieben ist. Entscheide für ein gegebenes Programm P, ob dieses syntaktisch korrekt ist.

     – Suchpattern für Textdateien sind häufig reguläre Ausdrücke. Suche die Dateien (Wörter), die das Suchpattern enthalten (zu der von ihm beschriebenen Sprache gehören).

   - *Leerheitsproblem*: gegeben eine Beschreibung der Sprache L. Ist L leer?

     Anwendungsbeispiel:
     Wenn ein Suchpattern die leere Sprache beschreibt, so müssen die Dateien nicht durchsucht werden, sondern es kann ohne weiteren Aufwand gemeldet werden, dass das Pattern nicht sinnvoll ist.

   - *Äquivalenzproblem*: Beschreiben zwei verschiedene Beschreibungen dieselbe Sprache?

     Anwendungsbeispiel:
     Jemand vereinfacht die Grammatik einer Programmiersprache, um sie übersichtlicher zu gestalten. Beschreibt die vereinfachte Grammatik wirklich dieselbe Sprache wie die ursprüngliche?

3. Welche **Abschlusseigenschaften** hat eine Sprachklasse?

   Z. B. Abschluss unter Schnitt, Vereinigung und Komplement: wenn L_1, L_2 in der Sprachklasse enthalten, sind es dann auch $L_1 \cap L_2$, $L_1 \cup L_2$, $\overline{L_1}$?

Anwendungsbeispiele:

- Suchpattern: Suche nach Dateien, die das Pattern *nicht* enthalten (Komplement) oder die zwei Pattern enthalten (Schnitt).
- Reduziere das Äquivalenzproblem auf das Leerheitsproblem, ohne die gewählte Klasse von Sprachen zu verlassen: Statt „$L_1 = L_2$?“ können wir entscheiden, ob $(L_1 \cap \overline{L_2}) \cup (L_2 \cap \overline{L_1})$ leer ist.

Abgesehen von ihrer direkten Nützlichkeit für verschiedene Informatik-Anwendungen stellen sich alle diese Fragestellungen als mathematisch sehr interessant heraus. Zusammengenommen bilden sie eine wichtige formale Grundlage der Informatik.

I. Endliche Automaten und reguläre Sprachen

1. Grundbegriffe

Die grundlegenden Begriffe der Vorlesung „Theoretische Informatik 1“ sind *Wörter* und *formale Sprachen*.

1.1. Wörter und formale Sprachen

Alphabet. Ein *Alphabet* ist eine endliche, nicht leere Menge von Symbolen.

Beispiele sind:

- $\Sigma_1 = \{a, b, c, \ldots, z\}$.
- $\Sigma_2 = \{0, 1\}$.
- $\Sigma_3 = \{0, \ldots, 9\} \cup \{,\}$.
- $\Sigma_4 = \{$`program`, `const`, `var`, `label`, `procedure`, `function`, `type`, `begin`, `end`, `if`, `then`, `else`, `case`, `of`, `repeat`, `until`, `while`, `do`, `for`, `to`$\} \cup \{$VAR, VALUE, FUNCTION$\}$.

Als Symbol (Platzhalter) für Alphabetssymbole benutzen wir in der Regel $a, b, c, \ldots$. Alphabete bezeichnen wir meist mit Σ.

Obwohl die Symbole von Σ_4 aus mehreren Buchstaben der üblichen Schriftsprache bestehen, betrachten wir sie doch als unteilbare Symbole. Die Elemente von Σ_4 sind genau die Schlüsselwörter der Programmiersprache Pascal. Konkrete Variablennamen, Werte und Funktionsaufrufe sind zu den Schlüsselwörtern VAR, VALUE, FUNCTION abstrahiert, um Endlichkeit des Alphabets zu gewährleisten.

Natürliche Zahlen und Folgen. Wir bezeichnen die Menge der natürlichen Zahlen mit $\mathbb{N} = \{1, 2, 3, \ldots\}$ und die Menge der natürlichen Zahlen mit 0 mit $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. Für $n \in \mathbb{N}$ bezeichnen wir mit $[n] = \{1, \ldots, n\}$ die Menge der ersten n natürlichen Zahlen. Wir definieren $[0] = \emptyset$. Eine endliche *Folge über einer Menge M* ist eine Abbildung $w\colon [n] \to M$ für ein $n \in N_0$. Die Zahl n bezeichnet die Länge der Folge. Die Folge $w\colon [0] \to M$ heißt *leere Folge*. Wir schreiben meist w_i für das Folgenglied $w(i)$ für $1 \leq i \leq n$.

Wort. Ein *Wort über einem Alphabet* Σ ist eine endliche Folge über Σ. Wir schreiben kurz $w = w_1 \cdots w_n$ mit $w_i \in \Sigma$ für das Wort $w\colon [n] \to \Sigma\colon i \mapsto w_i$.

Beispiele sind:

- $w = abc$ ist ein Wort über Σ_1.
- $w = 1000110$ ist ein Wort über Σ_2.
- $w = 10{,}0221{,}4292_{„}$ ist ein Wort über Σ_3.
- Jedes Pascal-Programm kann als Wort über Σ_4 betrachtet werden, wenn jede konkrete Variable durch das Schlüsselwort VAR ersetzt wird, und jeder Wert durch VALUE und jeden Funktionsaufruf durch FUNCTION.

Als Symbol für Wörter verwenden wir meist w, v, u. Die Länge eines Wortes w wird mit $|w|$ bezeichnet, es gilt also z. B. $|aba| = 3$. Manchmal ist es praktisch, auch die Anzahl der Vorkommen eines Symbols a in einem Wort w in kurzer Weise beschreiben zu können. Wir verwenden hierfür $|w|_a$, es gilt also z. B. $|aba|_a = 2$, $|aba|_b = 1$, $|aba|_c = 0$. Einen Spezialfall stellt das *leere Wort* dar, also die leere Folge von Symbolen. Dieses wird durch ε bezeichnet. Es ist das einzige Wort mit $|w| = 0$.

Formale Sprache. Mit Σ^* bezeichnen wir die Menge *aller* Wörter über einem Alphabet Σ. Eine *(formale) Sprache* über einem Alphabet Σ ist eine Menge $L \subseteq \Sigma^*$ von Wörtern über Σ

Beispiele sind:

- $\{a, b\}^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$.
- $L = \emptyset$.
- $L = \{abc\} \subseteq \{a, b, c\}^*$.
- $L = \{a, b, c, ab, ac, bc\} \subseteq \{a, b, c\}^*$.
- $L = \{w \in \{a, \dots, z\}^* \ : \ w \text{ ist ein Wort der deutschen Sprache}\}$.
- L als Menge aller Wörter über Σ_4, die wohlgeformte Pascal-Programme bilden.

Als Symbol für Sprachen verwenden wir meist L. Sprachen können sowohl endlich als auch unendlich sein. Interessant sind meist nur unendliche Sprachen. Als nützliche Abkürzung führen wir Σ^+ für die Menge $\Sigma^* \setminus \{\varepsilon\}$ aller nicht-leeren Wörter über Σ ein. Sowohl Σ^* als auch Σ^+ sind unendliche Sprachen.

1.2. Operationen auf Sprachen und Wörtern

Im Folgenden werden wir viel mit Wörtern und formalen Sprachen umgehen. Dazu verwenden wir in erster Linie die folgenden Operationen.

Konkatenation: Eine Operation, die auf Wörter sowie auf Sprachen angewendet werden kann. Auf Wörtern u und v bezeichnet die Konkatenation $u \cdot v$ das Wort uv, das wir durch einfaches „Hintereinanderschreiben“ erhalten. Formal: wenn $u\colon [n] \to \Sigma$, $v\colon [m] \to \Sigma$, dann ist uv definiert als

$$uv\colon [n+m] \to \Sigma : i \mapsto \begin{cases} u_i & \text{falls } 1 \leq i \leq n \\ v_j & \text{falls } n+1 \leq i \leq n+m, j = n+i \end{cases}$$

Es gilt also z. B. $abb \cdot ab = abbab$. Auf Sprachen bezeichnet die Konkatenation das Hintereinanderschreiben *beliebiger* Wörter aus den beteiligten Sprachen:

$$L_1 \cdot L_2 := \{u \cdot v \mid u \in L_1 \text{ und } v \in L_2\}$$

Es gilt also z. B.

$$\{aa, a\} \cdot \{ab, b, aba\} = \{aaab, aab, aaaba, ab, aaba\}.$$

Sowohl auf Sprachen als auch auf Wörtern wird der Konkatenationspunkt häufig weggelassen, wir schreiben also z. B. L_1L_2 statt $L_1 \cdot L_2$.

Es gilt $\emptyset \cdot L = L \cdot \emptyset = \emptyset$. Die Konkatenation ist assoziativ; es gilt also $(L_1 \cdot L_2) \cdot L_3 = L_1 \cdot (L_2 \cdot L_3)$. Sie ist nicht kommutativ; im Allgemeinen gilt also nicht $L_1 \cdot L_2 = L_2 \cdot L_1$.

Um wiederholte Konkatenation desselben Wortes zu beschreiben, verwenden wir folgende Notation: für ein Wort $w \in \Sigma^*$ und ein $n \geq 0$ bezeichnet w^n das Wort, das wir durch n-malige Konkatenation von w erhalten, also zum Beispiel $(abc)^3 = abcabcabc$ (aber $abc^3 = abccc$). Wir definieren $w^0 = \varepsilon$ für jedes Wort w.

Präfix, Suffix, Infix: Zu den natürlichsten und einfachsten Operationen auf Wörtern gehört das Bilden von Präfixen, Suffixen und Infixen:

u ist Präfix von v	wenn	$v = uw$ für ein $w \in \Sigma^*$.
u ist Suffix von v	wenn	$v = wu$ für ein $w \in \Sigma^*$.
u ist Infix von v	wenn	$v = w_1uw_2$ für $w_1, w_2 \in \Sigma^*$.

Präfixe und Suffixe eines Wortes w sind also beliebig lange Anfangs- bzw. Endstücke von w; Infixe sind beliebige Teilwörter, die auch in der Mitte auftreten können. Jedes Präfix und Suffix von w ist damit auch Infix. Das Wort $aabbcc$ beispielsweise hat die Präfixe $\varepsilon, a, aa, aab, aabb, aabbc, aabbcc$ sowie 19 Infixe.

Boolesche Operationen: Es handelt sich um die üblichen Booleschen Mengenoperationen, angewendet auf formale Sprachen:

Vereinigung	$L_1 \cup L_2$	:=	$\{w : w \in L_1 \text{ oder } w \in L_2\}$,
Schnitt	$L_1 \cap L_2$	:=	$\{w : w \in L_1 \text{ und } w \in L_2\}$,
Komplement	$\overline{L_1}$	:=	$\{w : w \in \Sigma^* \text{ und } w \notin L_1\}$.

Vereinigung und Schnitt sind sowohl assoziativ als auch kommutativ.

Kleene-Stern: Der Kleene-Stern bezeichnet die beliebig (aber nur endlich) oft iterierte Konkatenation. Gegeben eine Sprache L, so definieren wir zunächst induktiv Sprachen $L^0, L^1, \ldots$ und darauf basierend dann die durch Anwendung des Kleene-Sterns erhaltene Sprache L^*:

$$\begin{aligned} L^0 &:= \{\varepsilon\}, \\ L^{n+1} &:= L^n \cdot L, \\ L^* &:= \textstyle\bigcup_{n\geq 0} L^n. \end{aligned}$$

Für $L = \{a, ab\}$ gilt also z. B. $L^0 = \{\varepsilon\}$, $L^1 = L$, $L^2 = \{aa, aab, aba, abab\}$, etc. Offensichtlich ist L^* unendlich **gdw. (genau dann, wenn)** $L \neq \emptyset$ und $L \neq \{\varepsilon\}$.

Das leere Wort ist per Definition *immer* in L^* enthalten ist, unabhängig davon, was L für eine Sprache ist. Manchmal verwenden wir auch die Variante ohne L^0:

$$L^+ := \bigcup_{n\geq 1} L^n.$$

Einige einfache Beobachtungen sind $\emptyset^* = \{\varepsilon\}$, $(L^*)^* = L^*$ und $L^* \cdot L^* = L^*$. Es ist hier wichtig, $\emptyset$ (die leere Sprache), $\{\varepsilon\}$ (die Sprache, die das leere Wort enthält) und ε (das leere Wort) sorgsam auseinander zu halten.

Äquivalent könnten wir den Kleene-Stern auch wie folgt definieren:

$$L^* = \{\varepsilon\} \cup \{w \ : \ \text{es ex. } u_1, \ldots, u_n \in L, \text{so dass } w = u_1 \cdot u_2 \cdot \cdots \cdot u_n\}.$$

1.3. Das Induktionsprinzip

Induktive Definitionen. Es gibt eine wichtige Parallele zwischen der Menge Σ^* aller Wörter über einem Alphabet Σ und der Menge $\mathbb{N}$ der natürlichen Zahlen – sowohl was den Aufbau dieser beiden unendlichen Mengen betrifft als auch bezüglich einer grundlegenden Beweistechnik. Eine äquivalente, sehr nützliche Definition der Menge $\mathbb{N}_0$ ist die folgende: $\mathbb{N}_0$ ist die kleinste Menge mit den folgenden Eigenschaften.

(1) $0 \in \mathbb{N}_0$,

(2) Wenn $n \in \mathbb{N}_0$, dann auch $n + 1 \in \mathbb{N}_0$.

Damit „erreichen“ wir alle natürlichen Zahlen, und nur diese:

- wegen (1) ist $0 \in \mathbb{N}_0$,
- wegen $0 \in \mathbb{N}_0$ und (2) ist auch $1 \in \mathbb{N}_0$,
- wegen $1 \in \mathbb{N}_0$ und (2) ist auch $2 \in \mathbb{N}_0$,
- usw.

Andere Zahlen als $0, 1, 2, 3, \ldots$ können wegen der Einschränkung „kleinste Menge mit den Eigenschaften (1) und (2)“ nicht in $\mathbb{N}_0$ vorkommen. Diese Art der Definition wird *induktiv* genannt, denn ein Element n zieht wegen (2) das nächste $n + 1$ nach sich.

Ganz analog dazu können wir die Menge Menge Σ^* aller Wörter über Σ induktiv definieren: Σ^* ist die kleinste Menge mit den folgenden Eigenschaften.

(1) $\varepsilon \in \Sigma^*$

(2) Wenn $w \in \Sigma^*$, dann $wa \in \Sigma^*$ für jedes Symbol $a \in \Sigma$.

Diese Definition ist Äquivalent zur Definition in Abschnitt 1.1 und hat gegenüber dieser gewisse Vorteile, die wir gleich kennen lernen werden. Vorher sei jedoch noch angemerkt, dass wir die induktive Definition von Wörtern nutzen können, um die Länge eines Wortes $|w|$ und die Anzahl der Vorkommen eines Symbols a im Wort w (also $|w|_a$) ebenfalls induktiv zu definieren.

Die Wortlänge $|w|$ ist induktiv definiert als:

(1) $|\varepsilon| = 0$,

(2) $|wa| = |w| + 1$ für alle $w \in \Sigma^*$ und $a \in \Sigma$.

Für die Anzahl der Vorkommen $|w|_a$ gilt:

(1) $|\varepsilon|_a = 0$,

(2) $|wb|_a = \begin{cases} |w|_a + 1 & \text{falls } b = a \\ |w|_a & \text{sonst.} \end{cases}$ für alle $w \in \Sigma^*$ und $b \in \Sigma$

Auf diese beiden Definitionen kommen wir weiter unten im Beispiel eines Induktionsbeweises zurück.

Im Laufe der beiden Vorlesungen werden wir noch weitere Objekte induktiv definieren. Auch die Definition des Kleene-Sterns in Abschnitt 1.2 war induktiv: die unendlich vielen Mengen L^i haben wir per Induktion über die natürliche Zahl i definiert.

Induktive Beweise. Induktive Definitionen haben nicht nur den Vorteil, dass sie die Struktur der zu definierenden Objekte deutlich machen, sondern sie erlauben auch eine elegante und mächtige Beweistechnik für Aussagen der Form „für alle natürlichen Zahlen/Wörter/regulären Ausdrücke/etc. gilt . . . “. Diese Beweistechnik heißt *vollständige Induktion*. Sie wird im Folgenden für natürliche Zahlen und Wörter demonstriert; für alle weiteren induktiv definierten Objekte (reguläre Ausdrücke, Formeln, . . .) funktioniert sie ganz analog.

Stellen wir uns vor, wir möchten zeigen, dass eine Aussage P für alle natürlichen Zahlen gilt. Was P genau besagt, ist zunächst irrelevant; ein konkretes Beispiel schließt sich an. Wir schreiben $P(n)$ für: „die Aussage P gilt für n“, wobei n eine beliebige natürliche Zahl ist. Wir möchten also zeigen:

$$\text{Für alle } n \in \mathbb{N} \text{ gilt: } \; P(n). \qquad (*)$$

Ein Beweis dieser Aussage per vollständiger Induktion hat zwei Schritte:

Induktionsanfang: Wir zeigen $P(0)$.

Induktionsschritt: Wir zeigen, dass für alle natürlichen Zahlen $n \in \mathbb{N}_0$ gilt:

$$\text{wenn } P(n), \text{ dann auch } P(n+1).$$

Um dies zu tun, wählen wir ein $n \in \mathbb{N}_0$ *beliebig* (wegen „für alle“). Dann nehmen wir an, dass $P(n)$ gilt (*Induktionsvoraussetzung* oder *Induktionsannahme*), und zeigen, dass unter dieser Annahme $P(n+1)$ gilt (*Induktionsbeweis*). ❑

Auf diese Weise haben wir gezeigt:

- $P(0)$ (wegen Induktionsanfang),
- $P(1)$ (wegen $P(0)$ und Induktionsschritt),
- $P(2)$ (wegen $P(1)$ und Induktionsschritt),
- usw.,

also $P(n)$ für alle $n \in \mathbb{N}_0$.

Die Induktionsannahme $P(n)$ bedeutet *nicht*, dass wir unerlaubter Weise die zu zeigende Aussage $(*)$ voraussetzen würden: diese besagt, dass $P(n)$ für *alle* $n \in \mathbb{N}$ gilt; die Induktionsannahme wird nur in einer Implikation benutzt: wenn $P(n)$ für *ein* beliebig gewähltes n gilt, dann gilt auch $P(n+1)$.

Verdeutlichen wir uns dies an einem (absichtlich sehr einfach gewählten) Beispiel. Wir wollen zeigen, dass die Summe der ersten n ungeraden Zahlen genau n^2 beträgt (in diesem Fall beginnt die Induktion bei $n = 1$), d. h.:

$$\text{Für alle } n \in \mathbb{N} \text{ gilt: } \; 1 + 3 + 5 + \cdots + (2n-1) = n^2.$$

Hier ist also $P(n)$ die Aussage $1+3+5+\cdots+(2n-1) = n^2$. Der Induktionsbeweis sieht so aus:

Induktionsanfang: $P(1)$ ist $1 = 1^2$, und das gilt offensichtlich (die Induktion beginnt bei 1, da wir obige Aussage für $n \in \mathbb{N}$ und nicht für $n \in \mathbb{N}_0$ machen).

Induktionsschritt: Sei $n \in \mathbb{N}$ beliebig und angenommen es gilt

$$1 + 3 + 5 + \cdots + (2n-1) = n^2 \quad \text{(Induktionsannahme)}.$$

Wir zeigen, dass dann auch die Induktionsbehauptung

$$1 + 3 + 5 + \cdots + (2n-1) + (2n+1) = (n+1)^2$$

gilt. Diese erhalten wir aus der Induktionsannahme, indem wir auf beiden Seiten der Gleichung $2n+1$ addieren (Klammersetzung ist dabei nicht notwendig) und dann auf der rechten Seite die binomische Formel anwenden:

$$\begin{aligned} & 1 + 3 + 5 + \cdots + (2n-1) && = n^2 \\ \Leftrightarrow\ & 1 + 3 + 5 + \cdots + (2n-1) + 2n + 1 && = n^2 + 2n + 1 \\ \Leftrightarrow\ & 1 + 3 + 5 + \cdots + (2n-1) + 2n + 1 && = (n+1)^2 \end{aligned}$$

❑

Für Wörter tun wir nun im Prinzip dasselbe: Wir wollen zeigen, dass eine beliebige Aussage P für alle Wörter über einem Alphabet Σ gilt, d. h.:

$$\text{Für alle } w \in \Sigma^* \text{ gilt: } \; P(w).$$

Dazu müssen die zwei Schritte des Induktionsbeweises den Teilen (1) und (2) der induktiven Definition von Wörtern angepasst werden:

Induktionsanfang: Wir zeigen $P(\varepsilon)$.

Induktionsschritt: Wir zeigen, dass für alle Wörter $w \in \Sigma^*$ und alle Symbole $a \in \Sigma$ gilt:

$$\text{wenn } P(w), \text{ dann auch } P(wa).$$

Um dies zu tun, wählen wir ein $w \in \Sigma^*$ und $a \in \Sigma$ *beliebig* (wegen „für alle“). Dann nehmen wir an, dass $P(w)$ gilt (*Induktionsannahme*), und zeigen, dass unter dieser Annahme $P(wa)$ gilt (*Induktionsschritt*). ❑

Auf diese Weise haben wir gezeigt:

- $P(\varepsilon)$ (wegen Induktionsanfang),
- $P(w)$ für alle Wörter w der Länge 1 (wegen $P(\varepsilon)$ und Induktionsschritt),
- $P(w)$ für alle Wörter w der Länge 2 (wegen dem vorigen und Induktionsschritt),
- usw.,

also $P(w)$ für alle $w \in \Sigma^*$. Weil die Länge eines Wortes direkt mit dem induktiven Aufbau korrespondiert, sprechen wir hier auch oft von *Induktion über die Wortlänge*. Auch hier sei daran erinnert, dass die Induktionsannahme *nicht* mit der zu zeigenden Aussage übereinstimmt, wie oben erklärt.

Als Beispiel für einen Induktionsbeweis wählen wir die folgende einfache Aussage. Wir wollen zeigen, dass die Länge jedes Wortes mindestens so groß ist wie die Anzahl der Vorkommen eines festen Symbols a, d. h.:

$$\text{Für alle } w \in \Sigma^* \text{ und } a \in \Sigma \text{ gilt: } \; |w|_a \leq |w|.$$

Hier ist also $P(w)$ die Aussage „für alle $a \in \Sigma$ gilt: $|w|_a \leq |w|$“. Der Induktionsbeweis sieht so aus:

Induktionsanfang: $P(\varepsilon)$ ist: „für alle $a \in \Sigma$ gilt: $|\varepsilon|_a \leq |\varepsilon|$“, und das gilt offensichtlich wegen $|\varepsilon|_a = |\varepsilon| = 0$ aufgrund der obigen induktiven Definitionen von $|w|_a$ und $|w|$.

Induktionsschritt: Seien $w \in \Sigma^*$ und $a \in \Sigma$ beliebig und angenommen es gilt

$$|w|_a \leq |w|.$$

Wir müssen zeigen, dass dann auch die Induktionsbehauptung

$$|wb|_a \leq |wb|$$

für alle $b \in \Sigma$ gilt (wegen des induktiven Aufbaus von Wörtern!). Die Induktionsbehauptung erhalten wir nun aus der Induktionsannahme durch folgende Überlegungen:

$$|wb|_a \;\leq\; |w|_a + 1 \;\leq\; |w| + 1 \;=\; |wb|$$

Dabei gilt die erste Ungleichung wegen der induktiven Definition von $|wb|_a$ (siehe oben), und die zweite wegen der Induktionsannahme; die letzte Gleichheit gilt wegen der induktiven Definition von $|wb|$ (siehe oben). ❑

2. Endliche Automaten

Endliche Automaten stellen ein einfaches und dennoch sehr nützliches Mittel zur Beschreibung formaler Sprachen dar. Sie können als Abstraktion eines (Hardware- oder Software-)Systems aufgefasst werden. Die charakteristischen Merkmale eines endlichen Automaten sind

- eine endliche Menge von Zuständen, in denen sich der Automat befinden kann.

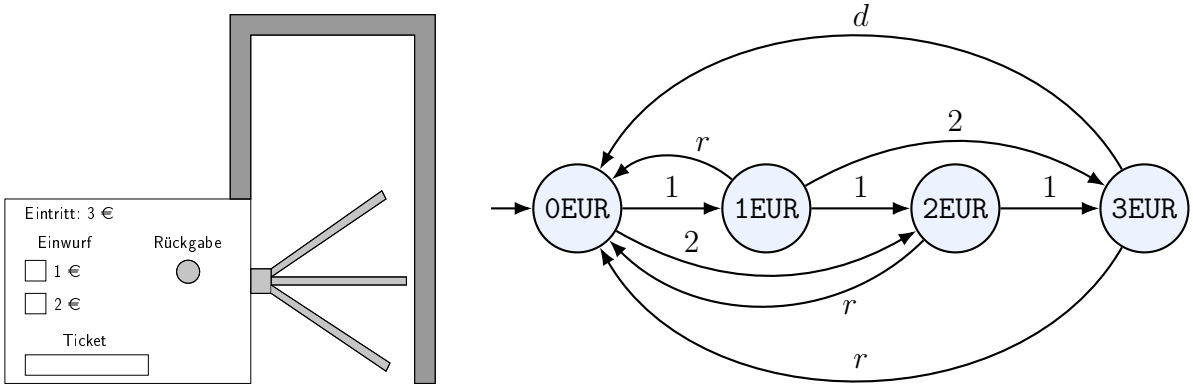
  Ein Zustand beschreibt die aktuelle Konfiguration des Systems. In unserem Kontext ist ein Zustand lediglich ein Symbol (bzw. ein Name) wie q_0, q_1, etc. Insbesondere wird nicht näher beschrieben, was genau diesen Zustand ausmacht (etwa eine bestimmte Belegung eines Registers mit einem konkreten Wert).

- feste Übergangsregeln zwischen Zuständen in Abhängigkeit von der Eingabe.

  Zustandswechsel werden dabei als augenblicklich angenommen, d. h. ein eventueller Zeitverbrauch wird nicht modelliert. Ein Lauf eines Systems ist also einfach eine Folge von Zuständen.

Beispiel (Eintrittsautomat).

Eingabealphabet: $1, 2, r, d$ *(r: Geldrückgabe, d: Drehsperre dreht sich)*
Zustände: `0EUR`, `1EUR`, `2EUR`, `3EUR`



Der dargestellte Automat regelt eine Drehsperre. Es können Münzen im Wert von 1 € oder 2 € eingeworfen werden. Nach Einwurf von 3 € wird die Arretierung der Drehsperre gelöst und der Eintritt freigegeben. Der Automat gibt kein Wechselgeld zurück, sondern nimmt einen zu hohen Betrag nicht an (Münzen fallen durch). Es kann jederzeit den Rückgabeknopf gedrdrückt werden, um den bereits gezahlten Betrag zurückzuerhalten.

In der schematischen Darstellung kennzeichnen die Kreise die internen Zustände und die Pfeile die Übergänge. Die Pfeilbeschriftung gibt die jeweilige Eingabe an, unter der der Übergang erfolgt.

Beachte, dass

- nur der Zustand `3EUR` einen Übergang vom Typ d erlaubt. Dadurch wird modelliert, dass nur durch Einwurf von 3 € der Eintritt ermöglicht wird.
- das Drehen der Sperre als Eingabe angesehen wird. Wir könnten dies auch als Ausgabe modellieren. Wir werden in dieser Vorlesung jedoch keine endlichen Automaten mit Ausgabe (sogenannte Transduktoren) betrachten.

Die Übergänge können als festes Programm betrachtet werden, das der Automat ausführt.

Wir stellen nun den Zusammenhang zu formalen Sprachen her: die Menge der möglichen Eingaben $\{1, 2, r, d\}$ bildet ein Alphabet. Jede (Gesamt-)Eingabe des Automaten ist ein Wort über diesem Alphabet. Wenn wir `3EUR` als Zielzustand betrachten, so bildet die Menge der Eingaben, mittels derer dieser Zustand erreicht werden kann, eine (unendliche) formale Sprache. Diese enthält zum Beispiel das Wort $11r21$.

Wir definieren endliche Automaten nun formal und unterscheiden dabei zwischen der *deterministischen* und der *nichtdeterministischen* Variante.

2.1. Deterministische endliche Automaten

Wir beginnen mit der intuitiven deterministischen Variante, bei der es in jedem Schritt einen eindeutig bestimmten Folgezustand gibt.

Definition 2.1 (DEA)

Ein *deterministischer endlicher Automat (DEA)* ist ein Tupel $\mathcal{A} = (Q, \Sigma, q_s, \delta, F)$, wobei

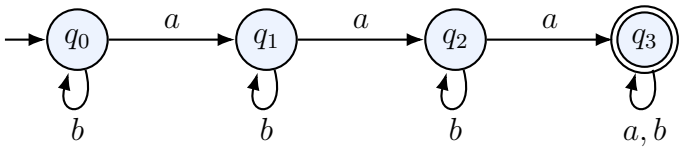
- Q eine endliche Menge von *Zuständen* ist,
- Σ ein *Eingabealphabet* ist,
- $q_s \in Q$ der *Anfangszustand* ist,
- $\delta : Q \times \Sigma \to Q$ die *Übergangsfunktion* ist,
- $F \subseteq Q$ eine Menge von *akzeptierenden Zuständen* ist.

Beispiel 2.2

Der DEA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ mit den Komponenten

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{a, b\}$,
- $q_s = q_0$,
- $\delta(q_0, a) = q_1 \qquad \delta(q_1, a) = q_2 \qquad \delta(q_2, a) = \delta(q_3, a) = q_3$
  $\delta(q_i, b) = q_i \quad$ für $i \in \{0, 1, 2, 3\}$,
- $F = \{q_3\}$

wird graphisch dargestellt als:



Wie im obigen Beispiel werden wir Automaten häufig als kantenbeschriftete Graphen darstellen, wobei die Zustände des Automaten die Knoten des Graphen sind und die Übergänge als Kanten gesehen werden (beschriftet mit einem Alphabetsymbol). Der Startzustand wird durch einen eingehenden Pfeil gekennzeichnet und die akzeptierenden Zustände durch einen Doppelkreis. (Manchmal werden in der Literatur die akzeptierenden Zustände auch durch einen ausgehenden Pfeil gekennzeichnet.)

Intuitiv arbeitet ein DEA, indem er im Startzustand beginnt und ein Wort Symbol für Symbol von links nach rechts liest und dabei entsprechend der Übergangsfunktion den Zustand wechselt. Zu diesem Zweck bestimmt die Übergangsfunktion für jeden Zustand und jedes mögliche gelesene Zeichen einen eindeutig bestimmten Folgezustand. Der DEA akzeptiert das Eingabewort, wenn er sich, *nachdem er das Wort vollständig gelesen hat,* in einem akzeptierenden Zustand befindet. Um dieses Verhalten präzise definieren zu können, müssen wir zunächst die Übergangsfunktion so erweitern, dass sie auch für alle möglichen gelesenen *(Teil-)Wörter* einen Folgezustand bestimmt.

Definition 2.3 (kanonische Fortsetzung von δ)

Die *kanonische Fortsetzung von* $\delta\colon Q \times \Sigma \to Q$ *von einzelnen Symbolen auf beliebige Wörter* ist die Funktion $\hat{\delta} : Q \times \Sigma^* \to Q$, die induktiv (über die Wortlänge) wie folgt definiert ist:

- $\hat{\delta}(q, \varepsilon) := q$,
- $\hat{\delta}(q, wa) := \delta(\hat{\delta}(q, w), a)$.

Beachte: für alle Symbole $a \in \Sigma$ und Zustände $q \in Q$ ist die obige Definition von $\hat{\delta}(q, a)$ identisch mit dem ursprünglichen δ, denn $\hat{\delta}(q, a) = \delta(\hat{\delta}(q, \varepsilon), a) = \delta(q, a)$.

Als Beispiel für Definition 2.3 betrachte wieder den Automaten $\mathcal{A}$ aus Beispiel 2.2. Es gilt $\hat{\delta}(q_0, bbbabbbb) = q_1$ und $\hat{\delta}(q_0, baaab) = q_3$.

Definition 2.4 (Akzeptiertes Wort, akzeptierte Sprache)

Ein DEA $\mathcal{A} = (Q, \Sigma, q_s, \delta, F)$ *akzeptiert* ein Wort $w \in \Sigma^*$, wenn $\hat{\delta}(q_s, w) \in F$. Die von $\mathcal{A}$ *akzeptierte Sprache* ist $L(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{A} \text{ akzeptiert } w\}$.

Der Automat $\mathcal{A}$ aus Beispiel 2.2 akzeptiert die Sprache

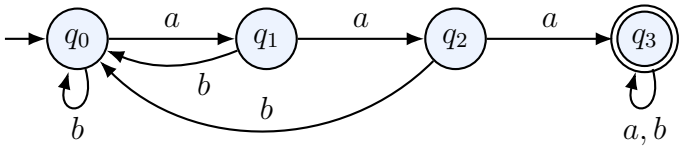
$$L(\mathcal{A}) = \{w \in \{a, b\}^* \ : \ |w|_a \geq 3\}.$$

Mit anderen Worten: er akzeptiert genau diejenigen Wörter über dem Alphabet $\{a, b\}$, in denen das Symbol a mindestens 3-mal vorkommt.

Definition 2.5 (Reguläre Sprachen)
Eine Sprache $L \subseteq \Sigma^*$ heißt *regulär*, wenn es einen DEA $\mathcal{A}$ gibt mit $L = L(\mathcal{A})$.

Wir haben also gerade gesehen, dass die Sprache $L = \{w \in \{a,b\}^* \ : \ |w|_a \geq 3\}$ regulär ist. Folgendes Beispiel liefert eine weitere reguläre Sprache.

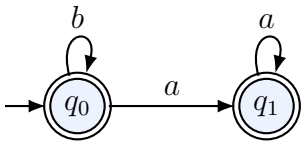
Beispiel 2.6
Der folgende DEA erkennt die Sprache $L = \{w = uaaav \ : \ u, v \in \Sigma^*\}$ mit $\Sigma = \{a, b\}$. Auch diese Sprache ist also regulär.



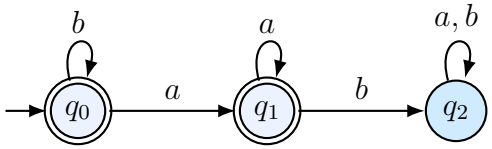
Anmerkung zur Begriffswahl. Anstelle des Begriffs „akzeptierender Zustand“ (auf Englisch: „accepting state“) wird oft „Endzustand“ (bzw. „final state“) verwendet. Dies erweckt jedoch den falschen Eindruck, dass die Berechnung zu Ende sei, sobald der DEA einen solchen Zustand erreicht. Tatsächlich ist die Berechnung aber erst zu Ende, wenn das gesamte Wort gelesen wurde (siehe Def. 2.4). Der Automat kann dabei zwischendurch einen akzeptierenden Zustand einnehmen; dies ist aber für das Akzeptieren des Wortes *am Ende der Berechnung* unerheblich. Reguläre Sprachen werden in der Literatur auch als *erkennbare Sprachen* bezeichnet.

Totalheit der Übergangsfunktion. Beachte: die Übergangsfunktion eines DEAs ist eine *totale Funktion*; es muss also für jede mögliche Kombination von Zustand und Symbol ein Folgezustand angegeben werden.

Beispiel 2.7
Folgendes ist also kein DEA, denn es fehlt ein Übergang für q_1 und b:



Wir erhalten aber leicht einen DEA durch Hinzunahme eines „Papierkorbzustandes“, der alle fehlenden Übergänge aufnimmt (und *kein* akzeptierender Zustand ist):



Die akzeptierte Sprache ist übrigens

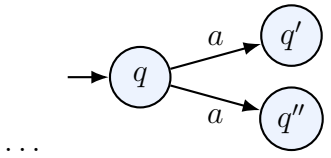
$$L \;=\; \{b\}^* \cdot \{a\}^* \;=\; \{w \in \{a,b\}^* \;:\; ab \text{ ist nicht Infix von } w\}.$$

Randbemerkung. Im Prinzip sind „echte Computer“ ebenfalls endliche Automaten: sie haben nur endlich viel Speicherplatz und daher nur eine endliche Menge möglicher Konfigurationen (Prozessorzustand + Belegung der Speicherzellen). Die Konfigurationsübergänge werden bestimmt durch Verdrahtung und Eingaben (Tastatur, Peripheriegeräte).

Wegen der extrem großen Anzahl von Zuständen sind endliche Automaten aber keine geeignete Abstraktion für Rechner. In einer geeigneten Abstraktion von Rechnern wird der Speicher sogar als unendlich großangenommen. Das wichtigste solche Modell ist die Turingmaschine, die wir später im Detail kennen lernen werden.

2.2. Nichtdeterministische endliche Automaten

Wir generalisieren nun das Automatenmodell des DEA dadurch, dass wir *Nichtdeterminismus* zulassen. In unserem konkreten Fall bedeutet das, dass wir für einen gegebenen Zustand und ein gelesenes Symbol *mehr als einen möglichen Übergang* erlauben; Folgendes ist also möglich:



Ein Automat hat dadurch unter Umständen *mehrere* Möglichkeiten, ein Wort zu verarbeiten. Er akzeptiert seine Eingabe, wenn *eine Möglichkeit existiert*, dabei einen akzeptierenden Zustand zu erreichen.

Nichtdeterminismus ist ein fundamentales Konzept der Informatik, das nicht nur für endliche Automaten eine wichtige Rolle spielt. Wir werden es in dieser Vorlesung noch häufiger verwenden. Dabei werden mehrere Möglichkeiten wie oben immer durch *existentielles Quantifizieren* behandelt (also z. B. wie hier: „es existiert eine Möglichkeit, einen akzeptierenden Zustand zu erreichen“). Natürlich gibt es in der Realität keine nichtdeterministischen Maschinen. Dennoch ist Nichtdeterminismus aus folgenden Gründen von großer Bedeutung:

- Als Modellierungsmittel bei unvollständiger Information.

  Es ist häufig nicht sinnvoll, Ereignisse wie Benutzereingaben, einkommende Nachrichten von anderen Prozessen usw. im Detail zu modellieren, da wir viel zu komplexe Modelle erhalten würden. Stattdessen verwenden wir nichtdeterministische Übergänge ohne genauer zu spezifizieren, wann welcher Übergang verwendet wird.

- Große Bedeutung in der Komplexitätstheorie.

  In der Komplexitätstheorie (Theoretische Informatik 2) geht es unter anderem um die prinzipielle Frage, was effizient berechenbar ist und was nicht. Interessanterweise spielt dabei das zunächst praxisfern wirkende Konzept des Nichtdeterminismus eine zentrale Rolle. Ein Paradebeispiel ist das Problem „P versus NP“ – eines der wichtigsten ungelösten Probleme der Informatik, welches uns in Teil IV begegnen wird.

NEAs ergeben sich dadurch, dass man die Übergangs*funktion* von DEAs durch eine Übergangs*relation* ersetzt.

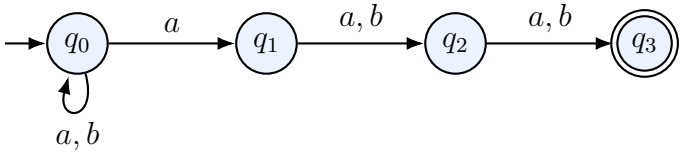
Definition 2.8 (NEA)

Ein *nichtdeterministischer endlicher Automat (NEA)* ist ein Tupel $\mathcal{A} = (Q, \Sigma, q_s, \Delta, F)$, wobei

- Q eine endliche Menge von Zuständen ist,
- Σ ein Eingabealphabet ist,
- $q_s \in Q$ der Anfangszustand ist,
- $\Delta \subseteq Q \times \Sigma \times Q$ *die Übergangsrelation ist*,
- $F \subseteq Q$ eine Menge von akzeptierenden Zuständen ist.

Beispiel 2.9

Folgenden NEA werden wir im Folgenden als durchgängiges Beispiel verwenden:



Dieser Automat ist kein DEA, da es an der Stelle q_0 für die Eingabe a zwei mögliche Übergänge gibt und an der Stelle q_3 keine möglichen Übergänge.

Um das Akzeptanzverhalten von NEAs zu beschreiben, verwenden wir eine etwas andere Notation als bei DEAs.

Definition 2.10 (Lauf)

Ein *Lauf* in einem NEA $\mathcal{A} = (Q, \Sigma, q_s, \Delta, F)$ von einem Zustand $p_0 \in Q$ zu einem Zustand $p_n \in Q$ ist eine Folge $\pi = p_0 \cdot a_1 \cdot p_1 \cdot a_2 \cdot \ldots \cdot a_n \cdot p_n$ über $Q \cup \Sigma$ für ein $n \in \mathbb{N}_0$ mit $\pi(1) = p_0$, $\pi(i) \in Q$ für ungerades i, $\pi(j) \in \Sigma$ für gerades j und $\pi(2n+1) = p_n$, so dass $(p_i, a_{i+1}, p_{i+1}) \in \Delta$ für $i = 0, \ldots, n-1$. Wir stellen den Lauf graphisch wie folgt dar:

$$\pi \;=\; p_0 \xrightarrow{a_1}_{\mathcal{A}} p_1 \xrightarrow{a_2}_{\mathcal{A}} \cdots \xrightarrow{a_n}_{\mathcal{A}} p_n.$$

Der Lauf π hat die *Beschriftung* $w := a_1 \cdots a_n$. Wenn es in $\mathcal{A}$ einen Lauf von $p \in Q$ nach $q \in Q$ mit der Beschriftung w gibt, so schreiben wir:

$$p \overset{w}{\Longrightarrow}_{\mathcal{A}} q.$$

Für $n = 0$ sprechen wir vom *leeren Lauf*, welcher die Beschriftung ε hat.

Im NEA aus Beispiel 2.9 gibt es unter anderem folgende Läufe für die Eingabe *aba*:

$$\begin{aligned} \pi_1 &= q_0 \xrightarrow{a}_{\mathcal{A}} q_1 \xrightarrow{b}_{\mathcal{A}} q_2 \xrightarrow{a}_{\mathcal{A}} q_3 \\ \pi_2 &= q_0 \xrightarrow{a}_{\mathcal{A}} q_0 \xrightarrow{b}_{\mathcal{A}} q_0 \xrightarrow{a}_{\mathcal{A}} q_1 \end{aligned}$$

Wie erwähnt basiert das Akzeptanzverhalten bei Nichtdeterminismus immer auf *existentieller Quantifizierung*:

Definition 2.11 (Akzeptiertes Wort, akzeptierte Sprache)
Der NEA $\mathcal{A} = (Q, \Sigma, q_s, \Delta, F)$ *akzeptiert* das Wort $w \in \Sigma^*$, wenn $q_s \overset{w}{\Longrightarrow}_{\mathcal{A}} q_f$ für ein $q_f \in F$; mit anderen Worten: wenn *es einen Lauf* $p_0 \xrightarrow{a_1}_{\mathcal{A}} \cdots \xrightarrow{a_n}_{\mathcal{A}} p_n$ *gibt*, so dass $p_0 = q_s$ und $p_n \in F$. Die von $\mathcal{A}$ *akzeptierte Sprache* ist $L(\mathcal{A}) = \{w \in \Sigma^* \ : \ \mathcal{A} \text{ akzeptiert } w\}$.

Der NEA aus Beispiel 2.9 akzeptiert also die Eingabe *aba*, weil der oben angegebene Lauf π_1 in einem akzeptierenden Zustand endet. Dabei ist es irrelevant, dass der ebenfalls mögliche Lauf π_2 in einem nicht-akzeptierenden Zustand endet. Nicht akzeptiert wird beispielsweise die Eingabe *baa*, da keiner der möglichen Läufe zu einem akzeptierenden Zustand führt. Der NEA aus Beispiel 2.9 akzeptiert die Sprache

$$L(\mathcal{A}) = \{w \in \{a, b\}^* \ : \ \text{das drittletzte Symbol in } w \text{ ist } a\}.$$

Eine gute Hilfe zum Verständnis von Nichtdeterminismus ist die Metapher des *Ratens*. Intuitiv „rät“ der NEA aus Beispiel 2.9 im Zustand q_0 bei Eingabe von a, ob er sich gerade an der drittletzten Stelle des Wortes befindet oder nicht. Der Automat hat keine Möglichkeit, das sicher zu „wissen“. Wenn er sich für „ja“ entscheidet, so wechselt er in den Zustand q_1 und *verifiziert* mittels der Kette von q_1 nach q_3, dass er richtig geraten hat:

- hat er in Wahrheit das zweitletzte oder letzte Symbol gelesen, so wird der akzeptierende Zustand nicht erreicht; der Automat akzeptiert also nicht;
- ist er weiter als drei Symbole vom Wortende entfernt, so ist in q_3 kein Übergang mehr möglich und der Automat „blockiert“ und akzeptiert ebenfalls nicht.

Die wichtigsten Eigenschaften eines solchen Rate-Ansatzes zum Erkennen einer Sprache L sind, dass (i) für Wörter $w \in L$ es die Möglichkeit gibt, richtig zu raten und (ii) für Wörter $w \notin L$ falsches Raten niemals zur Akzeptanz führt.

Da wir uns bei einem Automaten meist nur für die erkannte Sprache interessieren, bezeichnen wir zwei NEAs als *äquivalent*, wenn sie dieselbe Sprache akzeptieren.

Ohne Nichtdeterminismus, also mittels eines DEA, ist es schwieriger, die Sprache aus Beispiel 2.9 zu erkennen. Es gilt aber interessanterweise, dass wir zu jedem NEA einen äquivalenten DEA finden können. Nichtdeterminismus trägt in diesem Fall also nicht zur Erhöhung der Ausdrucksstärke bei – das ist aber keineswegs immer so, wie wir noch sehen werden. NEAs haben dennoch einen Vorteil gegenüber DEAs: manche Sprachen lassen sich im Vergleich zu DEAs mit erheblich (exponentiell) kleineren NEAs erkennen. Letzteres werden wir im Rahmen der Übungen kurz beleuchten. In der Vorlesung beweisen wir lediglich das folgende klassische Resultat.

Satz 2.12 (Rabin/Scott)

Zu jedem NEA kann ein äquivalenter DEA konstruiert werden.

Bevor wir den Beweis dieses Satzes angeben, skizzieren wir kurz die

Beweisidee:
Der Beweis dieses Satzes verwendet die sogenannte *Potenzmengenkonstruktion*: die Zustandsmenge des DEA ist die Potenzmenge 2^Q der Zustandsmenge Q des NEA. Jeder Zustand des DEA besteht also aus einer *Menge* von NEA-Zuständen.

Sei $\mathcal{A} = (Q, \Sigma, q_s, \Delta, F)$ ein NEA. Nach der Definition von NEAs gilt $w \in L(\mathcal{A})$ gdw. die Menge $\{q \in Q \mid q_s \overset{w}{\Longrightarrow}_{\mathcal{A}} q\} \in 2^Q$ mindestens einen akzeptierenden Zustand enthält. Wir definieren also die Übergangsfunktion δ und Menge F' der akzeptierenden Zustände des DEAs so, dass für alle $w \in \Sigma^*$ gilt:

1. $\hat{\delta}(\{q_s\}, w) = \{q \mid q_s \overset{w}{\Longrightarrow}_{\mathcal{A}} q\}$ und
2. $\{q \mid q_s \overset{w}{\Longrightarrow}_{\mathcal{A}} q\}$ ist akzeptierender Zustand des DEA, wenn mindestens ein akzeptierender Zustand des ursprünglichen NEAs enthalten ist.

Intuitiv simuliert damit der eindeutige Lauf des DEAs auf einer Eingabe w *alle* möglichen Läufe des ursprünglichen NEAs auf w.

Beweis. Sei der NEA $\mathcal{A} = (Q, \Sigma, q_s, \Delta, F)$ gegeben. Der DEA $\mathcal{A}' = (2^Q, \Sigma, \{q_s\}, \delta, F')$ ist definiert durch:

- $\delta(P, a) = \bigcup_{p \in P} \{p' \mid (p, a, p') \in \Delta\}$ für alle $P \in 2^Q$ und $a \in \Sigma$ und
- $F' = \{P \in 2^Q \mid P \cap F \neq \emptyset\}$.

Wir benötigen im Folgenden die

Hilfsaussage: $q' \in \hat{\delta}(\{q_s\}, w)$ gdw. $q_s \overset{w}{\Longrightarrow}_{\mathcal{A}} q'$. $(\star)$

Daraus folgt $L(\mathcal{A}) = L(\mathcal{A}')$, da:

$w \in L(\mathcal{A})$	gdw.	$\exists q \in F : q_s \overset{w}{\Longrightarrow}_{\mathcal{A}} q$	(Def. $L(\mathcal{A})$)
	gdw.	$\exists q \in F : q \in \hat{\delta}(\{q_s\}, w)$	(Hilfsaussage)
	gdw.	$\hat{\delta}(\{q_s\}, w) \cap F \neq \emptyset$	
	gdw.	$\hat{\delta}(\{q_s\}, w) \in F'$	(Def. F')
	gdw.	$w \in L(\mathcal{A}')$	

Beweis der Hilfsaussage mittels Induktion über $|w|$:

Induktionsanfang: $|w| = 0$

$q' \in \hat{\delta}(\{q_s\}, \varepsilon)$ gdw. $q_s = q'$ gdw. $q_s \overset{\varepsilon}{\Longrightarrow}_{\mathcal{A}} q'$

Induktionsvoraussetzung: Die Hilfsaussage ist bereits gezeigt für alle $w \in \Sigma^*$ mit $|w| \leq n$.

Induktionsschritt: $|w| = n + 1$

Sei $w = ua$ mit $u \in \Sigma^*$, $|u| = n$ und $a \in \Sigma$. Es gilt:

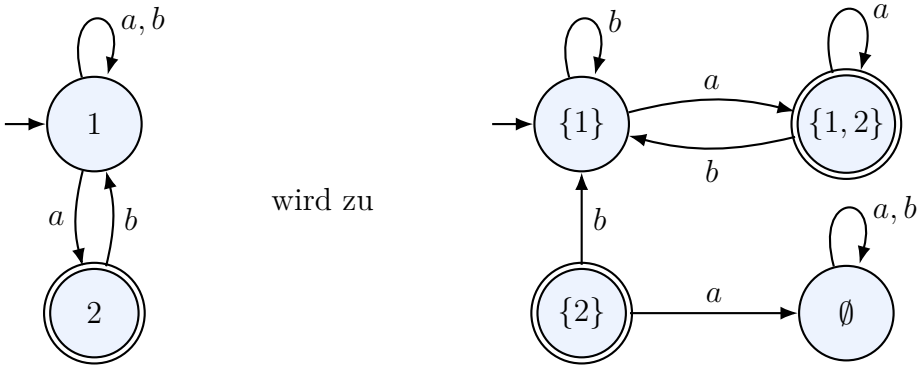
$$\begin{aligned}
\hat{\delta}(\{q_s\}, ua) &= \delta(\hat{\delta}(\{q_s\}, u), a) && \text{(Def. 2.3)} \\
&= \bigcup_{q' \in \hat{\delta}(\{q_s\}, u)} \{q'' \mid (q', a, q'') \in \Delta\} && \text{(Def. } \delta\text{)} \\
&= \bigcup_{q_s \overset{u}{\Longrightarrow}_{\mathcal{A}} q'} \{q'' \mid (q', a, q'') \in \Delta\} && \text{(Induktionsvoraussetzung)} \\
&= \{q'' \ : \ q_s \overset{ua}{\Longrightarrow}_{\mathcal{A}} q''\} && \text{(Def. Lauf)}
\end{aligned}$$

Daraus folgt die Hilfsaussage für $w = ua$.

❑

Beispiel 2.13

Der NEA $\mathcal{A}$ (links) wird mit der Potenzmengenkonstruktion transformiert in den DEA $\mathcal{A}'$ (rechts):



Nachteilig an dieser Konstruktion ist, dass die Zustandsmenge *exponentiell* vergrößert wird. Im Allgemeinen können wir dies wie erwähnt nicht vermeiden; in manchen Fällen kommen wir aber doch mit weniger Zuständen aus. Als einfache Optimierung können wir Zustände weglassen, die mit keinem Wort vom Startzustand aus erreichbar sind. In der Übung werden wir eine Methode kennenlernen, die Potenzmengenkonstruktion systematisch so anzuwenden, dass nicht erreichbare Zustände von vorn herein weggelassen werden – dies reicht allerdings nicht aus, wenn der erzeugte Automat so klein wie möglich werden soll. In Abschnitt 7 werden wir eine allgemeine Methode kennen lernen, um zu einer gegebenen erkennbaren Sprache den kleinstmöglichen DEA zu konstruieren.

2.3. Endliche Automaten mit Wortübergängen

Wir betrachten noch zwei natürliche Varianten von NEAs, die sich in einigen technischen Konstruktionen als sehr nützlich herausstellen. Wir werden sehen, dass sie dieselben Sprachen erkennen können wie NEAs.

Definition 2.14 (NEA mit Wortübergängen, ε-NEA)
Ein *NEA mit Wortübergängen* hat die Form $\mathcal{A} = (Q, \Sigma, q_s, \Delta, F)$, wobei Q, Σ, q_s, F wie beim NEA definiert sind und $\Delta \subseteq Q \times \Sigma^* \times Q$ eine endliche Menge von *Wortübergängen* ist.
Ein *ε-NEA* ist ein NEA mit Wortübergängen der Form (q, ε, q') und (q, a, q') mit $a \in \Sigma$.

Läufe, Laufbeschriftungen und erkannte Sprache werden entsprechend wie für NEAs definiert. Zum Beispiel hat der Lauf

$$q_0 \xrightarrow{ab}_{\mathcal{A}} q_1 \xrightarrow{\varepsilon}_{\mathcal{A}} q_2 \xrightarrow{bb}_{\mathcal{A}} q_3$$

die Beschriftung $ab \cdot \varepsilon \cdot bb = abbb$.

Beachte, dass $q \overset{a}{\Longrightarrow} p$ bedeutet, dass wir von Zustand q den Zustand p erreichen, indem wir zunächst beliebig viele ε-Übergänge ausführen, dann einen a-Übergang und danach wieder beliebig viele[1] ε-Übergänge (im Unterschied zu $q \xrightarrow{a} p$).

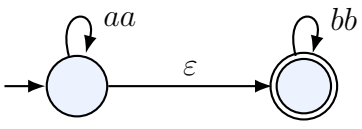
Satz 2.15
Zu jedem NEA mit Wortübergängen kann ein äquivalenter NEA konstruiert werden.

Wir zeigen Satz 2.15 mit einem Umweg über ε-NEAs.

Lemma 2.16
Zu jedem NEA mit Wortübergängen kann ein äquivalenter ε-NEA konstruiert werden.

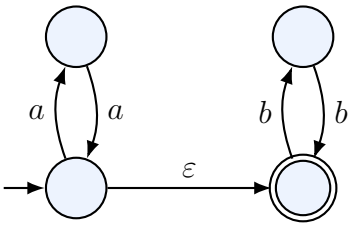
Beweis. Wir ersetzen jeden Wortübergang $(q, a_1 \cdots a_n, q')$ mit $n > 1$ durch Symbolübergänge $(q, a_1, p_1), (p_1, a_2, p_2), \ldots, (p_{n-1}, a_n, q')$, wobei $p_1, \ldots, p_{n-1}$ jeweils neue Hilfszustände sind (die nicht zur Menge der akzeptierenden Zustände hinzugenommen werden). Es ist leicht zu sehen, dass dies einen äquivalenten ε-NEA liefert. ❑

Beispiel 2.17
Der NEA mit Wortübergängen, der durch die folgende Darstellung gegeben ist:



[1] „Beliebig viele“ schließt dabei jeweils „keine“ mit ein.

wird überführt in einen äquivalenten ε-NEA:



Lemma 2.18

Zu jedem ε-NEA kann ein äquivalenter NEA konstruiert werden.

Beweis. Der ε-NEA $\mathcal{A} = (Q, \Sigma, q_s, \Delta, F)$ sei gegeben. Wir konstruieren daraus einen NEA $\mathcal{A}'$ ohne ε- Übergänge wie folgt:

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

- $\Delta' := \left\{ (p, a, q) \in Q \times \Sigma \times Q \mid p \stackrel{a}{\Longrightarrow}_{\mathcal{A}} q \right\}$ und
- $F' := \begin{cases} F \cup \{q_s\} & \text{falls} \quad q_s \stackrel{\varepsilon}{\Longrightarrow}_{\mathcal{A}} q_f \text{ für ein } q_f \in F \\ F & \text{sonst.} \end{cases}$

Intuitiv gesprochen macht also der NEA $\mathcal{A}'$ die „impliziten“ a-Übergänge des ε-NEAs $\mathcal{A}$ explizit: die Übergangsrelation Δ' enthält genau dann einen a-Übergang von p nach q, wenn $\mathcal{A}$ von q nach p kommt, indem er zunächst beliebig viele ε-Übergänge macht, dann einen a-Übergang und danach wieder beliebig viele ε-Übergänge. Entsprechend muss F' nicht nur alle akzeptierenden Zustände von $\mathcal{A}$ enthalten, sondern zusätzlich den Anfangszustand, falls von diesem aus in $\mathcal{A}$ über ε-Kanten ein akzeptierender Zustand erreichbar ist.

Wir zeigen, dass $L(\mathcal{A}) = L(\mathcal{A}')$.

1. $L(\mathcal{A}') \subseteq L(\mathcal{A})$:
   Sei $w = a_1 \cdots a_n \in L(\mathcal{A}')$. Dann gibt es einen Lauf

   $$p_0 \xrightarrow{a_1}_{\mathcal{A}'} p_1 \xrightarrow{a_2}_{\mathcal{A}'} \cdots \xrightarrow{a_{n-1}}_{\mathcal{A}'} p_{n-1} \xrightarrow{a_n}_{\mathcal{A}'} p_n \quad \text{mit} \quad p_0 = q_s,\ p_n \in F'.$$

   Nach Definition von Δ' gibt es auch in $\mathcal{A}$ einen Lauf π von p_0 nach p_n mit Beschriftung w (der unter Umständen zusätzliche ε-Schritte enthält).

   1. Fall: $p_n \in F$
   Dann zeigt π, dass $w \in L(\mathcal{A})$.

   2. Fall: $p_n \in F' \setminus F$, d. h. $p_n = q_s$
   Nach Definition von F' gilt $q_s \stackrel{\varepsilon}{\Longrightarrow}_{\mathcal{A}} p$ für ein $p \in F$. Es gibt also in $\mathcal{A}$ einen Lauf von p_0 über $p_n = q_s$ nach $p \in F$ mit Beschriftung w; daher $w \in L(\mathcal{A})$.

2. $L(\mathcal{A}) \subseteq L(\mathcal{A}')$: Sei $w \in L(\mathcal{A})$.

   1. Fall: $w \neq \varepsilon$. Sei

   $$\pi = p_0 \overset{\varepsilon}{\Longrightarrow}_{\mathcal{A}} p'_0 \xrightarrow{a_1}_{\mathcal{A}} p_1 \overset{\varepsilon}{\Longrightarrow}_{\mathcal{A}} p'_1 \xrightarrow{a_2}_{\mathcal{A}} p_2 \overset{\varepsilon}{\Longrightarrow}_{\mathcal{A}} \cdots \overset{\varepsilon}{\Longrightarrow}_{\mathcal{A}} p'_{n-1} \xrightarrow{a_n}_{\mathcal{A}} p_n$$

   ein Lauf in $\mathcal{A}$ mit $p_0 = q_s$, $p_n \in F$ und Beschriftung w. Nach Definition von Δ' ist

   $$p_0 \xrightarrow{a_1}_{\mathcal{A}'} p_1 \xrightarrow{a_2}_{\mathcal{A}'} \cdots \xrightarrow{a_{n-1}}_{\mathcal{A}'} p_{n-1} \xrightarrow{a_n}_{\mathcal{A}'} p_n$$

   ein Lauf in $\mathcal{A}'$. Aus $p_n \in F$ folgt $p_n \in F'$; also $w \in L(\mathcal{A}')$.

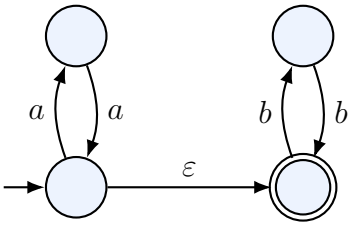
   2. Fall: $w = \varepsilon$

   Wegen $\varepsilon \in L(\mathcal{A})$ gibt es $p \in F$ mit $q_s \overset{\varepsilon}{\Longrightarrow}_{\mathcal{A}} p$. Also $q_s \in F'$ nach Definition von F' und damit $\varepsilon \in L(\mathcal{A}')$.

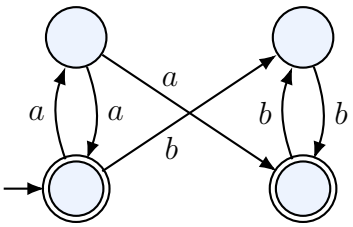
❑

Beispiel 2.19

Der ε-NEA aus Beispiel 2.17



wird in folgenden NEA überführt:



3. Nachweis der Nichtregularität

Nicht jede formale Sprache ist regulär. Es stellt sich also die Frage nach Methoden um nachzuweisen, dass eine Sprache *nicht* regulär ist.

Um nachzuweisen, dass eine gegebene Sprache regulär ist, genügt es, einen endlichen Automaten (DEA oder NEA) anzugeben, der die Sprache akzeptiert. Der Nachweis, dass eine Sprache nicht regulär ist, gestaltet sich schwieriger: wir können nicht alle unendlich viele existierende Automaten durchprobieren, und es genügt auch nicht zu sagen, dass wir keinen funktionierenden Automaten gefunden haben.

Darum verwenden wir die folgende Strategie. Wir etablieren eine allgemeine Eigenschaft, die von jeder regulären Sprache erfüllt wird. Um von einer Sprache zu zeigen, dass sie nicht regulär ist, genügt es dann nachzuweisen, dass sie die Eigenschaft verletzt. Die wichtigste solche Eigenschaft wird durch das bekannte *Pumping-Lemma* beschrieben. Bevor wir es formulieren, verdeutlichen wir die zugrunde liegenden Gedanken an einer Beispielsprache.

Beispiel 3.1

Die Sprache $L = \{a^n b^n \mid n \geq 0\}$ ist *nicht* regulär.

Bevor wir den Beweis erbringen, überlegen wir uns intuitiv, was ein endlicher Automat tun müsste, um L zu akzeptieren. Er müsste für jedes Eingabewort w die Anzahl der gelesenen a's mit denen der danach gelesenen b's vergleichen und w genau dann akzeptieren, wenn diese Anzahlen übereinstimmen. Da ein endlicher Automat jedoch nur eine endliche Anzahl von Zuständen hat, aber die Zahl n in der Beschreibung von L unbegrenzt groß werden kann, ist dieses Zählen von keinem endlichen Automaten zu bewerkstelligen.

Dieser Gedankengang gibt nur *Intuitionen* wieder. Er taugt nicht als formaler Beweis, denn er basiert auf der nur schwer exakt zu belegenden Annahme, dass die einzige Möglichkeit L zu erkennen auf dem Zählen von a's und b's basiert. Ein korrekter Beweis für die Nichtregularität von L sieht z.B. so aus:

Beweis. Angenommen L ist regulär. Dann gibt es einen NEA $\mathcal{A}$ mit $L(\mathcal{A}) = L$. Dieser NEA hat eine endliche Anzahl von Zuständen, sagen wir n_0 Stück. Wir betrachten das Wort $w = a^{n_0} b^{n_0} \in L$. Da $w \in L(\mathcal{A})$, gibt es einen Lauf vom Anfangszustand q_0 zu irgendeinem Endzustand q_f in $\mathcal{A}$, der mit w beschriftet ist. Die ersten $n_0 + 1$ Zustände (einschließlich q_0) auf diesem Lauf werden durch Lesen der a's erreicht. Da $\mathcal{A}$ nur n_0 Zustände hat, muss unter diesen $n_0 + 1$ Zuständen ein Zustand q doppelt vorkommen. Der genannte Lauf lässt sich also auch schreiben als

$$q_0 \overset{x}{\Longrightarrow}_{\mathcal{A}} q \overset{y}{\Longrightarrow}_{\mathcal{A}} q \overset{z}{\Longrightarrow}_{\mathcal{A}} q_f \,,$$

wobei x, y, z Teilwörter von w sind mit $w = xyz$ und $|y| > 0$. Weil q in diesem Lauf doppelt auftaucht, können wir den Teillauf zwischen diesen beiden Vorkommen verdoppeln

und erhalten einen Lauf

$$q_0 \overset{x}{\Longrightarrow}_{\mathcal{A}} q \overset{y}{\Longrightarrow}_{\mathcal{A}} q \overset{y}{\Longrightarrow}_{\mathcal{A}} q \overset{z}{\Longrightarrow}_{\mathcal{A}} q_f$$

auf dem Wort $xyyz$. Sei $k_1 = |x|, k_2 = |y| > 0, k_3 = |z| - n_0$. Damit enthält $L(\mathcal{A})$ das Wort $xyyz$, welches die Form

$$a^{k_1}\, a^{2k_2}\, a^{k_3}\, b^{n_0}$$

mit $k_1 + 2k_3 + k_3 > n_0$ hat. Also hat $xyyz$ mehr a's als b's und kann damit nicht zu L gehören, obwohl es von $\mathcal{A}$ akzeptiert wird. Dies ist ein Widerspruch; also war die Annahme falsch, dass L regulär ist! ❑

Der dem letzten Beweis zugrunde liegende Hauptgedanke lässt sich verallgemeinern: Wenn wir annehmen, dass eine Sprache L von einem endlichen Automaten $\mathcal{A}$ akzeptiert wird, und ein Wort $w \in L$ betrachten, dessen Länge größer ist als die Zahl der Zustände von $\mathcal{A}$, dann muss $\mathcal{A}$ beim Akzeptieren von w einen Zustand q doppelt besuchen. Deshalb enthält L auch sämtliche Wörter, die wir erhalten, wenn wir den Teil zwischen diesen beiden Besuchen von q beliebig vervielfachen. Dies ist im Wesentlichen die Aussage des Pumping-Lemmas.

Lemma 3.2 (Pumping-Lemma für reguläre Sprachen)
Es sei L eine reguläre Sprache. Dann gibt es eine natürliche Zahl $n_0 \geq 1$, so dass gilt: Jedes Wort $w \in L$ mit $|w| \geq n_0$ lässt sich zerlegen in $w = xyz$, so dass

- $y \neq \varepsilon$ *und* $|xy| \leq n_0$
- $xy^kz \in L$ *für alle* $k \geq 0$.

Beweis. Sei $\mathcal{A} = (Q, \Sigma, q_s, \Delta, F)$ ein NEA mit $L(\mathcal{A}) = L$. Wir wählen $n_0 = |Q|$. Sei nun $w = a_1 \cdots a_m \in L$ ein Wort mit $m \geq n_0$. Dann existiert ein Lauf

$$p_0 \xrightarrow{a_1}_{\mathcal{A}} p_1 \xrightarrow{a_2}_{\mathcal{A}} \cdots \xrightarrow{a_{n_0}}_{\mathcal{A}} p_{n_0} \xrightarrow{a_{n_0+1}}_{\mathcal{A}} \cdots \xrightarrow{a_m}_{\mathcal{A}} p_m$$

mit $p_0 = q_s$ und $p_m \in F$. Wegen $n_0 = |Q|$ können die $n_0 + 1$ Zustände $p_0, \ldots, p_{n_0}$ nicht alle verschieden sein. Es gibt also i, j mit $0 \leq i < j \leq n_0$ und $p_i = p_j$. Wir wählen

$$x := a_1 \cdots a_i, \quad y := a_{i+1} \cdots a_j, \quad z := a_{j+1} \cdots a_m.$$

Offensichtlich gilt $y \neq \varepsilon$ (da $i < j$) und $|xy| \leq n_0$ (da $j \leq n_0$) sowie

$$q_0 = p_0 \overset{x}{\Longrightarrow}_{\mathcal{A}} p_i \overset{y}{\Longrightarrow}_{\mathcal{A}} p_i = p_j \overset{z}{\Longrightarrow}_{\mathcal{A}} p_m \in F.$$

Es gilt für alle $k \geq 0$ auch $p_i \overset{y^k}{\Longrightarrow}_{\mathcal{A}} p_i$, also

$$q_s = p_0 \overset{x}{\Longrightarrow}_{\mathcal{A}} p_i \overset{y^k}{\Longrightarrow}_{\mathcal{A}} p_i = p_j \overset{z}{\Longrightarrow}_{\mathcal{A}} p_m \in F,$$

was $xy^kz \in L$ zeigt. ❑

Beispiel 3.3
Wir benutzen jetzt das Pumping-Lemma um zu zeigen:

$$L = \{a^n b^n \mid n \geq 0\} \text{ ist } \textit{nicht} \text{ regulär.}$$

Beweis. Wir führen einen Widerspruchsbeweis und nehmen an, L sei regulär. Es gibt also eine Zahl n_0 mit den in Lemma 3.2 beschriebenen Eigenschaften. Wähle das Wort

$$w = a^{n_0} b^{n_0} \in L.$$

Da $|w| \geq n_0$, gibt es eine Zerlegung $a^{n_0} b^{n_0} = xyz$ mit $|y| \geq 1$ und $|xy| \leq n_0$, so dass $xy^k z \in L$ für alle $k \geq 0$. Wegen $|xy| \leq n_0$ muss y ganz in a^{n_0} liegen. Also ist

$$x = a^{k_1}, \quad y = a^{k_2}, \quad z = a^{k_3} b^{n_0}$$

mit $k_2 > 0$ und $n_0 = k_1 + k_2 + k_3$. Damit ist aber

$$xy^0 z = xz = a^{k_1+k_3} b^{n_0} \notin L,$$

da $k_1 + k_3 < n_0$. Widerspruch. Deshalb muss die Annahme „L ist regulär“ falsch sein. ❑

Beweise mit Hilfe des Pumping-Lemmas werden einfacher, wenn wir die logische Struktur der Aussage genauer betrachten: Lemma 3.2 hat die Form einer Implikation

$$\text{wenn } X, \text{ dann } Y,$$

wobei X die Aussage „L ist eine reguläre Sprache“ und Y die Aussage „es gibt eine natürliche Zahl $n \geq 1, \ldots$“ darstellt. Anstatt wie in Beispiel 3.3 einen Widerspruchsbeweis zu führen, ist es bequemer, die *Kontraposition* der obigen Aussage zu verwenden:

$$\text{wenn nicht } Y, \text{ dann nicht } X.$$

Da eine Implikation und ihre Kontraposition logisch äquivalent sind, ergibt sich folgende direkte Konsequenz aus dem Pumping-Lemma.

Korollar 3.4 (Pumping-Lemma als Kontrapositiv)
Angenommen, für eine Sprache L gilt das Folgende:

Für alle *natürlichen Zahlen $n_0 \geq 1$*
  gibt es *ein Wort $w \in L$ mit $|w| \geq n_0$, so dass*
    für alle *Zerlegungen $w = xyz$ mit $y \neq \varepsilon$ und $|xy| \leq n_0$ gilt:*
      es gibt *$k \geq 0$ mit $xy^k z \notin L$.*

Dann ist L ***nicht*** *regulär.*

Diese äquivalente Formulierung des Pumping-Lemmas legt eine *spieltheoretische Sicht* nahe: Wir spielen in Runden gegen einen Gegner. Der Gegner möchte zeigen, dass die betrachtete Sprache L regulär ist; wir möchten zeigen, dass sie es nicht ist. In den Zeilen, die mit „für alle“ beginnen, ist der Gegner am Zug; in den Zeilen, die mit „es gibt“ beginnen, sind wir an der Reihe. Wenn wir das Spiel *stets gewinnen können* – und zwar unabhängig davon, was der Gegner tut – dann ist die Eigenschaft aus Korollar 3.4 erfüllt, also ist L nicht regulär.

Dieses Spiel verdeutlichen wir an den folgenden zwei Beispielen.

Beispiel 3.5

Die Sprache $L = \{a^n \ : \ n \text{ ist Quadratzahl}\}$ ist nicht regulär.

Beweis. Sei $n_0 \geq 1$ eine natürliche Zahl (vom Gegner gewählt, also können wir n_0 nicht näher bestimmen). Wir wählen das Wort $w = a^m$ mit $m = (n_0 + 1)^2$. Für dieses Wort gilt $w \in L$ und $|w| > n_0$. Sei nun xyz eine Zerlegung (vom Gegner gewählt) mit den Eigenschaften

$$y \neq \varepsilon \quad \text{und} \quad |xy| \leq n_0.$$

Wir müssen nun ein $k \geq 0$ finden, so dass $xy^kz \notin L$. Dazu beobachten wir zunächst, dass wegen $|w| > n_0$ und $|xy| \leq n_0$ gilt: $z \neq \varepsilon$. Wir möchten nun zeigen, dass wir y so „aufpumpen“ können, dass die Wortlänge die Form $s^2 + t$ bekommt, für geeignete s, t mit $0 < t < s$. So eine Zahl kann nämlich nie eine Quadratzahl sein, denn es gilt

$$s^2 \quad < \quad s^2 + t \quad < \quad s^2 + 2s + 1 \quad = \quad (s + 1)^2.$$

Wenn wir nun $k = 4 \cdot |y| \cdot |w|^2$ wählen, erreichen wir dieses Ziel, denn es gilt:

$$\begin{aligned} |xy^kz| &= |y| \cdot k + |x| + |z| \\ &= 4 \cdot |y|^2 \cdot |w|^2 + |x| + |z| \\ &= (2 \cdot |y| \cdot |w|)^2 + |x| + |z| \end{aligned}$$

Mit $s = 2 \cdot |y| \cdot |w|$ und $t = |x| + |z|$ gilt nun wie gewünscht $|xy^kz| = s^2 + t$ und $0 < t < s$. Also ist $xy^kz \notin L$, was zu zeigen war (d. h. wir gewinnen das Spiel). ❑

Der Beweis im vorangehenden Beispiel erfordert zwei kreative Schritte von uns: wir müssen (in Abhängigkeit von n_0, das der Gegner uns vorgibt) ein geeignetes Wort w wählen und dann (in Abhängigkeit von der Zerlegung $w = xyz$, die der Gegner uns vorgibt) ein geeignetes k finden, so dass wir logisch zwingend argumentieren können, dass xy^kz nicht in L sein kann. Der erste Schritt war in diesem Beispiel recht naheliegend: $w = a^{(n_0)^2}$ hätte es nicht erlaubt, $z \neq \varepsilon$ zu folgern ($n_0 = 1$ ist möglich); also haben wir $w = a^{(n_0+1)^2}$ gewählt. Im nächsten Beispiel müssen wir bei der Wahl von w etwas kreativer sein.

Beispiel 3.6

Die Sprache $L = \{a^n b^m \mid n \neq m\}$ ist *nicht* regulär.

Beweis. Sei wieder $n_0 \geq 1$ eine natürliche Zahl (vom Gegner gewählt). Wir wählen das Wort $w = a^{n_0} b^{n_0! + n_0}$, wobei $n_0! = 1 \cdot \ldots \cdot n_0$ (Fakultätsoperation). Der intuitive Grund für diese Wahl ist, dass wir (a) wie in Beispiel 3.3 erzwingen wollen, dass das y der Zerlegung vollständig in den a's liegt, und (b) wir durch „Aufpumpen“ von y erreichen müssen, dass das so gepumpte Wort genauso viele a's wie b's enthält. Da das Aufpumpen einer

a-Folge der Multiplikation der Anzahl der a's entspricht, müssen wir die Anzahl der b's im Wesentlichen so wählen, dass sie durch alle Zahlen bis n_0 teilbar ist, und das ist der Fall für die Fakultätsoperation.

Sei nun xyz eine Zerlegung (vom Gegner gewählt) mit den Eigenschaften

$$y \neq \varepsilon \quad \text{und} \quad |xy| \leq n_0,$$

d. h. wie gewünscht liegt y vollständig in den a's. Also gilt:

$$x = a^{k_1}, \quad y = a^{k_2}, \quad z = a^{k_3} b^{n_0! + n_0}$$

für geeignete k_1, k_2, k_3 mit $k_1 + k_2 + k_3 = n_0$ und $k_2 > 0$ (da $y \neq \varepsilon$). Jetzt zahlt sich die Wahl der Fakultätsoperation aus: wegen $0 < k_2 \leq n_0$ gibt es nämlich eine Zahl ℓ mit:

$$\ell \cdot k_2 = n_0!$$

Wir können nun $k = \ell + 1$ wählen, denn dann ist die Anzahl a's im Wort $xy^{\ell+1}z$:

$$\begin{aligned} k_1 + (\ell+1)k_2 + k_3 &= \underbrace{k_1 + k_2 + k_3}_{=n_0} + \underbrace{\ell \cdot k_2}_{=n_0!} \\ &= n_0 + n_0! \end{aligned}$$

Da die b's nur im Teilwort z auftreten, ist deren Anzahl unverändert gleich $n_0 + n_0!$. Also ist $xy^{\ell+1}z = a^{n_0+n_0!}b^{n_0+n_0!} \notin L$, was zu zeigen war (d. h. wir gewinnen das Spiel). ❑

Mit Hilfe des Pumping-Lemmas gelingt es leider nicht immer, die Nichtregularität einer Sprache nachzuweisen, denn es gibt Sprachen, die nicht regulär sind, aber trotzdem die in Lemma 3.2 beschriebene Pumping-Eigenschaft erfüllen. Anders ausgedrückt ist die Pumping-Eigenschaft aus Lemma 3.2 zwar *notwendig* für die Regularität einer Sprache, aber nicht *hinreichend*.

Beispiel 3.7

Die Sprache $L = \{a^m b^n c^n \ : \ m, n \geq 1\} \cup \{b^m c^n \ : \ m, n \geq 0\}$ ist nicht regulär, erfüllt aber die Eigenschaften von Lemma 3.2.

Wir zeigen, dass es eine natürliche Zahl $n_0 \geq 1$ gibt, so dass gilt: Jedes Wort $w \in L$ mit $|w| \geq n_0$ lässt sich zerlegen in $w = xyz$, so dass

- $y \neq \varepsilon$ und $|xy| \leq n_0$
- $xy^k z \in L$ für alle $k \geq 0$.

Wir betrachten $n_0 = 3$. Es sei nun $w \in L$ mit $|w| \geq 3$ beliebig. Es tritt einer der folgenden drei Fälle ein.

1. $w = a^m b^n c^n$ mit $m, n \geq 1$ (w ist aus dem „1. Teil“ von L),
2. $w = b^m c^n$ mit $m \geq 1$ und $n \geq 0$ (w ist aus dem „2. Teil“ von L und beginnt mit b),
3. $w = c^n$ mit $n \geq 1$ (w ist aus dem „2. Teil“ von L und beginnt mit c, da $|w| \geq 3$).

Wir zeigen: in jedem dieser drei Fälle lässt sich w zerlegen in $w = xyz$ mit $y \neq \varepsilon$ und $|xy| \leq n_0$ sowie $xy^k z \in L$ für alle $k \geq 0$.

1. Fall. Wenn $w = a^m b^n c^n$ mit $m, n \geq 1$, dann betrachte die Zerlegung

$$x = \varepsilon, \quad y = a, \quad z = a^{m-1} b^n c^n.$$

Dann ist für *jedes* $k \geq 0$ das Wort

$$xy^k z = a^{k+(m-1)} b^n c^n$$

in L, denn die Anzahlen der a's und b's sind im „1. Teil“ unabhängig (und falls $(m-1)+k = 0$, ist $xy^k z$ im „2. Teil“ von L).

2. Fall. Wenn $w = b^m c^n$ mit $m \geq 1$ und $n \geq 0$, dann betrachte die Zerlegung

$$x = \varepsilon, \quad y = b, \quad z = b^{m-1} c^n.$$

Dann ist für *jedes* $k \geq 0$ das Wort

$$xy^k z = b^{k+(m-1)} c^n$$

in L, denn die Anzahlen der b's und c's sind im „2. Teil“ unabhängig.

3. Fall. Wenn $w = c^n$ mit $n \geq 1$, dann betrachte die Zerlegung

$$x = \varepsilon, \quad y = c, \quad z = c^{n-1}.$$

Dann ist für *jedes* $k \geq 0$ das Wort

$$xy^k z = c^{k+(n-1)}$$

in L (wiederum im „2. Teil“).

Wir können also in keinem der drei Fälle erreichen, dass $xy^k z \notin L$. Dadurch misslingt der Nachweis, dass L nicht regulär ist.

In Kapitel 7 werden wir ein Werkzeug kennen lernen, mit dem der Nachweis der Nichtregularität dieser Sprache L gelingt.

In der Literatur existieren verschärfte (und kompliziertere) Varianten des Pumping-Lemmas, die dann auch hinreichend sind (z. B. Jaffes Pumping-Lemma). Diese Varianten liefern also eine automatenunabhängige Charakterisierung der regulären Sprachen.

4. Abschlusseigenschaften

Endliche Automaten definieren eine ganze *Klasse* von Sprachen: die regulären Sprachen. Statt die Eigenschaften einzelner Sprachen zu studieren (wie z. B. Regularität) können wir auch die Eigenschaften ganzer Sprachklassen betrachten. Wir interessieren uns hier insbesondere für Abschlusseigenschaften, zum Beispiel unter Schnitt: wenn L_1 und L_2 reguläre Sprachen sind, dann ist auch $L_1 \cap L_2$ eine reguläre Sprache.

Es stellt sich heraus, dass die Klasse der regulären Sprachen unter den meisten natürlichen Operationen abgeschlossen ist. Diese Eigenschaft ist für viele technische Konstruktionen und Beweise sehr nützlich. Wie wir beispielsweise sehen werden, können wir manchmal die Anwendung des Pumping-Lemmas durch ein viel einfacheres Argument ersetzen, das auf Abschlusseigenschaften beruht. Später werden wir sehen, dass andere interessante Sprachklassen *nicht* unter allen natürlichen Operationen abgeschlossen sind.

Satz 4.1 (Abschlusseigenschaften regulärer Sprachen)
Sind L_1 und L_2 regulär, so sind auch die folgenden Sprachen regulär.

- $L_1 \cup L_2$ *(Vereinigung),*
- $\overline{L_1}$ *(Komplement),*
- $L_1 \cap L_2$ *(Schnitt),*
- $L_1 \cdot L_2$ *(Konkatenation),*
- L_1^* *(Kleene-Stern).*

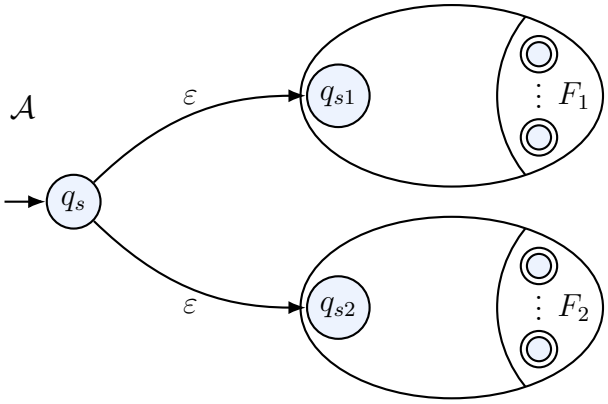
Beweis. Seien $\mathcal{A}_i = (Q_i, \Sigma, q_{si}, \Delta_i, F_i)$ zwei NEAs für L_i $(i = 1, 2)$. O. B. d. A. gelte $Q_1 \cap Q_2 = \emptyset$.

(1) Abschluss unter Vereinigung:

Der folgende ε-NEA erkennt $L_1 \cup L_2$:
$\mathcal{A} := (Q_1 \cup Q_2 \cup \{q_s\}, \Sigma, q_s, \Delta, F_1 \cup F_2)$, wobei

- $q_s \notin Q_1 \cup Q_2$ und
- $\Delta := \Delta_1 \cup \Delta_2 \cup \{(q_s, \varepsilon, q_{s1}), (q_s, \varepsilon, q_{s2})\}$.

Schematisch sieht der *Vereinigungsautomat* $\mathcal{A}$ so aus.



Nach Lemma 2.18 gibt es zu $\mathcal{A}$ einen äquivalenten NEA (ohne ε-Übergänge).

(2) Abschluss unter Komplement:

Einen DEA für $\overline{L_1}$ erhalten wir wie folgt.

Zunächst wenden wir die Potenzmengenkonstruktion an, um einen zu $\mathcal{A}_1$ äquivalenten DEA $\mathcal{A} = (Q, \Sigma, q_s, \delta, F)$ zu konstruieren. Den DEA für $\overline{L_1}$ erhalten wir nun durch Vertauschen der akzeptierenden Zustände mit den nicht akzeptierenden Zuständen:

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

Es gilt nämlich:

$$\begin{array}{lll} w \in \overline{L_1} & \text{gdw.} & w \notin L(\mathcal{A}_1) \\ & \text{gdw.} & w \notin L(\mathcal{A}) \\ & \text{gdw.} & \hat{\delta}(q_s, w) \notin F \\ & \text{gdw.} & \hat{\delta}(q_s, w) \in Q \setminus F \\ & \text{gdw.} & w \in L(\overline{\mathcal{A}}) \end{array}$$

Beachte: Es ist wichtig, dass der Automat zunächst in einen deterministischen Automat transformiert wird. Für einen NEA erhalten wir durch Komplementieren der akzeptierenden Zustände nicht notwendigerweise den Komplementautomaten.

(3) Abschluss unter Schnitt:

Wegen $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ folgt (3) aus (1) und (2).

Da die Potenzmengenkonstruktion, die wir für $\overline{L_1}$ und $\overline{L_2}$ benötigen, recht aufwändig ist und exponentiell große Automaten liefert, kann es günstiger sein, direkt einen NEA für $L_1 \cap L_2$ zu konstruieren, den so genannten *Produktautomaten*:

$$\mathcal{A} := (Q_1 \times Q_2, \Sigma, (q_{s1}, q_{s2}), \Delta, F_1 \times F_2)$$

mit

$$\Delta := \{((q_1, q_2), a, (q_1', q_2')) \mid (q_1, a, q_1') \in \Delta_1 \text{ und } (q_2, a, q_2') \in \Delta_2\}.$$

Ein Übergang in $\mathcal{A}$ ist also genau dann möglich, wenn der entsprechende Übergang in $\mathcal{A}_1$ *und* $\mathcal{A}_2$ möglich ist.

Behauptung. $L(\mathcal{A}) = L_1 \cap L_2$.

Sei $w = a_1 \cdots a_n$. Dann ist $w \in L(\mathcal{A})$ gdw. es gibt einen Lauf

$$(q_{1,0}, q_{2,0}) \xrightarrow{a_1}_{\mathcal{A}} (q_{1,1}, q_{2,1}) \cdots (q_{1,n-1}, q_{2,n-1}) \xrightarrow{a_n}_{\mathcal{A}} (q_{1,n}, q_{2,n})$$

mit $(q_{1,0}, q_{2,0}) = (q_{s1}, q_{s2})$ und $(q_{1,n}, q_{2,n}) \in F_1 \times F_2$. Nach Konstruktion von $\mathcal{A}$ ist das der Fall gdw. für jedes $i \in \{1, 2\}$

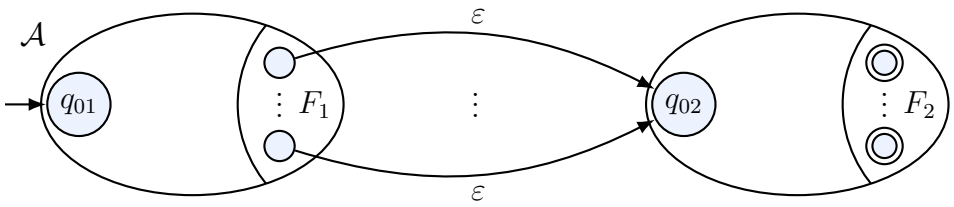
$$q_{i,0} \xrightarrow{a_1}_{\mathcal{A}_i} q_{i,1} \cdots q_{i,n-1} \xrightarrow{a_n}_{\mathcal{A}_i} q_{i,n}$$

ein Lauf ist mit $q_{i,0} = q_{si}$ und $q_{i,n} \in F_i$. Solche Läufe existieren gdw. $w \in L_1 \cap L_2$.

(4) Abschluss unter Konkatenation:

Der folgende ε-NEA erkennt $L_1 \cdot L_2$:

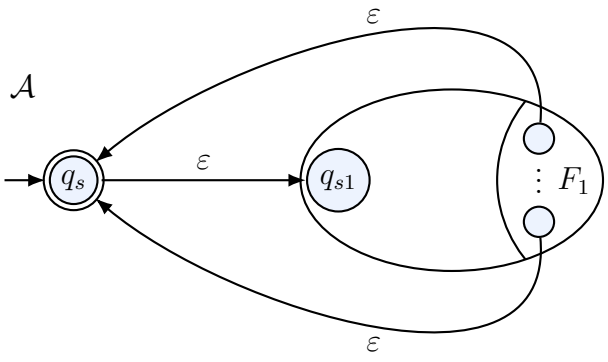
$$\mathcal{A} := (Q_1 \cup Q_2, \Sigma, q_{s1}, \Delta, F_2), \quad \text{wobei}$$
$$\Delta := \Delta_1 \cup \Delta_2 \cup \{(q_f, \varepsilon, q_{s2}) \ : \ q_f \in F_1\}$$



(5) Abschluss unter Kleene-Stern:

Der folgende ε-NEA erkennt L_1^*:

$$\mathcal{A} := (Q_1 \cup \{q_0\}, \Sigma, q_s, \Delta, \{q_s\}), \quad \text{wobei } q_s \notin Q_1 \text{ und}$$
$$\Delta := \Delta_1 \cup \{(q_f, \varepsilon, q_s) \ : \ q_f \in F_1\} \cup \{(q_s, \varepsilon, q_{s1})\}$$



Beachte: Diese Konstruktion funktioniert nicht, wenn wir anstelle des neuen Zustands q_s den ursprünglichen Startzustand q_{s1} verwenden!

❑

Beachte: In den Fällen 1, 3, 4 und 5 ist die Größe des konstruierten Automaten jeweils polynomiell in der Größe der Automaten für L_1 und L_2. In Fall 2 (Komplement) kann der konstruierte Automat exponentiell groß werden, wenn wir mit einem NEA beginnen.

Wir können derartige Abschlusseigenschaften verwenden, um Nichtregularität einer Sprache L nachzuweisen.

Beispiel 4.2

$L := \{a^n b^m \mid n \neq m\}$ ist *nicht* regulär (vgl. Beispiel 3.6). Anstatt dies direkt mit dem Pumping-Lemma (Lemma 3.2) zu zeigen, können wir auch verwenden, dass bereits bekannt ist, dass die Sprache $L' := \{a^n b^n \mid n \geq 0\}$ *nicht* regulär ist. Wäre nämlich L regulär, so auch $L' = \overline{L} \cap (\{a\}^* \cdot \{b\}^*)$. Da wir schon wissen, dass L' nicht regulär ist, kann auch L nicht regulär sein.

5. Entscheidungsprobleme

Wenn wir einen endlichen Automaten in einer konkreten Anwendung einsetzen möchten, so ist es wichtig, dass wir uns zunächst vor Augen führen, was *genau* wir mit dem Automaten anfangen möchten. In Abhängigkeit davon können wir dann die konkreten, in dieser Anwendung zu lösenden algorithmischen Probleme bestimmen.

Wir betrachten drei typische Probleme im Zusammenhang mit regulären Sprachen.

- **das Wortproblem:**

  Gegeben ein endlicher Automat $\mathcal{A}$ und eine Eingabe $w \in \Sigma^*$ für $\mathcal{A}$, entscheide ob $w \in L(\mathcal{A})$.

- **das Leerheitsproblem:**

  Gegeben ein endlicher Automat $\mathcal{A}$, entscheide ob $L(\mathcal{A}) = \emptyset$.

- **das Äquivalenzproblem:**

  Gegeben endliche Automaten $\mathcal{A}_1$ und $\mathcal{A}_2$, entscheide ob $L(\mathcal{A}_1) = L(\mathcal{A}_2)$.

Wir werden Algorithmen für alle diese Probleme entwerfen und deren Laufzeit analysieren: wie viele elementare Rechenschritte macht der Algorithmus auf Eingaben der Länge n; siehe Appendix A für weitere Erklärungen zur Laufzeit- bzw. Komplexitätsanalyse von Algorithmen.

In den obigen Problemen können die als Eingabe gegebenen endlichen Automaten entweder DEAs oder NEAs sein. Für die Anwendbarkeit der Algorithmen macht das im Prinzip keinen Unterschied, da wir zu jedem NEA ja einen äquivalenten DEA konstruieren können (die Potenzmengenkonstruktion aus Satz 2.12 kann leicht algorithmisch implementiert werden). Bezüglich der Laufzeit kann es aber einen erheblichen Unterschied geben, da der Übergang von NEAs zu DEAs einen exponentiell größeren Automaten liefert und damit auch die Laufzeit exponentiell größer wird.

5.1. Das Wortproblem für DEAs

Ist der Eingabeautomat $\mathcal{A}$ für das Wortproblem ein DEA $\mathcal{A} = (Q, \Sigma, q_s, \delta, F)$, so können wir beginnend mit q_s durch wiederholte Anwendung von δ berechnen, in welchem Zustand $\mathcal{A}$ nach dem Lesen der Eingabe w ist. Wir müssen dann nur noch prüfen, ob dies ein akzeptierender Zustand ist. Bei dieser Vorgehensweise müssen wir δ offensichtlich $|w|$-mal anwenden, und jede Anwendung benötigt $|\delta|$ Schritte (Durchsuchen von δ nach dem passenden Übergang).

Satz 5.1

Das Wortproblem für DEAs ist in Zeit $\mathcal{O}(|w| \cdot |\delta|)$ entscheidbar.

Für einen NEA ist dieser triviale Algorithmus nicht müglich, da es ja mehrere mit w beschriftete Pfade geben kann. In der Tat führen die naiven Ansätze zum Entscheiden des Wortproblems für NEAs zu exponentieller Laufzeit:

- Alle Pfade für das Eingabewort durchprobieren.

  Im schlimmsten Fall gibt es $|Q|^{|w|}$ viele solche Pfade, also exponentiell viele. Wenn $w \notin L(\mathcal{A})$, werden alle diese Pfade auch tatsächlich überprüft.

- Erst Potenzmengenkonstruktion anwenden, um DEA zu konstruieren.

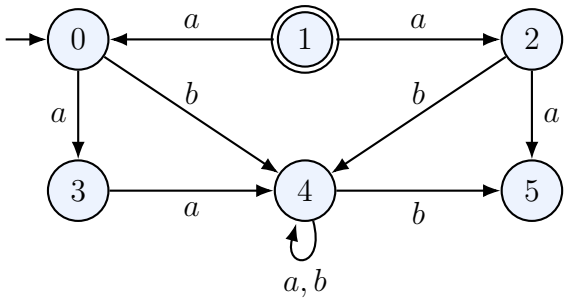
  Wie bereits erwähnt, führt die exponentielle Vergrößerung des Automaten zu exponentieller Laufzeit.

Es stellt sich allerdings heraus, dass auch das Wortproblem für NEAs effizient lösbar ist. Wir verwenden dazu (einen Trick und) den Algorithmus für das Leerheitsproblem, das wir im Folgenden betrachten.

5.2. Das Leerheitsproblem für DEAs und NEAs

Wir betrachten hier direkt NEAs. Da jeder DEA auch ein NEA ist, können die entwickelten Algorithmen natürlich auch für DEAs verwendet werden.

Im Gegensatz zum Wortproblem ist beim Leerheitsproblem keine konkrete Eingabe gegeben. Es scheint daher zunächst, als müssten wir alle (unendlich vielen) Eingaben durchprobieren, was natürlich unmöglich ist. Ein einfaches Beispiel zeigt aber sofort, dass das Leerheitsproblem sehr einfach zu lösen ist. Betrachte den folgenden NEA $\mathcal{A}$:



Offensichtlich ist $L(\mathcal{A}) = \emptyset$, da der akzeptierende Zustand vom Startzustand aus gar nicht erreichbar ist. Es ist leicht zu sehen, dass auch die umgekehrte Implikation gilt: wenn $L(\mathcal{A}) = \emptyset$, dann ist kein akzeptierender Zustand vom Startzustand aus erreichbar, denn sonst würde die Beschriftung eines den akzeptierenden Zustand erreichenden Laufs ein Wort $w \in L(\mathcal{A})$ liefern. Das Leerheitsproblem ist also nichts weiter als ein Erreichbarkeitsproblem auf gerichteten Graphen wie dem oben dargestellten.

Das konkrete Wort, mit dem ein akzeptierender Zustand erreicht wird, interessiert uns im Fall des Leerheitsproblems meist nicht. Da wir jedoch im Folgenden Aussagen über die Länge von Pfaden treffen müssen, schreiben wir $p \Longrightarrow_{\mathcal{A}}^{i} q$ (q wird in $\mathcal{A}$ von p mit einem Pfad der Länge höchtens i erreicht), wenn $p \overset{w}{\Longrightarrow}_{\mathcal{A}} q$ für ein Wort $w \in \Sigma^*$ mit $|w| \leq i$.

Es gibt verschiedene effiziente Algorithmen für Erreichbarkeitsprobleme auf gerichteten Graphen. Der folgende ist eine Variante der *Breitensuche* und entscheidet das Leerheitsproblem für einen gegebenen NEA $\mathcal{A} = (Q, \Sigma, q_s, \Delta, F)$ in polynomieller Zeit.

1. Berechne eine Folge von Zustandsmengen $P_0, P_1, \ldots$ wie folgt:

$$P_0 := \{q_s\}$$
$$P_{i+1} := P_i \cup \{q \in Q \ : \ \exists p \in P_i \ \exists a \in \Sigma : (p, a, q) \in \Delta\}.$$

   Stoppe sobald $P_{i+1} = P_i$.
2. Antworte „ja“, wenn $P_i \cap F = \emptyset$, und „nein“ sonst.

Lemma 5.2

Der Algorithmus terminiert nach maximal $|Q|$ Iterationen und gibt „ja“ zurück gdw. $L(\mathcal{A}) = \emptyset$.

Beweis. Die Terminierung in $|Q|$ Iterationen folgt unmittelbar aus der Beobachtung, dass

$$P_0 \subseteq P_1 \subseteq P_2 \subseteq \cdots \subseteq Q.$$

Bei Terminierung nach i Iterationen setze $P_j := P_i$ für alle $j > i$.

Der Beweis der Korrektheit basiert auf folgender Behauptung.

Behauptung: $q \in P_j$ gdw. $q_s \Longrightarrow_{\mathcal{A}}^{j} q$ für alle $j \geq 0$ und $q \in Q$.

Beweis per Induktion über j:

$j = 0$. $q \in P_0$ gdw. $q = q_s$ gdw. $q_s \Longrightarrow_{\mathcal{A}}^{0} q$.

$j > 0$. $q \in P_j$ gdw. $q \in P_{j-1}$ oder $(p, a, q) \in \Delta$ mit $p \in P_{j-1}$ und $a \in \Sigma$
gdw. $q \in P_{j-1}$ oder $(p, a, q) \in \Delta$ und $q_0 \Longrightarrow_{\mathcal{A}}^{j-1} p$
gdw. $q_s \Longrightarrow_{\mathcal{A}}^{j} q$.

Die Korrektheit des Algorithmus folgt nun aus der Behauptung:

Der Algorithmus gibt „ja“ zurück
gdw. $P_i \cap F = \emptyset$
gdw. $P_j \cap F = \emptyset$ für alle $j \geq 0$
gdw. $q_s \not\Longrightarrow_{\mathcal{A}}^{j} q_f$ für alle $q_f \in F$ und $j \geq 0$
gdw. $L(\mathcal{A}) = \emptyset$ ❑

Der Algorithmus stoppt also nach $|Q|$ Iterationen. Jede Iteration braucht bei naiver Implementierung Zeit $\mathcal{O}(|Q| \cdot |\Delta|)$: laufe über alle Zustände $p \in P_i$, für jeden solchen Zustand laufe über Δ und suche alle Übergänge (p, a, q). Insgesamt benötigt der Algorithmus also Zeit $\mathcal{O}(|Q|^2 \cdot |\Delta|)$. Durch geschickte Datenstrukturen können wir die Laufzeit verbessern auf Zeit $\mathcal{O}(|Q| + |\Delta|)$, also *Linearzeit*. Mehr Informationen finden sich beispielsweise im Buch „Introduction to Algorithms“ [CLRS01] unter dem Thema *Erreichbarkeit in gerichteten Graphen/reachability in directed graphs.*

Satz 5.3

Das Leerheitsproblem für NEAs ist in Zeit $\mathcal{O}(|Q| + |\Delta|)$ entscheidbar.

5.3. Das Wortproblem für NEAs

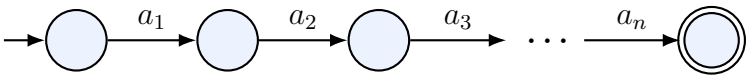
Wir verwenden Satz 5.3 um nachzuweisen, dass das Wortproblem für NEAs nicht schwieriger ist als das für DEAs. Die exponentielle Laufzeit der naiven Ansätze kann also vermieden werden.

Satz 5.4

Das Wortproblem für NEAs ist in Zeit $\mathcal{O}(|w| \cdot (|Q| + |\Delta|))$ entscheidbar.

Wir verwenden eine *Reduktion* des Wortproblems auf das Leerheitsproblem: der schon gefundene Algorithmus für das Leerheitsproblem wird verwendet, um das Wortproblem zu lösen.

Beweis. Konstruiere zunächst einen Automaten $\mathcal{A}_w$, der genau das Wort $w = a_1 \cdots a_n$ akzeptiert:



Dieser Automat hat $|w| + 1$ Zustände. Offenbar ist

$$w \in L(\mathcal{A}) \qquad \text{gdw.} \qquad L(\mathcal{A}) \cap L(\mathcal{A}_w) \neq \emptyset.$$

Wir können also entscheiden, ob $w \in L(\mathcal{A})$ ist, indem wir zunächst den Produktautomaten zu $\mathcal{A}$ und $\mathcal{A}_w$ konstruieren und dann unter Verwendung von Satz 5.3 prüfen, ob dieser eine nicht-leere Sprache erkennt.

Wir analysieren zunächst die Größe des Produktautomaten:

- Anzahl der Zustände: $|Q| \cdot (|w| + 1)$
- Anzahl der Übergänge:

  Da $\mathcal{A}_w$ genau $|w|$ Übergänge hat, hat der Produktautomat höchstens $|w| \cdot |\Delta|$ Übergänge.

Nach Satz 5.3 ist daher der Aufwand zum Testen von $L(\mathcal{A}) \cap L(\mathcal{A}_w) \neq \emptyset$ also:

$$\mathcal{O}(|Q| \cdot (|w| + 1) + |w| \cdot |\Delta|) \;=\; \mathcal{O}(|w| \cdot (|Q| + |\Delta|)).$$

Auch die Konstruktion des Produktautomaten benötigt Zeit. Es ist leicht zu sehen, dass auch hierfür die Zeit $\mathcal{O}(|w| \cdot (|Q| + |\Delta|)$ ausreichend ist. Als Gesamtlaufzeit ergibt sich

$$2 \cdot \mathcal{O}(|w| \cdot (|Q| + |\Delta|)) \;=\; \mathcal{O}(|w| \cdot (|Q| + |\Delta|)).$$

❑

5.4. Das Äquivalenzproblem für DEAs und NEAs

Wir verwenden sowohl für DEAs als auch für NEAs eine Reduktion auf das Leerheitsproblem:

$$L_1 = L_2 \qquad \text{gdw.} \qquad (L_1 \cap \overline{L_2}) \cup (L_2 \cap \overline{L_1}) = \emptyset$$

Im Fall des Äquivalenzproblems wollen wir auf eine ganz exakte Analyse der Laufzeit des sich ergebenden Algorithmus verzichten. Allerdings gibt es einen interessanten Unterschied zwischen DEAs und NEAs, der im folgenden Satz herausgearbeitet wird.

Satz 5.5

Das Äquivalenzproblem für DEAs ist in polynomieller Zeit entscheidbar. Für NEAs ist es in exponentieller Zeit entscheidbar.

Beweis. Die Konstruktion für Schnitt (Produktautomat) und Vereinigung ist sowohl für DEAs als auch für NEAs polynomiell. Bei der Komplementierung ist dies nur dann der Fall, wenn bereits DEAs vorliegen. Bei NEAs muss zunächst die Potenzmengenkonstruktion angewendet werden, daher kann der auf Leerheit zu testende Automat exponentiell groß sein. Damit ergibt sich exponentielle Laufzeit. ❑

Vorgreifend auf die Theoretische Informatik 2 sei erwähnt, dass das Äquivalenzproblem für NEAs PSpace-vollständig ist und damit zu einer Klasse von Problemen gehört, die wahrscheinlich nicht in polynomieller Zeit lösbar sind.

6. Reguläre Ausdrücke und Sprachen

Wir haben bereits einige *verschiedene Charakterisierungen* der Klasse der regulären Sprachen gesehen:

Eine Sprache $L \subseteq \Sigma^*$ ist *regulär* gdw.

(1) $L = L(\mathcal{A})$ für einen DEA $\mathcal{A}$.

(2) $L = L(\mathcal{A})$ für einen NEA $\mathcal{A}$.

(3) $L = L(\mathcal{A})$ für einen ε-NEA $\mathcal{A}$.

(4) $L = L(\mathcal{A})$ für einen NEA mit Wortübergängen $\mathcal{A}$.

Im Folgenden betrachten wir eine weitere Charakterisierung mit Hilfe *regulärer Ausdrücke*. Diese Stellen einen bequemen Formalismus zur Verfügung um reguläre Sprachen zu beschreiben. Varianten von regulären Ausdrücken werden in Tools wie `Emacs`, `Perl` und `sed` zur Beschreibung von Mustern („Patterns“) verwendet.

Definition 6.1 (Syntax regulärer Ausdrücke)

Sei Σ ein endliches Alphabet. Die Menge Reg_Σ der *regulären Ausdrücke über* Σ ist induktiv definiert:

- $\emptyset$, ε, a (für $a \in \Sigma$) sind Elemente von Reg_Σ.
- Sind $r, s \in \mathsf{Reg}_\Sigma$, so auch $(r + s), (r \cdot s), r^* \in \mathsf{Reg}_\Sigma$.

Beispiel 6.2

- $((a \cdot b^*) + \emptyset^*)^* \in \mathsf{Reg}_\Sigma$ für $\Sigma = \{a, b\}$
- $(((a \cdot b)^* + (c \cdot b \cdot a)^*) + a^*) \in \mathsf{Reg}_\Sigma$ für $\Sigma = \{a, b, c\}$

Notation:

Um Klammern zu sparen, lassen wir Außenklammern weg und vereinbaren,

- dass * stärker bindet als $\cdot$
- dass $\cdot$ stärker bindet als $+$
- $\cdot$ lassen wir meist ganz wegfallen.

Der Ausdruck aus Beispiel 6.2 kann also geschrieben werden als $(ab^* + \emptyset^*)^*$.

Außerdem wird gleich aus Definition 6.3 folgen, dass $+$ assoziativ ist, d. h. $((\alpha + \beta) + \gamma)$ und $(\alpha + (\beta + \gamma))$ beschreiben dieselbe Sprache. Also lassen wir auch hier Klammern weg und schreiben $\alpha + \beta + \gamma$.

Statt $(((a \cdot b)^* + (c \cdot b \cdot a)^*) + a^*)$ schreiben wir also $(ab)^* + (cba)^* + a^*$.

Um die Bedeutung bzw. Semantik von regulären Ausdrücken zu fixieren, wird jedem regulären Ausdruck r über Σ eine formale Sprache $L(r)$ zugeordnet.

Definition 6.3 (Semantik regulärer Ausdrücke)

Die durch den regulären Ausdruck r definierte Sprache $L(r)$ ist induktiv definiert:

$$\begin{aligned} L(\emptyset) &:= \emptyset & L(r+s) &:= L(r) \cup L(s) \\ L(\varepsilon) &:= \{\varepsilon\} & L(r \cdot s) &:= L(r) \cdot L(s) \\ L(a) &:= \{a\} & L(r^*) &:= L(r)^* \end{aligned}$$

Beispiel 6.4

- $(a+b)^*ab(a+b)^*$ definiert die Sprache aller Wörter über $\{a, b\}$, die Infix ab haben.
- $L(ab^* + b) = \{ab^i \mid i \geq 0\} \cup \{b\}$

Bemerkung:

Statt $L(r)$ schreiben wir häufig einfach r. Dies ermöglicht es uns z. B. zu schreiben:

- $(ab)^*a = a(ba)^*$ eigentlich: $L((ab)^*a) = L(a(ba)^*)$
- $L(\mathcal{A}) = ab^* + b$ eigentlich: $L(\mathcal{A}) = L(ab^* + b)$

Wir zeigen nun, dass wir mit regulären Ausdrücken genau die regulären Sprachen definieren können.

Satz 6.5 (Kleene)

Für eine Sprache $L \subseteq \Sigma^*$ sind äquivalent:

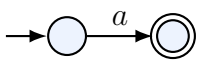
1) Es gibt einen regulären Ausdruck r mit $L = L(r)$.

2) L ist regulär.

Beweis.

„1 $\Rightarrow$ 2“: Per Induktion über den Aufbau regulärer Ausdrücke.

Induktionsanfang:

- $L(\emptyset) = \emptyset$ regulär:  ist NEA für $\emptyset$ (kein akz. Zustand).
- $L(\varepsilon) = \{\varepsilon\}$ regulär:  ist NEA für $\{\varepsilon\}$.
- $L(a) = \{a\}$ regulär:  ist NEA für $\{a\}$.

Induktionsschritt:

Wenn $L(r)$ und $L(s)$ regulär sind, so folgt mit Satz 4.1 (Abschlusseigenschaften), dass auch $L(r+s)$, $L(r \cdot s)$ und $L(r^*)$ regulär sind, denn laut Definition 6.3 gilt:

- $L(r + s) = L(r) \cup L(s)$,
- $L(r \cdot s) = L(r) \cdot L(s)$,
- $L(r^*) = L(r)^*$.

„2 ⇒ 1“: Sei $\mathcal{A} = (Q, \Sigma, q_s, \Delta, F)$ ein NEA mit $L = L(\mathcal{A})$. Für alle $p, q \in Q$ und $X \subseteq Q$ sei $L^X_{p,q}$ die Sprache aller Wörter $w = a_1 \cdots a_n$, für die es einen Pfad

$$p_0 \xrightarrow{a_1}_{\mathcal{A}} p_1 \xrightarrow{a_2}_{\mathcal{A}} \cdots \xrightarrow{a_{n-1}}_{\mathcal{A}} p_{n-1} \xrightarrow{a_n}_{\mathcal{A}} p_n$$

gibt mit $p_0 = p$, $p_n = q$ und $\{p_1, \ldots, p_{n-1}\} \subseteq X$. Offensichtlich gilt:

$$L(\mathcal{A}) = \bigcup_{q_f \in F} L^Q_{q_s, q_f}$$

Wir zeigen, dass es für alle Sprachen $L^X_{p,q}$ einen regulären Ausdruck gibt. Dies erfolgt per Induktion über die Größe von X.

Induktionsanfang: $X = \emptyset$.

- 1. Fall: $p \neq q$

  Dann ist $L^\emptyset_{p,q} = \{a \in \Sigma \ : \ (p, a, q) \in \Delta\}$. Damit hat $L^\emptyset_{p,q}$ die Form $\{a_1, \ldots, a_k\}$ und der entsprechende reguläre Ausdruck ist $a_1 + \cdots + a_k$.

- 2. Fall: $p = q$

  Dann ist $L^\emptyset_{p,q} = \{a \in \Sigma \ : \ (p, a, q) \in \Delta\} \cup \{\varepsilon\}$. Damit hat $L^\emptyset_{p,q}$ die Form $\{a_1, \ldots, a_k\} \cup \{\varepsilon\}$ und der entsprechende reguläre Ausdruck ist $a_1 + \cdots + a_k + \varepsilon$.

Induktionsschritt: $X \neq \emptyset$.

Wähle ein beliebiges $\widehat{q} \in X$. Dann gilt:

$$L^X_{p,q} \ = \ L^{X \setminus \{\widehat{q}\}}_{p,q} \ \cup \ \left(L^{X \setminus \{\widehat{q}\}}_{p,\widehat{q}} \ \cdot \ \left(L^{X \setminus \{\widehat{q}\}}_{\widehat{q},\widehat{q}} \right)^* \ \cdot \ L^{X \setminus \{\widehat{q}\}}_{\widehat{q},q} \right). \qquad (*)$$

Für die Sprachen, die auf der rechten Seite verwendet werden, gibt es nach Induktionsvoraussetzung reguläre Ausdrücke. Außerdem sind alle verwendeten Operationen in regulären Ausdrücken verfügbar. Es bleibt also, $(*)$ zu zeigen.

„⊆“: Sei $w \in L^X_{p,q}$. Dann gibt es einen Pfad

$$p_0 \xrightarrow{a_1}_{\mathcal{A}} p_1 \xrightarrow{a_2}_{\mathcal{A}} \cdots \xrightarrow{a_{n-1}}_{\mathcal{A}} p_{n-1} \xrightarrow{a_n}_{\mathcal{A}} p_n$$

mit $p_0 = p$, $p_n = q$ und $\{p_1, \ldots, p_{n-1}\} \subseteq X$. Wenn $\widehat{q}$ nicht in $\{p_1, \ldots, p_{n-1}\}$ vorkommt, dann $w \in L^{X \setminus \{\widehat{q}\}}_{p,q}$. Andernfalls seien $i_1, \ldots, i_k$ alle Indizes mit $p_{i_j} = \widehat{q}$ (und $i_1 < \cdots < i_k$). Es gilt:

- $a_0 \cdots a_{i_1} \in L^{X \setminus \{\widehat{q}\}}_{p,\widehat{q}}$,
- $a_{i_j+1} \cdots a_{i_{j+1}} \in L^{X \setminus \{\widehat{q}\}}_{\widehat{q},\widehat{q}}$ für $1 \leq j < k$ und
- $a_{i_k+1} \cdots a_n \in L^{X \setminus \{\widehat{q}\}}_{\widehat{q},q}$.

„⊇“: Wenn $w \in L_{p,q}^{X\setminus\{\widehat{q}\}}$, dann auch $w \in L_{p,q}^{X}$. Wenn

$$w \in \left(L_{p,\widehat{q}}^{X\setminus\{\widehat{q}\}} \cdot \left(L_{\widehat{q},\widehat{q}}^{X\setminus\{\widehat{q}\}} \right)^* \cdot L_{\widehat{q},q}^{X\setminus\{\widehat{q}\}} \right),$$

dann $w = xyz$ mit $x \in L_{p,\widehat{q}}^{X\setminus\{\widehat{q}\}}$, $y \in (L_{\widehat{q},\widehat{q}}^{X\setminus\{\widehat{q}\}})^*$, und $z \in L_{\widehat{q},q}^{X\setminus\{\widehat{q}\}}$. Setzen wir die entsprechenden Pfade für x, y und z zusammen, so erhalten wir einen mit w beschrifteten Pfad

$$p_0 \xrightarrow{a_1}_{\mathcal{A}} p_1 \xrightarrow{a_2}_{\mathcal{A}} \cdots \xrightarrow{a_{n-1}}_{\mathcal{A}} p_{n-1} \xrightarrow{a_n}_{\mathcal{A}} p_n$$

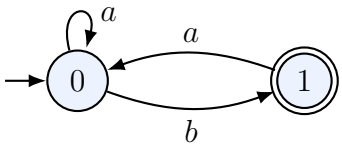
mit $p_0 = p$, $p_n = q$ und $\{p_1, \ldots, p_{n-1}\} \subseteq X$. Also $w \in L_{p,q}^{X}$.

❑

Wenn wir die Konstruktion aus „$2 \Rightarrow 1$“ in der Praxis anwenden, so ist es meist sinnvoll, die Zustände $\widehat{q}$ so zu wählen, dass der Automat in möglichst viele nicht-verbundene Teile zerfällt.

Beispiel 6.6

Betrachte den folgenden NEA $\mathcal{A}$:



Da 1 der einzige akzeptierende Zustand ist, gilt $L(\mathcal{A}) = L_{0,1}^{Q}$. Wir wenden wiederholt $(*)$ an:

$$\begin{aligned} L_{0,1}^{Q} &= L_{0,1}^{\{0\}} \cup L_{0,1}^{\{0\}} \cdot (L_{1,1}^{\{0\}})^* \cdot L_{1,1}^{\{0\}} \\ L_{0,1}^{\{0\}} &= L_{0,1}^{\emptyset} \cup L_{0,0}^{\emptyset} \cdot (L_{0,0}^{\emptyset})^* \cdot L_{0,1}^{\emptyset} \\ L_{1,1}^{\{0\}} &= L_{1,1}^{\emptyset} \cup L_{1,0}^{\emptyset} \cdot (L_{0,0}^{\emptyset})^* \cdot L_{0,1}^{\emptyset} \end{aligned}$$

Im ersten Schritt hätten wir natürlich auch 0 anstelle von 1 aus X eliminieren können. Der Induktionsanfang liefert:

$$\begin{aligned} L_{0,1}^{\emptyset} &= b \\ L_{0,0}^{\emptyset} &= a + \varepsilon \\ L_{1,1}^{\emptyset} &= \varepsilon \\ L_{1,0}^{\emptyset} &= a \end{aligned}$$

Einsetzen und Vereinfachen liefert nun:

$$\begin{aligned} L_{0,1}^{\{0\}} &= b + (a+\varepsilon) \cdot (a+\varepsilon)^* \cdot b = a^*b \\ L_{1,1}^{\{0\}} &= \varepsilon + a \cdot (a+\varepsilon)^* \cdot b = \varepsilon + aa^*b \\ L_{0,1}^{Q} &= a^*b + a^*b \cdot (\varepsilon + aa^*b)^* \cdot (\varepsilon + aa^*b) = a^*b(aa^*b)^* \end{aligned}$$

Der zu $\mathcal{A}$ gehörende reguläre Ausdruck ist also $a^*b(aa^*b)^*$.

Der reguläre Ausdruck, der in der Richtung „$2 \Rightarrow 1$“ aus einem NEA konstruiert wird, ist im Allgemeinen exponentiell größer als der ursprüngliche NEA. Es kann gezeigt werden, dass dies nicht vermeidbar ist.

Beachte: Aus Satz 4.1 und Satz 6.5 folgt, dass es zu allen regulären Ausdrücken r und s

- einen Ausdruck t gibt mit $L(t) = L(r) \cap L(s)$;
- einen Ausdruck t' gibt mit $L(t') = \overline{L(r)}$.

Es ist offensichtlich sehr schwierig, diese Ausdrücke direkt aus r und s (also ohne den Umweg über Automaten) zu konstruieren.

7. Minimale DEAs und die Nerode-Rechtskongruenz

7.1. Minimale DEAs

Wir werden im Folgenden ein Verfahren angeben, welches zu einem gegebenen DEA einen äquivalenten DEA mit minimaler Zustandszahl konstruiert. Das Verfahren besteht aus 2 Schritten:

1. Schritt: Eliminieren unerreichbarer Zustände.

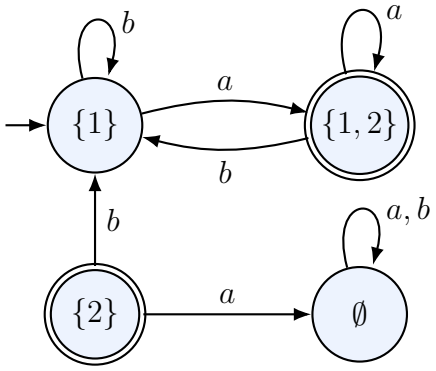
Definition 7.1 (Erreichbarkeit eines Zustandes)
Ein Zustand q des DEA $\mathcal{A} = (Q, \Sigma, q_s, \delta, F)$ heißt *erreichbar*, falls es ein Wort $w \in \Sigma^*$ gibt mit $\hat{\delta}(q_s, w) = q$. Sonst heißt q *unerreichbar*.

Da für die erkannte Sprache nur Zustände wichtig sind, welche von q_s erreicht werden, erhalten wir durch Weglassen unerreichbarer Zustände einen äquivalenten Automaten:

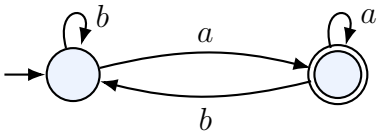
$\mathcal{A}_0 = (Q_0, \Sigma, q_s, \delta_0, F_0)$ mit

- $Q_0 = \{q \in Q \ : \ q \text{ ist erreichbar}\}$
- $\delta_0 = \delta|_{Q_0 \times \Sigma}$ (also: δ_0 ist wie δ, aber eingeschränkt auf die Zustände in Q_0)
- $F_0 = F \cap Q_0$

Beispiel 7.2
Betrachte als Resultat der Potenzmengenkonstruktion den Automaten $\mathcal{A}'$ aus Beispiel 2.13:



Die Zustände $\{2\}$ und $\emptyset$ sind nicht erreichbar. Durch Weglassen dieser Zustände erhalten wir den DEA $\mathcal{A}'_0$:

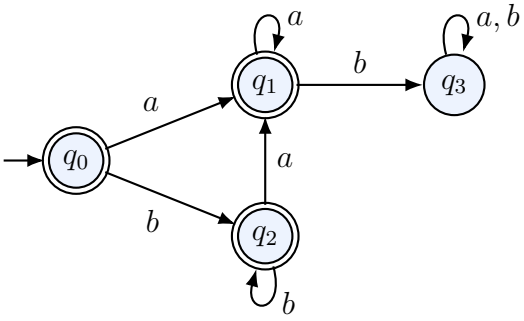


2. Schritt: Zusammenfassen äquivalenter Zustände

Ein DEA ohne unerreichbare Zustände muss noch nicht minimal sein, da er noch verschiedene Zustände enthalten kann, die sich „gleich“ verhalten in Bezug auf die akzeptierte Sprache.

Beispiel 7.3

Im folgenden DEA $\mathcal{A}$ mit $L(\mathcal{A}) = b^*a^*$ sind alle Zustände erreichbar. Er erkennt dieselbe Sprache wie der DEA aus Beispiel 2.7, hat aber einen Zustand mehr. Dies kommt daher, dass q_0 und q_2 äquivalent sind.



Im Allgemeinen definieren wir die Äquivalenz von Zuständen wie folgt.

Definition 7.4 (Äquivalenz von Zuständen)

Es sei $\mathcal{A} = (Q, \Sigma, q_s, \delta, F)$ ein DEA. Für $q \in Q$ sei $\mathcal{A}_q = (Q, \Sigma, q, \delta, F)$. Zwei Zustände $q, q' \in Q$ heißen *$\mathcal{A}$-äquivalent* ($q \sim_{\mathcal{A}} q'$) gdw. $L(\mathcal{A}_q) = L(\mathcal{A}_{q'})$.

In Beispiel 7.3 gilt $q_0 \sim_{\mathcal{A}} q_2$, aber z. B. nicht $q_0 \sim_{\mathcal{A}} q_1$, da $b \in L(\mathcal{A}_{q_0}) \setminus L(\mathcal{A}_{q_1})$.

Um äquivalente Zustände auf mathematisch elegante Weise zusammenzufassen, nutzen wir aus, dass es sich bei der Relation $\sim_{\mathcal{A}}$ um eine Äquivalenzrelation handelt. Diese erfüllt zusätzlich einige weitere angenehme Eigenschaften.

Lemma 7.5

1) $\sim_{\mathcal{A}}$ ist eine Äquivalenzrelation auf Q, d. h. reflexiv, transitiv und symmetrisch.

2) $\sim_{\mathcal{A}}$ ist verträglich mit der Übergangsfunktion, d. h.

$$q \sim_{\mathcal{A}} q' \Rightarrow \forall a \in \Sigma : \delta(q, a) \sim_{\mathcal{A}} \delta(q', a)$$

3) $\sim_{\mathcal{A}}$ kann in Polynomialzeit berechnet werden.

Beweis.

1) ist klar, da die Relation „=“ reflexiv, transitiv und symmetrisch ist.

2) lässt sich wie folgt herleiten:

$$\begin{aligned}
q \sim_{\mathcal{A}} q' &\Rightarrow L(\mathcal{A}_q) = L(\mathcal{A}_{q'}) \\
&\Rightarrow \forall w \in \Sigma^* : \hat{\delta}(q,w) \in F \Leftrightarrow \hat{\delta}(q',w) \in F \\
&\Rightarrow \forall a \in \Sigma\ \forall v \in \Sigma^* : \hat{\delta}(q,av) \in F \Leftrightarrow \hat{\delta}(q',av) \in F \\
&\Rightarrow \forall a \in \Sigma\ \forall v \in \Sigma^* : \hat{\delta}(\delta(q,a),v) \in F \Leftrightarrow \hat{\delta}(\delta(q',a),v) \in F \\
&\Rightarrow \forall a \in \Sigma : L(\mathcal{A}_{\delta(q,a)}) = L(\mathcal{A}_{\delta(q',a)}) \\
&\Rightarrow \forall a \in \Sigma : \delta(q,a) \sim_{\mathcal{A}} \delta(q',a).
\end{aligned}$$

3) folgt unmittelbar daraus, dass das Äquivalenzproblem für DEAs in Polynomialzeit entscheidbar ist. ❑

Die $\sim_{\mathcal{A}}$-Äquivalenzklasse eines Zustands $q \in Q$ bezeichnen wir von nun an mit

$$[q]_{\mathcal{A}} := \{q' \in Q \ : \ q \sim_{\mathcal{A}} q'\}.$$

Auch wenn wir mit Punkt 3) des Lemma 7.5 bereits wissen, dass die Relation $\sim_{\mathcal{A}}$ berechenbar ist, geben wir hier noch eine direktere Methode an. Wir definieren eine Folge von Relationen $\sim_0, \sim_1, \sim_2, \cdots$:

- $q \sim_0 q'$ gdw. $q \in F \Leftrightarrow q' \in F$
- $q \sim_{k+1} q'$ gdw. $q \sim_k q'$ und $\forall a \in \Sigma : \delta(q,a) \sim_k \delta(q',a)$

Diese sind (über-)Approximationen von $\sim_{\mathcal{A}}$ im folgenden Sinn.

Behauptung.

Für alle $k \geq 0$ gilt: $q \sim_k q'$ gdw. für alle $w \in \Sigma^*$ mit $|w| \leq k$: $w \in L(\mathcal{A}_q) \Leftrightarrow w \in L(\mathcal{A}_{q'})$.

Beweis.

Per Induktion über k:

Induktionsanfang: Nach Def. von $\sim_0$ gilt $q \sim_0 q'$ gdw. $\varepsilon \in L(\mathcal{A}_q) \Leftrightarrow \varepsilon \in L(\mathcal{A}_{q'})$.

Induktionsschritt:

$$\begin{aligned}
q \sim_{k+1} q' \ \text{gdw.}\ & q \sim_k q' \text{ und } \forall a \in \Sigma : \delta(q,a) \sim_k \delta(q',a) \\
\text{gdw.}\ & \forall w \in \Sigma^* \text{ mit } |w| \leq k : w \in L(\mathcal{A}_q) \Leftrightarrow w \in L(\mathcal{A}_{q'}) \text{ und} \\
& \forall a \in \Sigma : \forall w \in \Sigma^* \text{ mit } |w| \leq k : w \in L(\mathcal{A}_{\delta(q,a)}) \Leftrightarrow w \in L(\mathcal{A}_{\delta(q',a)}) \\
\text{gdw.}\ & \forall w \in \Sigma^* \text{ mit } |w| \leq k+1 : w \in L(\mathcal{A}_q) \Leftrightarrow w \in L(\mathcal{A}_{q'}).
\end{aligned}$$

❑

Offensichtlich gilt $Q \times Q \supseteq \sim_0 \supseteq \sim_1 \supseteq \sim_2 \supseteq \ldots$. Da Q endlich ist, gibt es ein $k \geq 0$ mit $\sim_k = \sim_{k+1}$. Wir zeigen, dass $\sim_k$ die gewünschte Relation $\sim_{\mathcal{A}}$ ist. Nach obiger Behauptung und Definition von $\sim_{\mathcal{A}}$ gilt offensichtlich $\sim_{\mathcal{A}} \subseteq \sim_k$. Um $\sim_k \subseteq \sim_{\mathcal{A}}$ zu zeigen, nehmen wir das Gegenteil $\sim_k \not\subseteq \sim_{\mathcal{A}}$ an. Wähle q, q' mit $q \sim_k q'$ und $q \not\sim_{\mathcal{A}} q'$. Es gibt also ein $w \in \Sigma^*$ mit $w \in L(\mathcal{A}_q)$ und $w \notin L(\mathcal{A}_{q'})$. Mit obiger Behauptung folgt $q \not\sim_n q'$ für $n = |w|$. Da $\sim_k \subseteq \sim_i$ für all $i \geq 0$ folgt $q \not\sim_k q'$, ein Widerspruch.

Beispiel 7.3 (Fortsetzung)

Für den Automaten aus Beispiel 7.3 gilt:

- $\sim_0$ hat die Klassen $F = \{q_0, q_1, q_2\}$ und $Q \setminus F = \{q_3\}$.
- $\sim_1$ hat die Klassen $\{q_1\}, \{q_0, q_2\}, \{q_3\}$.
  Zum Beispiel ist $\delta(q_0, b) = \delta(q_2, b) \in F$ und $\delta(q_1, b) \notin F$.
- $\sim_2 = \sim_1 = \sim_{\mathcal{A}}$.

Wir können diese Vorgehensweise beim Berechnen von $\sim_{\mathcal{A}}$ wie folgt als Algorithmus in Pseudocode formulieren; siehe Algorithmus 1.

Algorithmus 1 : Berechnung von $\sim_{\mathcal{A}}$

Algorithmus `equiv(`$\mathcal{A}$`)`**:**

input : DEA $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$
output : Äquivalenzrelation $\sim_{\mathcal{A}} \subseteq Q \times Q$

$k := 0$
$\sim_0$ ist die Äquivalenzrelation mit den Äquivalenzklassen F und $Q \setminus F$
repeat
  Berechne $\sim_{k+1}$ aus $\sim_k$ wie oben definiert
  $k := k + 1$
until $\sim_k = \sim_{k-1}$
return $\sim_{k+1}$

In der nachfolgenden Konstruktion werden äquivalente Zustände zusammengefasst, indem die Äquivalenzklassen $[q]_{\mathcal{A}}$ selbst zu den Zuständen gemacht werden. Jede Klasse verhält sich dann genau wie die in ihr enthaltenen Zustände im ursprünglichen Automaten.

Definition 7.6 (Quotientenautomat)

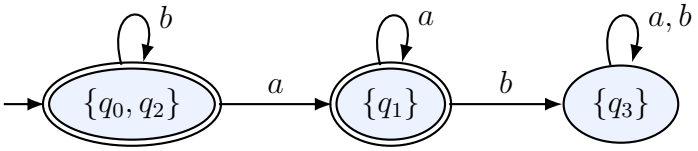
Der *Quotientenautomat* $\widetilde{\mathcal{A}} = (\widetilde{Q}, \Sigma, [q_0]_{\mathcal{A}}, \widetilde{\delta}, \widetilde{F})$ *zu* $\mathcal{A} = (Q, \Sigma, q_s, \delta, F)$ ist definiert durch:

- $\widetilde{Q} := \{[q]_{\mathcal{A}} \mid q \in Q\}$
- $\widetilde{\delta}([q]_{\mathcal{A}}, a) := [\delta(q, a)]_{\mathcal{A}}$ (repräsentantenunabhängig wegen Lemma 7.5)
- $\widetilde{F} := \{[q]_{\mathcal{A}} \mid q \in F\}$ (repräsentantenunabhängig wegen Definition von $\sim_{\mathcal{A}}$ 7.4)

Nach Lemma 7.5 kann der Quotientenautomat in polynomieller Zeit konstruiert werden.

Beispiel 7.3 (Fortsetzung)

Für den Automaten aus Beispiel 7.3 ergibt sich der folgende Quotientenautomat:



Lemma 7.7

$\widetilde{\mathcal{A}}$ ist äquivalent zu $\mathcal{A}$.

Beweis. Es folgt leicht per Induktion über $|w|$:

$$\hat{\widetilde{\delta}}([q_0]_{\mathcal{A}}, w) = [\hat{\delta}(q_0, w)]_{\mathcal{A}} \text{ für alle } w \in \Sigma^*. \qquad (*)$$

Nun gilt:

$$\begin{aligned} w \in L(\mathcal{A}) \quad & \text{gdw.} \quad \hat{\delta}(q_0, w) \in F \\ & \text{gdw.} \quad [\hat{\delta}(q_0, w)]_{\mathcal{A}} \in \widetilde{F} \quad (\text{Def. } \widetilde{F}) \\ & \text{gdw.} \quad \hat{\widetilde{\delta}}([q_0]_{\mathcal{A}}, w) \in \widetilde{F} \quad (*) \\ & \text{gdw.} \quad w \in L(\widetilde{\mathcal{A}}). \end{aligned}$$

❑

Die folgende Definition fasst die beiden Minimierungs-Schritte zusammen:

Definition 7.8 (reduzierter Automat zu einem DEA)

Für einen DEA $\mathcal{A}$ bezeichnet $\mathcal{A}_{\mathsf{red}}$ den *reduzierten Automaten*, den man aus $\mathcal{A}$ durch Eliminieren unerreichbarer Zustände und nachfolgendes Bilden des Quotientenautomaten erhält.

Wir wollen zeigen, dass der reduzierte Automat nicht weiter vereinfacht werden kann: $\mathcal{A}_{\mathsf{red}}$ ist der kleinste DEA (bezüglich der Zustandszahl), der $L(\mathcal{A})$ erkennt. Um den Beweis führen zu können, benötigen wir als Hilfsmittel eine Äquivalenzrelation auf Wörtern, die Nerode-Rechtskongruenz.

7.2. Die Nerode-Rechtskongruenz

Die Nerode-Rechtskongruenz ist unabhängig von reduzierten Automaten von Interesse und hat neben dem bereits erwähnten Beweis weitere interessante Anwendungen, von denen wir zwei kurz darstellen werden: sie liefert eine von Automaten unabhängige Charakterisierung der regulären Sprachen und stellt ein weiteres Mittel zur Verfügung, um von einer Sprache nachzuweisen, dass sie *nicht* regulär ist.

Im Gegensatz zur Relation $\sim_{\mathcal{A}}$ auf den Zuständen eines Automaten handelt es sich hier um eine Relation *auf Wörtern*.

Definition 7.9 (Nerode-Rechtskongruenz)
Es sei $L \subseteq \Sigma^*$ eine beliebige Sprache. Für $u, v \in \Sigma^*$ definieren wir:
$u \simeq_L v$ gdw. $\forall w \in \Sigma^* : uw \in L \Leftrightarrow vw \in L$.

Beachte, dass das Wort w in Definition 7.9 auch gleich ε sein kann. Darum folgt aus $u \simeq_L v$, dass $u \in L \Leftrightarrow v \in L$.

Beispiel 7.10
Wir betrachten die Sprache $L = b^*a^*$ (vgl. Beispiele 2.7, 7.3).

- Es gilt:
  $\varepsilon \simeq_L b$: $\forall w : \varepsilon w \in L$ gdw. $w \in L$
  gdw. $w \in b^*a^*$
  gdw. $bw \in b^*a^*$
  gdw. $bw \in L$
- $\varepsilon \not\simeq_L a$: $\varepsilon b \in L$, aber $a \cdot b \notin L$

Wir zeigen nun, dass es sich bei $\simeq_L$ wirklich um eine Äquivalenzrelation handelt. In der Tat ist $\simeq_L$ sogar eine Kongruenzrelation bezüglich Konkatenation von beliebigen Wörtern „von rechts“. Im Folgenden bezeichnet der *Index* einer Äquivalenzrelation die Anzahl ihrer Klassen.

Lemma 7.11 (Eigenschaften von $\simeq_L$)

1) $\simeq_L$ ist eine Äquivalenzrelation.

2) $\simeq_L$ ist Rechtskongruenz, d.h. zusätzlich zu 1) gilt: $u \simeq_L v \Rightarrow \forall w \in \Sigma^* : uw \simeq_L vw$.

3) L ist Vereinigung von $\simeq_L$-Klassen:

$$L = \bigcup_{u \in L} [u]_L$$

wobei $[u]_L := \{v \mid u \simeq_L v\}$.

4) Ist $L = L(\mathcal{A})$ für einen DEA $\mathcal{A}$, so ist die Anzahl der Zustände von $\mathcal{A}$ größer oder gleich dem Index von $\simeq$.

Beweis.

1) folgt aus der Definition von $\simeq_L$, da die Relation „$\Leftrightarrow$“ reflexiv, transitiv und symmetrisch ist.

2) Damit $uw \simeq_L vw$ gilt, muss für alle $w' \in \Sigma^*$ gelten:

$$uww' \in L \Leftrightarrow vww' \in L \qquad (*)$$

Wegen $ww' \in \Sigma^*$ folgt $(*)$ aus $u \simeq_L v$.

3) Dafür zeigen wir, dass für alle $v \in \Sigma^*$ gilt:

$$v \in L \qquad \text{gdw.} \qquad v \in \bigcup_{u \in L} [u]_L$$

„⇒“: Wenn $v \in L$, dann ist $[v]_L$ in der Vereinigung rechts; zudem gilt $v \in [v]_L$.

„⇐“: Sei $v \in [u]_L$ für ein $u \in L$.
Wegen $\varepsilon \in \Sigma^*$ folgt aus $u = u \cdot \varepsilon \in L$ und $v \simeq_L u$ auch $v = v \cdot \varepsilon \in L$.

4) Es sei $\mathcal{A} = (Q, \Sigma, q_s, \delta, F)$ ein DEA mit $L = L(\mathcal{A})$.
Wir zeigen: $\hat{\delta}(q_s, u) = \hat{\delta}(q_s, v)$ impliziert $u \simeq_L v$:

$$\begin{aligned} \forall w : uw \in L \quad & \text{gdw.} \quad \hat{\delta}(q_s, uw) \in F \\ & \text{gdw.} \quad \hat{\delta}(\hat{\delta}(q_s, u), w) \in F \\ & \text{gdw.} \quad \hat{\delta}(\hat{\delta}(q_s, v), w) \in F \\ & \text{gdw.} \quad \hat{\delta}(q_s, vw) \in F \\ & \text{gdw.} \quad vw \in L \end{aligned}$$

Also folgt aus $u \not\simeq_L v$, dass $\hat{\delta}(q_s, u) \neq \hat{\delta}(q_s, v)$. Damit gibt es mindestens so viele Zustände wie $\simeq$-Klassen (Schubfachprinzip). ❑

Beispiel 7.10 (Fortsetzung)

$\simeq_L$ hat drei Klassen:

- $[\varepsilon]_L = b^*$
- $[a]_L = b^*aa^*$
- $[ab]_L = (a+b)^*ab(a+b)^*$

Es ist $L = [\varepsilon]_L \cup [a]_L$ (vgl. Lemma 7.11, Punkt 3) und $\Sigma^* \setminus L = [ab]_L$.

Eine interessante Eigenschaft von $\simeq_L$ ist, dass die Äquivalenzklassen zur Definition eines kanonischen Automaten $\mathcal{A}_L$ verwendet werden können, der L erkennt. Dieser Automat ergibt sich *direkt* und *auf eindeutige Weise* aus der Sprache L (im Gegensatz zum reduzierten Automaten, für dessen Konstruktion wir bereits einen Automaten für die betrachtete Sprache haben müssen). Damit wir einen endlichen Automaten erhalten, dürfen wir die Konstruktion nur auf Sprachen L anwenden, für die $\simeq_L$ nur endlich viele Äquivalenzklassen hat.

Definition 7.12 (Kanonischer DEA $\mathcal{A}_L$ zu einer Sprache L)

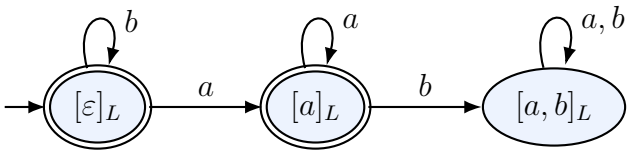
Sei $L \subseteq \Sigma^*$ eine Sprache, so dass $\simeq_L$ endlichen Index hat. Der *kanonische DEA* $\mathcal{A}_L = (Q', \Sigma, q_s', \delta', F')$ zu L ist definiert durch:

- $Q' := \{[u]_L \mid u \in \Sigma^*\}$
- $q_s' := [\varepsilon]_L$
- $\delta'([u]_L, a) := [ua]_L$ (repräsentantenunabhängig wegen Lemma 7.11, Punkt 2)
- $F' := \{[u]_L \mid u \in L\}$ (repräsentantenunabhängig wegen Bemerkung nach Def. 7.9).

Beachte, dass $\mathcal{A}_L$ mit Punkt 4 von Lemma 7.11 eine minimale Anzahl von Zuständen hat: es gibt keinen DEA, der $L(\mathcal{A}_L)$ erkennt und weniger Zustände hat.

Beispiel 7.10 (Fortsetzung)

Für die Sprache $L = b^*a^*$ ergibt sich damit folgender kanonischer Automat $\mathcal{A}_L$:



Lemma 7.13

Hat $\simeq_L$ endlichen Index, so ist $\mathcal{A}_L$ ein DEA mit $L = L(\mathcal{A}_L)$.

Beweis. Es gilt:

$$\begin{aligned}
L(\mathcal{A}_L) &= \{u \ : \ \hat{\delta}'(q'_s, u) \in F'\} \\
&= \{u \ : \ \hat{\delta}'([\varepsilon]_L, u) \in F'\} && \text{(Def. } q'_s\text{)} \\
&= \{u \ : \ [u]_L \in F'\} && \text{(wegen } \hat{\delta}'([\varepsilon]_L, u) = [u]_L\text{)} \\
&= \{u \ : \ u \in L\} && \text{(Def. } F'\text{)} \\
&= L
\end{aligned}$$

❑

Das folgende Resultat ist eine interessante Anwendung der Nerode-Rechtskongruenz und des kanonischen Automaten. Es liefert eine Charakterisierung von regulären Sprachen, die vollkommen unabhängig von endlichen Automaten ist.

Satz 7.14 (Satz von Myhill und Nerode)

Eine Sprache L ist regulär gdw. $\simeq_L$ endlichen Index hat.

Beweis.

„⇒“: Ergibt sich unmittelbar aus Lemma 7.11, 4).

„⇐“: Ergibt sich unmittelbar aus Lemma 7.13, da $\mathcal{A}_L$ DEA ist, der L akzeptiert.

❑

Der Satz von Myhill und Nerode liefert uns als Nebenprodukt eine weitere Methode, von einer Sprache zu beweisen, dass sie *nicht* regulär ist.

Beispiel 7.15 (nicht reguläre Sprache)

Die Sprache $L = \{a^n b^n \mid n \geq 0\}$ ist nicht regulär, da für $n \neq m$ gilt: $a^n \not\simeq_L a^m$. In der Tat gilt $a^n b^n \in L$, aber $a^m b^n \notin L$. Daher hat $\simeq_L$ unendlichen Index.

Wir zeigen nun, dass der reduzierte DEA minimal ist.

Satz 7.16 (Minimalität des reduzierten DEA)
Sei $\mathcal{A}$ ein DEA. Dann hat jeder DEA, der $L(\mathcal{A})$ erkennt, mindestens so viele Zustände wie der reduzierte DEA $\mathcal{A}_{\mathsf{red}}$.

Beweis. Sei $\mathcal{A} = (Q, \Sigma, q_s, \delta, F)$ und $\mathcal{A}_{\mathsf{red}} = (\widetilde{Q}, \Sigma, [q_s]_{\mathcal{A}}, \widetilde{\delta}, \widetilde{F})$. Wir definieren eine injektive Abbildung π, die jedem Zustand aus $\widetilde{Q}$ eine Äquivalenzklasse von $\simeq_L$ zuordnet. Es folgt, dass $\mathcal{A}_{\mathsf{red}}$ *höchstens* so viele Zustände hat, wie $\simeq_L$ Äquivalenzklassen (Schubfachprinzip), also ist er nach Punkt 4 von Lemma 7.11 minimal.

Sei $[q]_{\mathcal{A}} \in \widetilde{Q}$. Nach Definition von $\mathcal{A}_{\mathsf{red}}$ ist q in $\mathcal{A}$ von q_s aus erreichbar mit einem Wort w_q. Setze $\pi([q]_{\mathcal{A}}) = [w_q]_L$.

Es bleibt zu zeigen, dass π injektiv ist. Seien $[q]_{\mathcal{A}}, [p]_{\mathcal{A}} \in \widetilde{Q}$ mit $[q]_{\mathcal{A}} \neq [p]_{\mathcal{A}}$. Dann gilt $q \not\sim_{\mathcal{A}} p$ und es gibt $w \in \Sigma^*$ so dass

$$\hat{\delta}(q, w) \in F \Leftrightarrow \hat{\delta}(p, w) \in F$$

nicht gilt. Nach Wahl von w_p und w_q gilt dann aber auch

$$\hat{\delta}(q_s, w_q w) \in F \Leftrightarrow \hat{\delta}(q_s, w_p w) \in F$$

nicht und damit auch nicht

$$w_q w \in L \Leftrightarrow w_p w \in L$$

Es folgt $w_q \not\simeq_L w_p$, also $\pi([q]_{\mathcal{A}}) \neq \pi([p]_{\mathcal{A}})$ wie gewünscht. ❑

Es ist also sowohl der reduzierte Automat als auch der kanonische Automat von minimaler Größe. In der Tat ist der Zusammenhang zwischen beiden Automaten sogar noch viel enger: wir können zeigen, dass sie identisch bis auf Zustandsumbenennung sind. Dies wird durch den Begriff der Isomorphie beschrieben.

Definition 7.17 (Isomorphie von DEAs)
Zwei DEAs $\mathcal{A} = (Q, \Sigma, q_s, \delta, F)$ und $\mathcal{A}' = (Q', \Sigma, q'_s, \delta', F')$ sind *isomorph* (geschrieben $\mathcal{A} \simeq \mathcal{A}'$) gdw. es eine Bijektion $\pi : Q \to Q'$ gibt mit:

- $\pi(q_s) = q'_s$,
- $\pi(F) = \{\pi(q) \ : \ q \in F\} = F'$ und
- $\pi(\delta(q, a)) = \delta'(\pi(q), a)$ für alle $q \in Q$, $a \in \Sigma$.

Lemma 7.18
$\mathcal{A} \simeq \mathcal{A}' \ \Rightarrow \ L(\mathcal{A}) = L(\mathcal{A}')$.

Beweis. Es sei $\pi : Q \to Q'$ der Isomorphismus. Es folgt leicht per Induktion über $|w|$, dass $\pi(\hat{\delta}(q, w)) = \hat{\delta}'(\pi(q), w)$. Daher gilt:

$$\begin{aligned} w \in L(\mathcal{A}) \quad & \text{gdw.} \quad \hat{\delta}(q_s, w) \in F \\ & \text{gdw.} \quad \pi(\hat{\delta}(q_s, w)) \in F' \quad (\text{wegen } \pi(F) = F') \\ & \text{gdw.} \quad \hat{\delta}'(\pi(q_s), w) \in F' \\ & \text{gdw.} \quad \hat{\delta}'(q'_s, w) \in F' \quad (\text{wegen } q'_s = \pi(q_s)) \\ & \text{gdw.} \quad w \in L(\mathcal{A}'). \end{aligned}$$

❑

Wir können nun Minimalität und Eindeutigkeit des reduzierten Automaten zeigen.

Satz 7.19 (Isomorphie reduzierter und kanonischer DEA)
Es sei L eine reguläre Sprache und $\mathcal{A}$ ein DEA mit $L(\mathcal{A}) = L$. Dann gilt: der reduzierte Automat $\mathcal{A}_{\mathsf{red}}$ ist isomorph zum kanonischen Automaten $\mathcal{A}_L$.

Beweis. Es sei $\mathcal{A}_{\mathsf{red}} = (Q, \Sigma, q_s, \delta, F)$[2] und $\mathcal{A}_L = (Q', \Sigma, q'_0, \delta', F')$. Wir definieren eine Funktion $\pi : Q \to Q'$ und zeigen, dass sie ein Isomorphismus ist. Für jedes $q \in Q$ existiert (mindestens) ein $w_q \in \Sigma^*$ mit $\hat{\delta}(q_s, w_q) = q$, da in $\mathcal{A}_{\mathsf{red}}$ alle Zustände erreichbar sind. O. B. d. A. fixieren wir $w_{q_s} = \varepsilon$. Wir definieren $\pi(q) := [w_q]_L$.

I) π ist injektiv:
Wir müssen zeigen, dass aus $p \neq q$ auch $[w_p]_L \neq [w_q]_L$ folgt.
Da $\mathcal{A}_{\mathsf{red}}$ reduziert ist, sind verschiedene Zustände nicht äquivalent. Es gibt also mindestens ein w, für das

$$\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$$

nicht gilt. Das heißt aber, dass

$$\hat{\delta}(q_s, w_p w) \in F \Leftrightarrow \hat{\delta}(q_s, w_q w) \in F$$

nicht gilt und damit wiederum, dass $w_p w \in L \Leftrightarrow w_q w \in L$ *nicht* gilt. Also ist $w_p \not\equiv_L w_q$, d. h. $[w_p]_L \neq [w_q]_L$.

II) π ist surjektiv:
Folgt aus Injektivität und $|Q| \geq |Q'|$ (Punkt 4 Lemma 7.11).

III) $\pi(q_s) = q'_s$:
Da $w_{q_s} = \varepsilon$ und $q'_s = [\varepsilon]_L$.

[2]Wir lassen die Tilden in diesem Beweis der Einfachheit halber weg.

IV) $\pi(F) = F'$:

$$\begin{array}{lll} q \in F & \text{gdw.} & \hat{\delta}(q_s, w_q) \in F \quad \text{(Wahl } w_q\text{)} \\ & \text{gdw.} & w_q \in L \\ & \text{gdw.} & [w_q]_L \in F' \qquad \text{(Def. } F'\text{)} \\ & \text{gdw.} & \pi(q) \in F'. \end{array}$$

V) $\pi(\delta(q,a)) = \delta'(\pi(q), a)$:
Als Abkürzung setze $p = \delta(q,a)$. Die Hilfsaussage

$$\hat{\delta}(q_s, w_p) = \hat{\delta}(q_s, w_q a) \qquad (*)$$

gilt wegen:

$$\begin{array}{rcll} \hat{\delta}(q_s, w_p) & = & p & \text{(Wahl } w_p\text{)} \\ & = & \delta(q,a) & \\ & = & \delta(\hat{\delta}(q_s, w_q), a) & \text{(Wahl } w_q\text{)} \\ & = & \hat{\delta}(q_s, w_q a). & \end{array}$$

Mittels $(*)$ zeigen wir nun:

$$\begin{array}{rcll} \pi(p) & = & [w_p]_L & \text{(Def. } \pi\text{)} \\ & = & [w_q a]_L & \text{(folgt aus } (*) \text{ und } L(\mathcal{A}_{\mathsf{red}}) = L\text{)} \\ & = & \delta'([w_q]_L, a) & \text{(Def. } \mathcal{A}_L\text{)} \\ & = & \delta'(\pi(q), a). & \text{(Def. } \pi\text{)} \end{array}$$

❑

Dieser Satz zeigt auch folgende interessante Eigenschaften:

- Der reduzierte Automat $\mathcal{A}_{\mathsf{red}}$ ist unabhängig vom ursprünglichen DEA $\mathcal{A}$:
  wenn $L(\mathcal{A}) = L(\mathcal{B}) = L$, dann gilt wegen $\mathcal{A}_{\mathsf{red}} \simeq \mathcal{A}_L \simeq \mathcal{B}_{\mathsf{red}}$ auch $\mathcal{A}_{\mathsf{red}} \simeq \mathcal{B}_{\mathsf{red}}$ (denn die Komposition zweier Isomorphismen ergibt wieder einen Isomorphismus).
- Für jede reguläre Sprache L gibt es einen eindeutigen minimalen DEA:
  Wenn $L(\mathcal{A}) = L(\mathcal{B}) = L$ und $\mathcal{A}$ und $\mathcal{B}$ minimale Zustandszahl unter allen DEAs haben, die L akzeptieren, dann enthalten $\mathcal{A}$ und $\mathcal{B}$ weder unerreichbare noch äquivalente Zustände und der jeweilige reduzierte Automat ist identisch zum ursprünglichen Automaten. Damit gilt $\mathcal{A} = \mathcal{A}_{\mathsf{red}} \simeq \mathcal{A}_L \simeq \mathcal{B}_{\mathsf{red}} = \mathcal{B}$, also auch $\mathcal{A} \simeq \mathcal{B}$.

Im Prinzip liefert Satz 7.19 zudem eine Methode, um für zwei Automaten zu entscheiden, ob sie dieselbe Sprache akzeptieren:

Korollar 7.20

Es seien $\mathcal{A}$ und $\mathcal{A}'$ DEAs. Dann gilt: $L(\mathcal{A}) = L(\mathcal{A}')$ gdw. $\mathcal{A}_{\mathsf{red}} \simeq \mathcal{A}'_{\mathsf{red}}$.

Wir können die reduzierten Automaten wie beschrieben konstruieren. Für gegebene Automaten können wir feststellen, ob sie isomorph sind. Da wir allerdings keinen Polynomialzeitalgorithmus für das Isomorphieproblem kennen, ist diese Methode nicht optimal.

Sind NEAs anstelle von DEAs gegeben, so können wir diese zuerst determinisieren und dann das Korollar anwenden.

Schlussbemerkungen zu Teil I

Zum Abschluss von Teil I erwähnen wir einige hier aus Zeitgründen nicht behandelte Themenbereiche:

Andere Varianten von endlichen Automaten:

NEAs/DEAs mit Ausgabe (sogenannte *Transduktoren*) haben Übergänge $p \xrightarrow{a/v}_{\mathcal{A}} q$, wobei $v \in \Gamma^*$ ein Wort über einem Ausgabealphabet ist. Solche Automaten beschreiben Funktionen $\Sigma^* \to \Gamma^*$. *2-Wege-Automaten* können sich sowohl vorwärts als auch rückwärts auf der Eingabe bewegen. *Alternierende Automaten* generalisieren NEAs: zusätzlich zu den nichtdeterministischen Übergängen, die einer existentiellen Quantifizierung entsprechen, gibt es hier auch universell quantifizierte Übergänge.

Algebraische Theorie formaler Sprachen:

Jeder Sprache L wird ein Monoid M_L (syntaktisches Monoid) zugeordnet. Klassen von Sprachen entsprechen dann Klassen von Monoiden, z. B. L ist regulär gdw. M_L endlich ist. Dies ermöglicht einen sehr fruchtbaren algebraischen Zugang zur Automatentheorie.

Automaten auf unendlichen Wörtern:

Hier geht es um Automaten, die unendliche Wörter (unendliche Folgen von Symbolen) als Eingabe erhalten. Die Akzeptanz via Erreichen eines akzeptierenden Zustandes funktioniert hier natürlich nicht mehr und es werden verschiedene Akzeptanzbedingungen studiert wie z. B. Büchi-Akzeptanz und Rabin-Akzeptanz.

Baumautomaten:

Diese Automaten erhalten Bäume statt Wörter als Eingabe. Eine streng lineare Abarbeitung wie bei Wörtern ist in diesem Fall natürlich nicht möglich. Es wird zwischen Top-Down- und Bottom-Up-Automaten unterschieden.

II. Grammatiken, kontextfreie Sprachen und Kellerautomaten

Einführung

Der zweite Teil der Vorlesung beschäftigt sich hauptsächlich mit der Klasse der kontextfreien Sprachen sowie mit Kellerautomaten, dem zu dieser Sprachfamilie passenden Automatenmodell. Die Klasse der kontextfreien Sprachen ist allgemeiner als die der regulären Sprachen, und dementsprechend können Kellerautomaten als eine Erweiterung von endlichen Automaten verstanden werden. Kontextfreie Sprachen spielen in der Informatik eine wichtige Rolle, da durch sie z. B. die Syntax von Programmiersprachen (zumindest in großen Teilen) beschreibbar ist.

Bevor wir uns im Detail den kontextfreien Sprachen zuwenden, führen wir Grammatiken als allgemeines Mittel zum Generieren formaler Sprachen ein. Wir werden sehen, dass sich sowohl die regulären als auch die kontextfreien Sprachen und weitere, noch allgemeinere Sprachklassen mittels Grammatiken definieren lassen. Diese Sprachklassen sind angeordnet in der bekannten Chomsky-Hierarchie der formalen Sprachen.

7. Die Chomsky-Hierarchie

Grammatiken dienen dazu, Wörter zu erzeugen. Sie enthalten *Regeln*, die es erlauben, ein Wort durch ein anderes Wort zu ersetzen (aus ihm *abzuleiten*). Die *erzeugte Sprache* ist die Menge der Wörter, die ausgehend von einem *Startsymbol* durch wiederholtes Ersetzen erzeugt werden können.

Beispiel 7.1

Regeln:	$S \longrightarrow aSb$	(1)
	$S \longrightarrow \varepsilon$	(2)
Startsymbol:	S	

Eine mögliche Ableitung eines Wortes ist:

$S \xrightarrow{1} aSb \xrightarrow{1} aaSbb \xrightarrow{1} aaaSbbb \xrightarrow{2} aaabbb$

Das Symbol S ist hier ein Hilfssymbol (*nichtterminales Symbol*) und wir sind nur an erzeugten Wörtern interessiert, die das Hilfssymbol nicht enthalten (*Terminalwörter*). Es ist leicht zu sehen, dass dies in diesem Fall genau die Wörter $a^n b^n$ mit $n \geq 0$ sind.

Definition 7.2 (Grammatik)

Eine *Grammatik* ist von der Form $G = (N, \Sigma, P, S)$, wobei

- N und Σ endliche, disjunkte Alphabete von *Nichtterminalsymbolen* bzw. *Terminalsymbolen* sind,
- $S \in N$ das *Startsymbol* ist,
- $P \subseteq (N \cup \Sigma)^+ \times (N \cup \Sigma)^*$ eine endliche Menge von Ersetzungsregeln (*Produktionen*) ist.

Der besseren Lesbarkeit halber schreiben wir Produktionen $(u, v) \in P$ gewöhnlich als $u \longrightarrow v$.

Beachte, dass die rechte Seite von Produktionen aus dem leeren Wort bestehen darf, die linke jedoch nicht. Außerdem sei darauf hingewiesen, dass sowohl Nichtterminalsymbole als auch Terminalsymbole durch eine Ersetzungsregel ersetzt werden dürfen.

Beispiel 7.3

Folgendes Tupel ist eine Grammatik: $G = (N, \Sigma, P, S)$ mit

- $N = \{S, B\}$,
- $\Sigma = \{a, b, c\}$,
- $P = \{S \longrightarrow aSBc,$
  $S \longrightarrow abc,$
  $cB \longrightarrow Bc,$
  $bB \longrightarrow bb\}.$

Im Folgenden schreiben wir meistens Elemente von N mit Großbuchstaben und Elemente von Σ mit Kleinbuchstaben.

Wir definieren nun, was es heißt, dass ein Wort durch Anwenden der Regeln aus einem anderen ableitbar ist.

Definition 7.4 (durch eine Grammatik erzeugte Sprache)
Sei $G = (N, \Sigma, P, S)$ eine Grammatik und x, y Wörter aus $(N \cup \Sigma)^*$.

1) *y aus x direkt ableitbar:*
   $x \vdash_G y$ gdw. $x = x_1 u x_2$ und $y = x_1 v x_2$ mit $u \longrightarrow v \in P$ und $x_1, x_2 \in (N \cup \Sigma)^*$.
2) *y aus x in n Schritten ableitbar:*
   $x \vdash_G^n y$ gdw. $x \vdash_G x_1 \vdash_G \cdots \vdash_G x_{n-1} \vdash_G y$ für $x_1, \ldots, x_{n-1} \in (N \cup \Sigma)^*$.
3) *y aus x ableitbar:*
   $x \vdash_G^* y$ gdw. $x \vdash_G^n y$ für ein $n \geq 0$.
4) Die durch G *erzeugte Sprache* ist
   $L(G) := \{w \in \Sigma^* \ : \ S \vdash_G^* w\}$.

Wir sind also bei der erzeugten Sprache nur an den in G aus S ableitbaren Terminalwörtern interessiert.

Beispiel 7.3 (Fortsetzung)
Zwei Beispielableitungen in der Grammatik aus Beispiel 7.3:

$$
\begin{aligned}
S &\vdash_G abc, && \text{d. h. } abc \in L(G)\\
S &\vdash_G aSBc\\
&\vdash_G aaSBcBc\\
&\vdash_G aaabcBcBc\\
&\vdash_G aaabBccBc\\
&\vdash_G^2 aaabBBccc\\
&\vdash_G^2 aaabbbccc, && \text{d. h. } aaabbbccc \in L(G).
\end{aligned}
$$

Die erzeugte Sprache ist $L(G) = \{a^n b^n c^n \ : \ n \geq 1\}$.

Beweis.

„$\supseteq$“: Für $n = 1$ ist $abc \in L(G)$ klar. Für $n > 1$ ist leicht zu sehen, dass:

$$S \ \vdash_G^{n-1} \ a^{n-1} S (Bc)^{n-1} \ \vdash_G \ a^n bc (Bc)^{n-1} \ \vdash_G^* \ a^n b B^{n-1} c^n \ \vdash_G^{n-1} \ a^n b^n c^n.$$

„$\subseteq$“: Sei $S \vdash_G^* w$ mit $w \in \Sigma^*$. Offenbar wird die Regel $S \longrightarrow abc$ in der Ableitung von w genau einmal angewendet. Vor dieser Anwendung können nur die Regeln

$S \longrightarrow aSBc$ und $cB \longrightarrow Bc$ angewendet werden, da noch keine b's generiert wurden. Die Anwendung von $S \longrightarrow abc$ hat damit die Form

$$a^n Su \vdash a^{n+1}bcu \text{ mit } u \in \{c, B\}^* \text{ und } |u|_B = |u|_c = n$$

(formaler Beweis per Induktion über die Anzahl der Regelanwendungen). Nun sind nur noch $cB \longrightarrow Bc$ und $bB \longrightarrow bb$ anwendbar. Alle dabei entstehenden Wörter haben die folgende Form (formaler Beweis per Induktion über die Anzahl der Regelanwendungen):

$$a^{n+1}b^{k+1}v \text{ mit } v \in \{c, B\}^*, \quad |v|_B = n - k \text{ und } |v|_c = n + 1.$$

Jedes Terminalwort dieser Form erfüllt $|v|_B = 0$, also $n = k$, und damit hat das Wort die Form $a^n b^n c^n$, $n \geq 1$.

❑

Beispiel 7.5

Betrachte die Grammatik $G = (N, \Sigma, P, S)$ mit

- $N = \{S, B\}$,
- $\Sigma = \{a, b\}$,
- $P = \{S \longrightarrow aS,$
  $S \longrightarrow bS,$
  $S \longrightarrow abB,$
  $B \longrightarrow aB,$
  $B \longrightarrow bB,$
  $B \longrightarrow \varepsilon\}$.

Die erzeugte Sprache ist $L(G) = \Sigma^* \cdot \{a\} \cdot \{b\} \cdot \Sigma^*$.

Die Grammatiken aus Beispiel 7.5, 7.3 und 7.1 gehören zu unterschiedlichen Sprachklassen, die alle durch Grammatiken erzeugt werden können. Sie sind angeordnet in der Chomsky-Hierarchie.

Definition 7.6 (Chomsky-Hierarchie, Typen von Grammatiken)

Es sei $G = (N, \Sigma, P, S)$ eine Grammatik.

- Jede Grammatik G ist Grammatik vom **Typ 0.**
- G ist Grammatik vom **Typ 1 (monoton),** falls alle Regeln *nicht verkürzend* sind, also die Form $u \longrightarrow w$ haben wobei $u, w \in (\Sigma \cup N)^+$ und $|w| \geq |u|$.

  Ausnahme: Die Regel $S \longrightarrow \varepsilon$ ist erlaubt, wenn S in keiner Produktion auf der rechten Seite vorkommt.
- G ist Grammatik vom **Typ 2 (kontextfrei),** falls alle Regeln die Form $A \longrightarrow w$ haben mit $A \in N, w \in (\Sigma \cup N)^*$.
- G ist Grammatik vom **Typ 3 (rechtslinear),** falls alle Regeln die Form $A \longrightarrow wB$ oder $A \longrightarrow w$ haben mit $A, B \in N$, $w \in \Sigma^*$.

Die *kontextfreien* Sprachen heißen deshalb so, weil die linke Seite jeder Produktion nur aus einem Nichtterminalsymbol A besteht, das unabhängig vom *Kontext im Wort* (also dem Teilwort links von A und dem Teilwort rechts von A) ersetzt wird. Bei monotonen Grammatiken sind hingegen Regeln

$$u_1 A u_2 \longrightarrow u_1 w u_2$$

erlaubt, wenn $|w| \geq 1$. Hier ist die Ersetzung von A durch w abhängig davon, dass der richtige Kontext (u_1 links und u_2 rechts, beides aus $(N \cup \Sigma)^*$) im Wort vorhanden ist. Es kann sogar gezeigt werden, dass es keine Beschränkung der Allgemeinheit ist, monotone Grammatiken ausschließlich durch Regeln der obigen Form zu definieren. Die durch monotone Grammatiken erzeugten Sprachen werden daher auch *kontextsensitive Sprachen* genannt.

Eine sehr wichtige Eigenschaft von monotonen Grammatiken ist, dass die Anwendung einer Produktion das Wort nicht verkürzen kann. Vor diesem Hintergrund ist auch die Ausnahme $S \longrightarrow \varepsilon$ zu verstehen: sie dient dazu, das leere Wort generieren zu können, was ohne Verkürzen natürlich nicht möglich ist. Wenn diese Regel verwendet wird, dann sind Regeln wie $aAb \to aSb$ aber implizit verkürzend, da Sie das Ableiten von ab aus aAb erlauben. Um das zu verhindern, darf in der Gegenwart von $S \longrightarrow \varepsilon$ das Symbol S nicht auf der rechten Seite von Produktionen verwendet werden.

Beispiel 7.7

Die Grammatik aus Beispiel 7.1 ist vom Typ 2. Sie ist nicht vom Typ 1, da $S \longrightarrow \varepsilon$ vorhanden ist, aber S auf der rechten Seite von Produktionen verwendet wird. Es gibt aber eine Grammatik vom Typ 1, die dieselbe Sprache erzeugt:

$$\begin{array}{lcl} S & \longrightarrow & \varepsilon \\ S & \longrightarrow & S' \\ S' & \longrightarrow & ab \\ S' & \longrightarrow & aS'b \end{array}$$

Die Grammatik aus Beispiel 7.3 ist vom Typ 1. Wir werden später sehen, dass es keine Grammatik vom Typ 2 gibt, die die Sprache aus diesem Beispiel generiert. Die Grammatik aus Beispiel 7.5 ist vom Typ 3.

Die unterschiedlichen Typen von Grammatiken führen zu unterschiedlichen *Typen von Sprachen.*

Definition 7.8 (Klasse der Typ-i-Sprachen)

Für $i = 0, 1, 2, 3$ ist die *Klasse der Typ-i-Sprachen definiert* als

$$\mathcal{L}_i := \{L(G) \mid G \text{ ist Grammatik vom Typ } i\}.$$

Nach Definition kann eine Grammatik von Typ i auch Sprachen höheren Typs $j \geq i$ generieren (aber nicht umgekehrt). So erzeugt beispielsweise die folgende Typ-1-Grammatik die Sprache $\{a^n b^n \ : \ n \geq 0\}$, welche nach Beispiel 7.1 von Typ 2 ist:

$$\begin{aligned} S &\longrightarrow \varepsilon \\ S &\longrightarrow ab \\ S &\longrightarrow aXb \\ aXb &\longrightarrow aaXbb \\ X &\longrightarrow ab \end{aligned}$$

Offensichtlich ist jede Grammatik von Typ 3 auch eine von Typ 2 und jede Grammatik von Typ 1 auch eine von Typ 0. Da Grammatiken von Typ 2 und 3 das Verkürzen des abgeleiteten Wortes erlauben, sind solche Grammatiken nicht notwendigerweise von Typ 1. Wir werden jedoch später sehen, dass jede Typ-2-Grammatik in eine Typ-1-Grammatik umgewandelt werden kann, die dieselbe Sprache erzeugt. Daher bilden die assoziierten Sprachtypen eine Hierarchie. Diese ist sogar strikt.

Lemma 7.9

$\mathcal{L}_3 \subsetneq \mathcal{L}_2 \subsetneq \mathcal{L}_1 \subsetneq \mathcal{L}_0$

Beweis. Nach Definition der Grammatiktypen gilt offenbar $\mathcal{L}_3 \subseteq \mathcal{L}_2$ und $\mathcal{L}_1 \subseteq \mathcal{L}_0$. Die Inklusion $\mathcal{L}_2 \subseteq \mathcal{L}_1$ werden wir später zeigen (Satz 9.16). Auch die Striktheit der Inklusionen werden wir erst später beweisen. ❑

8. Rechtslineare Grammatiken und reguläre Sprachen

Wir zeigen, dass rechtslineare Grammatiken genau die regulären Sprachen generieren. Damit haben wir eine weitere, unabhängige Charakterisierung dieser Klasse von Sprachen gefunden, und alle Resultate, die wir bereits für die reguläre Sprachen bewiesen haben, gelten auch für Typ-3-Sprachen.

Satz 8.1

Die Typ-3-Sprachen sind genau die regulären Sprachen, d. h.:

$$\mathcal{L}_3 = \{L \mid L \text{ ist regulär}\}.$$

Beweis.

„$\subseteq$“: **Jede Typ-3-Sprache ist regulär:**

Sei $L \in \mathcal{L}_3$, d. h. $L = L(G)$ für eine Typ-3-Grammatik $G = (N, \Sigma, P, S)$. Es gilt $w_1 \cdots w_n \in L(G)$ (für $w_i \in \Sigma^*$) gdw. es eine Ableitung

$$S = B_0 \vdash_G w_1 B_1 \vdash_G w_1 w_2 B_2 \vdash_G \ldots \vdash_G w_1 \ldots w_{n-1} B_{n-1} \vdash_G w_1 \ldots w_{n-1} w_n \quad (*)$$

gibt mit Produktionen $B_{i-1} \longrightarrow w_i B_i \in P$ $(i = 1, \ldots, n)$ und $B_{n-1} \longrightarrow w_n \in P$.

Diese Ableitung ähnelt dem Lauf eines NEA mit Wortübergängen auf dem Wort $w_1 \cdots w_n$, wobei die Nichtterminale die Zustände sind und die Produktionen die Übergänge beschreiben.

Wir konstruieren nun einen *NEA mit Wortübergängen*, der die Nichtterminalsymbole von G und einen neuen Zustand Ω als Zustände hat:

$\mathcal{A} = (N \cup \{\Omega\}, \Sigma, S, \Delta, \{\Omega\})$, wobei

- $\Omega \notin N$ akzeptierender Zustand ist und
- $\Delta = \{(A, w, B) \mid A \longrightarrow wB \in P\} \cup \{(A, w, \Omega) \mid A \longrightarrow w \in P\}$.

Ableitungen der Form $(*)$ entsprechen nun genau Läufen in $\mathcal{A}$:

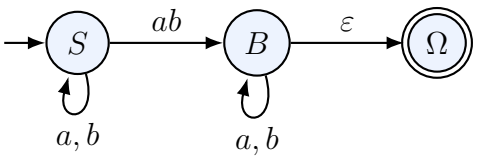
$$S \xrightarrow{w_1}_{\mathcal{A}} B_1 \xrightarrow{w_2}_{\mathcal{A}} \cdots \xrightarrow{w_n}_{\mathcal{A}} \Omega.$$

Dies zeigt $L(\mathcal{A}) = L(G)$.

Beispiel 7.5 (Fortsetzung)

Die Grammatik

$$\begin{aligned} P = \{ & S \longrightarrow aS, \\ & S \longrightarrow bS, \\ & S \longrightarrow abB, \\ & B \longrightarrow aB, \\ & B \longrightarrow bB, \\ & B \longrightarrow \varepsilon\} \end{aligned}$$



liefert den abgebildeten NEA mit Wortübergängen.

„$\supseteq$“: **Jede reguläre Sprache ist eine Typ-3-Sprache:**

Sei $L = L(\mathcal{A})$ für einen NEA $\mathcal{A} = (Q, \Sigma, q_s, \Delta, F)$. Wir definieren daraus eine Typ-3-Grammatik $G = (N, \Sigma, P, S)$ wie folgt:

$$\begin{aligned} N &:= Q, \\ S &:= q_s, \\ P &:= \{p \longrightarrow aq \mid (p, a, q) \in \Delta\} \cup \{p \longrightarrow \varepsilon \mid p \in F\}. \end{aligned}$$

Ein Lauf in $\mathcal{A}$ der Form

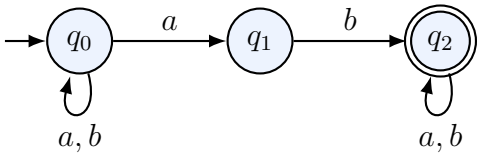
$$q_s \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_n} q_n$$

mit $q_n \in F$ entspricht nun genau einer Ableitung

$$q_s \vdash_G a_1 q_1 \vdash_G a_1 a_2 q_2 \vdash_G \ldots \vdash_G a_1 \ldots a_n q_n \vdash_G a_1 \ldots a_n.$$

Beispiel 8.2

Der NEA



liefert die Grammatik mit den rechtslinearen Produktionen

$$\begin{aligned} P = \{ & q_0 \longrightarrow aq_0, \\ & q_0 \longrightarrow bq_0, \\ & q_0 \longrightarrow aq_1, \\ & q_1 \longrightarrow bq_2, \\ & q_2 \longrightarrow aq_2, \\ & q_2 \longrightarrow bq_2, \\ & q_2 \longrightarrow \varepsilon\} \end{aligned}$$

❑

Korollar 8.3

$\mathcal{L}_3 \subsetneq \mathcal{L}_2$.

Beweis. Wir wissen bereits, dass $\mathcal{L}_3 \subseteq \mathcal{L}_2$ gilt. Außerdem haben wir mit Beispiel 7.1 $L := \{a^n b^n \ : \ n \geq 0\} \in \mathcal{L}_2$. Wir haben bereits gezeigt, dass L nicht regulär ist, d. h. mit Satz 8.1 folgt $L \notin \mathcal{L}_3$. ❑

Beispiel 8.4

Als ein weiteres Beispiel für eine kontextfreie Sprache, die nicht regulär ist, betrachten wir $L = \{a^n b^m \ : \ n \neq m\}$ (vgl. Beispiele 3.7, 4.2). Wir können diese Sprache mit der folgenden kontextfreien Grammatik erzeugen:

$G = (N, \Sigma, P, S)$ mit

- $N = \{S, A, B\}$
- $\Sigma = \{a, b\}$
- $P = \{S \longrightarrow aA, \quad S \longrightarrow Bb,$
  $A \longrightarrow aAb, \quad B \longrightarrow aBb,$
  $A \longrightarrow aA, \quad B \longrightarrow Bb,$
  $A \longrightarrow \varepsilon, \quad B \longrightarrow \varepsilon\}$

Es gilt nun:

- $A \vdash_G^* w \in \{a, b\}^* \quad \Rightarrow \quad w = a^n b^m$ mit $n \geq m$,
- $B \vdash_G^* w \in \{a, b\}^* \quad \Rightarrow \quad w = a^n b^m$ mit $n \leq m$,

woraus sich ergibt:

- $S \vdash_G^* w \in \{a, b\}^* \quad \Rightarrow \quad w = aa^n b^m$ mit $n \geq m$ oder $w = a^n b^m b$ mit $n \leq m$,

d. h. $L(G) = \{a^n b^m \ : \ n \neq m\}$.

9. Normalformen und Entscheidungsprobleme

Es existieren verschiedene *Normalformen* für kontextfreie Grammatiken, bei denen die syntaktische Form der Regeln weiter eingeschränkt wird, ohne dass die Klasse der erzeugten Sprachen sich ändert. Wir werden insbesondere die *Chomsky-Normalform* kennen lernen und sie verwenden, um einen effizienten Algorithmus für das Wortproblem für kontextfreie Sprachen zu entwickeln. Zudem ist jede kontextfreie Grammatik in Chomsky-Normalform auch eine Typ-1-Grammatik, was die noch ausstehende Inklusion $\mathcal{L}_2 \subseteq \mathcal{L}_1$ zeigt. Wir werden auch das Leerheitsproblem und das Äquivalenzproblem diskutieren.

Zwei Grammatiken heißen *äquivalent*, falls sie dieselbe Sprache erzeugen.

9.1. Vereinfachung kontextfreier Grammatiken und das Leerheitsproblem

Zunächst zeigen wir, wie wir „überflüssige“ Symbole aus kontextfreien Grammatiken eliminieren können. Das ist zum späteren Herstellen der Chomsky-Normalform nicht unbedingt notwendig; es ist aber trotzdem ein natürlicher erster Schritt zur Vereinfachung einer Grammatik.

Definition 9.1 (terminierende, erreichbare Symbole; reduzierte Grammatik)
Es sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik.

1) $A \in N$ heißt *terminierend*, falls es ein $w \in \Sigma^*$ gibt mit $A \vdash_G^* w$.
2) $A \in N$ heißt *erreichbar*, falls es $u, v \in (\Sigma \cup N)^*$ gibt mit $S \vdash_G^* uAv$.
3) G heißt *reduziert*, falls alle Elemente von N *erreichbar* und *terminierend* sind.

Die folgenden zwei Lemmata bilden die Grundlage zum Wandeln einer kontextfreien Grammatik in eine reduzierte kontextfreie Grammatik.

Lemma 9.2
Für jede kontextfreie Grammatik $G = (N, \Sigma, P, S)$ ist die Menge der terminierenden Symbole berechenbar.

Beweis. Wir definieren dazu

$$T_1 := \{A \in N \mid \exists w \in \Sigma^* : A \longrightarrow w \in P\},$$
$$T_{i+1} := T_i \cup \{A \in N \mid \exists w \in (\Sigma \cup T_i)^* : A \longrightarrow w \in P\}.$$

Es gilt

$$T_1 \subseteq T_2 \subseteq \cdots \subseteq N.$$

Da N endlich ist, gibt es ein k mit $T_k = T_{k+1} = \bigcup_{i \geq 1} T_i$.

Behauptung: $T_k = \{A \in N \ : \ A \text{ ist terminierend}\}$, denn:

„$\subseteq$“: Zeige per Induktion über i: alle Elemente von T_i, $i \geq 1$, sind terminierend:

- $i = 1$: $\quad A \in T_1 \Rightarrow A \vdash_G w \in \Sigma^*$, also ist A terminierend.
- $i \to i+1$:

  Wenn $A \in T_{i+1}$, dann gibt es nach Definition T_{i+1} zwei Fälle:

  - $A \in T_i$: Dann ist A nach Induktionsvoraussetzung (IV) terminierend.
  - $A \longrightarrow w \in P$ für ein $w \in (\Sigma \cup T_i)^*$. Sei $w = u_1 B_1 u_2 B_2 \cdots u_n B_n u_{n+1}$ mit $u_j \in \Sigma^*$ und $B_j \in T_i$. Nach IV sind alle B_j terminierend, also $B_j \vdash_G^* w_j \in \Sigma^*$. Es folgt $A \vdash_G^* u_1 w_1 u_2 w_2 \cdots u_n w_n u_{n+1} \in \Sigma^*$, also ist A terminierend.

„$\supseteq$“: Zeige per Induktion über i:

$A \vdash_G^i w \in \Sigma^* \Rightarrow A \in T_\ell$ für alle $\ell \geq i$.

- $i = 1$: $\quad A \vdash_G^1 w \in \Sigma^* \Rightarrow A \longrightarrow w \in P \Rightarrow A \in T_1$.
- $i \to i+1$: $A \vdash_G^{i+1} w \in \Sigma^*$
  $\Rightarrow A \vdash_G u_1 B_1 \cdots u_n B_n u_{n+1} \vdash_G^i u_1 w_1 \cdots u_n w_n u_{n+1} = w$,
  mit $u_j \in \Sigma^*$ und $B_j \vdash_G^{i_j} w_j \in \Sigma^*$ für $i_j \leq i$, $1 \leq \ell \leq n$
  $\Rightarrow B_j \in T_i$ für alle j (Induktion)
  $\Rightarrow A \in T_{i+1}$.

❑

Beispiel 9.3

$$P = \{ \begin{array}{ll} S \longrightarrow A, & S \longrightarrow aCbBa, \\ A \longrightarrow aBAc, & B \longrightarrow Cab, \\ C \longrightarrow AB, & C \longrightarrow aa \qquad \} \end{array}$$

$$T_1 = \{C\} \quad \subsetneq \quad T_2 = \{C, B\} \quad \subsetneq \quad T_3 = \{C, B, S\} \quad = \quad T_4$$

Es ist also A das einzige nichtterminierende Symbol.

Das Leerheitsproblem für kontextfreie Grammatiken besteht darin, für eine gegebene kontextfreie Grammatik G zu entscheiden, ob $L(G) = \emptyset$. Lemma 9.2 hat folgende Konsequenz.

Satz 9.4

Das Leerheitsproblem ist für kontextfreie Grammatiken in polynomieller Zeit entscheidbar.

Beweis. Offenbar gilt $L(G) \neq \emptyset$ gdw. $\exists w \in \Sigma^* : S \vdash_G^* w$ gdw. S ist terminierend. Es ist leicht zu sehen, dass unsere obige Konstruktion zum Berechnen aller terminierenden Symbole in polynomieller Zeit ausgeführt werden kann. ❑

Wir werden in *Theoretische Informatik 2* sehen, dass das Leerheitsproblem für Grammatiken der Typen 0 und 1 nicht entscheidbar ist.

Lemma 9.5

Für jede kontextfreie Grammatik $G = (N, \Sigma, P, S)$ ist die Menge der erreichbaren Nichtterminalsymbole berechenbar.

Beweis. Wir definieren dazu

$$E_0 := \{S\},$$
$$E_{i+1} := E_i \cup \{A \ : \ \exists B \in E_i \text{ mit Regel } B \longrightarrow u_1 A u_2 \in P\}.$$

Es gilt

$$E_0 \subseteq E_1 \subseteq E_2 \subseteq \cdots \subseteq N.$$

Da N endlich ist, gibt es ein k mit $E_k = E_{k+1}$ und damit $E_k = \bigcup_{i \geq 0} E_i$.

Behauptung: $E_k = \{A \in N \mid A \text{ ist erreichbar}\}$, denn:

„$\subseteq$“: Zeige durch Induktion über i: E_i enthält nur erreichbare Symbole.

„$\supseteq$“: Zeige durch Induktion über i: $S \vdash_G^i uAv \Rightarrow A \in E_i$.

❏

Beispiel 9.6

$$\begin{aligned} P \ = \ \{ \quad & S \longrightarrow aS, \quad A \longrightarrow ASB, \\ & S \longrightarrow SB, \quad A \longrightarrow C, \\ & S \longrightarrow SS, \quad B \longrightarrow Cb, \\ & S \longrightarrow \varepsilon \qquad\qquad\quad \} \end{aligned}$$

$$E_0 = \{S\} \quad \subsetneq \quad E_1 = \{S, B\} \quad \subsetneq \quad E_2 = \{S, B, C\} \quad = \quad E_3$$

Es ist also A das einzige unerreichbare Symbol.

Lemma 9.2 und 9.5 zusammen zeigen, wie unerreichbare und nichtterminierende Symbole eliminiert werden können.

Satz 9.7

Zu jeder kontextfreien Grammatik G mit $L(G) \neq \emptyset$ kann eine äquivalente reduzierte kontextfreie Grammatik konstruiert werden.

Beweis. Sei $G = (N, \Sigma, P, S)$.

Erster Schritt: Eliminieren nichtterminierender Symbole.

G' ist die Einschränkung von G auf terminierende Nichtterminale, also
$G' := (N', \Sigma, P', S)$ mit

- $N' := \{A \in N \mid A \text{ ist terminierend in } G\}$,
- $P' := \{A \longrightarrow w \in P \mid A \in N', w \in (N' \cup \Sigma)^*\}$.

Beachte: Wegen $L(G) \neq \emptyset$ ist S terminierend, also $S \in N'$.

Zweiter Schritt: Eliminieren unerreichbarer Symbole.

$G'' := (N'', \Sigma, P'', S)$ ist die Einschränkung von G' auf erreichbare Nichtterminale, wobei

- $N'' := \{A \in N' \mid A \text{ ist erreichbar in } G'\}$,
- $P'' := \{A \longrightarrow w \in P' \mid A \in N''\}$.

Es ist leicht zu sehen, dass $L(G) = L(G') = L(G'')$ und dass G'' reduziert ist. ❑

Vorsicht: Die Reihenfolge der beiden Schritte ist wichtig, dann das Eliminieren der nichtterminierenden Symbole kann zusätzliche Symbole unerreichbar machen (aber nicht umgekehrt). Betrachte zum Beispiel die Grammatik $G = (N, \Sigma, P, S)$ mit

$$P = \{S \longrightarrow \varepsilon, \quad S \longrightarrow AB, \quad A \longrightarrow a\}.$$

In G sind alle Symbole erreichbar. Eliminieren wir also zuerst die unerreichbaren Symbole, so ändert sich die Grammatik nicht. Das einzige nichtterminierende Symbol ist B und dessen Elimination liefert

$$P' = \{S \to \varepsilon, \quad A \longrightarrow a\}$$

Diese Grammatik ist nicht reduziert, da nun A unerreichbar ist.

Beachte auch, dass eine Grammatik G mit $L(G) = \emptyset$ niemals reduziert sein kann, da jede Grammatik ein Startsymbol S enthalten muss und S in G nicht terminierend sein kann.

9.2. Die Chomsky-Normalform

Wie zeigen nun, dass jede kontextfreie Grammatik in eine äquivalente Grammatik in *Chomsky-Normalform* gewandelt werden kann.

Definition 9.8

Eine kontextfreie Grammatik (N, Σ, P, S) ist in *Chomsky-Normalform*, wenn alle Ableitungsregeln die folgende Form haben:

- $A \longrightarrow a$ oder $A \longrightarrow BC$ mit $A, B, C \in N,\ a \in \Sigma$
- $S \longrightarrow \varepsilon$ ist erlaubt, wenn es keine Regeln $A \to BC$ mit $S \in \{B, C\}$ gibt.

Dies geschieht in vier Schritten:

1. Aufheben der Mischung von Terminalen und Nichtterminalen auf den rechten Seiten von Produktionen.
2. Aufbrechen langer Wörter auf den rechten Seiten von Produktionen.
3. Eliminieren von Regeln der Form $A \longrightarrow B$ (*Kettenregeln*).
4. Eliminieren von Regeln der Form $A \longrightarrow \varepsilon$ (ε-*Regeln*).

Am Ende werden wir die Chomsky-Normalform hergestellt haben, bei der alle Regeln die Form $A \longrightarrow BC$ und $A \longrightarrow a$ haben. Wie bei Typ-1-Grammatiken ist die Ausnahme $S \longrightarrow \varepsilon$ erlaubt, wenn S nicht auf der rechten Seite von Produktionen vorkommt. Wir messen die Größe einer Grammatik durch die Zahl der Nichtterminale und Terminale in allen Regeln.

Wir beginnen mit der Aufhebung der Mischung von Terminalen und Nichtterminalen auf den rechten Seiten von Produktionen.

Definition 9.9

Eine kontextfreie Grammatik (N, Σ, P, S) heißt *separiert*, wenn alle Ableitungsregeln die folgende Form haben:

- $A \longrightarrow \varepsilon$ für $A \in N$,
- $A \longrightarrow a$ für $A \in N$, $a \in \Sigma$,
- $A \longrightarrow N^*$ für $A \in N$.

Lemma 9.10

Jede kontextfreie Grammatik lässt sich umformen in eine äquivalente separierte kontextfreie Grammatik.

Beweis. Sei (N, Σ, P, S) eine kontextfreie Grammatik. Wir konstruieren eine äquivalente separierte Grammatik (N', Σ, P', S) wie folgt.

1. Führe für jedes $a \in \Sigma$ ein neues Nichtterminalsymbol X_a und die Produktion $X_a \longrightarrow a$ ein.
2. Ersetze in jeder Produktion $A \longrightarrow w$ mit $w \notin N^*$ jedes Terminalsymbol a durch X_a.

Dadurch wird offensichtlich die erzeugte Sprache nicht verändert. ❑

Die separierten Grammatik enthält $|\Sigma|$ neue Nichtterminale und $|\Sigma|$ neue Regeln $X_a \to a$, während jede Regel $A \to (N \cup \Sigma)^+$ durch eine Regel $A \to N^+$ ersetzt wurde. Die Größe der Grammatik verdreifacht sich also höchstens.

Wir fahren fort mit dem Aufbrechen langer Wörter.

Lemma 9.11

Jede separierte kontextfreie Grammatik lässt sich umformen in eine äquivalente separierte kontextfreie Grammatik, die nur Regeln der folgenden Form enthält:

- $A \longrightarrow \varepsilon$ *für* $A \in N$,
- $A \longrightarrow a$ *für* $A \in N,\ a \in \Sigma$,
- $A \longrightarrow B$ *für* $A, B \in N$,
- $A \longrightarrow BC$ *für* $A, B, C \in N$.

Beweis. Sei (N, Σ, P, S) eine separierte kontextfreie Grammatik. Also haben alle Ableitungsregeln die Form $A \longrightarrow \varepsilon$ für $A \in N$, $A \longrightarrow a$ für $A \in N$, $a \in \Sigma$ oder $A \longrightarrow N^*$ für $A \in N$. Nur Produktionen der letzten Form können unsere Anforderung verletzen.

Wir ersetzen alle Produktionen $A \longrightarrow B_1 \cdots B_n$ für $n > 2$ durch

$$A \longrightarrow B_1C_1,\ C_1 \longrightarrow B_2C_2,\ \ldots,\ C_{n-2} \longrightarrow B_{n-1}B_n$$

wobei die C_i jeweils neue Symbole sind.

Offensichtlich ist die resultierende Grammatik äquivalent und erfüllt unsere Anforderungen. ❑

In der obigen Konstruktion werden Regeln mit n Variablen durch $n-2$ Regeln mit je 3 Variablen ersetzt. Die Größe der Grammatik verdreifacht sich also höchstens.

Wir fahren fort mit dem Eliminieren von ε-Regeln.

Definition 9.12 (ε-freie kontextfreie Grammatik)
Eine kontextfreie Grammatik (N, Σ, P, S) heißt *ε-frei*, falls gilt:

1) $A \longrightarrow \varepsilon \notin P$ für $A \in N \setminus \{S\}$.
2) Falls $S \longrightarrow \varepsilon \in P$, so kommt S nicht auf der rechten Seite einer Regel vor.

Lemma 9.13
Jede kontextfreie Grammatik lässt sich umformen in eine äquivalente ε-freie kontextfreie Grammatik.

Beweis. Sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik.

1) Wir finden zunächst alle $A \in N$ mit $A \vdash_G^* \varepsilon$:

$$\begin{aligned} N_1 &:= \{A \in N \mid A \longrightarrow \varepsilon \in P\}, \\ N_{i+1} &:= N_i \cup \{A \in N \mid A \longrightarrow B_1 \cdots B_n \in P \text{ mit } B_1, \ldots, B_n \in N_i\}. \end{aligned}$$

Es gibt ein k mit $N_k = N_{k+1} = \bigcup_{i \geq 1} N_i$. Für dieses k gilt:

Behauptung: $N_k = \{A \ :\ A \vdash_G^* \varepsilon\}$, denn:

„$\subseteq$“: Zeige per Induktion über i: Wenn $A \in N_i$, dann $A \vdash_G^* \varepsilon$.

„$\supseteq$“: Zeige per Induktion über i: Wenn $A \vdash_G^i \varepsilon$, dann $A \in N_i$.

2) Nun eliminieren wir in G alle Regeln $A \longrightarrow \varepsilon$. Um dies auszugleichen, nehmen wir für alle Regeln

$$A \longrightarrow u_1B_1 \cdots u_nB_nu_{n+1} \text{ mit } B_1, \ldots, B_n \in N_k \text{ und } u_1, \ldots, u_{n+1} \in (\Sigma \cup N \setminus N_k)^*$$

die Regeln

$$A \longrightarrow u_1\beta_1u_2 \cdots u_n\beta_nu_{n+1}$$

hinzu für alle $\beta_1 \in \{B_1, \varepsilon\}, \ldots, \beta_n \in \{B_n, \varepsilon\}$ mit $u_1\beta_1u_2 \cdots u_n\beta_nu_{n+1} \neq \varepsilon$.

3) Falls $\varepsilon \notin L(G)$ (d. h. $S \not\vdash_G^* \varepsilon$, also $S \notin N_k$), so ist G' die gesuchte ε-freie Grammatik. Sonst erweitere G' um ein neues Startsymbol S_0 und die Produktionen $S_0 \longrightarrow S$ und $S_0 \longrightarrow \varepsilon$.

Es ist leicht zu sehen, dass G' äquivalent zu G ist und die gewünschten Eigenschaften besitzt. ❑

Beispiel 9.14

$$\begin{aligned} P \;=\; & \{S \longrightarrow aS,\ S \longrightarrow SS,\ S \longrightarrow bA,\\ & A \longrightarrow BB,\\ & B \longrightarrow CC,\ B \longrightarrow aAbC,\\ & C \longrightarrow \varepsilon\} \end{aligned}$$

$$N_0 \;=\; \{C\}, \quad N_1 \;= \{C, B\}, \quad N_2 \;= \{C, B, A\} \;=\; N_3$$

$$\begin{aligned} P' \;=\; & \{S \longrightarrow aS,\ S \longrightarrow SS,\ S \longrightarrow bA,\ S \longrightarrow b,\\ & A \longrightarrow BB,\ A \longrightarrow B,\\ & B \longrightarrow CC,\ B \longrightarrow C,\\ & B \longrightarrow aAbC,\ B \longrightarrow abC,\ B \longrightarrow aAb, B \longrightarrow ab\} \end{aligned}$$

Die Ableitung $S \vdash bA \vdash bBB \vdash bCCB \vdash bCCCC \vdash^* b$ kann nun in G' direkt durch $S \vdash b$ erreicht werden.

Im zweiten Schritt der obigen Konstruktion werden für eine Regel

$$A \longrightarrow u_1 B_1 \cdots u_n B_n u_{n+1} \text{ mit } B_1, \ldots, B_n \in N_k$$

möglicherweise 2^n neue Regeln eingefügt. Die Grammatik kann also exponentiell größer werden. Aus diesem Grund überführen wir jede kontextfreie Grammatik zunächst mit Lemma 9.11 in eine äquivalente Grammatik, in der keine langen Regeln vorkommen.

Korollar 9.15

$\mathcal{L}_2 \subseteq \mathcal{L}_1$.

Beweis. Offenbar ist jede ε-freie kontextfreie Grammatik eine Typ-1-Grammatik, da keine der verbleibenden Regeln verkürzend ist – mit Ausnahme von $S \longrightarrow \varepsilon$, wobei dann S aber wie auch bei Typ 1 gefordert auf keiner rechten Regelseite auftritt. ❑

Das folgende Lemma zeigt, dass wir auch auf Kettenregeln verzichten können.

Lemma 9.16

Jede kontextfreie Grammatik lässt sich umformen in eine äquivalente kontextfreie Grammatik, die keine Kettenregeln enthält.

Beweis. Sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik.

1) Bestimme die Relation $K := \{(A, B) \ : \ A \vdash_G^* B\}$:

$$\begin{aligned} K_0 &:= \{(A, A) \ : \ A \in N\}, \\ K_{i+1} &:= K_i \cup \{(A, B) \in N \times N \ : \ \exists (A, B') \in K_i : \ B' \longrightarrow B \in P\}. \end{aligned}$$

Es gibt ein k mit $K_k(A) = K_{k+1}(A)$. Für dieses k gilt

Behauptung: $K_k = K$, denn:

„$\subseteq$“: Zeige durch Induktion über i: Wenn $(A, B) \in K_i$, dann $A \vdash_G^* B$.

„$\supseteq$“: Zeige durch Induktion über i: Wenn $A \vdash_G^i B$, dann $(A, B) \in K_i$.

2) Entferne alle Kettenregeln. Um das auszugleichen, füge für jede verbleibende Regel $B \longrightarrow w$ und jedes $(A, B) \in K$ auch $A \longrightarrow w$ hinzu:

$$P' = \{A \longrightarrow w \ : \ (A, B) \in K, \ B \longrightarrow w \in P \ \text{ und } w \notin N\}.$$

Es ist leicht zu sehen, dass die erhaltene Grammatik äquivalent zu G ist und die gewünschte Form hat.

❑

Beispiel 9.17

Sei

$$P \ = \ \{S \longrightarrow A, \ A \longrightarrow B, \ B \longrightarrow aA, \ B \longrightarrow b\}.$$

Dann ist

$$\begin{aligned} K_0 &= \{(S, S), (A, A), (B, B)\}, \\ K_1 &= \{(S, S), (A, A), (B, B), (S, A), (A, B)\}, \\ K_2 &= \{(S, S), (A, A), (B, B), (S, A), (A, B), (S, B)\} \ = \ K_3 \ = \ K, \quad \text{also} \\ P' &= \{B \longrightarrow aA, \ A \longrightarrow aA, \ S \longrightarrow aA, \ B \longrightarrow b, \quad A \longrightarrow b, \quad S \longrightarrow b\}. \end{aligned}$$

Wir etablieren nun die Chomsky-Normalform.

Satz 9.18 (Chomsky-Normalform)

Jede kontextfreie Grammatik lässt sich umformen in eine äquivalente Grammatik in Chomsky-Normalform.

Beweis. Sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik.

1) Überführe G mittels Lemma 9.10 in eine äquivalente kontextfreie separierte Grammatik $G' = (N', \Sigma, P', S)$, d.h., alle Regeln von G' haben die Form
   - $A \longrightarrow \varepsilon$ für $A \in N'$,
   - $A \longrightarrow a$ für $A \in N'$, $a \in \Sigma$
   - $A \longrightarrow N^*$ für $A \in N'$.
2) Überführe G' mittels Lemma 9.11 in eine äquivalente Grammatik $G'' = (N'', \Sigma, P'', S)$, die nur Regeln der folgenden Form enthält:
   - $A \longrightarrow \varepsilon$ für $A \in N''$
   - $A \longrightarrow a$ für $A \in N'',\ a \in \Sigma$
   - $A \longrightarrow B$ für $A, B \in N''$
   - $A \longrightarrow BC$ für $A, B, C \in N''$
3) Überführe mittels Lemma 9.13 in eine äquivalente ε-freie Grammatik $G''' = (N''', \Sigma, P''', S''')$. Alle Regeln haben nun die Form
   - $A \longrightarrow a$ für $A \in N''',\ a \in \Sigma$
   - $A \longrightarrow B$ für $A, B \in N'''$
   - $A \longrightarrow BC$ für $A, B, C \in N'''$
   - Möglicherweise $S \longrightarrow \varepsilon$, und falls dies der Fall ist dann gibt es keine Regel $A \to BC$ mit $S \in \{B, C\}$
4) Eliminiere Kettenregeln aus G''' mittels Lemma 9.16. Die erhaltene äquivalente Grammatik $G^\star = (N^\star, \Sigma, P^\star, S^\star)$ enthält nur Regeln der Form
   - $A \longrightarrow a$ für $A \in N^\star,\ a \in \Sigma$
   - $A \longrightarrow BC$ für $A, B, C \in N^\star$
   - Möglicherweise $S \longrightarrow \varepsilon$, und falls dies der Fall ist dann gibt es keine Regel $A \to BC$ mit $S \in \{B, C\}$

Also ist $G^\star$ in Chomsky-Normalform. ❑

Beachte: Das Herstellen der Chomsky-Normalform wie im Beweis angegeben kann dazu führen, dass Nichtterminale überflüssig im Sinne von Definition 9.1 sind – dies verstößt nicht gegen die Chomsky-Normalform. Wir können die entstandene Grammatik nochmals reduzieren (siehe Abschnitt 9.1), ohne die Chomsky-Normalform zu verletzen. Es empfiehlt sich aber trotzdem, die Grammatik bereits *vor* dem Herstellen der Chomsky-Normalform zu reduzieren, da dies die anderen Schritte deutlich vereinfachen kann.

9.3. Das Wortproblem für kontextfreie Sprachen

Wir betrachten nun das *Wortproblem* für kontextfreie Sprachen. Im Gegensatz zu den regulären Sprachen fixieren wir dabei eine kontextfreie Sprache L, die durch eine Grammatik G gegeben ist, anstatt G als Eingabe zu betrachten. Die zu entscheidende Frage lautet dann: gegeben ein Wort $w \in \Sigma^*$, ist $w \in L$? Das Fixieren der Sprache ist für kontextfreie Sprachen in vielen Anwendungen durchaus sinnvoll: wenn wir z. B. einen Parser für eine Programmiersprache erstellen, so tun wir dies in der Regel für eine fixierte Sprache und betrachten nur das in der Programmiersprache formulierte Programm (= Wort) als Eingabe, nicht aber die Grammatik für die Programmiersprache selbst.

Wir nehmen an, dass die Grammatik in Chomsky-Normalform vorliegt. Wie wir gleich sehen werden, ist dies eine Voraussetzung, um den bekannten CYK-Algorithmus anwenden zu können. Zuvor soll aber kurz eine angenehme Eigenschaft der Chomsky-Normalform erwähnt werden, die auf sehr einfache Weise einen (wenngleich ineffizienten) Algorithmus für das Wortproblem liefert: wenn G eine Grammatik in Chomsky-Normalform ist, dann hat jede Ableitung eines Wortes $w \in L(G)$ *höchstens die Länge* $2|w| + 1$:

- Produktionen der Form $A \longrightarrow BC$ verlängern um 1, d. h. sie können maximal $(|w| - 1)$-mal angewendet werden.
- Produktionen der Form $A \longrightarrow a$ erzeugen genau ein Terminalsymbol von w, d. h. sie werden genau $|w|$-mal angewendet.

Die „$+1$“ in „$2|w| + 1$“ wird nur wegen des leeren Wortes benötigt, das Länge 0 hat, aber einen Ableitungsschritt benötigt. Im Gegensatz dazu können wir zu kontextfreien Grammatiken, die *nicht* in Chomsky-Normalform sind, *überhaupt keine* Schranke für die maximale Länge von Ableitungen angeben. Im Wesentlichen liegt dies daran, dass Kettenregeln und ε-Regeln gestattet sind.

Für Grammatiken in Chomsky-Normalform liefert die obige Beobachtung folgenden Algorithmus für das Wortproblem: gegeben ein Wort w kann man rekursiv *alle möglichen Ableitungen* der Länge $\leq 2|w| + 1$ durchprobieren, denn davon gibt es nur endlich viele. Wie schon erwähnt ist die Laufzeit dieses Verfahrens aber exponentiell. Einen besseren Ansatz liefert die folgende Überlegung:

Definition 9.19

Es sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik in Chomsky-Normalform und $w = a_1 \cdots a_n \in \Sigma^*$. Wir definieren:

- $w_{ij} := a_i \cdots a_j \quad$ (für $i \leq j$)
- $N_{ij} := \{A \in N \mid A \vdash_G^* w_{ij}\}$

Mit dieser Notation gilt nun:

1)	$S \in N_{1n}$	gdw.	$w \in L(G)$
2)	$A \in N_{ii}$	gdw.	$A \vdash_G^* a_i$
		gdw.	$A \longrightarrow a_i \in P$
3)	$A \in N_{ij}$ für $i < j$	gdw.	$A \vdash_G^* a_i \cdots a_j$
		gdw.	$\exists A \longrightarrow BC \in P$ und ein k mit $i \leq k < j$ mit $B \vdash_G^* a_i \cdots a_k$ und $C \vdash_G^* a_{k+1} \cdots a_j$
		gdw.	$\exists A \longrightarrow BC \in P$ und k mit $i \leq k < j$ mit $B \in N_{ik}$ und $C \in N_{(k+1)j}$

Diese Überlegungen liefern einen dynamischen Algorithmus zur Berechnung von N_{1n}: siehe Algorithmus 2. Die generelle Idee dabei ist, das eigentliche Problem in Teilprobleme zu zerlegen und diese dann beginnend mit den einfachsten und fortschreitend zu immer komplexeren Teilproblemen zu lösen. Im vorliegenden Fall sind die Teilprobleme das Berechnen der N_{ij} mit $i \leq j$. Die „einfachsten“ Teilprobleme sind dann diejenigen mit $i = j$, und die Teilprobleme werden immer schwieriger, je größer die Teilwortlänge $j - i$ wird.

Algorithmus 2 : CYK-Algorithmus von Cocke, Younger, Kasami

```
for i := 1 to n do
    N_ii := {A : A ⟶ a_i ∈ P}

for ℓ := 1 to n − 1 do   // wachsende Teilwortlänge ℓ = j − i
    for i := 1 to n − ℓ do   // Startposition i von Teilwort
        j := i + ℓ   // Endposition j von Teilwort
        N_ij := ∅
        for k := i to j − 1 do   // Mögliche Trennpositionen k
            N_ij := N_ij ∪ {A | ∃A ⟶ BC ∈ P mit B ∈ N_ik und C ∈ N_(k+1)j}
```

Beachte: In der innersten Schleife sind N_{ik} und $N_{(k+1)j}$ bereits berechnet, da die Teilwortlängen $k - i$ und $j - k + 1$ kleiner als das aktuelle ℓ.

Satz 9.20

Für jede Grammatik G in Chomsky-Normalform entscheidet der CYK-Algorithmus die Frage „gegeben w, ist $w \in L(G)$?“ in Zeit $\mathcal{O}(|w|^3)$.

Beweis. Die erste for-Schleife macht n Schritte; der Rest des Algorithmus besteht aus drei geschachtelte Schleifen, die jeweils $\leq |w| = n$ Schritte machen; also werden in der innersten Schleife $\leq |w|^3$ Schritte gemacht.

Es ist zu beachten, dass die Suche nach den Produktionen $A \longrightarrow a_i$ $A \longrightarrow BC$ und im Inneren der Schleifen auch nur konstante Zeit benötigt, denn wir haben die Größe von G hier als konstant angenommen (fest vorgegebenes G). ❑

Beispiel 9.21

Betrachte die Grammatik $G = (N, \Sigma, P, S)$ mit

$$\begin{aligned} P = \quad & \{S \longrightarrow SA,\ S \longrightarrow a, \\ & A \longrightarrow BS, \\ & B \longrightarrow BB,\ B \longrightarrow BS,\ B \longrightarrow b,\ B \longrightarrow c\} \end{aligned}$$

und $w = abacba$.

In der folgenden graphischen Darstellung werden die Zeilen der Tabelle von unten nach oben gefüllt.

			$N_{1,6} = \{S\}$			
			$N_{1,5} = \emptyset$	$N_{2,6}=\{A,B\}$		
		$N_{1,4} = \emptyset$	$N_{2,5} = \{B\}$	$N_{3,6} = \{S\}$		
	$N_{1,3} = \{S\}$	$N_{2,4} = \{B\}$	$N_{3,5} = \emptyset$	$N_{4,6}=\{A,B\}$		
$N_{1,2} = \emptyset$	$N_{2,3}=\{A,B\}$	$N_{3,4} = \emptyset$	$N_{4,5} = \{B\}$	$N_{2,3}=\{A,B\}$		
$N_{1,1} = \{S\}$	$N_{2,2} = \{B\}$	$N_{3,3} = \{S\}$	$N_{4,4} = \{B\}$	$N_{5,5} = \{B\}$	$N_{6,6} = \{S\}$	
$w =$	a	b	a	c	b	a

Eine andere Darstellung in einer Tabelle ist wie folgt.

i \ j	1	2	3	4	5	6
1	S	$\emptyset$	S	$\emptyset$	$\emptyset$	S
2		B	A, B	B	B	A, B
3			S	$\emptyset$	$\emptyset$	S
4				B	B	A, B
5					B	A, B
6						S
$w =$	a	b	a	c	b	a

In jedem Fall gilt $S \in N_{1,6} = \{S\}$, also gilt $w \in L(G)$.

Wir werden in der VL *Theoretische Informatik 2* beweisen, dass das Äquivalenzproblem für kontextfreie Sprachen unentscheidbar ist. Es gibt also keinen Algorithmus, der für zwei gegebene kontextfreie Grammatiken G_1 und G_2 entscheidet, ob $L(G_1) = L(G_2)$.

9.4. Die Greibach-Normalform

Eine weitere interessante Normalform für kontextfreie Grammatiken ist die *Greibach-Normalform*, bei der es für jedes Wort in der Sprache eine Ableitung gibt, die die Terminale *streng von links nach rechts* erzeugt, und zwar *genau ein* Nichtterminal pro Ableitungsschritt. Wir geben den folgenden Satz ohne Beweis an.

Satz 9.22 (Greibach-Normalform)
Jede kontextfreie Grammatik lässt sich umformen in eine äquivalente Grammatik, die nur Regeln der Form

- $A \longrightarrow aw \quad (A \in N, a \in \Sigma, w \in N^*)$,
- *und möglicherweise* $S \longrightarrow \varepsilon$, *wobei* S *nicht rechts vorkommt*

enthält. Eine derartige Grammatik heißt Grammatik in Greibach-Normalform.

10. Abschlusseigenschaften und Pumping-Lemma

Die kontextfreien Sprachen verhalten sich bezüglich der Abschlusseigenschaften nicht ganz so gut wie die regulären Sprachen. Wir beginnen mit positiven Resultaten.

Satz 10.1

Die Klasse $\mathcal{L}_2$ der kontextfreien Sprachen ist unter Vereinigung, Konkatenation und Kleene-Stern abgeschlossen.

Beweis. Es seien $G_i = (N_i, \Sigma, P_i, S_i), i = 1, 2$ kontextfreie Grammatiken. Wir nehmen o. B. d. A. an, dass $N_1 \cap N_2 = \emptyset$.

1) $G := (N_1 \cup N_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \longrightarrow S_1, S \longrightarrow S_2\}, S)$ mit $S \notin N_1 \cup N_2$ ist eine kontextfreie Grammatik mit $L(G) = L(G_1) \cup L(G_2)$.
2) $G' := (N_1 \cup N_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \longrightarrow S_1S_2\}, S)$ mit $S \notin N_1 \cup N_2$ ist eine kontextfreie Grammatik mit $L(G') = L(G_1) \cdot L(G_2)$.
3) $G'' := (N_1 \cup \{S\}, \Sigma, P_1 \cup \{S \longrightarrow \varepsilon, S \longrightarrow SS_1\}, S)$ mit $S \notin N_1$ ist eine kontextfreie Grammatik für $L(G_1)^*$.

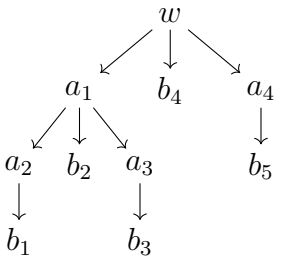
❑

Wir werden zeigen, dass Abschluss unter *Schnitt* und *Komplement nicht* gilt. Dazu benötigen wir zunächst eine geeignete Methode, von einer Sprache nachzuweisen, dass sie *nicht* kontextfrei ist. Dies gelingt wieder mit Hilfe eines Pumping-Lemmas. In dessen Beweis stellen wir Ableitungen als Bäume dar, so genannte *Ableitungsbäume*.

Definition 10.2

Ein *gewurzelter Baum* ist ein gerichteter kreisfreier Graph, der einen Knoten w besitzt, von dem jeder andere Knoten erreichbar ist. Der Knoten w wird *Wurzel* genannt. Er hat Eingangsgrad 0 und ist der einzige Knoten mit dieser Eigenschaft. Alle Knoten mit Ausgangsgrad 0 heißen *Blätter*. Alle Knoten mit positivem Ausgangsgrad heißen *innere Knoten*.

Im folgenden Beispiel ist die Wurzel eines gewurzelten Baumes mit w bezeichnet, die inneren Knoten sind mit a_i benannt und die Blätter sind mit b_i benannt.



Definition 10.3

Sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik. Ein *partieller Ableitungsbaum* ist ein gewurzelter Baum, dessen Knoten mit Symbolen aus $N \cup \Sigma$ beschriftet sind, so dass

- jeder innere Knoten mit einem Symbol aus N beschriftet ist,
- jedes Blatt mit einem Symbol aus $N \cup \Sigma$ beschriftet ist und
- wenn ein innerer Knoten v mit $A \in N$ beschriftet ist, so sind die Nachfolger von v (von rechts nach links[1]) mit $w_1, \ldots, w_n$ beschriftet für eine Regel $A \to w_1 \cdots w_n \in P$.

Ein *(vollständiger) Ableitungsbaum* ist ein partieller Ableitungsbaum, so dass

- die Wurzel mit S beschriftet ist,
- jedes Blatt mit einem Symbol aus Σ beschriftet ist.

Ein partieller Ableitungsbaum, dessen Wurzel mit A beschriftet ist und dessen Blätter (von links nach rechts) mit $w_1, \ldots, w_n \in \Sigma \cup N$ beschriftet sind, beschreibt eine Ableitung $A \vdash_G^* \alpha_1 \cdots \alpha_n$.

Beispiel 10.4

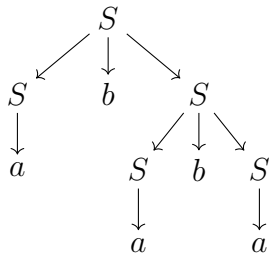
$P = \{S \longrightarrow SbS, S \longrightarrow a\}$

Drei *Ableitungen* des Wortes *ababa*:

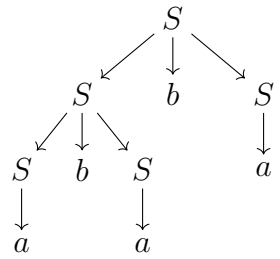
1) $S \vdash SbS \vdash abS \vdash abSbS \vdash ababS \vdash ababa$
2) $S \vdash SbS \vdash abS \vdash abSbS \vdash abSba \vdash ababa$
3) $S \vdash SbS \vdash Sba \vdash SbSba \vdash Sbaba \vdash ababa$

Die zugehörigen *Ableitungsbäume*:

für 1) und 2):



für 3):



Ein Ableitungsbaum kann also für mehr als eine Ableitung stehen, und dasselbe Wort kann verschiedene Ableitungsbäume haben.

[1]Wir setzen hier eine Ordnung der Nachfolger eines inneren Knotens voraus, die es erlaubt, von *links* und *rechts* zu sprechen

Wir können nun das Pumping-Lemma für kontextfreie Sprachen beweisen.

Lemma 10.5 (Pumping-Lemma für kontextfreie Sprachen)
Für jede kontextfreie Sprache L gibt es ein $n_0 \geq 1$, so dass gilt:
für jedes $z \in L$ mit $|z| \geq n_0$ existiert eine Zerlegung $z = uvwxy$ mit:

- $vx \neq \varepsilon$ *und* $|vwx| \leq n_0$ *und*
- $uv^iwx^iy \in L$ *für alle* $i \geq 0$.

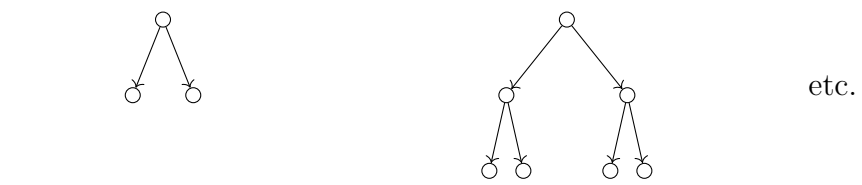
Beweis. Sei G eine kontextfreie Grammatik mit $L(G) = L$. O. B. d. A. können wir annehmen, dass $\varepsilon \notin L$ (da $n_0 \geq 1$ muss das leere Wort nicht zerlegt werden). Nach Satz 9.18 können wir annehmen, dass G in Chomsky Normalform vorliegt. Also gibt es keine Kettenregeln und da nach Annahme $\varepsilon \notin L$, gibt es keine Regel $A \longrightarrow \varepsilon$.

Sei

- m die Anzahl der Nichtterminale in G und
- $n_0 = 2^{m+1}$.

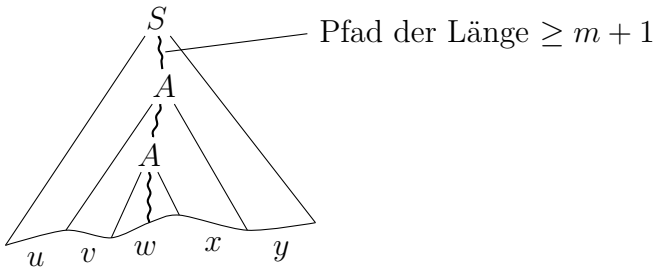
Wir verfahren nun wie folgt.

1) Ein Baum der Tiefe $\leq t$ und der Verzweigungsgrad ≤ 2 hat maximal 2^t viele Blätter:

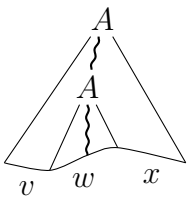


   Jeder Ableitungsbaum für z mit $|z| \geq n_0$ hat $|z| \geq n_0 \geq 2^{m+1}$ Blätter; also gibt es einen Pfad (Weg von Wurzel zu Blatt) der Länge $\geq m + 1$ (Anzahl der Kanten).

2) Auf diesem Pfad kommen $> m + 1$ Symbole vor, davon $> m$ Nichtterminale. Da es nur m verschiedene Nichtterminale gibt, kommt ein Nichtterminal A zweimal vor. Dies führt zu folgender Wahl von u, v, w, x, y:



Wir wählen hier o. B. d. A. die erste Wiederholung eines Nichtterminals A von den Blättern aus gesehen; mit den Argumenten in 1) hat dann der Teilbaum



die Tiefe $\leq m+1$, was $|vwx| \leq 2^{m+1} = n_0$ zeigt.

3) Es gilt: $S \vdash_G^* uAy, \quad A \vdash_G^* vAx, \quad A \vdash_G^* w$, woraus folgt:

$$S \ \vdash_G^* \ uAy \ \vdash_G^* \ uv^iAx^iy \ \vdash_G^* \ uv^iwx^iy.$$

4) Außerdem gilt $vx \neq \varepsilon$: Da G ε-frei ist, wäre sonst $A \vdash_G^+ vAx$ nur bei Anwesenheit von Regeln der Form $A \longrightarrow B$ möglich. ❑

Wir verwenden nun das Pumping-Lemma, um beispielhaft von einer Sprache nachzuweisen, dass sie nicht kontextfrei ist.

Lemma 10.6
$L = \{a^nb^nc^n \mid n \geq 1\} \notin \mathcal{L}_2$.

Beweis. Angenommen $L \in \mathcal{L}_2$. Es sei $n_0 \geq 1$ die zugehörige Zahl aus Lemma 10.5. Wir betrachten $z = a^{n_0}b^{n_0}c^{n_0} \in L$. Mit Lemma 10.5 gibt es eine Zerlegung

$$z = uvwxy, \ vx \neq \varepsilon \text{ und } uv^iwx^iy \in L \text{ für alle } i \geq 0.$$

1. Fall: v enthält verschiedene Symbole. Dann gilt

$$uv^2wx^2y \notin a^*b^*c^* \supseteq L.$$

2. Fall: x enthält verschiedene Symbole. Dies führt zu entsprechendem Widerspruch.

3. Fall: v enthält lauter gleiche Symbole und x enthält lauter gleiche Symbole. Dann gibt es ein Symbol aus $\{a, b, c\}$, das in xv nicht vorkommt. Daher kommt dieses in $uv^0wx^0y = uwy$ weiterhin n_0-mal vor. Aber es gilt $|uwy| < 3n_0$, was $uwy \notin L$ zeigt. ❑

Dieses Beispiel zeigt auch, dass die kontextfreien Sprachen eine *echte* Teilmenge der Typ-1-Sprachen sind.

Satz 10.7
$\mathcal{L}_2 \subsetneq \mathcal{L}_1$.

Beweis. Wir haben bereits gezeigt, dass $\mathcal{L}_2 \subseteq \mathcal{L}_1$ gilt (Korollar 9.15). Es bleibt zu zeigen, dass die Inklusion echt ist. Dafür betrachten wir die Sprache $L = \{a^n b^n c^n \mid n \geq 1\}$. Nach Lemma 10.6 ist $L \notin \mathcal{L}_2$. Nach Beispiel 7.3 gilt aber $L \in \mathcal{L}_1$. ❏

Außerdem können wir nun zeigen, dass die kontextfreien Sprachen unter zwei wichtigen Operationen nicht abgeschlossen sind.

Korollar 10.8

Die Klasse $\mathcal{L}_2$ der kontextfreien Sprachen ist nicht *unter Schnitt und Komplement abgeschlossen.*

Beweis.

1) Die Sprachen $\{a^n b^n c^m \mid n \geq 1, m \geq 1\}$ und $\{a^m b^n c^n \mid n \geq 1, m \geq 1\}$ sind in $\mathcal{L}_2$:
   - $\{a^n b^n c^m \mid n \geq 1, m \geq 1\} = \underbrace{\underbrace{\{a^n b^n \mid n \geq 1\}}_{\in \mathcal{L}_2} \cdot \underbrace{\{c^m \mid m \geq 1\}}_{= c^+ \in \mathcal{L}_3 \subseteq \mathcal{L}_2}}_{\in \mathcal{L}_2 \text{ (Konkatenation)}}$
   - $\{a^m b^n c^n \mid n \geq 1, m \geq 1\}$ — analog
2) $\{a^n b^n c^n \mid n \geq 1\} = \{a^n b^n c^m \mid n, m \geq 1\} \cap \{a^m b^n c^n \mid n, m \geq 1\}$.
   Wäre $\mathcal{L}_2$ unter $\cap$ abgeschlossen, so würde $\{a^n b^n c^n \mid n \geq 1\} \in \mathcal{L}_2$ folgen.
   Dies ist ein Widerspruch zu Teil 1) des Beweises von Satz 10.7.
3) $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.
   Wäre $\mathcal{L}_2$ unter Komplement abgeschlossen, so auch unter $\cap$, da $\mathcal{L}_2$ unter $\cup$ abgeschlossen ist. Dies ist ein Widerspruch zu 2). ❏

Beachte:

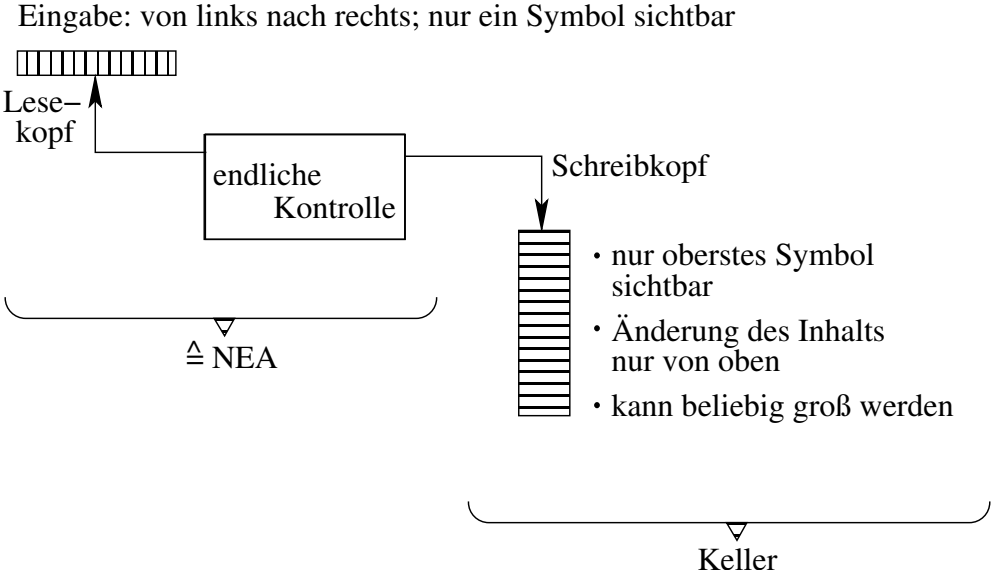
Wir können daher das Äquivalenzproblem für kontextfreie Sprachen nicht einfach auf das Leerheitsproblem reduzieren (wir brauchen dazu sowohl Schnitt als auch Komplement). Wie bereits erwähnt werden wir später sogar sehen, dass das Äquivalenzproblem für kontextfreie Sprachen *unentscheidbar* ist.

11. Kellerautomaten

Bisher haben wir kontextfreie Sprachen nur mit Hilfe von Grammatiken charakterisiert. Wir haben gesehen, dass endliche Automaten *nicht* in der Lage sind, alle kontextfreien Sprachen zu akzeptieren.

Um die *Beschreibung von kontextfreien Sprachen* mit Hilfe von endlichen Automaten zu ermöglichen, müssen wir diese um eine unbeschränkte *Speicherkomponente*, einen so genannten *Keller (englisch: Stack)* erweitern. Dieser Keller speichert zwar zu jedem Zeitpunkt nur endlich viel Information, kann aber unbeschränkt wachsen.

Die folgende Abbildung zeigt eine schematische Darstellung eines *Kellerautomaten*:



Diese Idee wird in folgender Weise formalisiert.

Definition 11.1 (Kellerautomat)

Ein *Kellerautomat (pushdown automaton, kurz PDA)* hat die Form $\mathcal{A} = (Q, \Sigma, \Gamma, q_s, Z_s, \Delta, F)$, wobei

- Q eine endliche Menge von *Zuständen* ist,
- Σ das *Eingabealphabet* ist,
- Γ das *Kelleralphabet* ist,
- $q_s \in Q$ der *Anfangszustand* ist,
- $Z_s \in \Gamma$ das *Kellerstartsymbol* ist,
- $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times \Gamma^* \times Q$ eine endliche *Übergangsrelation* ist und
- $F \subseteq Q$ eine Menge von *akzeptierenden Zuständen* ist.

Anschaulich bedeutet die Übergangsrelation:

(q, a, Z, γ, q')**:** Im Zustand q mit aktuellem Eingabesymbol a und oberstem Kellersymbol Z darf der Automat Z auf dem Keller durch γ ersetzen, in den Zustand q' wechseln und zum nächsten Eingabesymbol übergehen.

$(q, \varepsilon, Z, \gamma, q')$**:** wie oben, nur dass das aktuelle Eingabesymbol *nicht relevant* ist und wir nicht zum nächsten Eingabesymbol übergehen (der Lesekopf ändert seine Position nicht).

Übergänge der zweiten Form können nicht so einfach eliminiert werden, wie wir das für ε-Übergänge von ε-NEAs kennen, denn durch ε-Übergänge kann ein PDA beliebig viele Kellersymbole löschen oder hinzufügen, ohne ein weiteres Symbol im Eingabewort zu lesen.[2]

Um die Sprache zu definieren, die von einem Kellerautomaten akzeptiert wird, benötigen wir den Begriff der *Konfiguration*, die den aktuellen Stand einer Kellerautomatenberechnung beschreibt. Eine Konfiguration ist bestimmt durch

- den *noch zu lesenden Rest* $w \in \Sigma^*$ der Eingabe (Lesekopf steht auf dem ersten Symbol von w),
- den *Zustand* $q \in Q$ und
- den *Kellerinhalt* $\gamma \in \Gamma^*$ (Schreibkopf steht auf dem ersten Symbol von γ).

Definition 11.2

Sei $\mathcal{A} = (Q, \Sigma, \Gamma, q_s, Z_s, \Delta, F)$ ein Kellerautomat. Eine *Konfiguration* von $\mathcal{A}$ hat die Form

$$\mathcal{K} = (q, w, \gamma) \in Q \times \Sigma^* \times \Gamma^*.$$

Die Übergangsrelation ermöglicht die folgenden *Konfigurationsübergänge*:

- $(q, aw, Z\gamma) \vdash_{\mathcal{A}} (q', w, \beta\gamma)$ falls $(q, a, Z, \beta, q') \in \Delta$,
- $(q, w, Z\gamma) \vdash_{\mathcal{A}} (q', w, \beta\gamma)$ falls $(q, \varepsilon, Z, \beta, q') \in \Delta$,
- $\mathcal{K} \vdash_{\mathcal{A}}^* \mathcal{K}'$ gdw. Konfigurationen $\mathcal{K}_0, \ldots, \mathcal{K}_n$ existieren mit

$$\mathcal{K}_0 = \mathcal{K},\ \mathcal{K}_n = \mathcal{K}' \text{ und } \mathcal{K}_i \vdash_{\mathcal{A}} \mathcal{K}_{i+1} \text{ für } 0 \leq i < n.$$

Der Automat $\mathcal{A}$ *akzeptiert* das Wort $w \in \Sigma^*$ gdw.

$$(q_s, w, Z_s) \vdash_{\mathcal{A}}^* (q, \varepsilon, \gamma) \quad \text{für ein } q \in F \text{ und } \gamma \in \Gamma^*.$$

[2]Wir können jedoch die Greibach-Normalform für kontextfreie Grammatiken (Satz 9.22), sowie die im Folgenden gezeigte Äquivalenz von PDAs und kontextfreien Grammatiken (Satz 11.9) benutzen um zu zeigen, dass es zu jedem PDA einen äquivalenten PDA ohne ε-Übergänge gibt. Dies gilt wiederum nicht für die deterministische Variante (Def. 11.12): dPDAs ohne ε-Übergänge sind echt schwächer als allgemeine PDAs.

Also akzeptiert ein Kellerautomat ein Wort w, wenn er w komplett gelesen hat und sich danach in einem akzeptierenden Zustand befindet. Auf dem Keller dürfen sich noch beliebige Symbole befinden. Die von $\mathcal{A}$ *akzeptierte Sprache* ist

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

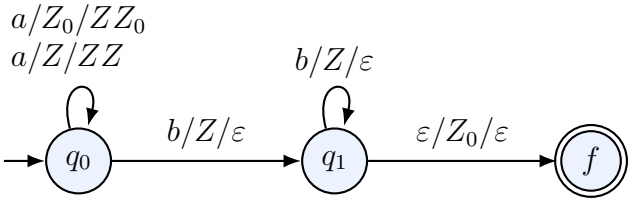
Im Folgenden zwei einfache Beispiele für Kellerautomaten.

Beispiel 11.3

Ein PDA für $\{a^n b^n \mid n \geq 1\}$.

- $Q = \{q_0, q_1, f\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{Z, Z_0\}$
- $q_s = q_0$
- $Z_s = Z_0$
- $\Delta = \{(q_0, a, Z_0, ZZ_0, q_0)$, (lese erstes a, lege Z auf den Keller)
  (q_0, a, Z, ZZ, q_0), (lese weitere a's, lege Z auf den Keller)
  $(q_0, b, Z, \varepsilon, q_1)$, (erstes b, entnimm Z)
  $(q_1, b, Z, \varepsilon, q_1)$, (weitere b's, entnimm Z)
  $(q_1, \varepsilon, Z_0, \varepsilon, f)\}$ (lösche das Kellerstartsymbol)
- $F = \{f\}$

Wir stellen einen Kellerautomaten graphisch in der folgenden Weise dar, wobei die Kantenbeschriftung $a/Z/\gamma$ bedeutet, dass der Automat bei Z als oberstem Kellersymbol das Eingabesymbol a lesen und Z durch γ ersetzen kann.



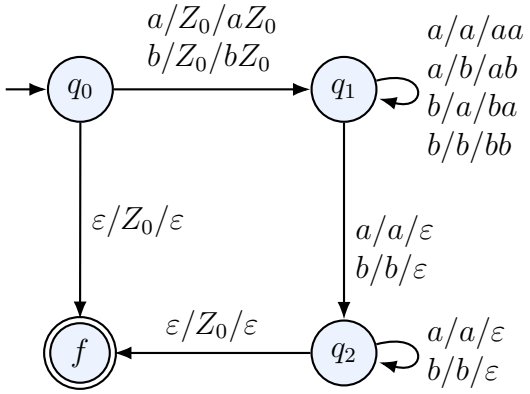
Betrachten wir beispielhaft einige Konfigurationsübergänge:

1) $(q_0, aabb, Z_0) \vdash_{\mathcal{A}} (q_0, abb, ZZ_0) \vdash_{\mathcal{A}} (q_0, bb, ZZZ_0) \vdash_{\mathcal{A}} (q_1, b, ZZ_0) \vdash_{\mathcal{A}} (q_1, \varepsilon, Z_0) \vdash_{\mathcal{A}} (f, \varepsilon, \varepsilon)$
   – akzeptiert
2) $(q_0, aab, Z_0) \vdash^*_{\mathcal{A}} (q_0, b, ZZZ_0) \vdash_{\mathcal{A}} (q_1, \varepsilon, ZZ_0)$
   – kein Übergang mehr möglich, nicht akzeptiert
3) $(q_0, abb, Z_0) \vdash_{\mathcal{A}} (q_0, bb, ZZ_0) \vdash_{\mathcal{A}} (q_1, b, Z_0) \vdash_{\mathcal{A}} (f, b, \varepsilon)$
   – nicht akzeptiert (weil das Wort nicht zu Ende gelesen wurde – „b“ in (f, b, ε))

Beispiel 11.4

Ein PDA für $L = \{ww^R \ : \ w \in \{a,b\}^*\}$ (wobei für $w = a_1 \cdots a_n$ gilt $w^R = a_n \cdots a_1$).

- $Q = \{q_0, q_1, q_2, f\}$
- $\Gamma = \{a, b, Z_0\}$
- $\Sigma = \{a, b\}$
- $F = \{f\}$
- Δ wird graphisch dargestellt:



In q_1 wird die erste Worthälfte im Keller gespeichert. Der nichtdeterministische Übergang von q_1 nach q_2 „rät“ die Wortmitte. In q_2 wird die zweite Hälfte des Wortes gelesen und mit dem Keller verglichen.

Die in den Definitionen 11.1 und 11.2 eingeführte Version von Kellerautomaten akzeptiert wie auch ein NEA *per akzeptierendem Zustand*. Wir können stattdessen auch Akzeptanz *per leerem Keller* definieren.

Definition 11.5

Ein *PDA mit Akzeptanz per leerem Keller* ist ein Tupel

$$\mathcal{A} = (Q, \Sigma, \Gamma, q_s, Z_s, \Delta),$$

wobei alle Komponenten wie in Definition 11.1 sind. Ein solcher PDA *akzeptiert* ein Eingabewort $w \in \Sigma^*$ gdw. $(q_s, w, Z_0) \vdash_{\mathcal{A}}^* (q, \varepsilon, \varepsilon)$ für ein $q \in Q$.

Dabei bedeutet die Konfiguration $(q, \varepsilon, \varepsilon)$ also, dass sich $\mathcal{A}$ in einem beliebigen Zustand befinden kann, das Eingabewort zu Ende gelesen haben muss und den Keller vollständig geleert haben muss (einschließlich des Kellerstartsymbols Z_s).

Es ist nicht schwer zu zeigen, dass PDAs mit Akzeptanz per leerem Keller und solche mit akzeptierenden Zuständen dieselbe Sprachklasse definieren. Der Beweis wird als Übung gelassen.

Satz 11.6

Jede Sprache, die von einem PDA (mit akzeptierenden Zuständen) akzeptiert wird, wird auch von einem PDA mit Akzeptanz per leerem Keller akzeptiert und umgekehrt.

Wir werden nun zeigen, dass die kontextfreien Sprachen genau die Sprachen sind, die von Kellerautomaten akzeptiert werden. Dazu führen wir zunächst den Begriff der Linksableitung ein.

Definition 11.7

Sei G eine kontextfreie Grammatik. Eine Ableitung $S = w_0 \vdash_G w_1 \vdash_G w_2 \vdash_G \cdots \vdash_G w_n$ heißt *Linksableitung*, wenn sich w_{i+1} aus w_i durch Anwendung einer Regel auf das am weitesten links stehende Nichtterminal in w_i ergibt, für alle $i < n$.

Das folgende Lemma zeigt, dass sich die von einer kontextfreien Grammatik erzeugte Sprache nicht ändert, wenn wir statt beliebigen Ableitungen nur noch Linksableitungen zulassen.

Lemma 11.8

Für jede kontextfreie Grammatik G gilt:

$$L(G) = \{w \in \Sigma^* \ : \ w \text{ kann in } G \text{ mit Linksableitung erzeugt werden}\}.$$

Zum Beweis von Lemma 11.8 genügt es zu zeigen, dass jedes $w \in L(G)$ mittels einer Linksableitung erzeugt werden kann. Wir argumentieren nur informell: wenn $w \in L(G)$, dann gibt es eine Ableitung von w in G. Diese kann als Ableitungsbaum dargestellt werden. Aus dem Ableitungsbaum lässt sich nun wiederum eine Linksableitung von w ablesen (zum Beispiel durch Traversieren des Baumes in Pre-Order).

Satz 11.9

Für jede formale Sprache L sind äquivalent:

1) *L wird von einer kontextfreien Grammatik erzeugt.*
2) *L wird von einem PDA akzeptiert.*

Beweis.

„1 $\Rightarrow$ 2“.

Es sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik. Wegen Satz 11.6 genügt es, einen PDA $\mathcal{A}$ *mit Akzeptanz per leerem Keller* zu konstruieren, so dass $L(\mathcal{A}) = L(G)$. Die Idee ist, dass $\mathcal{A}$ die Linksableitungen von G auf dem Keller simulieren soll.

Dazu definieren wir $\mathcal{A} = (Q, \Sigma, \Gamma, q_s, Z_s, \Delta)$ wobei $Q = \{q\}$, $\Gamma = \Sigma \cup N$, $q_s = q$, $Z_s = S$ und Δ aus folgenden Übergängen besteht:

- Übergänge zum Anwenden von Produktionen auf das oberste Kellersymbol:

  für jede Regel $A \to \gamma$ den Übergang $(q, \varepsilon, A, \gamma, q)$
- Übergänge zum Entfernen bereits erzeugter Terminalsymbole von der Kellerspitze, wenn sie in der Eingabe vorhanden sind:

  für jedes Terminalsymbol $a \in \Sigma$ den Übergang $(q, a, a, \varepsilon, q)$.

Beispiel:

$P = \{S \longrightarrow \varepsilon,\ S \longrightarrow aSa,\ S \longrightarrow bSb\}$ liefert die Übergänge:

$$\begin{array}{llllll} (q, & \varepsilon, & S, & \varepsilon, & q), & (S \longrightarrow \varepsilon) \\ (q, & \varepsilon, & S, & aSa, & q), & (S \longrightarrow aSa) \\ (q, & \varepsilon, & S, & bSb, & q), & (S \longrightarrow bSb) \\ (q, & a, & a, & \varepsilon, & q), & (a \text{ entfernen}) \\ (q, & b, & b, & \varepsilon, & q) & (b \text{ entfernen}) \end{array}$$

Die Ableitung $S \vdash_G aSa \vdash_G abSba \vdash_G abba$ entspricht der *Konfigurationsfolge*

$$\begin{array}{llll} (q, abba, S) \vdash_{\mathcal{A}} (q, abba, aSa) & \vdash_{\mathcal{A}} (q, bba, Sa) & \vdash_{\mathcal{A}} (q, bba, bSba) \vdash_{\mathcal{A}} \\ (q, ba, Sba) \vdash_{\mathcal{A}} (q, ba, ba) & \vdash_{\mathcal{A}} (q, a, a) & \vdash_{\mathcal{A}} (q, \varepsilon, \varepsilon) \end{array}$$

Zu zeigen:

Für alle $w \in \Sigma^*$ gilt: $S \vdash_G^* w$ mittels Linksableitung gdw. $(q, w, S) \vdash_{\mathcal{A}}^* (q, \varepsilon, \varepsilon)$.

Beweis der Behauptung.

„⇒“: Sei

$$S = w_0 \vdash_G w_1 \vdash_G \cdots \vdash_G w_n = w$$

eine Linksableitung. Für alle $i \leq n$ sei $w_i = u_i \alpha_i$ so dass u_i nur aus Terminalsymbolen besteht und α_0 mit Nichtterminal beginnt oder leer ist. Jedes u_i ist Präfix von w. Sei v_i das zugehörige Suffix, also $w = u_i v_i$.

Wir zeigen, dass

$$(q, w, S) = (q, v_0, \alpha_0) \vdash_{\mathcal{A}}^* (q, v_1, \alpha_1) \vdash_{\mathcal{A}}^* \cdots \vdash_{\mathcal{A}}^* (q, v_n, \alpha_n) = (q, \varepsilon, \varepsilon).$$

Sei $i < n$ und $A \to \gamma$ die Produktion für $w_i \vdash_G w_{i+1}$. Also beginnt α_i mit A. Dann ergibt sich $(q, v_i, \alpha_i) \vdash_{\mathcal{A}}^* (q, v_{i+1}, \alpha_{i+1})$ durch folgende Übergänge:

- zunächst verwende $(q, \varepsilon, A, \gamma, q)$;
- verwende dann Übergänge $(q, a, a, \varepsilon, q)$ solange das oberste Stacksymbol ein Terminalsymbol a ist.

„⇐“: Gelte umgekehrt

$$(q, w, S) = (q, v_0, \alpha_0) \vdash_{\mathcal{A}}^* (q, v_1, \alpha_1) \vdash_{\mathcal{A}}^* \cdots \vdash_{\mathcal{A}}^* (q, v_n, \alpha_n) = (q, \varepsilon, \varepsilon).$$

Wir können o. B. d. A. annehmen, dass

- jede der Teilberechnungen $(q, v_i, \alpha_i) \vdash_{\mathcal{A}}^* (q, v_{i+1}, \alpha_{i+1})$ genau einen Übergang der Form $(q, \varepsilon, A, \gamma, q)$ und beliebig viele der Form $(q, a, a, \varepsilon, q)$ verwendet und
- das erste Symbol von α_i ein Nichtterminal ist.

Jedes v_i ist Suffix von w. Sei u_i das zugehörige Präfix, also $w = u_i v_i$.

Wir zeigen, dass

$$S = u_0\alpha_0 \vdash_G u_1\alpha_1 \vdash_G \cdots \vdash_G u_n\alpha_n.$$

Sei $i < n$ und $(q, \varepsilon, A, \gamma, q)$ der erste Übergang in $(q, v_i, \alpha_i) \vdash_G^* (q, v_{i+1}, \alpha_{i+1})$ gefolgt von $(q, a_1, a_1, \varepsilon, q), \ldots, (q, a_m, a_m, \varepsilon, q)$. Dann hat α_i die Form $A\alpha_i'$ und γ die Form $a_1 \cdots a_m B\gamma'$ mit $B \in N$. Anwendung der Regel $A \to \gamma$ auf $u_i\alpha_i$ ergibt $u_i\gamma\alpha_i' = u_i a_1 \cdots a_m B\gamma'\alpha_i' = u_{i+1}\alpha_{i+1}$.

„2 ⇒ 1“.

Es sei $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, Z_0, \Delta)$ ein PDA mit Akzeptanz per leerem Keller. Wir wollen eine kontextfreie Grammatik G konstruieren mit $L(G) = L(\mathcal{A})$. Die Nichtterminalsymbole von G sind alle Tripel $[p, Z, q] \in Q \times \Gamma \times Q$.

Idee:

Es soll gelten: $[p, Z, q] \vdash_G^* u \in \Sigma^*$ gdw.

1. $\mathcal{A}$ erreicht *vom Zustand p aus den Zustand q* (in beliebig vielen Schritten),
2. durch *Lesen von u* auf dem Eingabeband und
3. *Löschen von Z* aus dem Keller (ohne dabei die Symbole unter Z anzutasten).

Die Produktionen beschreiben die intendierte Bedeutung jedes Nichtterminals $[p, Z, q]$ auf folgende Weise: um q von p aus unter Lesen der Eingabe $u = av$ und Löschen des Stacksymbols Z zu erreichen, brauchen wir einen Übergang $(p, a, Z, X_1 \cdots X_n, p_0)$, der Z durch Symbole $X_1 \cdots X_n$ ersetzt und das erste Symbol a von u liest (hier ist auch $a = \varepsilon$ möglich). Nun müssen wir noch den neu erzeugten Stackinhalt $X_1 \cdots X_n$ loswerden. Das tun wir Schritt für Schritt mittels Zerlegung des Restwortes $v = v_1 \cdots v_n$ und über Zwischenzustände $p_1, \ldots, p_{n-1}$, so dass

- $[p_0, X_1, p_1]$ unter Lesen von v_1,
- $[p_1, X_2, p_2]$ unter Lesen von v_2,
- $\cdots$
- $[p_{n-1}, X_n, q]$ unter Lesen von v_n.

(Hier ist $[p, X, q]$ jeweils gemäss den obigen Punkten 1-3 zu lesen).

Da die benötigten Zwischenzustände $p_1, \ldots, p_{n-1}$ nicht bekannt sind, fügen wir einfach eine Produktion

$$[p, Z, q] \longrightarrow a[p_0, X_1, p_1] \cdots [p_{n-1}, X_n, q]$$

für *alle möglichen* Zustandsfolgen $p_1, \ldots, p_{n-1}$ hinzu. In einer Ableitung der resultierenden Grammatik können wir dann die Regel mit den „richtigen“ Zwischenzuständen auswählen (und eine falsche Auswahl führt einfach zu keiner Ableitung).

Präzise Definition:

$$
\begin{aligned}
G &:= (N, \Sigma, P, S) \text{ mit} \\
N &:= \{S\} \cup \{[p, Z, q] \mid p, q \in Q, Z \in \Gamma\}, \\
P &:= \{S \longrightarrow [q_0, Z_0, q] \mid q \in Q\} \cup \\
&\quad \{[p, Z, q] \longrightarrow a \mid (p, a, Z, \varepsilon, q) \in \Delta \text{ mit } a \in \Sigma \cup \{\varepsilon\}\ \} \cup \\
&\quad \{[p, Z, q] \longrightarrow a[p_0, X_1, p_1][p_1, X_2, p_2] \ldots [p_{n-1}, X_n, q] \mid \\
&\qquad (p, a, Z, X_1 \ldots X_n, p_0) \in \Delta,\ q \in Q \text{ sowie} \\
&\qquad a \in \Sigma \cup \{\varepsilon\}, \\
&\qquad p_1, \ldots, p_{n-1} \in Q, \\
&\qquad n \geq 1\}.
\end{aligned}
$$

Beachte:

Für $n = 0$ haben wir den Übergang $(p, a, Z, \varepsilon, q) \in \Delta$, welcher der Produktion $[p, Z, q] \longrightarrow a$ entspricht.

Behauptung:

Für alle $p, q \in Q, u \in \Sigma^*, Z \in \Gamma, \gamma \in \Gamma^*$ gilt:

$$(\star) \qquad [p, Z, q] \vdash_G^* u \ \text{ gdw. } \ (p, u, Z) \vdash_{\mathcal{A}}^* (q, \varepsilon, \varepsilon).$$

Für $p = q_0$ und $Z = Z_0$ folgt daraus:

$$S \vdash_G [q_0, Z_0, q] \vdash_G^* u \ \text{ gdw. } \ (q_0, u, Z_0) \vdash_{\mathcal{A}}^* (q, \varepsilon, \varepsilon)$$

d. h. $u \in L(G)$ gdw. $u \in L(\mathcal{A})$.

Der Beweis dieser Behauptung kann durch Induktion über die Länge der Ableitung („$\Rightarrow$“) bzw. über die Länge der Konfigurationsfolge („$\Leftarrow$“) geführt werden.

❑

Beispiel:

Gegeben sei der PDA aus Beispiel 11.3, diesmal aber mit Akzeptanz per leerem Keller; also ist f zwar weiterhin ein Zustand, aber kein akzeptierender Zustand mehr. Dieser PDA erkennt ebenfalls $\{a^n b^n \ : \ n \geq 1\}$.[3] Die Berechnung dieses PDA

$$(q_0, ab, Z_0) \overset{(q_0, a, Z_0, ZZ_0, q_0)}{\vdash_{\mathcal{A}}} (q_0, b, ZZ_0) \overset{(q_0, b, Z, \varepsilon, q_1)}{\vdash_{\mathcal{A}}} (q_1, \varepsilon, Z_0) \overset{(q_1, \varepsilon, Z_0, \varepsilon, f)}{\vdash_{\mathcal{A}}} (f, \varepsilon, \varepsilon)$$

entspricht der Ableitung

$$S \vdash_G [q_0, Z_0, f] \vdash_G a[q_0, Z, q_1][q_1, Z_0, f] \vdash_G ab[q_1, Z_0, f] \vdash_G ab.$$

[3] Das klappt bei diesem PDA gerade, weil der Übergang von q_2 zu f der einzige ist, bei dem das Kellerstartsymbol gelöscht wird.

Aus Satz 11.9 ergibt sich leicht folgendes Korollar. Wir nennen zwei PDAs $\mathcal{A}$ und $\mathcal{A}'$ *äquivalent*, wenn $L(\mathcal{A}) = L(\mathcal{A}')$.

Korollar 11.10

Zu jedem PDA $\mathcal{A}$ gibt es einen PDA $\mathcal{A}'$ mit Akzeptanz per leerem Keller, so dass $L(\mathcal{A}) = L(\mathcal{A}')$ und $\mathcal{A}'$ nur einen Zustand hat.

Beweis. Gegeben einen PDA $\mathcal{A}$ können wir erst die Konstruktion aus dem Teil „$2 \Rightarrow 1$“ des Beweises von Satz 11.9 anwenden und dann die aus dem Teil „$1 \Rightarrow 2$“. Wir erhält einen äquivalenten PDA, der nach Konstruktion nur einen einzigen Zustand enthält. ❑

Wegen der gerade gezeigten Äquivalenz zwischen kontextfreien Sprachen und PDA-akzeptierbaren Sprachen können wir Eigenschaften von kontextfreien Sprachen mit Hilfe von Eigenschaften von Kellerautomaten zeigen. Als Beispiel betrachten wir den Durchschnitt von kontextfreien Sprachen mit regulären Sprachen. Wir wissen: Der Durchschnitt zweier kontextfreier Sprachen muss nicht kontextfrei sein. Dahingegen gilt:

Satz 11.11

Es sei $L \subseteq \Sigma^*$ kontextfrei und $R \subseteq \Sigma^*$ regulär. Dann ist $L \cap R$ kontextfrei.

Beweis. Es sei $L = L(\mathcal{A})$ für einen PDA $\mathcal{A} = (Q, \Sigma, \Gamma, q_s, Z_s, \Delta, F)$ (also mit akzeptierenden Zuständen) und $R = L(\mathcal{A}')$ für einen DEA $\mathcal{A}' = (Q', \Sigma, q'_s, \delta', F')$. Wir wenden eine Produktkonstruktion an, um einen PDA zu konstruieren, der $L \cap R$ erkennt:

$$\begin{aligned}
\mathcal{B} &:= (Q \times Q', \Sigma, \Gamma, (q_s, q'_s), Z_s, \Delta', F \times F') \text{ mit} \\
\Delta' &:= \{((p,p'), a, Z, \gamma, (q,q')) \;:\; (p, a, Z, \gamma, q) \in \Delta \text{ und } \delta(p', a) = q'\} \cup \\
&\quad\; \{((p,p'), \varepsilon, Z, \gamma, (q,p')) \;:\; (p, \varepsilon, Z, \gamma, q) \in \Delta\}
\end{aligned}$$

Es ist nun leicht (per Induktion über k) zu zeigen, dass
$((p,p'), uv, \gamma) \vdash_{\mathcal{B}}^{k} ((q,q'), v, \beta)$ gdw. $(p, uv, \gamma) \vdash_{\mathcal{A}}^{k} (q, v, \beta)$ und $p' \xrightarrow{u}_{\mathcal{A}'} q'$ ❑

Beachte: mit zwei PDAs als Eingabe funktioniert eine solche Produktkonstruktion nicht, da die beiden PDAs den Keller im Allgemeinen nicht „synchron“ nutzen (der eine kann das obere Kellersymbol löschen, während der andere Symbole zum Keller hinzufügt).

Deterministische Kellerautomaten

Analog zu endlichen Automaten können wir auch bei Kellerautomaten eine deterministische Variante betrachten. Intuitiv ist der PDA aus Beispiel 11.3 deterministisch, da es zu jeder Konfiguration höchstens eine Folgekonfiguration gibt. Der PDA aus Beispiel 11.4 ist hingegen nichtdeterministisch, da er die Wortmitte „rät“. Interessanterweise stellt es sich heraus, dass im Gegensatz zu DEAs/NEAs bei PDAs die deterministische Variante

echt schwächer ist als die nichtdeterministische. Daher definieren die deterministischen PDAs eine eigene Sprachklasse, die *deterministisch kontextfreien Sprachen.*

Deterministische PDAs akzeptieren immer per akzeptierendem Zustand (aus gutem Grund, wie wir noch sehen werden).

Definition 11.12 (deterministischer Kellerautomat)

Ein *deterministischer Kellerautomat (dPDA)* ist ein PDA

$$\mathcal{A} = (Q, \Sigma, \Gamma, q_s, Z_s, \Delta, F),$$

der folgende Eigenschaften erfüllt:

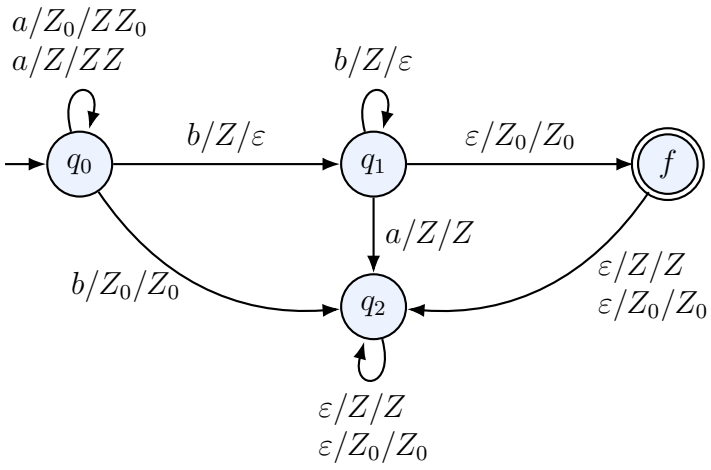
1. Für alle $q \in Q$, $a \in \Sigma$ und $Z \in \Gamma$ gibt es genau einen Übergang der Form (q, a, Z, γ, q') oder $(q, \varepsilon, Z, \gamma, q')$.
2. Wenn ein Übergang das Kellerstartsymbol Z_s entfernt, so muss er es direkt wieder zurückschreiben; alle Übergänge, in denen Z_s vorkommt, müssen also die Form $(q, a, Z_s, \gamma Z_s, q')$ haben, mit $\gamma \in \Gamma^*$ beliebig.

Es ist leicht zu sehen, dass es zu jeder Konfiguration eines dPDA, bei der der Keller nicht leer ist, genau eine Folgekonfiguration gibt. Die Bedingung 2 ist notwendig, damit der Keller tatsächlich nie leer wird (denn eine Konfiguration mit leerem Keller kann keine Folgekonfiguration haben). Wie ein PDA (mit akzeptierenden Zuständen) akzeptiert ein dPDA $\mathcal{A}$ ein Wort w gdw. $(q_s, w, Z_s) \vdash_{\mathcal{A}}^* (q_f, \varepsilon, \gamma)$ für ein $q_f \in F$ und $\gamma \in \Gamma^*$.

Beispiel 11.13

Als Beispiel für einen dPDA betrachte die folgende Variante des PDAs aus Beispiel 11.3, die ebenfalls $L = \{a^n b^n \ : \ n \geq 1\}$ erkennt:

- $Q = \{q_0, q_1, q_2, f\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{Z, Z_0\}$
- $q_s = q_0$
- $Z_s = Z_0$
- $F = \{f\}$
- $\Delta =$



Im Unterschied zum PDA aus Beispiel 11.3 entfernt der Übergang von q_1 nach f nicht mehr Z_0 vom Keller (weil das nicht erlaubt wäre), und der „Papierkorbzustand“ q_2 ist hinzugekommen.

Da die Arbeitsweise von dPDAs durchaus etwas subtil ist, hier zwei Hinweise zum vorhergehenden Beispiel:

- Auf manchen Eingaben gibt es mehr als eine Berechnung. Als Beispiel betrachten wir die Eingabe *aabb*. Nachdem diese gelesen wurde, befindet sich der PDA im Zustand q_1, der kein akzeptierender Zustand ist. Wir können jedoch den akzeptierenden Zustand f in einem weiteren Schritt erreichen, also wird die Eingabe *aabb* akzeptiert. Danach können wir im Prinzip noch den nicht-akzeptierenden Zustand q_2 erreichen und in diesem beliebig oft loopen (was aber nicht zur Akzeptanz beiträgt – und damit auch nicht zur erkannten Sprache).
- Trotz des Determinismus können manche Eingaben wie z. B. *ba* nicht vollständig gelesen werden.

Eine interessante Eigenschaft von deterministischen PDAs ist, dass für sie das Wortproblem in *Linearzeit* (also sehr effizient) entschieden werden kann. Aus diesem Grund spielen dPDAs im Gebiet des Compilerbaus eine wichtige Rolle.

Definition 11.14

Eine formale Sprache L heißt *deterministisch kontextfrei*, wenn es einen dPDA $\mathcal{A}$ gibt mit $L(\mathcal{A}) = L$. Die Menge aller deterministisch kontextfreien Sprachen ist $\mathcal{L}_2^d$.

Folgende Einordnung der deterministisch kontextfreien Sprachen ist leicht vorzunehmen.

Satz 11.15

$\mathcal{L}_3 \subsetneq \mathcal{L}_2^d \subseteq \mathcal{L}_2$.

Beweis. Es gilt $\mathcal{L}_3 \subsetneq \mathcal{L}_2^d$, da jeder DEA $\mathcal{A}$ als dPDA ohne ε-Übergänge und mit nur einem Kellersymbol Z_s betrachtet werden kann, der zudem seinen Keller nie modifiziert: aus jedem Übergang $\delta(q, a) = q'$ des DEA wird der Übergang (q, a, Z_s, Z_s, q') des dPDA. Die Inklusion ist echt, da mit Beispiel 11.13 $L = \{a^n b^n \ : \ n \geq 1\} \in \mathcal{L}_2^d$, wohingegen $L \notin \mathcal{L}_3$. Die Inklusion $\mathcal{L}_2^d \subseteq \mathcal{L}_2$ gilt nach Satz 11.6. ❑

Wie bereits erwähnt sind dPDAs echt schwächer als PDAs, d. h. die deterministisch kontextfreien Sprachen sind eine *echte Teilmenge* der kontextfreien Sprachen. Der Beweis beruht auf dem folgenden Resultat. Wir verzichten hier auf den etwas aufwändigen Beweis und verweisen z. B. auf [Koz07].

Satz 11.16

$\mathcal{L}_2^d$ ist unter Komplement abgeschlossen.

Zur Erinnerung: die kontextfreien Sprachen selbst sind mit Korollar 10.8 nicht unter Komplement abgeschlossen. Es kann gezeigt werde, dass die deterministisch kontextfreien Sprachen nicht unter Schnitt, Vereinigung, Konkatenation und Kleene-Stern abgeschlossen sind.

Satz 11.17
$\mathcal{L}_2^d \subsetneq \mathcal{L}_2$.

Beweis. Mit Satz 11.16 ist der folgende sehr einfache Beweis möglich: wäre $\mathcal{L}_2^d = \mathcal{L}_2$, so wäre mit Satz 11.16 $\mathcal{L}_2$ unter Komplement abgeschlossen, was jedoch ein Widerspruch zu Korollar 10.8 ist.

Dieser Beweis liefert jedoch keine konkrete Sprache, die kontextfrei aber nicht deterministisch kontextfrei ist. In der Übung zeigen wir, dass die Sprache

$$L = \{w \in \{a,b\}^* \ : \ \forall v \in \{a,b\}^* : w \neq vv\}$$

kontextfrei ist (durch Angabe einer Grammatik), ihr Komplement

$$\overline{L} = \{w \in \{a,b\}^* \ : \ \exists v \in \{a,b\}^* : w = vv\}$$

aber nicht (Pumping-Lemma für kontextfreie Sprachen). Wäre $L \in \mathcal{L}_2^d$, so wäre mit Satz 11.16 auch $\overline{L} \in \mathcal{L}_2^d \subseteq \mathcal{L}_2$, womit ein Widerspruch hergestellt ist. ❑

Auch die Sprache $\{ww^R \ : \ w \in \{a,b\}^*\}$ aus Beispiel 11.4 ist kontextfrei, aber nicht deterministisch kontextfrei; der Beweis ist allerdings aufwändig. Intuitiv ist der Grund aber, dass das nicht-deterministische „Raten“ der Wortmitte essentiell ist. Dies wird auch dadurch illustriert, dass die Sprache $\{wcw^R \ : \ w \in \{a,b\}^*\}$, bei der die Wortmitte explizit durch das Symbol c angezeigt wird, sehr einfach mit einem dPDA erkannt werden kann.

Zum Abschluss bemerken wir noch, dass Akzeptanz per leerem Keller bei dPDAs zu Problemen führt.

Lemma 11.18
Es gibt keinen dPDA mit Akzeptanz per leerem Keller, der die endliche (also reguläre) Sprache $L = \{a, aa\}$ erkennt.

Beweis. Angenommen, der dPDA $\mathcal{A}$ mit Akzeptanz per leerem Keller erkennt L. Da $a \in L$, gibt es eine Konfigurationsfolge

$$\Omega = (q_0, a, Z_0) \vdash_{\mathcal{A}} (q_1, w_1, \gamma_1) \vdash_{\mathcal{A}} \cdots \vdash_{\mathcal{A}} (q_n, w_n, \gamma_n)$$

mit $w_n = \gamma_n = \varepsilon$. Also gibt es auch eine Konfigurationsfolge

$$\Omega' = (q_0, aa, Z_0) \vdash_{\mathcal{A}} (q_1, u_1, \gamma_1) \vdash_{\mathcal{A}} \cdots \vdash_{\mathcal{A}} (q_n, u_n, \gamma_n)$$

mit $u_n = a$ und $\gamma_n = \varepsilon$; da $\mathcal{A}$ deterministisch ist, ist das die *einzige* Konfigurationsfolge, die das Präfix a der Eingabe aa verarbeitet. Wegen $\gamma_n = \varepsilon$ hat (q_n, u_n, γ_n) keine Folgekonfiguration, also wird aa nicht akzeptiert (dies kann per Definition nur nach Lesen der gesamten Eingabe geschehen). Widerspruch. ❑

Diese Probleme können behoben werden, indem ein explizites Symbol für das Wortende eingeführt wird, das auch in Übergängen von dPDAs verwendet werden kann. Mit einem solchen Symbol sind Akzeptanz per akzeptierendem Zustand und Akzeptanz per leerem Keller auch für dPDAs äquivalent.

12. Die Struktur kontextfreier Sprachen

In diesem Abschnitt gehen wir der Frage nach, was die kontextfreien von den regulären Sprachen unterscheidet. Wir können kontextfreie Sprachen mit Typ-2-Grammatiken erzeugen und mit Kellerautomaten akzeptieren; das Pumping-Lemma zeigt die Grenzen der kontextfreien Sprachen auf. Eine wichtige Erkenntnis war dabei, dass PDAs (bzw. kontextfreie Grammatiken) unbeschränkt zählen können, NEAs jedoch nicht.

Ein wichtiges Beispiel kontextfreier Sprachen sind die *Dyck-Sprachen*. Dyck-Sprachen sind nach Walther von Dyck (1856–1934), einem deutschen Mathematiker, benannt und bilden die Grundlage von Programmiersprachen, HTML und XML.

Definition 12.1

Für jede natürliche Zahl n ist die Dyck-Sprache D_n die Wortmenge der korrekt geklammerten (wohlgeformten) Ausdrücke mit n unterschiedlichen Klammerpaaren. Formal werden sie gebildet durch kontextfreie Grammatiken der Form

$$S \longrightarrow \underset{1}{(}S\underset{1}{)},\ \ldots,\ S \longrightarrow \underset{n}{(}S\underset{n}{)},\ S \longrightarrow SS,\ S \longrightarrow \varepsilon.$$

Hierbei stehen die Symbole $\underset{i}{(}, \underset{i}{)}$ für die unterschiedlichen Klammerpaare.

In diesem Abschnitt wollen wir zeigen, dass jede kontextfreie Sprache dargestellt werden kann als der Schnitt einer Dyck-Sprache und einer regulären Sprache, plus „Umbenennung“.

Beispiel 12.2

Es gilt $\{a^n b^n \ :\ n \geq 0\} = h(D_1 \cap R)$, wobei

- D_1 die *Dyck-Sprache mit einem Klammerpaar* ist, erzeugt durch die kontextfreie Grammatik $G = (N, \Sigma, P, S)$ mit $N = \{S\}$, $\Sigma = \{(,)\}$ und
$$P = \{S \longrightarrow (S),\ S \longrightarrow SS,\ S \longrightarrow \varepsilon\};$$
- R die reguläre Sprache $(^*)^*$ ist;
- h eine Abbildung ist, die das Symbol (in a und das Symbol) in b umbenennt.

Um genauer zu formulieren, was eine zulässige Umbenennung h ist, benötigen wir den Begriff des Homomorphismus.

Definition 12.3

Seien Σ und Γ Alphabete. Ein *Homomorphismus von Σ^* nach Γ^** ist eine Abbildung $h : \Sigma^* \to \Gamma^*$, so dass $h(wv) = h(w)h(v)$ für alle $w, v \in \Sigma^*$.

Ist $L \subseteq \Sigma^*$ eine Sprache, so bezeichnen wir die Sprache $h(L) := \{h(w) \ :\ w \in L\}$ als das *homomorphe Bild von L unter h*.

Anschaulich bildet ein Homomorphismus Wörter über Σ auf Wörter über Γ so ab, dass dabei die Konkatenation respektiert wird.

Aus Definition 12.3 folgt fast unmittelbar:

- $h(\varepsilon) = \varepsilon$
- $h(a_1 \cdots a_n) = h(a_1) \cdots h(a_n)$ für alle $a_1 \cdots a_n \in \Sigma^*$

Deshalb reicht es zur Definition eines Homomorphismus h immer aus, nur $h(a)$ für alle $a \in \Sigma$ anzugeben.

Homomorphismen liefern eine weitere wichtige Abschlusseigenschaft der regulären und kontextfreien Sprachen, welche wir hier jedoch nicht beweisen wollen.

Satz 12.4

Ist L eine reguläre (bzw. kontextfreie) Sprache und h ein Homomorphismus, dann ist auch das homomorphe Bild $h(L)$ regulär (bzw. kontextfrei).

Das zentrale Resultat dieses Abschnitts besagt nun, dass jede kontextfreie Sprache das homomorphe Bild des Schnittes einer Dyck-Sprache und einer regulären Sprache ist. Genauer:

Satz 12.5 (Chomsky-Schützenberger)

Für jede kontextfreie Sprache L gibt es eine Dyck-Sprache D, eine reguläre Sprache R und einen Homomorphismus h, so dass:

$$L = h(D \cap R)$$

Beweis. Sei L eine kontextfreie Sprache und $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik mit $L(G) = L$. Wegen Satz 9.18 können wir o. B. d. A. annehmen, dass G *in Chomsky-Normalform* ist; jede Produktion $\pi \in P$ hat also die Form

(1) $A \longrightarrow BC$ mit $A, B, C \in N$ oder

(2) $A \longrightarrow a$ mit $A \in N$ und $a \in \Sigma$.

Wir ignorieren dabei die mögliche Produktion $S \longrightarrow \varepsilon$. Für jede Produktion $\pi \in P$ führen wir nun zwei neue Klammerpaare $\overset{1}{\underset{\pi}{(}}, \overset{1}{\underset{\pi}{)}}, \overset{2}{\underset{\pi}{(}}, \overset{2}{\underset{\pi}{)}}$ ein und definieren:

$$\pi' = \begin{cases} A \longrightarrow \overset{1}{\underset{\pi}{(}} B \overset{1}{\underset{\pi}{)}} \overset{2}{\underset{\pi}{(}} C \overset{2}{\underset{\pi}{)}} & \text{wenn } \pi = A \longrightarrow BC \\ A \longrightarrow \overset{1}{\underset{\pi}{(}} \overset{1}{\underset{\pi}{)}} \overset{2}{\underset{\pi}{(}} \overset{2}{\underset{\pi}{)}} & \text{wenn } \pi = A \longrightarrow a \end{cases}$$

Aus diesen neuen Produktionen bilden wir nun die Grammatik $G' = (N, \Gamma, P', S)$ mit

$$\Gamma = \{\overset{1}{\underset{\pi}{(}}, \overset{1}{\underset{\pi}{)}}, \overset{2}{\underset{\pi}{(}}, \overset{2}{\underset{\pi}{)}} \ : \ \pi \in P\} \quad \text{und}$$
$$P' = \{\pi' \ : \ \pi \in P\}.$$

Des Weiteren betrachten wir D_Γ, die Dyck-Sprache mit den Klammern aus Γ.

Anhand der Definition der π' ist leicht zu sehen, dass $L(G') \subseteq D_\Gamma$ ist; die Umkehrung gilt jedoch nicht: zum Beispiel sind die Klammerwörter

$$\underset{\pi\,\pi\,\pi\,\pi}{\overset{1\,1\,1\,1}{()\,()}} \quad \text{und} \quad \underset{\pi\,\pi\,\pi\,\pi}{\overset{2\,2\,1\,1}{()\,()}}$$

in D_Γ, aber nicht von G' erzeugbar.

Alle Wörter $w \in L(G')$ erfüllen jedoch folgende zusätzliche Eigenschaften.

1. Auf jedes $\underset{\pi}{\overset{1}{)}}$ folgt $\underset{\pi}{\overset{2}{(}}$.
2. Auf $\underset{\pi}{\overset{2}{)}}$ folgt nie eine öffnende Klammer.
3. Wenn $\pi = A \longrightarrow BC$, dann
   - folgt auf $\underset{\pi}{\overset{1}{(}}$ immer $\underset{\rho}{\overset{1}{(}}$ mit $\rho = B \longrightarrow \cdots$;
   - folgt auf $\underset{\pi}{\overset{2}{(}}$ immer $\underset{\sigma}{\overset{1}{(}}$ mit $\sigma = C \longrightarrow \cdots$.
4. Wenn $\pi = A \longrightarrow a$, dann folgt auf $\underset{\pi}{\overset{1}{(}}$ immer $\underset{\pi}{\overset{1}{)}}$ und auf $\underset{\pi}{\overset{2}{(}}$ immer $\underset{\pi}{\overset{2}{)}}$.

Diese Eigenschaften können leicht per Induktion über die Anzahl der Regelanwendungen bewiesen werden.

Weiterhin gilt für jedes Nichtterminal $A \in N$ und alle Wörter w mit $A \vdash_{G'}^* w$:

5_A. w beginnt mit $\underset{\pi}{\overset{1}{(}}$, wobei $\pi = A \longrightarrow \cdots$

Eigenschaften 1–4 und 5_A lassen sich jedoch durch reguläre Ausdrücke beschreiben! Für jedes $A \in N$ ist also die Sprache

$$R_A := \{w \in \Gamma^* \ : \ w \text{ erfüllt Eigenschaften 1–4 und } 5_A\}$$

regulär. Wir wollen als nächstes zeigen, dass sich $L(G')$ von D_Γ *nur* durch Eigenschaften 1–4 und 5_A unterscheidet, Dann ist nämlich $L(G') = D_\Gamma \cap R_S$, und wir können durch Umbenennen der Klammern in Terminale auch L wie gewünscht darstellen. Zu diesem Zweck zeigen wir:

Behauptung. Für alle $A \in N$ gilt: $\quad A \vdash_{G'}^* w \quad$ gdw. $\quad w \in D_\Gamma \cap R_A$

„⇒“: dies folgt leicht per Induktion über die Anzahl der Regelanwendungen beim Ableiten von w.

„⇐“: per Induktion über die Länge von w.

Induktionsanfang. $w = \varepsilon$ erfüllt Eigenschaft 5_A nicht, ist also nicht in R_A. Damit ist die Behauptung trivialerweise erfüllt.

Induktionsschritt. Sei $w \in D_\Gamma \cap R_A$ mit $w \neq \varepsilon$. Weil w wohlgeklammert ist und Eigenschaften 1–4 und 5_A erfüllt, ist leicht zu zeigen, dass

$$w = \overset{1}{\underset{\pi}{(}} u \overset{1}{\underset{\pi}{)}} \overset{2}{\underset{\pi}{(}} v \overset{2}{\underset{\pi}{)}} \quad \text{für } u, v \in \Gamma^* \text{ und } \pi = A \longrightarrow \cdots$$

Nun kann π eine von zwei Formen haben.

- Wenn $\pi = A \longrightarrow BC$, dann sind auch u und v wohlgeklammert und erfüllen die Eigenschaften 1–4 und 5_B bzw. 5_C (z. B. folgt 5_B für u aus 3 für w). Die Induktionsvoraussetzung liefert nun $B \vdash^*_{G'} u$ und $C \vdash^*_{G'} v$, und damit erhalten wir

$$A \;\vdash_{G'}\; \overset{1}{\underset{\pi}{(}} B \overset{1}{\underset{\pi}{)}} \overset{2}{\underset{\pi}{(}} C \overset{2}{\underset{\pi}{)}} \;\vdash^*_{G'}\; \overset{1}{\underset{\pi}{(}} u \overset{1}{\underset{\pi}{)}} \overset{2}{\underset{\pi}{(}} v \overset{2}{\underset{\pi}{)}} \;=\; w$$

  wie gewünscht.

- Wenn $\pi = A \longrightarrow a$, dann liefert Eigenschaft 4 für w, dass $u = v = \varepsilon$ ist. Nach Definition von G' erhalten wir

$$A \;\vdash_{G'}\; \overset{1}{\underset{\pi}{(}} \overset{1}{\underset{\pi}{)}} \overset{2}{\underset{\pi}{(}} \overset{2}{\underset{\pi}{)}} \;=\; w$$

  wie gewünscht.

Aus der Behauptung folgt nun insbesondere:

$$L(G') = D_\Gamma \cap R_S$$

Wir definieren nun den Homomorphismus h wie folgt.

- $h\Big(\overset{1}{\underset{\pi}{(}}\Big) = h\Big(\overset{1}{\underset{\pi}{)}}\Big) = h\Big(\overset{2}{\underset{\pi}{(}}\Big) = h\Big(\overset{2}{\underset{\pi}{)}}\Big) = \varepsilon, \qquad$ falls $\pi = A \longrightarrow BC$
- $h\Big(\overset{1}{\underset{\pi}{(}}\Big) = a$ und $h\Big(\overset{1}{\underset{\pi}{)}}\Big) = h\Big(\overset{2}{\underset{\pi}{(}}\Big) = h\Big(\overset{2}{\underset{\pi}{)}}\Big) = \varepsilon, \qquad$ falls $\pi = A \longrightarrow a$.

Für jede Produktion $\pi \in P$ gilt dann: h angewendet auf π' ergibt π. Damit erhalten wir wie gewünscht:

$$L(G) = h(L(G')) = h(D_\Gamma \cap R_S).$$

❑

Anschaulich gesprochen bedeutet der Satz von Chomsky-Schützenberger, dass jede kontextfreie Sprache modulo Umbenennen durch Homomorphismen eine durch eine reguläre Sprache beschreibbare Teilmenge einer Dyck-Sprache ist.

Überblick Theoretische Informatik 1 (Teile I & II)

Zum Abschluss von Teil II geben wir hier einen Überblick über die in *Theoretische Informatik 1* (Teile I & II) behandelten Themen.

Themenübersicht

- Sprachklassen (Chomsky-Hierarchie, Typen 0–3)
  - $\mathcal{L}_3$: regulär = erkennbar = rechtslinear
  - $\mathcal{L}_2^d$: deterministisch kontextfrei
  - $\mathcal{L}_2$: kontextfrei
  - $\mathcal{L}_1$: kontextsensitiv
  - $\mathcal{L}_0$
- Automatenmodelle zur Beschreibung von Sprachen
  - nichtdeterministische endliche Automaten (NEAs)
  - deterministische endliche Automaten (DEAs)
  - NEAs mit ε- und Wortübergängen
  - Kellerautomaten (PDAs)
  - deterministische Kellerautomaten (dPDAs)
- Andere Mechanismen, um Sprachen endlich zu beschreiben
  - reguläre Ausdrücke
  - Grammatiken (Typen 0–3: allgemeine, monotone, kontextfreie, reguläre)
- Eigenschaften von Sprachklassen
  - Abschlusseigenschaften
  - Entscheidbarkeit und Komplexität von Problemen
- Konstruktionen und Beweistechniken
  - Potenzmengenkonstruktion
  - Produktautomat
  - Quotientenautomat
  - Nerode-Rechtskongruenz
  - Pumping-Lemma für reguläre Sprachen
  - Pumping-Lemma für kontextfreie Sprachen
  - Normalformen von Grammatiken
  - etc.

Überblick Abschlusseigenschaften und Entscheidungsprobleme

Die Abschlusseigenschaften der Sprachklassen $\mathcal{L}_3$, $\mathcal{L}_2$ und $\mathcal{L}_2^d$ (unterste 3 Zeilen der folgenden Tabelle) haben wir in Abschnitten 4, 10 und 11 behandelt (für $\mathcal{L}_2^d$ aber nur den Fall des Komplements, Satz 11.16).

Klasse	$\cap$	$\cup$	$^-$	$\cdot$	*
$\mathcal{L}_0$	✓	✓	✗	✓	✓
$\mathcal{L}_1$	✓	✓	✓	✓	✓
$\mathcal{L}_2$	✗	✓	✗	✓	✓
$\mathcal{L}_2^d$	✗	✗	✓	✗	✗
$\mathcal{L}_3$	✓	✓	✓	✓	✓

Die Entscheidungsprobleme der Sprachklassen $\mathcal{L}_3$, $\mathcal{L}_2$ und $\mathcal{L}_2^d$ (unterste 4 Zeilen der folgenden Tabelle) haben wir in Abschnitten 5, 9 und 11 behandelt; für die regulären Sprachen ($\mathcal{L}_3$) haben wir gesehen, dass es dabei auf die Repräsentation ankommt (DEA versus NEA oder Grammatik oder regulärer Ausdruck). Deshalb ist diese Tabelle nicht nach Sprachklassen, sondern nach der Art der Repräsentation (Automat/Grammatik/regulärer Ausdruck) geordnet.

Repräsentation	Wortproblem	Leerheitsproblem	Äquivalenzproblem
Typ-0-Grammatik	unentscheidbar	unentscheidbar	unentscheidbar
Typ-1-Grammatik	PSpace-vollständig	unentscheidbar	unentscheidbar
Typ-2-Grammatik, PDA	in Polynomialzeit	in Polynomialzeit	unentscheidbar
dPDA	in Linearzeit	in Polynomialzeit	entscheidbar
Typ-3-Grammatik, NEA, reg. Ausdr.	in Linearzeit	in Linearzeit	PSpace-vollständig
DEA	in Linearzeit	in Linearzeit	in Polynomialzeit

III. Appendix

A. Laufzeitanalyse von Algorithmen und $\mathcal{O}$-Notation

Ein gegebenes Berechnungsproblem lässt sich meist durch viele verschiedene Algorithmen lösen. Um die „Qualität“ dieser vielen möglichen Algorithmen zu bestimmen, analysieren wir deren Ressourcenverbrauch. In diesem Zusammenhang ist die wichtigste Ressource der *Zeitverbrauch*, also die Anzahl Rechenschritte, die der Algorithmus ausführt. Andere Ressourcen, die eine Rolle spielen können, sind beispielsweise der Speicherverbrauch oder der Kommunikationsbedarf, wenn der Algorithmus auf mehreren Prozessoren oder Rechnern verteilt ausgeführt wird. Ein Algorithmus mit geringem Ressourcenbedarf ist dann einem Algorithmus mit höherem Ressourcenbedarf vorzuziehen.

Laufzeitanalyse

Die *Laufzeit* eines Algorithmus A auf einer Eingabe x ist die Anzahl *elementarer Rechenschritte*, die A gestartet auf x ausführt, bevor er anhält. Was ein elementarer Rechenschritt ist, wird in der VL „Theoretische Informatik II“ genauer untersucht. Bis dahin betrachten wir die üblichen Operationen eines Prozessors als elementare Rechenschritte, also z. B. Addition und Multiplikation. Die zentrale Eigenschaft eines elementaren Rechenschritts ist, dass er in konstanter Zeit ausgeführt werden kann – das soll heißen, dass die benötigte Zeit unabhängig von den konkreten Argumenten ist, auf die der Rechenschritt angewendet wird.

Wir beschreiben die Laufzeit eines Algorithmus immer in Abhängigkeit von der *Größe seiner Eingabe*. Dem liegt die Intuition zugrunde, dass das Verarbeiten einer größeren Eingabe im Allgemeinen länger dauert als das einer kleinen. So können wir z. B. offensichtlich schneller entscheiden, ob ein gegebener NEA ein Eingabewort der Länge 1 akzeptiert als eines der Länge 128. Der Zeitverbrauch eines Algorithmus kann also beschrieben werden durch eine Funktion

$$f\colon \mathbb{N} \to \mathbb{N},$$

die jeder Eingabelänge $n \in \mathbb{N}$ eine Laufzeit $f(n)$ zuordnet. Beachte, dass diese Darstellung von der *konkreten Eingabe* abstrahiert, d. h. die Laufzeit des Algorithmus auf verschiedenen Eingaben derselben Länge wird nicht unterschieden. Diese kann durchaus sehr unterschiedlich sein: für einen gegebenen NEA $\mathcal{A}$, in dem der Startzustand keinen a-Übergang erlaubt, ist es trivial zu entscheiden, dass das Wort

$$abbbababababbbbbbbbababbabbabbbbbbbbaaabbbbaaaaaab$$

nicht akzeptiert wird, wohingegen dies für das gleichlange, aber mit b beginnende Wort

$$bbbababababbbbbbbbababbabbabbbbbbbbaaabbbbaaaaaaaba$$

viel mehr Schritte erfordern kann. Die durch die Funktion $f\colon \mathbb{N} \to \mathbb{N}$ beschriebene Laufzeit ist als *Worst-Case-Komplexität* zu verstehen: $f(n)$ beschreibt eine obere Schranke, also eine maximale Schrittzahl, die für keine Eingabe der Länge n überschritten wird (die aber für „einfache“ Eingaben unter Umständen stark *unter*schritten werden kann).

Effiziente Laufzeit

Welche Laufzeit werden als *effizient* angesehen und welche nicht? Die wichtigste Grenze verläuft zwischen polynomiellem und super-polynomiellem Laufzeitverhalten, wobei ersteres im allgemeinen mit „machbar“ gleichgesetzt wird und letzteres mit steigender Eingabegröße so schnell wächst, dass größere Eingaben nicht verarbeitet werden können. Bei *polynomieller Laufzeit* ist die Funktion f also ein Polynom wie zum Beispiel

$$n, \quad n^2, \quad 5n, \quad 7n^2, \quad 8n^3, \quad 3n^3 + 2n^2 + 5n + 7.$$

Bei *super-polynomieller Laufzeit* ist sie eine Funktion $f(n)$, so dass

$$\limsup_{n\to\infty} \frac{f(n)}{p(n)} = \infty$$

für jedes Polynom p, z.B.

$$2^{\sqrt{n}}, 6^{3n}, n^n.$$

Hierbei ist der Limes superior (größter Häufungspunkt) einer Folge $(x_n)_{n\in\mathbb{N}}$ definiert als

$$\limsup_{n\to\infty} x_n = \inf\{\sup\{x_k : k \geq n\} : n \in \mathbb{N}\}.$$

Dabei stehen inf und sup für *Infimum* und *Supremum* und wir erlauben für ihre Werte die erweiterten natürlichen Zahlen $\mathbb{N} \cup \{\infty\}$. Wir benutzen den Limes superior als Ersatz für den Grenzwert (lim), falls dieser nicht existiert.

Es ist wichtig sich vor Augen zu führen, dass exponentielle Laufzeit so dramatisch ist, dass sie von schnellerer Hardware nicht kompensiert werden kann. Betrachte z. B. die Laufzeit 2^n auf Eingaben der (sehr moderaten!) Größe $n = 128$. Die sich ergebende Anzahl Schritte ist so gewaltig, dass ein moderner 16,34-GHz-Prozessor länger rechnen müsste als vom Anfang des Universums bis heute. Aber auch Polynome höheren Grades sind recht schnell wachsende Funktionen. Interessanterweise finden wir aber nur äußerst selten Algorithmen, deren Laufzeit zwar polynomiell ist, aber nicht durch ein Polynom kleinen Grades (meist ≤ 5, oft ≤ 3) beschrieben werden kann. Dies rechtfertigt die übliche Gleichsetzung von „polynomiell“ und „machbar“. Für sehr zeitkritische Probleme kann aber auch eine Laufzeit von n^2 zu lang sein. Als besonders effizient gilt *Linearzeit*, also Laufzeiten der Form $c \cdot n$, wobei $c \in \mathbb{N}$ eine Konstante ist.

$\mathcal{O}$-Notation

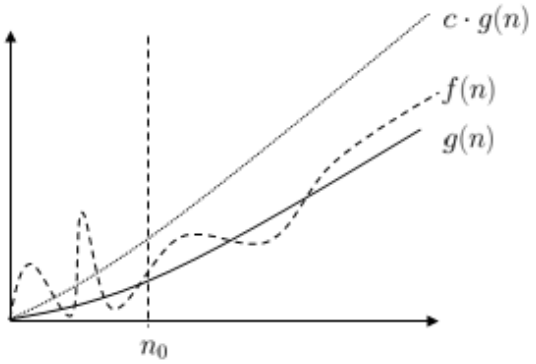
Bei der Laufzeitanalyse von Algorithmen abstrahieren wir so gut wie immer von konkreten Konstanten. Wir würden also beispielsweise nicht zwischen einem Algorithmus mit Laufzeit $2n$ und einem mit Laufzeit $4n$ unterscheiden (außer natürlich in einer konkreten Implementierung, wo derartige Unterschiede sehr wichtig sein können!). Dies hat mehrere Gründe. Erstens ist es schwierig und fehleranfällig, alle involvierten Konstanten zu identifizieren und während der für die Laufzeitabschätzung benötigten Rechnungen „mitzuschleppen“. Zweitens sind die konkreten Konstanten oft abhängig von Implementierungsdetails wie den konkret zur Verfügung stehenden elementaren Rechenschritten und den zur Repräsentation der Eingabe verwendeten Datenstrukturen. Die so genannte „$\mathcal{O}$-Notation“ stellt eine einfache Methode zur Verfügung, um konkrete Konstanten auszublenden.

Definition A.1 ($\mathcal{O}$-Notation)
Seien f und g Funktionen von $\mathbb{N}$ nach $\mathbb{N}$. Wir schreiben

$$f \in \mathcal{O}(g),$$

wenn es eine Konstante $c > 0$ und Schranke $n_0 \geq 0$ gibt, so dass $f(n) \leq c \cdot g(n)$ für alle $n > n_0$.

Mit anderen Worten bedeutet $f \in \mathcal{O}(g)$, dass f „schließlich nicht wesentlich schneller wächst“ als g. Die folgende Graphik illustriert dies anhand zweier Funktionen $f(n)$ und $g(n)$ mit $f \in \mathcal{O}(g)$:



Wie in Definition A.1 gefordert, liegt f *schließlich* (d. h. für *alle* Argumente größer als die Schranke n_0) unterhalb der mit der Konstanten c skalierten Funktion $c \cdot g(n)$. Auch wenn der absolute Wert von $f(n)$ an vielen Stellen größer ist als der von $g(n)$, wächst f also nicht wesentlich schneller als g.

Wir verwenden die Laufzeitbeschreibung $\mathcal{O}(f(n))$, wenn wir ausdrücken wollen, dass die Laufzeit $f(n)$ ist, bis auf Konstanten (repräsentiert durch c) und endlich viele Ausnahmen (repräsentiert durch n_0).

Insbesondere beschreibt

- $\mathcal{O}(n)$ Linearzeit,
- $\mathcal{O}(n^2)$ quadratische Zeit und
- $\bigcup_{i \geq 1} n^i$ polynomielle Zeit.

Es existieren verschiedene Rechenregeln für die $\mathcal{O}$-Notation wie zum Beispiel:

- $\mathcal{O}(\mathcal{O}(f)) = \mathcal{O}(f)$
- $\mathcal{O}(f) + \mathcal{O}(g) = \mathcal{O}(f + g)$
- $\mathcal{O}(f) \cdot \mathcal{O}(g) = \mathcal{O}(f \cdot g)$

Mehr Informationen zur $\mathcal{O}$-Notation finden sich beispielsweise im Buch „Concrete Mathematics“ [GKP94].

Abkürzungsverzeichnis

bzw. ..... beziehungsweise
DEA ..... deterministischer endlicher Automat
d. h. ..... das heißt
DTM ..... deterministische Turingmaschine
EBNF ..... erweiterte Backus-Naur-Form
etc. ..... et cetera
gdw. ..... genau dann wenn
LBA ..... linear beschränkter Automat
MPKP ..... modifiziertes Postsches Korrespondenzproblem
NEA ..... nichtdeterministischer endlicher Automat
NP ..... nichtdeterministisch polynomiell
NTM ..... nichtdeterministische Turingmaschine
o. B. d. A. ..... ohne Beschränkung der Allgemeinheit
PDA ..... pushdown automaton (Kellerautomat)
dPDA ..... Deterministischer Kellerautomat
PKP ..... Postsches Korrespondenzproblem
SAT ..... satisfiability problem (Erfüllbarkeitstest der Aussagenlogik)
TM ..... Turingmaschine (allgemein)
usw. ..... und so weiter
vgl. ..... vergleiche
z. B. ..... zum Beispiel
□ ..... was zu beweisen war (q. e. d.)

Mathematische Symbole und Notation

Allgemeines

$\{\ \}$ ..... Mengenklammern

$x \in M$ ..... x ist Element der Menge M

$M_1 \subseteq M_2$ ..... M_1 Teilmenge von M_2

$M_1 \subset M_2$ oder $M_1 \subsetneq M_2$ ..... M_1 *echte* Teilmenge von M_2

$\emptyset$ ..... leere Menge

$M_1 \cup M_2$ ..... Vereinigung von Mengen M_1 und M_2

$M_1 \cap M_2$ ..... Schnitt von Mengen M_1 und M_2

$M_1 \setminus M_2$ ..... Mengendifferenz: M_1 ohne die Elemente von M_2

$\overline{M}$ ..... Komplement der Menge M
(bezüglich einer als bekannt angenommenen „Gesamtmenge“)

$M_1 \times M_2$ ..... Kreuzprodukt der Mengen M_1 und M_2

$|M|$ ..... Kardinalität der Menge M (Anzahl Elemente)

2^M ..... Potenzmenge von M (Menge aller Teilmengen)

$\wedge$ ..... logisches „und“

$\vee$ ..... logisches „oder“

$\neg$ ..... logisches „nicht“

$\Rightarrow$ (auch $\rightarrow$) ..... logisches „impliziert“

$\Leftrightarrow$ (auch $\leftrightarrow$) ..... logisches „genau dann, wenn“

$\exists$ ..... „es existiert“

$\forall$ ..... „für alle“

$\mathbb{N}$ ..... Menge der natürlichen Zahlen (einschließlich der Null)

$f : M \rightarrow M'$ ..... Funktion f bildet von Menge M in Menge M' ab

$n!$ ..... n Fakultät ($n! = 1 \cdot 2 \cdot \ldots \cdot n$)

$f \in O(g)$ ..... Funktion f wächst schließlich nicht schneller als Funktion g

$f|_D$ ..... Einschränkung der Funktion f auf Definitionsbereich D

$\sim$ ..... Äquivalenzrelation

$:=$ ..... „ist definiert als“

Wörter und Sprachen

Σ ..... Alphabet (oft auch: Eingabealphabet)
a, b, c ..... Alphabetsymbole
w, u, v, x, y, z ..... Wörter
ε ..... leeres Wort
a^n ..... Wort, das aus n mal dem Symbol a besteht
$|w|$ ..... Länge des Wortes w
$|w|_a$ ..... Anzahl Vorkommen des Symbols a im Wort w
$w_1 \cdot w_2$ ..... Konkatenation der Wörter w_1 und w_2
L ..... formale Sprache
$L_1 \cdot L_2$ ..... Konkatenation der Sprachen L_1 und L_2
L^i ..... i-fache Konkatenation der Sprache L mit sich selbst
L^* ..... Kleene-Stern, angewendet auf die Sprache L
L^+ ..... $= L \cdot L^*$

Endliche Automaten

$\mathcal{A}$ ..... Automat (DEA, NEA, Kellerautomat, LBA, Turingmaschine)
Q ..... Zustandsmenge
F ..... Endzustandsmenge
q, p ..... Zustände von Automaten
q_0 ..... Startzustand von Automat
δ ..... Übergangsfunktion von deterministischen Automaten
$\hat{\delta}$ ..... Kanonische Fortsetzung der Übergangsfunktion
Δ ..... Übergangsrelation von nichtdeterministischen Automaten / Turingmaschinen
$L(\mathcal{A})$ ..... Vom Automaten $\mathcal{A}$ erkannte Sprache
$p \xrightarrow{a}_{\mathcal{A}} q$ ..... NEA $\mathcal{A}$ kann in einem Schritt unter Lesen von Symbol a von Zustand p in Zustand q übergehen
$p \xRightarrow{w}_{\mathcal{A}} q$ ..... NEA $\mathcal{A}$ kann unter Lesen von Wort w von Zustand p in Zustand q übergehen
$p \Longrightarrow^i_{\mathcal{A}} q$ ..... NEA $\mathcal{A}$ kann in i Schritten von Zustand p in Zustand q übergehen

Reguläre Ausdrücke

r, s ..... reguläre Ausdrücke
$r \cdot s$ ..... regulärer Ausdruck für Konkatenation
$r + s$ ..... regulärer Ausdruck für Vereinigung
r^* ..... regulärer Ausdruck für Kleene-Stern
Reg_Σ ..... Menge aller regulärer Ausdrücke über Σ

Nerode-Rechtskongruenz

$q \sim_{\mathcal{A}} q'$ ..... Zustände q und q' sind $\mathcal{A}$-äquivalent
$[q]_{\mathcal{A}}$ ..... Äquivalenzklasse von Zustand q bzgl. $\sim_{\mathcal{A}}$
$\simeq_L$ ..... Nerode-Rechtskongruenz für Sprache L
$[u]_L$ ..... Äquivalenzklasse von Wort u bzgl. $\simeq_L$
$\mathcal{A} \simeq \mathcal{A}'$ ..... Die DEAs $\mathcal{A}$ und $\mathcal{A}'$ sind isomorph

Grammatiken und Chomsky-Hierarchie

G ..... Grammatik
N ..... Menge der nichtterminalen Symbole einer Grammatik
P ..... Menge der Regeln (Produktionen) einer Grammatik
A, B, C ..... Nichtterminale Symbole
S ..... Startsymbol
$u \longrightarrow v$ ..... Produktion einer Grammatik
$x \vdash_G y$ ..... Wort y aus Wort x in Grammatik G in einem Schritt ableitbar
$x \vdash_G^i y$ ..... Wort y aus Wort x in Grammatik G in i Schritten ableitbar
$x \vdash_G^* y$ ..... Wort y aus Wort x in Grammatik G ableitbar
$L(G)$ ..... die von Grammatik G erzeugte Sprache
$\mathcal{L}_0$ ..... Klasse der Sprachen vom Typ 0
$\mathcal{L}_1$ ..... Klasse der kontextsensitiven Sprachen (Typ 1)
$\mathcal{L}_2$ ..... Klasse der kontextfreien Sprachen (Typ 2)
$\mathcal{L}_3$ ..... Klasse der regulären Sprachen (Typ 3)

Kellerautomaten und Turingmaschinen

Γ ..... Kelleralphabet (Kellerautomat), Bandalphabet (Turingmaschine)
Z_0 ..... Kellerstartsymbol
$k \vdash_{\mathcal{A}} k'$ ..... Kellerautomat/Turingmaschine $\mathcal{A}$ kann in einem Schritt von Konfiguration k in Konfiguration k' übergehen
$k \vdash_{\mathcal{A}}^i k'$ ..... Kellerautomat/Turingmaschine $\mathcal{A}$ kann in i Schritten von Konfiguration k in Konfiguration k' übergehen
$k \vdash_{\mathcal{A}}^* k'$ ..... Kellerautomat/Turingmaschine $\mathcal{A}$ kann von Konfiguration k in Konfiguration k' übergehen
$\mathcal{L}_2^d$ ..... Klasse der deterministisch kontextfreien Sprachen

$\not\flat$ ..... Blank-Symbol (Turingmaschine)
$\$, \not\epsilon$ ..... Bandbegrenzungssymbole (LBA)

Griechische Buchstaben

Kleinbuchstaben und einige typische Verwendungen im Skript

α ..... alpha
β ..... beta
γ ..... gamma
δ (bezeichnet meist Übergangsfunktion) ..... delta
ε (bezeichnet meist das leere Wort) ..... epsilon
ζ ..... zeta
η ..... eta
ϑ, θ ..... theta
ι ..... iota
κ ..... kappa
λ ..... lambda
μ ..... my
ν ..... ny
ξ xi
o ..... omikron
π (bezeichnet meist Pfad in NEA) ..... pi
ρ ..... rho
σ, ς ..... sigma
τ ..... tau
υ ..... ypsilon
φ, ϕ ..... phi
χ ..... chi
ψ ..... psi
ω ..... omega

Großbuchstaben und einige typische Verwendungen im Skript

Γ (bezeichnet meist Keller- bzw. Bandalphabet) ..... Gamma
Δ (bezeichnet meist Übergangsrelation) ..... Delta
Θ ..... Theta
Λ ..... Lambda
Π ..... Pi
Ξ Xi
Σ (bezeichnet meist Alphabet) ..... Sigma
Υ ..... Ypsilon
Φ ..... Phi
Ψ ..... Psi
Ω ..... Omega

Die restlichen griechischen Großbuchstaben werden genauso geschrieben wie die entsprechenden lateinischen.

[GJ79]

Literaturverzeichnis

[CLRS01] Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest und Clifford Stein: *Introduction to Algorithms*. The MIT Press, 2. Auflage, 2001.

[GJ79] Garey, Michael R. und David S. Johnson: *Computers and Intractability: A Guide to the Theory of* NP*-Completeness*. Freeman, 1979.

[GKP94] Graham, Ronald L., Donald E. Knuth und Oren Patashnik: *Concrete Mathematics*. Addison-Wesley, 2. Auflage, 1994.

[Koz07] Kozen, Dexter: *Automata and Computability*. Springer Verlag, 2007.