TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut für Theoretische Informatik, Lehrstuhl Automatentheorie

Skript

Formale Systeme

Modul INF-B-270

Teil 1 - Automatentheorie und formale Sprachen

Prof. Dr.-Ing. Franz Baader

Oktober 2010

Inhaltsverzeichnis

Organisation der Lehrveranstaltung

Die Lehrveranstaltung

„Formale Systeme“ (Modul INF-B-270)

gliedert sich in

- **Teil 1:** Automatentheorie und formale Sprachen und
- **Teil 2:** Logik in der Informatik

Verantwortlicher Hochschullehrer:

Prof. Dr.-Ing. Franz Baader

Technische Universität Dresden
Fakultät Informatik
Institut für Theoretische Informatik
01062 Dresden

`http://lat.inf.tu-dresden.de/~baader/`

Aktuelle Informationen zur Lehrveranstaltung:

`http://lat.inf.tu-dresden.de/teaching/ws2010-2011/FormaleSysteme/`

Hinweis

Dieses Skript ist als Hilfestellung für Studenten gedacht. Trotz großer Mühe beim Erstellen kann keine Garantie für Fehlerfreiheit übernommen werden. Es wird nochmals darauf hingewiesen, dass der prüfungsrelevante Stoff durch die Vorlesung bestimmt wird und nicht mit dem Skriptinhalt vollständig übereinstimmen muss.

Einführung

Die Lehrveranstaltung „Formale Systeme“ gibt eine Einführung in zwei wichtige Bereiche der Theoretischen Informatik:

Automatentheorie und formale Sprachen (Teil 1)

Hier werden *Wörter* und Mengen von Wörtern (*formale Sprachen*) untersucht.

- Wie beschreibt man solche Mengen?
  $\longrightarrow$ *Automaten* oder *Grammatiken* ...
- Beispiel: Daten (d.h. Eingabe und Ausgabe von Programmen) sind Wörter. Die formale Sprache beschreibt die syntaktisch korrekten Daten.

Logik in der Informatik (Teil 2)

Hier betrachtet man *logische Formeln* über atomaren Aussagen (Aussagenlogik).

- Wie sieht eine logische Formel aus?
  Wann ist eine logische Formel wahr bzw. nicht wahr?
  $\longrightarrow$ *Syntax und Semantik der Aussagenlogik*
- Wie kann man aus einer Menge von logischen Formeln weitere Aussagen schlussfolgern?
  $\longrightarrow$ *Kalküle des logischen Schließens*

Hinweis

Das vorliegende Skript umfasst Teil 1. Die Lehrunterlagen für den Teil 2 der Lehrveranstaltung werden gesondert veröffentlicht.

Automatentheorie und formale Sprachen

Dieser Teil der Theoretischen Informatik beschäftigt sich mit *Wörtern* und Mengen von Wörtern (*formalen Sprachen*).

Wichtige Fragestellungen:

1. **Charakterisierung**:

   Wie beschreibt man die (meist unendlichen) Mengen von Wörtern mit endlichem Aufwand?

   - *Automaten* oder *Maschinen*, die genau die Elemente der Menge akzeptieren.
   - *Grammatiken*, welche genau die Elemente der Menge generieren (vgl. LV „Programmierung“: kontextfreie Grammatiken (EBNF) zur Beschreibung der Syntax von Programmiersprachen).
   - *Ausdrücke*, welche beschreiben, wie man die Sprache aus Basissprachen mit Hilfe gewisser Operationen (z.B. Vereinigung) erzeugen kann.

   Abhängig von dem verwendeten Automaten-/Grammatiktyp erhält man verschiedene Klassen von Sprachen. Wir werden hier die vier wichtigsten Klassen betrachten, welche in der **Chomsky-Hierarchie** zusammengefasst sind:

Klasse	Automatentyp	Grammatiktyp
Typ 0	Turingmaschine (TM)	allgemeine Chomsky-Grammatik
Typ 1	TM mit linearer Bandbeschränkung	kontextsensitive Grammatik
Typ 2	Kellerautomat	kontextfreie Grammatik
Typ 3	endlicher Automat	einseitig lineare Grammatik

   Bei Typ 3 existiert auch eine Beschreibung durch reguläre Ausdrücke. Am wichtigsten sind die Typen 2 und 3; beispielsweise kann Typ 2 weitgehend die Syntax von Programmiersprachen beschreiben.

2. Welche **Problemstellungen** sind für eine Sprachklasse entscheidbar und mit welchem Aufwand?
   - *Wortproblem*: geg. eine Beschreibung der Sprache L (z.B. durch Automat, Grammatik, ...) und ein Wort w. Gilt $w \in L$?
     Anwendungsbeispiele:
     - Programmiersprache, deren Syntax durch eine kontextfreie Grammatik beschrieben ist. Entscheide für ein geg. Programm P, ob dieses syntaktisch korrekt ist.
     - Suchpattern für Textdateien sind häufig reguläre Ausdrücke. Suche die Dateien (Wörter), welche das Suchpattern enthalten (zu der von ihm beschriebenen Sprache gehören).
   - *Leerheitsproblem*: geg. eine Beschreibung der Sprache L. Gilt $L \neq \emptyset$?
     Anwendungsbeispiel:
     Wenn ein Suchpattern die leere Sprache beschreibt, so muss man die Dateien nicht durchsuchen.
   - *Äquivalenzproblem*: Beschreiben zwei verschiedene Beschreibungen dieselbe Sprache?
     Anwendungsbeispiel:
     Zwei Lehrbücher über eine Programmiersprache beschreiben die Syntax durch verschiedene Grammatiken. Sind diese wirklich äquivalent?

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

   z.B. Abschluss unter Durchschnitt, Vereinigung und Komplement (L_1, L_2 enthalten, so 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 (Durchschnitt).
   - Reduziere das Äquivalenzproblem auf das Leerheitsproblem: Statt „$L_1 = L_2$?“ entscheidet man, ob $(L_1 \cap \overline{L_2}) \cup (L_2 \cap \overline{L_1}) = \emptyset$.

I Endliche Automaten und reguläre Sprachen

Einführung

Endliche Automaten sind gekennzeichnet durch:

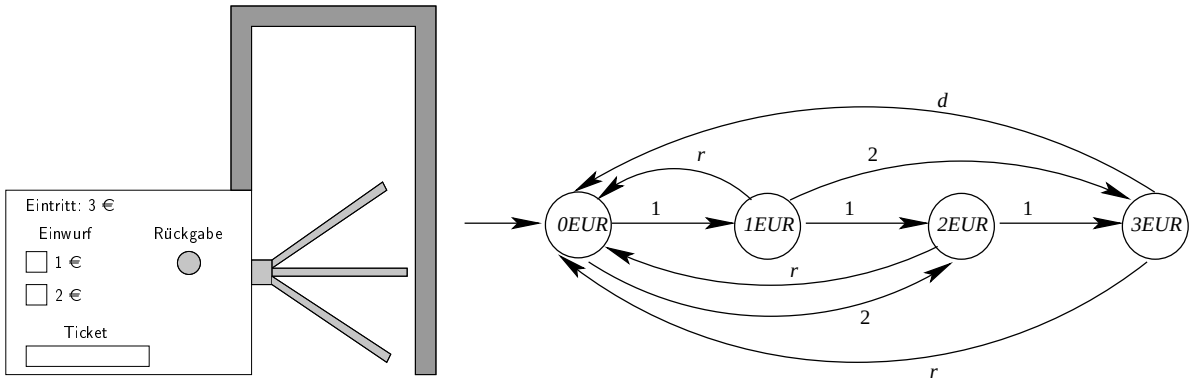
- endliche Menge von (internen) Zuständen
- Übergänge zwischen Zuständen; abhängig von der internen Struktur des Automaten und von der Eingabe

Im Prinzip sind Rechner ebenfalls endliche Automaten: Sie haben nur endlich viel Speicherplatz und daher nur eine endliche Menge möglicher Konfigurationen (Belegung der Speicherzellen). Die Konfigurationsübergänge werden bestimmt durch Verdrahtung und Eingaben (Tastatur, Peripheriegeräte).

Aber:

Wegen der extrem großen Anzahl von Zuständen sind endliche Automaten nicht die geeignete Beschreibungsebene für das Verhalten von Rechenmaschinen. Man abstrahiert daher von der Speicherplatzbeschränkung und nimmt potentiell unendlich großen Speicher an (z.B. unendliches Band bei Turingmaschine).

Beispiel: (Eintrittsautomat)
Beispiel für einen endlichen Automaten, bei dem dies die richtige Beschreibungsebene ist:
Eingabe: $1, 2, r, d$ *(d: Drehgitter dreht sich)*
Zustände: 0EUR, 1EUR, 2EUR, 3EUR



Teilweise interessiert man sich auch für Automaten mit Ausgabe (z.B. Rückgabe des Geldes, Freigeben der Sperre des Drehgitters). Wir werden diesen Aspekt hier nicht betrachten.

Bevor wir den Begriff des endlichen Automaten formal einführen können, benötigen wir noch einige grundlegende Definitionen:

Alphabet:
Eine endliche Menge von Buchstaben (z.B. $a, b, c, \ldots$) heißt Alphabet und wird meist mit Σ bezeichnet.

Wort:
Eine endliche Folge von Buchstaben $w = a_1 \ldots a_n$ mit $a_i \in \Sigma$ heißt Wort über dem Alphabet Σ.

Länge des Wortes:
$|w| = n$; z.B. $|aba| = 3$

Anzahl der Vorkommen von Buchstaben:
$|w|_a$ zählt die Vorkommen von a in w;
z.B. $|aba|_a = 2$, $|aba|_b = 1$, $|aba|_c = 0$

Leeres Wort:
ε hat Länge 0

Menge aller Wörter über Σ:
Σ^*

Menge aller nichtleeren Wörter über Σ:
$\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$

Formale Sprache:
Eine Menge L von Wörtern mit $L \subseteq \Sigma^*$ heißt formale Sprache über Σ.
Beispiele:

- Menge aller Wörter über $\{a, \ldots, z\}$, die korrekte Wörter der deutschen Sprache sind
- Menge der Wörter, die das Stichwort „theorie“ enthalten

Weiterhin benötigen wir verschiedene Operationen auf Sprachen oder Wörtern:

Boolesche Operationen:

$L_1 \cup L_2, \quad L_1 \cap L_2, \quad \overline{L_1} := \Sigma^* \setminus L_1, \quad L_1 \setminus L_2 := L_1 \cap \overline{L_2}$

Konkatenation:

$u \cdot v := uv$; z.B. $abb \cdot ab = abbab$

$L_1 \cdot L_2 := \{u \cdot v \mid (u \in L_1) \wedge (v \in L_2)\}$

Beachte: Der Konkatenationspunkt wird häufig weggelassen ($L_1 L_2$ statt $L_1 \cdot L_2$).

Kleene-Stern:

iterierte Konkatenation

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

Präfix, Suffix, Infix:

u ist Präfix von v gdw. $v = uw$ für ein $w \in \Sigma^*$.

u ist Suffix von v gdw. $v = wu$ für ein $w \in \Sigma^*$.

u ist Infix von v gdw. $v = w_1 u w_2$ für $w_1, w_2 \in \Sigma^*$.

1 Nichtdeterministische endliche Automaten

Wir betrachten zunächst den etwas allgemeineren Begriff des *Transitionssystems*.

Definition 1.1 (Transitionssystem)
Ein *Transitionssystem* ist von der Form $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$, wobei

- Q eine Menge von Zuständen ist (nicht notwendigerweise endlich!),
- Σ ein endliches Alphabet ist,
- $I \subseteq Q$ eine Menge von Anfangszuständen ist,
- $\Delta \subseteq Q \times \Sigma \times Q$ eine Übergangsrelation (Transitionsrelation) ist,
- $F \subseteq Q$ eine Menge von Endzuständen ist.

Ein *endlicher Automat* hat endlich viele Zustände und nur einen Anfangszustand.

Definition 1.2 (nichtdeterministischer endlicher Automat)
Ein Transitionssystem $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ nennt man *nichtdeterministischer endlicher Automat (NEA)*, wenn

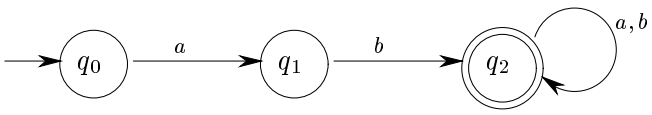
- $|Q| < \infty$
- $|I| = 1$

Anstelle von $(Q, \Sigma, \{q_0\}, \Delta, F)$ schreiben wir $(Q, \Sigma, q_0, \Delta, F)$.

Beispiel 1.3
Der NEA $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ mit den Komponenten

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{a, b\}$
- $F = \{q_2\}$
- $\Delta = \{(q_0, a, q_1), (q_1, b, q_2), (q_2, a, q_2), q_2, b, q_2)\}$

wird graphisch dargestellt als:



Definition 1.4 (Pfad)
Ein *Pfad* in einem Transitionssystem $\mathcal{A}$ ist eine Folge
$\pi = (p_0, a_1, p_1)(p_1, a_2, p_2) \dots (p_{n-1}, a_n, p_n)$ mit $(p_i, a_{i+1}, p_{i+1}) \in \Delta$ für $i = 0, \dots, n-1$.
π ist ein Pfad von p nach q, wenn $p = p_0$ und $q = p_n$.
Die *Beschriftung* des Pfades π ist das Wort $\beta(\pi) := a_1 \dots a_n$.
Die *Länge* des Pfades π ist n.
Für $n = 0$ sprechen wir vom *leeren Pfad*, welcher die Beschriftung ε hat.

Die Existenz eines Pfades in $\mathcal{A}$ von p nach q mit Beschriftung w beschreiben wir durch

$$p \xrightarrow{w}_{\mathcal{A}} q.$$

Für Mengen $Q_1, Q_2 \subseteq Q$ heißt $Q_1 \xrightarrow{w}_{\mathcal{A}} Q_2$, dass es Zustände $p_1 \in Q_1$ und $p_2 \in Q_2$ gibt mit $p_1 \xrightarrow{w}_{\mathcal{A}} p_2$.

Für den NEA aus Beispiel 1.3 gilt für alle $w \in \{a, b\}^*$: $q_0 \xrightarrow{abw}_{\mathcal{A}} q_2$.

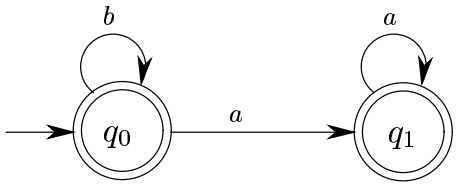
Definition 1.5 (akzeptierte Sprache)
Das Transitionssystem $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ *akzeptiert* das Wort $w \in \Sigma^*$ gdw. $I \xrightarrow{w}_{\mathcal{A}} F$ gilt. Die von $\mathcal{A}$ *akzeptierte (erkannte) Sprache* ist $L(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{A} \text{ akzeptiert } w\}$.

Für den NEA aus Beispiel 1.3 ist $L(\mathcal{A}) = \{abw \mid w \in \Sigma^*\}$, d.h. $\mathcal{A}$ akzeptiert genau die Wörter über $\Sigma = \{a, b\}$, welche mit ab beginnen.

Definition 1.6 (Erkennbarkeit einer Sprache)
Eine Sprache $L \subseteq \Sigma^*$ heißt *erkennbar*, wenn es einen NEA $\mathcal{A}$ gibt mit $L = L(\mathcal{A})$.

Da wir uns bei einem Transitionssystem i.a. nur für die akzeptierten Sprachen interessieren, bezeichnen wir zwei Transitionssysteme als äquivalent, wenn sie dieselbe Sprache akzeptieren.

Beispiel 1.7
$L = \{w \in \{a, b\}^* \mid ab \text{ ist nicht Infix von } w\}$ ist erkennbar, denn L wird z.B. von dem folgenden Automaten akzeptiert:



Für manche Konstruktionen ist es günstig, auch endliche Automaten zu betrachten, bei denen Übergänge mit Wörtern beschriftet sind.

Definition 1.8 (NEA mit Wortübergängen, ε-NEA)
Ein *NEA mit Wortübergängen* hat die Form $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$, wobei Q, Σ, q_0, 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, wobei für alle $(q, w, q') \in \Delta$ gilt: $|w| \leq 1$ (d.h. $w \in \Sigma$ oder $w = \varepsilon$).

Pfade, Pfadbeschriftungen und akzeptierte Sprache werden entsprechend wie für NEAs definiert. Zum Beispiel hat der Pfad $(q_0, ab, q_1)(q_1, \varepsilon, q_2)(q_2, bb, q_3)$ die Beschriftung $ab \cdot \varepsilon \cdot bb = abbb$.

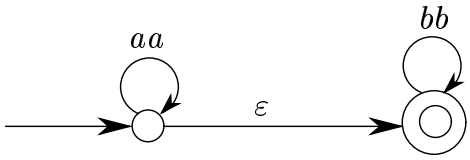
Satz 1.9
Zu jedem NEA mit Wortübergängen kann man effektiv einen äquivalenten NEA konstruieren.

Man zeigt dies mit Umweg über ε-NEAs.

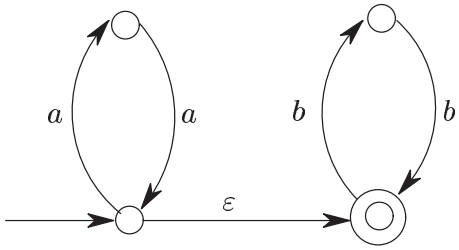
Lemma 1.10
Zu jedem NEA mit Wortübergängen kann man effektiv einen äquivalenten ε-NEA konstruieren.

Beweis. Man ersetzt jeden Wortübergang $(q, a_1 \dots a_n, q')$ mit $n > 1$ durch Buchstabenübergänge $(q, a_1, p_1), (p_1, a_2, p_2), \dots, (p_{n-1}, a_n, q')$, wobei $p_1, \dots, p_{n-1}$ jeweils neue Hilfszustände sind (die nicht zur Endzustandsmenge dazugenommen werden). Man sieht leicht, dass dies einen äquivalenten ε-NEA liefert. □

Beispiel 1.11
Der NEA mit Wortübergängen, der durch die folgende Darstellung gegeben ist:



wird überführt in einen äquivalenten ε-NEA:



Lemma 1.12
Zu jedem ε-NEA kann man effektiv einen äquivalenten NEA konstruieren.

Beweis. Der ε-NEA $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ sei gegeben. Wir konstruieren daraus einen NEA $\mathcal{A}'$ ohne ε- Übergänge wie folgt:

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

- $\Delta' := \left\{ (p, a, q) \in Q \times \Sigma \times Q \mid p \xrightarrow{a}_{\mathcal{A}} q \right\}$
- $F' := \begin{cases} F \cup \{q_0\} & \text{falls} \quad q_0 \xrightarrow{\varepsilon}_{\mathcal{A}} F \\ F & \text{sonst} \end{cases}$

Noch zu zeigen: $\mathcal{A}'$ kann effektiv bestimmt werden und $L(\mathcal{A}) = L(\mathcal{A}')$.

1. Δ' und F' können effektiv bestimmt werden:
   - $p \xrightarrow{a}_{\mathcal{A}} q$ gilt genau dann, wenn es Zustände $p', q' \in Q$ gibt, für die gilt:
   $$p \xrightarrow{\varepsilon}_{\mathcal{A}} p', \quad (p', a, q') \in \Delta, \quad q' \xrightarrow{\varepsilon}_{\mathcal{A}} q$$
   Die Übergangsrelation Δ ist eine endliche Menge. Weiterhin sind „$p \xrightarrow{\varepsilon}_{\mathcal{A}} p'$?“ sowie „$q' \xrightarrow{\varepsilon}_{\mathcal{A}} q$?“ Erreichbarkeitsprobleme in einem endlichen Graphen. (Gibt es einen Pfad von p (q') nach p' (q), dessen Übergänge nur mit ε beschriftet sind?)
   - „$q_0 \xrightarrow{\varepsilon}_{\mathcal{A}} F$?“ ist ebenfalls ein Erreichbarkeitsproblem in einem endlichen Graphen.
2. $L(\mathcal{A}) \subseteq L(\mathcal{A}')$:
   Es sei $w = a_1 \dots a_n \in L(\mathcal{A})$ und $\pi = (p_0, a_1, p_1)(p_1, a_2, p_2) \dots (p_{n-1}, a_n, p_n)$ ein Pfad in $\mathcal{A}$ mit $p_0 = q_0$, $p_n \in F$ und $a_i \in \Sigma \cup \{\varepsilon\}$. Wir betrachten die Indizes $i_1, \dots, i_m$ mit $a_{i_j} \neq \varepsilon$. Offenbar ist $w = a_{i_1} \dots a_{i_m}$.
   1. Fall: $(m = 0)$
   Es gibt keinen solchen Index, d.h. $a_i = \varepsilon$ für $i = 1, \dots, n$ und damit gilt $w = \varepsilon$. Es gilt also $q_0 = p_0 \xrightarrow{\varepsilon}_{\mathcal{A}} p_n \in F$, was $q_0 \in F'$ liefert. Es ist daher $w = \varepsilon \in L(\mathcal{A}')$.
   2. Fall: $(m > 0)$
   Nach Definition von Δ' ist nun aber
   $$(p_0, a_{i_1}, p_{i_1})(p_{i_1}, a_{i_2}, p_{i_2}) \dots (p_{i_{m-1}}, a_{i_m}, p_n)$$
   ein Pfad im NEA $\mathcal{A}'$. Aus $p_n \in F$ folgt $p_n \in F'$, was $w = a_{i_1} \dots a_{i_m} \in L(\mathcal{A}')$ zeigt.
3. $L(\mathcal{A}') \subseteq L(\mathcal{A})$:
   Es sei $w = a_1 \dots a_n \in L(\mathcal{A}')$ und
   $$(p_0, a_1, p_1)(p_1, a_2, p_2) \dots (p_{n-1}, a_n, p_n) \quad \text{mit} \quad p_0 = q_0,\ p_n \in F'$$
   ein entsprechender Pfad. Nach Definition von Δ' gilt nun in $\mathcal{A}$: $p_i \xrightarrow{a_{i+1}}_{\mathcal{A}} p_{i+1}$ für $i = 0, \dots, n-1$. Also gibt es in $\mathcal{A}$ einen Pfad von p_0 nach p_n mit Beschriftung $w = a_1 \dots a_n$.
   1. Fall: $(p_n \in F)$
   Dann folgt sofort $w \in L(\mathcal{A})$.
   2. Fall: $(p_n \in F' \setminus F$, d.h. $p_n = q_0)$
   Es gilt nun aber (nach Definition von F'): $p_n = q_0 \xrightarrow{\varepsilon}_{\mathcal{A}} \hat{p}$ für ein $\hat{p} \in F$. Daher gibt es in $\mathcal{A}$ einen Pfad von p_0 nach $\hat{p}$ mit Beschriftung w.

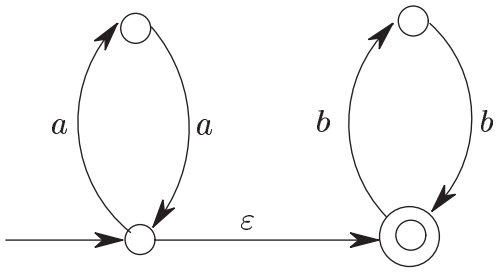
□

Lemma 1.12 kann man auch verwenden, wenn man zeigen will, dass es zu jedem *endlichen* Transitionssystem $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ einen äquivalenten NEA $\mathcal{A}'$ gibt. Man definiert für einen neuen Zustand $q_0 \notin Q$ einen ε-NEA $\mathcal{A}'' = (Q \cup \{q_0\}, \Sigma, q_0, \Delta'', F)$ mit

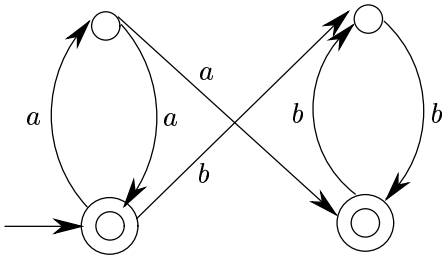
$$\Delta'' = \Delta \cup \{(q_0, \varepsilon, q) \mid q \in I\}$$

$\mathcal{A}''$ ist äquivalent zu $\mathcal{A}$. Man erhält nun $\mathcal{A}'$ aus $\mathcal{A}''$ wie im Beweis des Lemmas beschrieben.

Beispiel: (zu Lemma 1.12)
Der ε-NEA aus Beispiel 1.11



wird in folgenden NEA überführt:



2 Deterministische endliche Automaten

Die bisher betrachteten NEAs heißen nichtdeterministisch, weil es in ihnen zu einem Paar $(q, a) \in Q \times \Sigma$ mehr als einen Übergang geben kann, d.h. $q', q'' \in Q$ mit $q' \neq q''$ und $(q, a, q') \in \Delta$, $(q, a, q'') \in \Delta$.

Definition 2.1 (deterministischer endlicher Automat)
Ein NEA $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ heißt *deterministisch* (DEA), falls es für alle $q \in Q$, $a \in \Sigma$ genau ein $q' \in Q$ gibt mit $(q, a, q') \in \Delta$.

Anstelle der Übergangsrelation Δ verwenden wir dann die *Übergangsfunktion*

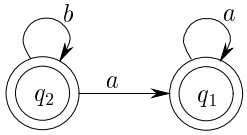
$$\delta : Q \times \Sigma \to Q$$

mit $\delta(q, a) = q'$ gdw. $(q, a, q') \in \Delta$. DEAs werden in der Form $(Q, \Sigma, q_0, \delta, F)$ geschrieben.

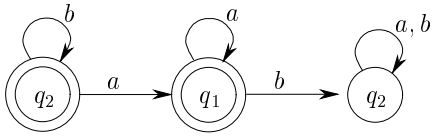
Beachte:
Zusätzlich zur *Eindeutigkeit* eines Übergangs haben wir auch die *Existenz* eines Übergangs für jedes Paar (q, a) gefordert.

In der Literatur wird für deterministische Automaten manchmal nur die Eindeutigkeit des Übergangs gefordert, d.h. für (q, a) darf es höchstens ein q' mit $(q, a, q') \in \Delta$ geben. Hingegen spricht bei Existenz aller Übergänge von *vollständigen* Automaten, d.h. wenn es für (q, a) mindestens ein q' gibt mit $(q, a, q') \in \Delta$. Bezüglich der akzeptierten Sprachklasse macht dies keinen Unterschied.

Beispiel 2.2
Der NEA



erfüllt zwar die Eindeutigkeitsanforderung, er ist aber kein DEA in unserem Sinn, da für (q_1, b) kein Übergang existiert. Einen äquivalenten DEA erhält man durch Hinzunahme eines „Papierkorbzustandes“:



Definition 2.3 (kanonische Fortsetzung von δ)
Die *kanonische Fortsetzung von* $\delta : Q \times \Sigma \to Q$ *auf eine Funktion* $\delta^* : Q \times \Sigma^* \to Q$ wird induktiv (über die Wortlänge) definiert:

- $\delta^*(q, \varepsilon) := q$
- $\delta^*(q, wa) := \delta(\delta^*(q, w), a)$

Der Einfachheit halber werden wir in Zukunft auch für Wörter w die Schreibweise $\delta(q, w)$ statt $\delta^*(q, w)$ verwenden.

Beachte:
Für einen DEA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ gilt:

a) $\delta(q, w) = q'$ gdw. q' ist der eindeutige Zustand mit $q \xrightarrow{w}_{\mathcal{A}} q'$.

b) $L(\mathcal{A}) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}$

c) $\delta(q, uv) = \delta(\delta(q, u), v)$

Wir zeigen nun, dass die bei der Definition von DEAs gemachten Einschränkungen (Existenz und Eindeutigkeit von Übergängen) keine echten Einschränkungen sind:

Satz 2.4 (Rabin/Scott)
Zu jedem NEA kann man effektiv einen äquivalenten DEA konstruieren.

Damit folgt, dass NEAs und DEAs dieselbe Klasse von Sprachen akzeptieren. 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.
Sei $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ ein NEA. Um für $w \in \Sigma^*$ zu entscheiden, ob $w \in L(\mathcal{A})$ ist, betrachtet man $\{q \in Q \mid q_0 \xrightarrow{w}_{\mathcal{A}} q\} \in 2^Q$. Es ist $w \in L(\mathcal{A})$ gdw. diese Menge enthält mindestens einen Endzustand.
Wir definieren einen DEA mit Zustandsmenge 2^Q und Übergangsfunktion δ so, dass $\delta(\{q_0\}, w) = \{q \mid q_0 \xrightarrow{w}_{\mathcal{A}} q\}$.

Beweis. Sei der NEA $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ gegeben. Der DEA $\mathcal{A}' = (2^Q, \Sigma, \{q_0\}, \delta, F')$ ist definiert durch:

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

Wir benötigen im Folgenden die

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

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

$$
\begin{aligned}
w \in L(\mathcal{A}) \quad & \underline{\text{gdw}}. \quad \exists q \in F : q_0 \xrightarrow{w}_{\mathcal{A}} q \\
& \underline{\text{gdw}}. \quad \exists q \in F : q \in \delta(\{q_0\}, w) \quad (\star) \\
& \underline{\text{gdw}}. \quad \delta(\{q_0\}, w) \cap F \neq \emptyset \\
& \underline{\text{gdw}}. \quad \delta(\{q_0\}, w) \in F' \\
& \underline{\text{gdw}}. \quad w \in L(\mathcal{A}')
\end{aligned}
$$

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

$|w| = 0$:
$\delta(\{q\}, \varepsilon) = \{q\}$. q ist der einzige Zustand, der mit ε von q aus erreichbar ist (keine ε-Übergänge!).

$|w| = n + 1$:
Für Wörter der Länge n gelte die Behauptung schon. Es ist $w = ua$ für $a \in \Sigma$ und für das Wort $u \in \Sigma^*$ der Länge n.
Nach Definition 2.3 ist

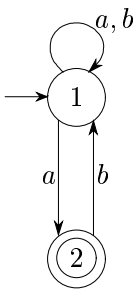
$$\delta(\{q\}, ua) = \delta(\delta(\{q\}, u), a) = \bigcup_{q' \in \delta(\{q\}, u)} \{q'' \mid (q', a, q'') \in \Delta\}$$

Nach Induktionsvoraussetzung gilt $q' \in \delta(\{q\}, u)$ $\underline{\text{gdw}}.$ $q \xrightarrow{u}_{\mathcal{A}} q'$.
Aus $q \xrightarrow{u}_{\mathcal{A}} q'$ und $(q', a, q'') \in \Delta$ ergibt sich $q \xrightarrow{ua}_{\mathcal{A}} q''$.
Umgekehrt gilt $q \xrightarrow{ua}_{\mathcal{A}} q''$ $\underline{\text{gdw}}.$ es gibt ein q' mit $q \xrightarrow{u}_{\mathcal{A}} q'$ und $(q', a, q'') \in \Delta$.
Es folgt daher $q'' \in \delta(\{q\}, ua)$ $\underline{\text{gdw}}.$ $q \xrightarrow{ua}_{\mathcal{A}} q''$.

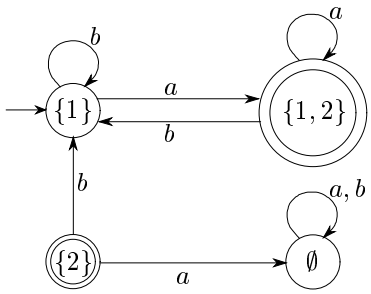
□

Beispiel 2.5

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



wird zu



Nachteilig an dieser Konstruktion ist, dass die Zustandsmenge *exponentiell* vergrößert wird. Im Allgemeinen kann man dies nicht vermeiden, in manchen Fällen kommt man aber doch mit weniger Zuständen aus. Wir werden im Folgenden ein Verfahren angeben, welches zu einem gegebenen DEA einen äquivalenten DEA mit minimaler Zustandszahl, also einen reduzierten DEA konstruiert. Dieses Verfahren besteht aus 2 Schritten:

1. Schritt: Eliminieren unerreichbarer Zustände

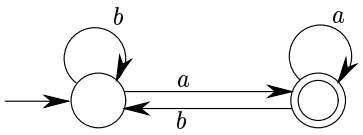
Definition 2.6 (Erreichbarkeit eines Zustandes)
Ein Zustand q des DEA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ heißt *erreichbar*, falls es ein Wort $w \in \Sigma^*$ gibt mit $\delta(q_0, w) = q$. Sonst heißt q *unerreichbar*.

Da für die akzeptierte Sprache nur Zustände wichtig sind, welche von q_0 erreicht werden, erhält man durch Weglassen unerreichbarer Zustände einen äquivalenten Automaten:

$\mathcal{A}_0 = (Q_0, \Sigma, q_0, \delta_0, F_0)$ mit

- $Q_0 = \{q \in Q \mid q \text{ ist erreichbar}\}$
- $\delta_0 = \delta\,|_{Q_0 \times \Sigma}$ Beachte: Für $q \in Q_0$ und $a \in \Sigma$ ist auch $\delta(q, a) \in Q_0$!
- $F_0 = F \cap Q_0$

Beispiel 2.5 (Fortsetzung)
Im Automaten $\mathcal{A}'$ von Beispiel 2.5 sind die Zustände $\{2\}$ und $\emptyset$ nicht erreichbar. Durch Weglassen dieser Zustände erhält man den DEA $\mathcal{A}'_0$:

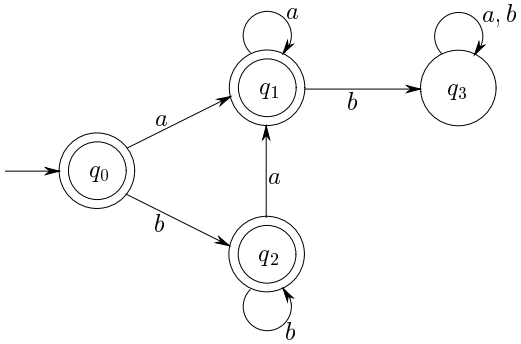


Bei der Potenzmengenkonstruktion kann man die Betrachtung unerreichbarer Elemente von 2^Q vermeiden, indem man mit $\{q_0\}$ beginnt und nur die davon erreichbaren Mengen sukzessive konstruiert.

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 2.7
Der folgende DEA erkennt dieselbe Sprache wie der DEA aus Beispiel 2.2, hat aber einen Zustand mehr. Dies kommt daher, dass q_0 und q_2 äquivalent sind.



Definition 2.8 (Äquivalenz von Zuständen)
Es sei $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ 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. $L(\mathcal{A}_q) = L(\mathcal{A}_{q'})$.

Lemma 2.9

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 effektiv bestimmt werden.*

Beweis.

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

2) lässt sich wie folgt herleiten:

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

3) Wir definieren Approximationen $\sim_k$ von $\sim_{\mathcal{A}}$:

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

Es gilt $Q \times Q \supseteq \sim_0 \supseteq \sim_1 \supseteq \sim_2 \supseteq \ldots \supseteq \sim_{\mathcal{A}}$. Da Q endlich ist, gibt es ein k mit $\sim_k = \sim_{k+1}$. Man zeigt leicht, dass dann $\sim_k = \sim_{\mathcal{A}}$ ist.

□

Die $\sim_{\mathcal{A}}$-Äquivalenzklasse eines Zustands $q \in Q$ notieren wir mit $\widetilde{q} := \{q' \in Q \mid q \sim_{\mathcal{A}} q'\}$.

Beispiel 2.7 (Fortsetzung)
Für den Automaten aus Beispiel 2.7 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 $\delta(q_1, b) \notin F$.
- $\sim_2 = \sim_1 = \sim_{\mathcal{A}}$.

Definition 2.10 (Quotientenautomat)
Der *Quotientenautomat* $\widetilde{\mathcal{A}} = (\widetilde{Q}, \Sigma, \widetilde{q}_0, \widetilde{\delta}, \widetilde{F})$ *zu* $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ ist definiert durch:

- $\widetilde{Q} := \{\widetilde{q} \mid q \in Q\}$
- $\widetilde{\delta}(\widetilde{q}, a) := \widetilde{\delta(q, a)}$ (repräsentantenunabhängig wegen Lemma 2.9)
- $\widetilde{F} := \{\widetilde{q} \mid q \in F\}$

Lemma 2.11
$\widetilde{\mathcal{A}}$ *ist äquivalent zu* $\mathcal{A}$.

Beweis.

$$\begin{aligned} w \in L(\mathcal{A}) \quad & \underline{\text{gdw}}. \quad \delta(q_0, w) \in F \\ & \underline{\text{gdw}}. \quad \widetilde{\delta(q_0, w)} \in \widetilde{F} \\ & \underline{\text{gdw}}. \quad \widetilde{\delta}(\widetilde{q}_0, w) \in \widetilde{F} \\ & \underline{\text{gdw}}. \quad w \in L(\widetilde{\mathcal{A}}) \end{aligned}$$

□

Definition 2.12 (reduzierter Automat zu einem DEA)
Für einen DEA $\mathcal{A}$ bezeichnet $\mathcal{A}_{red} := \widetilde{\mathcal{A}}_0$ den *reduzierten Automaten*, den man aus $\mathcal{A}$ durch Eliminieren unerreichbarer Zustände und Zusammenfassen äquivalenter Zustände erhält.

Wir werden im Folgenden zeigen:

- $\mathcal{A}_{red}$ kann nicht weiter vereinfacht werden, d.h. er ist der kleinste DEA, mit dem man $L(\mathcal{A})$ akzeptieren kann.
- $\mathcal{A}_{red}$ hängt nur von $L(\mathcal{A})$ und nicht von $\mathcal{A}$ ab, d.h. gilt $L(\mathcal{A}) = L(\mathcal{B})$, so auch $\mathcal{A}_{red} \simeq \mathcal{B}_{red}$ (gleich bis auf Zustandsumbenennung, isomorph).

Dazu konstruieren wir zu einer erkennbaren Sprache L einen „kanonischen“ Automaten $\mathcal{A}_L$ und zeigen, dass jeder reduzierte Automat, der L akzeptiert, isomorph zu $\mathcal{A}_L$ ist.

Definition 2.13 (Nerode-Rechtskongruenz)
Es sei $L \subseteq \Sigma^*$ eine beliebige Sprache. Für $u, v \in \Sigma^*$ definieren wir:
$u \simeq_L v \quad \underline{\text{gdw}}. \quad \forall w \in \Sigma^* : uw \in L \Leftrightarrow vw \in L.$

Beispiel 2.14
Wir betrachten die Sprache

$$L = \{w \in \{a, b\}^* \mid ab \text{ ist nicht Infix von } w\}$$

(vgl. Beispiele 1.7, 2.2, 2.7)

- Es gilt:
  $\varepsilon \simeq_L b$: $\quad \forall w : \varepsilon w \in L$ $\quad$ gdw. $\quad w \in L$
  $\quad$ gdw. $\quad w$ enthält ab nicht
  $\quad$ gdw. $\quad bw$ enthält ab nicht
  $\quad$ gdw. $\quad bw \in L$
- $\varepsilon \not\simeq_L a$: $\quad \varepsilon b \in L$, aber $a \cdot b \notin L$

Lemma 2.15 (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:

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

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

4) Ist $L = L(\mathcal{A})$ für einen DEA $\mathcal{A}$, so ist die Anzahl der $\simeq_L$-Klassen $\leq$ der Zustandszahl von $\mathcal{A}$. (Die Anzahl der $\simeq_L$-Klassen heißt Index von $\simeq_L$.)

Beweis.

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

2) Damit $uw \simeq_L vw$ gilt, muss für alle $w' \in \Sigma^*$ gelten:
   $(\star) \quad uww' \in L \Leftrightarrow vww' \in L$
   Nun ist aber $ww' \in \Sigma^*$ und daher folgt $(\star)$ aus $u \simeq_L v$.

3) Zeige $L = \bigcup_{u \in L} [u]$.
   „$\subseteq$“: klar, da $u \in [u]$.
   „$\supseteq$“: Sei $u \in L$ und $v \in [u]$.
   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 $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ ein DEA mit $L = L(\mathcal{A})$.
   Hilfsaussage: Aus $\delta(q_0, u) = \delta(q_0, v)$ folgt $u \simeq_L v$, denn:

   $\forall w : uw \in L$ $\quad$ gdw. $\quad \delta(q_0, uw) \in F$
   $\quad$ gdw. $\quad \delta(\delta(q_0, u), w) \in F$
   $\quad$ gdw. $\quad \delta(\delta(q_0, v), w) \in F$
   $\quad$ gdw. $\quad \delta(q_0, vw) \in F$
   $\quad$ gdw. $\quad vw \in L$

   Also gibt es maximal so viele verschiedene Klassen, wie es verschiedene Zustände gibt (Schubfachprinzip).

□

Beispiel 2.14 (Fortsetzung)
$\simeq_L$ hat drei Klassen:

- $[\varepsilon] = \{b\}^*$
- $[a] = \{b\}^* \cdot \{a\}^+$
- $[ab] = \Sigma^* \cdot \{ab\} \cdot \Sigma^*$

Man kann die $\simeq_L$-Klassen nun als Zustände eines Automaten für L verwenden.

Definition 2.16 (Transitionssystem $\mathcal{A}_L$ zu einer Sprache L)
Zu $L \subseteq \Sigma^*$ ist das Transitionssystem $\mathcal{A}_L := (Q', \Sigma, q_0', \delta', F')$ definiert durch:

- $Q' := \{[u] \mid u \in \Sigma^*\}$
- $q_0' := [\varepsilon]$
- $\delta'([u], a) := [ua]$ (repräsentantenunabhängig wegen Lemma 2.15, 2))
- $F' := \{[u] \mid u \in L\}$

Lemma 2.17
Hat $\simeq_L$ endlichen Index, so ist $\mathcal{A}_L$ ein DEA mit $L = L(\mathcal{A}_L)$.

Beweis. Hat $\simeq_L$ endlich viele Klassen, so hat $\mathcal{A}_L$ endlich viele Zustände. Außerdem gilt:

$$\begin{aligned} L(\mathcal{A}_L) &= \{u \mid \delta'(q_0', u) \in F'\} \\ &= \{u \mid \delta'([\varepsilon], u) \in F'\} \\ &= \{u \mid [u] \in F'\} \qquad (\text{wegen } \delta'([u], v) = [uv]) \\ &= \{u \mid u \in L\} \\ &= L \end{aligned}$$

□

Satz 2.18 (Satz von Nerode)
Eine Sprache L ist erkennbar gdw. $\simeq_L$ hat endlichen Index (d.h. endlich viele Klassen).

Beweis.
„⇒“: Ergibt sich unmittelbar aus Lemma 2.15, 4).
„⇐“: Ergibt sich unmittelbar aus Lemma 2.17, da $\mathcal{A}_L$ DEA ist, der L akzeptiert.

□

Beispiel 2.19 (nichterkennbare Sprache)
Die Sprache $L = \{a^n b^n \mid n \geq 0\}$ ist nicht erkennbar, 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 werden im nächsten Abschnitt noch eine weitere Möglichkeit sehen, wie man für eine Sprache zeigen kann, dass sie *nicht* regulär ist.

Zunächst untersuchen wir aber den Zusammenhang zwischen *reduzierten Automaten* und der *Nerode-Rechtskongruenz*.

Definition 2.20 (isomorph)
Zwei DEAs $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ und $\mathcal{A}' = (Q', \Sigma, q_0', \delta', F')$ sind *isomorph* ($\mathcal{A} \simeq \mathcal{A}'$) gdw. es eine Bijektion $\mathrm{f} : Q \to Q'$ gibt mit:

- $\mathrm{f}(q_0) = q_0'$
- $\mathrm{f}(F) = F'$, wobei $\mathrm{f}(F) := \{\mathrm{f}(q) \mid q \in F\}$
- $\mathrm{f}(\delta(q, a)) = \delta'(\mathrm{f}(q), a)$ für alle $q \in Q$, $a \in \Sigma$

Lemma 2.21
$\mathcal{A} \simeq \mathcal{A}' \;\Rightarrow\; L(\mathcal{A}) = L(\mathcal{A}')$

Beweis. Es sei $\mathrm{f} : Q \to Q'$ der Isomorphismus. Durch Induktion über $|w|$ zeigt man leicht, dass $\mathrm{f}(\delta(q, w)) = \delta'(\mathrm{f}(q), w)$. Daher gilt:

$$\begin{aligned}
w \in L(\mathcal{A}) \quad & \text{gdw.} \quad \delta(q_0, w) \in F \\
& \text{gdw.} \quad \mathrm{f}(\delta(q_0, w)) \in F' \qquad (\text{wegen } F' = \mathrm{f}(F)) \\
& \text{gdw.} \quad \delta'(\mathrm{f}(q_0), w) \in F' \\
& \text{gdw.} \quad \delta'(q_0', w) \in F' \qquad (\text{wegen } q_0' = \mathrm{f}(q_0)) \\
& \text{gdw.} \quad w \in L(\mathcal{A}')
\end{aligned}$$

□

Wir können nun Minimalität und Eindeutigkeit des reduzierten Automaten zeigen.

Satz 2.22
Es sei L eine erkennbare Sprache. Dann gilt:

1) $\mathcal{A}_L$ hat minimale Zustandszahl unter allen DEAs, welche L erkennen.

2) Ist $\mathcal{A}$ ein DEA mit $L(\mathcal{A}) = L$, so ist der reduzierte Automat $\mathcal{A}_{red} := \widetilde{\mathcal{A}}_0$ isomorph zu $\mathcal{A}_L$.

Beweis.

1) Mit Satz 2.18 hat $\simeq_L$ endlichen Index. Daher ist mit Lemma 2.17 $\mathcal{A}_L$ ein DEA für L. Mit Lemma 2.15, 4) hat jeder DEA, der L akzeptiert, mindestens soviele Zustände, wie $\simeq_L$ Klassen (d.h. wie $\mathcal{A}_L$ Zustände) hat.

2) Es sei $\mathcal{A}_{red} = (Q, \Sigma, q_0, \delta, F)$ und $\mathcal{A}_L = (Q', \Sigma, q_0', \delta', F')$. Wir definieren eine Funktion $\mathrm{f} : Q \to Q'$ und zeigen, dass sie ein Isomorphismus ist. Für $q \in Q$ existiert (mindestens) ein Wort $w_q \in \Sigma^*$ mit $\delta(q_0, w_q) = q$, da in $\mathcal{A}_{red}$ alle Zustände erreichbar sind. O.B.d.A. sei $w_{q_0} = \varepsilon$. Wir definieren $\mathrm{f}(q) := [w_q]$.

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

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

   nicht gilt.

Das heißt aber, dass

$$\delta(q_0, w_p w) \in F \Leftrightarrow \delta(q_0, w_q w) \in F$$

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

II) f ist surjektiv:
Folgt aus Injektivität und $|Q| \geq |Q'|$ (Aussage 1) des Lemmas).

III) $\mathrm{f}(q_0) = q_0'$:
Da $w_{q_0} = \varepsilon$ und $q_0' = [\varepsilon]$.

IV) $\mathrm{f}(F) = F'$:
$q \in F$ gdw. $\delta(q_0, w_q) = q \in F$ gdw. $w_q \in L$ gdw. $[w_q] \in F'$

V) $\mathrm{f}(\delta(q, a)) = \delta'(\mathrm{f}(q), a)$:
Es sei $\delta(q, a) =: p$. Dann ist $\mathrm{f}(\delta(q, a)) = [w_p]$.

Außerdem ist $\delta'(\mathrm{f}(q), a) = \delta'([w_q], a) = [w_q a]$.
Es bleibt also $[w_p] = [w_q a]$ zu zeigen, d.h. $\forall w : w_p w \in L$ gdw. $w_q a w \in L$.
Dies ist offenbar dann der Fall, wenn $\delta(q_0, w_p) = \delta(q_0, w_q a)$ ist:
$\delta(q_0, w_q a) = \delta(\delta(q_0, w_q), a) = \delta(q, a) = p = \delta(q_0, w_p)$

□

Im Prinzip liefert dieser Satz eine Methode, um von zwei Automaten zu entscheiden, ob sie dieselbe Sprache akzeptieren:

Korollar 2.23
Es seien $\mathcal{A}$ und $\mathcal{A}'$ DEAs. Dann gilt: $L(\mathcal{A}) = L(\mathcal{A}')$ gdw. $\mathcal{A}_{red} \simeq \mathcal{A}'_{red}$.

Man kann die reduzierten Automaten wie beschrieben konstruieren. Für gegebene Automaten kann man feststellen, ob sie isomorph sind (teste alle Bijektionen). Hat man NEAs gegeben, so kann man diese zuerst deterministisch machen und dann das Korollar anwenden.

3 Nachweis der Nichterkennbarkeit

Um nachzuweisen, dass eine gegebene Sprache erkennbar ist, genügt es, einen endlichen Automaten (DEA oder NEA) dafür anzugeben. Der Nachweis, dass eine Sprache nicht erkennbar ist, gestaltet sich schwieriger: Es genügt ja nicht zu sagen, dass man keinen Automaten dafür gefunden hat. (Es gibt unendlich viele Automaten).

Wir haben bereits gesehen, dass man den Satz von Nerode (Satz 2.18) dazu verwenden kann. Ein anderes Hilfsmittel hierfür ist das Pumping-Lemma:

Lemma 3.1 (Pumping-Lemma, einfache Version)
Es sei L eine erkennbare 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 $w = xyz$ mit

- $y \neq \varepsilon$
- $xy^k z \in L$ *für alle* $k \geq 0$.

Beweis. Es sei $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ ein NEA mit $L(\mathcal{A}) = L$. Wir wählen $n_0 = |Q|$. Für jedes Wort $w = a_1 \dots a_m \in L$ existiert ein Pfad $(p_0, a_1, p_1)(p_1, a_2, p_2) \dots (p_{m-1}, a_m, p_m)$ in $\mathcal{A}$ mit $p_0 = q_0$ und $p_m \in F$.
Ist $m \geq n_0$, so können die $m+1$ Zustände $p_0, \dots, p_m$ nicht alle verschieden sein, da $|Q| = n_0 < m+1$. Es gibt daher ein $i < j$ mit $p_i = p_j$.
Für $x := a_1 \dots a_i, \quad y := a_{i+1} \dots a_j, \quad z := a_{j+1} \dots a_m$ gilt daher $y \neq \varepsilon$ (da $i < j$) und $q_0 = p_0 \xrightarrow{x}_{\mathcal{A}} p_i \xrightarrow{y}_{\mathcal{A}} p_i = p_j \xrightarrow{z}_{\mathcal{A}} p_m \in F$.
Folglich gilt für alle $k \geq 0$ auch $p_i \xrightarrow{y^k}_{\mathcal{A}} p_i$, was $xy^k z \in L$ zeigt. □

Wir zeigen mit Hilfe dieses Lemmas (nochmals), dass die Sprache $\{a^n b^n \mid n \geq 0\}$ (vgl. Beispiel 2.19) nicht erkennbar ist.

Beispiel:
$L = \{a^n b^n \mid n \geq 0\}$ ist *nicht* erkennbar.

Beweis. Angenommen, L ist erkennbar. Dann gibt es eine Zahl n_0 mit den in Lemma 3.1 beschriebenen Eigenschaften. Es sei nun n so, dass $2n \geq n_0$ ist. Da $a^n b^n \in L$ ist, hat es eine Zerlegung $a^n b^n = xyz$ mit $|y| \geq 1$ und $xy^k z \in L$ für alle $k \geq 0$.

1. Fall: y liegt ganz in a^n.
D.h. $x = a^{n_1},\ y = a^{n_2},\ z = a^{n_3} b^n$ mit $n_2 > 0$ und $n = n_1 + n_2 + n_3$. Damit ist aber $xz = xy^0 z = a^{n_1+n_3} b^n \notin L$, da $n_1 + n_3 < n$. Widerspruch.

2. Fall: y liegt ganz in b^n.
Führt entsprechend zu einem Widerspruch.

3. Fall: y enthält as und bs.
D.h. $x = a^{n_1},\ y = a^{n_2} b^{n_1'},\ z = b^{n_2'}$ mit $n_2 \neq 0 \neq n_1'$. Dann ist aber

$$xyyz = a^{n_1} a^{n_2} b^{n_1'} a^{n_2} b^{n_1'} b^{n_2'} \notin L$$

(da nach bs nochmal as kommen). Widerspruch.

In allen drei Fällen haben wir also einen Widerspruch erhalten, d.h. die Annahme „L ist erkennbar“ war falsch. □

Als eine weitere Konsequenz von Lemma 3.1 erhält man, dass das Leerheitsproblem für erkennbare Sprachen entscheidbar ist.

Satz 3.2
Es sei L eine erkennbare Sprache (gegeben durch NEA oder DEA). Dann kann man effektiv entscheiden, ob $L = \emptyset$ oder nicht.

Beweis. Es sei L erkennbar und n_0 die zugehörige Zahl aus Lemma 3.1. Dann gilt:

$$(\star) \qquad L \neq \emptyset \ \text{ gdw. } \ \exists w \in L \text{ mit } |w| < n_0$$

Um $L = \emptyset$ zu entscheiden, muss man also nur für die *endlich vielen* Wörter $w \in \Sigma^*$ der Länge $< n_0$ entscheiden, ob w zu L gehört. Dies kann man für jedes einzelne Wort (z.B. durch Eingabe in den zugehörigen DEA) einfach entscheiden (vgl. Wortproblem, Satz 4.4).

Beweis von $(\star)$:

„⇐“: trivial

„⇒“: Es sei $L \neq \emptyset$. Wähle ein Wort w kürzester Länge in L. Wäre $|w| \geq n_0$, so müsste es eine Zerlegung $w = xyz,\ y \neq \varepsilon$ geben mit $xy^0z = xz \in L$. Dies ist ein Widerspruch zur Minimalität von w. □

Mit Hilfe der einfachen Variante des Pumping-Lemmas gelingt es nicht immer, Nichterkennbarkeit nachzuweisen.

Beispiel 3.3
Ist $L = \{a^n b^m \mid n \neq m\}$ erkennbar? Versucht man, Nichterkennbarkeit mit Lemma 3.1 zu zeigen, so scheitert man, da das Lemma für L zutrifft:
Wähle $n_0 := 3$. Es sei nun $w \in L$ mit $|w| \geq 3$, d.h. $w = a^n b^m$, $n \neq m$ und $n + m \geq 3$.

1. Fall: $n > m$
D.h. $n = m + i$ für $i \geq 1$.

1.1.: $i > 1$, d.h. $n - 1 > m$.
Für $x = \varepsilon,\ y = a,\ z = a^{n-1}b^m$ gilt für alle $k \geq 0 : xy^kz = a^k a^{n-1} b^m \in L$, da $k + n - 1 \geq n - 1 > m$ (wegen $n - m = i > 1$).

1.2.: $i = 1$, d.h. $n = m + 1$.
Wegen $n + m \geq 3$ folgt $n = m + 1 \geq 2$.
Für $x = \varepsilon$, $y = a^2,\ z = a^{n-2}b^m$ gilt
a) $xy^0z \in L$, da $n - 2 = m - 1 \neq m$
b) $xy^kz \in L$ für $k \geq 1$, da $(n - 2) + k \cdot 2 \geq n > m$

2. Fall: $n < m$
kann entsprechend behandelt werden.

Trotzdem ist $L = \{a^n b^m \mid n \neq m\}$ nicht erkennbar, was man mit der folgenden verschärften Variante des Pumping-Lemmas nachweisen kann.

Lemma 3.4 (Pumping-Lemma, verschärfte Variante)
Es sei L erkennbar. Dann gibt es eine natürliche Zahl $n_0 \geq 1$, so dass gilt:
Für alle Wörter u, v, w mit $uvw \in L$ und $|v| \geq n_0$ gibt es eine Zerlegung $v = xyz$ mit

- $y \neq \varepsilon$
- $uxy^k zw \in L$ *für alle* $k \geq 0$

Beweis. Es sei wieder $n_0 := |Q|$, wobei Q die Zustände eines NEA $\mathcal{A}$ für L sind. Ist $uvw \in L$, so gibt es Zustände $p, q, f \in Q$ mit

$$q_0 \xrightarrow{u}_{\mathcal{A}} p \xrightarrow{v}_{\mathcal{A}} q \xrightarrow{w}_{\mathcal{A}} f \in F$$

Auf dem Pfad von p nach q liegen $|v| + 1 > n_0$ Zustände, also müssen zwei davon gleich sein. Jetzt kann man wie im Beweis von Lemma 3.1 weitermachen. □

Was ist der Vorteil dieses Lemmas beim Nachweis der Nichterkennbarkeit? Man weiß, dass man bereits beliebig lange Teilstücke zerlegen kann. Dadurch kann man das y geeignet positionieren (im Beispiel 3.3 im kürzeren Block).

Beispiel 3.3 (Fortsetzung)
$L = \{a^n b^m \mid n \neq m\}$ ist *nicht* erkennbar.

Beweis. Angenommen, L ist doch erkennbar; dann gibt es $n_0 \geq 1$, das die in Lemma 3.4 geforderten Eigenschaften hat. Wir betrachten nun die Wörter

$$u := \varepsilon,\ v := a^{n_0},\ w := b^{n_0! + n_0}$$

Offenbar ist $uvw = a^{n_0} b^{n_0! + n_0} \in L$. Mit Lemma 3.4 gibt es eine Zerlegung $v = xyz$ mit $y \neq \varepsilon$ und $uxy^k zw \in L$ für alle $k \geq 0$. Es sei

$$x = a^{n_1},\ y = a^{n_2},\ z = a^{n_3},\ n_1 + n_2 + n_3 = n_0,\ n_2 > 0$$

Offenbar existiert ein l mit $n_2 \cdot l = n_0!$ (da $0 < n_2 \leq n_0$). Es ist nun aber

$$n_1 + (l+1) \cdot n_2 + n_3 = n_0 + n_0!$$

was $uxy^{l+1}zw \notin L$ liefert (Widerspruch). □

4 Abschlusseigenschaften und Entscheidungsprobleme

Wir zeigen zunächst, dass die Klasse der erkennbaren Sprachen unter den Booleschen Operationen sowie Konkatenation und Kleene-Stern abgeschlossen ist.

Satz 4.1 (Abschluss erkennbarer Sprachen)
Sind L_1 und L_2 erkennbar, so sind auch

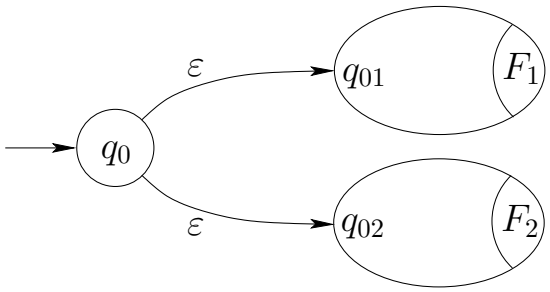
- $L_1 \cup L_2$ *(Vereinigung)*
- $\overline{L_1}$ *(Komplement)*
- $L_1 \cap L_2$ *(Durchschnitt)*
- $L_1 \setminus L_2$ *(Differenz)*
- $L_1 \cdot L_2$ *(Konkatenation)*
- L_1^* *(Kleene-Stern)*

erkennbar.

Beweis. Es seien $\mathcal{A}_i = (Q_i, \Sigma, q_{0i}, \Delta_i, F_i)$ 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 akzeptiert $L_1 \cup L_2$:
$\mathcal{A} := (Q_1 \cup Q_2 \cup \{q_0\}, \Sigma, q_0, \Delta, F_1 \cup F_2)$, wobei
   - $q_0 \notin Q_1 \cup Q_2$ und
   - $\Delta := \Delta_1 \cup \Delta_2 \cup \{(q_0, \varepsilon, q_{01}), (q_0, \varepsilon, q_{02})\}$.

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



   Mit Lemma 1.12 gibt es zu $\mathcal{A}$ einen äquivalenten NEA.

2) **Abschluss unter Komplement:**
Einen DEA für $\overline{L_1}$ erhält man wie folgt:
Zunächst verwendet man die Potenzmengenkonstruktion, um zu $\mathcal{A}_1$ einen äquivalenten DEA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ zu konstruieren. Der DEA für $\overline{L_1}$ ist nun $\overline{\mathcal{A}} := (Q, \Sigma, q_0, \delta, Q \setminus F)$. Es gilt nämlich:

$$\begin{array}{lll} w \in \overline{L_1} & \underline{\text{gdw}}. & w \notin L(\mathcal{A}_1) \\ & \underline{\text{gdw}}. & w \notin L(\mathcal{A}) \\ & \underline{\text{gdw}}. & \delta(q_0, w) \notin F \\ & \underline{\text{gdw}}. & \delta(q_0, w) \in Q \setminus F \\ & \underline{\text{gdw}}. & w \in L(\overline{\mathcal{A}}) \end{array}$$

Beachte: Man darf dies nicht direkt mit dem NEA machen!

3) **Abschluss unter Durchschnitt:**
Durch Ausnutzen von $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ folgt 3) aus 1) und 2).
Da die Potenzmengenkonstruktion aber sehr aufwendig sein kann, ist es günstiger, direkt einen NEA für $L_1 \cap L_2$ zu konstruieren, den sogenannten *Produktautomaten*:

$$\mathcal{A} := (Q_1 \times Q_2, \Sigma, (q_{01}, q_{02}), \Delta, F_1 \times F_2)$$

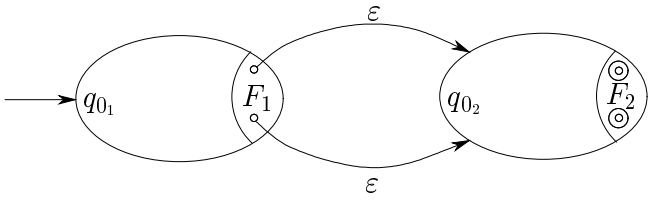
mit

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

Ein Übergang in $\mathcal{A}$ ist also genau dann möglich, wenn der entsprechende Übergang in $\mathcal{A}_1$ und $\mathcal{A}_2$ möglich ist. Daraus ergibt sich leicht $L(\mathcal{A}) = L_1 \cap L_2$.

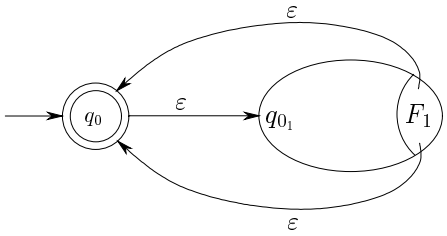
4) **Abschluss unter Differenz:**
Folgt aus 1) und 2), da
$L_1 \setminus L_2 = L_1 \cap \overline{L_2}$.

5) **Abschluss unter Konkatenation:**
Der folgende ε-NEA akzeptiert $L_1 \cdot L_2$:
$\mathcal{A} := (Q_1 \cup Q_2, \Sigma, q_{01}, \Delta, F_2)$, wobei
$\Delta := \Delta_1 \cup \Delta_2 \cup \{(f, \varepsilon, q_{02}) \mid f \in F_1\}$



6) **Abschluss unter Kleene-Stern:**
Der folgende ε-NEA akzeptiert L_1^*:
$\mathcal{A} := (Q_1 \cup \{q_0\}, \Sigma, q_0, \Delta, \{q_0\})$, wobei

- $q_0 \notin Q_1$
- $\Delta := \Delta_1 \cup \{(f, \varepsilon, q_0) \mid f \in F_1\} \cup \{(q_0, \varepsilon, q_{01})\}$.



□

Beachte:

Alle angegebenen Konstruktionen sind effektiv. Die Automaten für die Sprachen $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 \cdot L_2$ und L_1^* sind polynomiell in der Größe der Automaten für L_1, L_2. Beim Komplement kann die Konstruktion exponentiell sein, wenn man von einem NEA ausgeht.

Man kann derartige Abschlusseigenschaften auch dazu verwenden, Nichterkennbarkeit einer Sprache L nachzuweisen.

Beispiel 4.2

$L := \{a^n b^m \mid n \neq m\}$ ist *nicht* erkennbar (vgl. Beispiel 3.3). Anstatt dies direkt mit Lemma 3.4 zu zeigen, kann man auch verwenden, dass bereits bekannt ist, dass die Sprache $L' := \{a^n b^n \mid n \geq 0\}$ *nicht* erkennbar ist. Wäre nämlich L erkennbar, so auch $L' = \overline{L} \cap \{a\}^* \cdot \{b\}^*$. Da wir schon wissen, dass L' nicht erkennbar ist, kann auch L nicht erkennbar sein.

Wir betrachten nun drei Probleme im Zusammenhang mit erkennbaren Sprachen, für die wir Entscheidbarkeit und Komplexität untersuchen, und zwar:

- das *Leerheitsproblem*
- das *Wortproblem* und
- das *Äquivalenzproblem.*

Dabei gehen wir davon aus, dass die erkennbare Sprache durch einen DEA oder NEA gegeben ist. Bezüglich der Entscheidbarkeit dieser Probleme macht es keinen Unterschied, ob man einen DEA oder einen NEA gegeben hat, da man aus einem NEA effektiv einen äquivalenten DEA konstruieren kann (Satz 2.4). Bezüglich der Komplexität kann der Unterschied aber relevant sein, da der Übergang NEA $\longrightarrow$ DEA exponentiell sein kann.

Leerheitsproblem:

Geg.: erkennbare Sprache L (durch DEA oder NEA)
Frage: Ist $L \neq \emptyset$?

Wir wissen bereits (Satz 3.2), dass das Problem entscheidbar ist. Allerdings ist das im Beweis des Satzes beschriebene Entscheidungsverfahren viel zu aufwendig (*exponentiell* viele Wörter der Länge $< n_0$, falls $|\Sigma| > 1$).

Satz 4.3

Es sei $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ ein NEA. Dann kann man das Problem „$L(\mathcal{A}) \neq \emptyset$?“ in der Zeit $O(|Q| + |\Delta|)$ entscheiden.

Beweis. Man kann $\mathcal{A}$ als gerichteten Graphen $G = (Q, E)$ auffassen mit

$$E := \{(q_1, q_2) \mid (q_1, a, q_2) \in \Delta \text{ für ein } a \in \Sigma\}$$

Dann gilt: $L(\mathcal{A}) \neq \emptyset$ gdw. in der von q_0 aus erreichbaren Knotenmenge befindet sich ein Endzustand. Die von q_0 aus erreichbaren Knoten kann man mit Aufwand $O(|Q| + |E|)$ berechnen (vgl. Vorlesung „Algorithmen und Datenstrukturen“). Insbesondere ist also das Leerheitsproblem für erkennbare Sprachen mit polynomiellem Aufwand lösbar. □

Wortproblem:

Geg.: erkennbare Sprache L, Wort $w \in \Sigma^*$
Frage: Gilt $w \in L$?

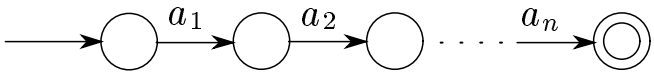
Ist $L = L(\mathcal{A})$ für einen DEA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$, so kann man einfach, beginnend mit q_0, durch Anwendung von δ berechnen, zu welchem Zustand in $\mathcal{A}$ man mit w kommt und prüfen, ob dies ein Endzustand ist. Nimmt man Σ als konstant an, so benötigt eine Anwendung von δ nur konstante Zeit. Dies liefert:

Satz 4.4
Es sei $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ ein DEA *und $w \in \Sigma^*$. Dann kann man „$w \in L(\mathcal{A})$?“ in der Zeit $O(|w|)$ entscheiden.*

Für einen NEA ist dies nicht so einfach, da man ja verschiedene mit w beschriftete Pfade haben kann und man diese (im schlimmsten Fall) alle betrachten muss, um festzustellen, ob einer davon mit einem Endzustand aufhört.

Satz 4.5
Es sei $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ ein NEA *und $w \in \Sigma^*$. Dann kann man „$w \in L(\mathcal{A})$?“ in der Zeit $O(|w| \cdot (|Q| + |\Delta|))$ entscheiden.*

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



Dieser Automat hat $|w|+1$ Zustände. Offenbar ist $w \in L(\mathcal{A})$ gdw. $L(\mathcal{A}) \cap L(\mathcal{A}_w) \neq \emptyset$. Wie groß ist nun der Produktautomat zu $\mathcal{A}$ und $\mathcal{A}_w$?

Zustände: $|Q| \cdot (|w| + 1)$

Übergänge: Für jeden Übergang  gibt es maximal $|\Delta|$ mögliche Übergänge in $\mathcal{A}$.
Also maximal $|w| \cdot |\Delta|$ viele Übergänge im Produktautomaten.

Nach Satz 4.3 ist daher der Aufwand zum Testen von $L(\mathcal{A}) \cap L(\mathcal{A}_w) \neq \emptyset$:

$$O(|Q| \cdot (|w| + 1) + |w| \cdot |\Delta|) = O(|w| \cdot (|Q| + |\Delta|)) \qquad \square$$

Äquivalenzproblem:

Geg.: erkennbare Sprachen L_1, L_2
Frage: Gilt $L_1 = L_2$?

Wie bereits erwähnt, kann man das Äquivalenzproblem auf das Leerheitsproblem *reduzieren*:

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

Satz 4.6
Das Äquivalenzproblem ist für erkennbare Sprachen entscheidbar. Sind die Sprachen durch DEAs gegeben, so ist dies in polynomieller Zeit möglich.

Beweis. Wir haben gesehen, dass die Automatenkonstruktionen, welche Abschluss unter Vereinigung, Durchschnitt und Komplement zeigen, effektiv sind. Daraus ergibt sich direkt die Entscheidbarkeit des Äquivalenzproblems (auch bei NEAs). Die Konstruktionen für Vereinigung und Durchschnitt sind polynomiell. Beim Komplement ist dies nur dann der Fall, wenn bereits DEAs vorliegen. □

Wir werden später sehen, dass sich der exponentielle Zeitaufwand für die Potenzmengenkonstruktion bei NEAs (wahrscheinlich) nicht vermeiden lässt: das Äquivalenzproblem für NEAs ist $PSPACE$-vollständig (schlimmer als NP).

5 Reguläre Ausdrücke und Sprachen

Wir haben bisher *verschiedene äquivalente Charakterisierungen* der Klasse der erkennbaren Sprachen mit Hilfe von Transitionssystemen und Rechtskongruenzen gesehen:

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

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

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

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

(4) $L = L(\mathcal{A})$ für ein endliches Transitionssystem $\mathcal{A}$.

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

(6) Die Nerode-Rechtskongruenz $\simeq_L$ hat endlichen Index.

Im folgenden betrachten wir eine weitere Charakterisierung mit Hilfe *regulärer Ausdrücke*.

Definition 5.1 (Syntax regulärer Ausdrücke)
Es 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 $r, s \in Reg_\Sigma$, so auch $(r + s), (r \cdot s), r^* \in Reg_\Sigma$.

Beispiel 5.2
$((a \cdot b^*) + \emptyset^*)^* \in Reg_\Sigma$ für $\Sigma = \{a, b\}$

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 5.2 kann also geschrieben werden als $(ab^* + \emptyset^*)^*$.

Jedem regulären Ausdruck r über Σ wird eine formale Sprache $L(r)$ zugeordnet.

Definition 5.3 (Semantik regulärer Ausdrücke)
Die durch den regulären Ausdruck r definierte Sprache $L(r)$ ist induktiv definiert:

- $L(\emptyset) := \emptyset, \quad L(\varepsilon) := \{\varepsilon\}, \quad L(a) := \{a\}$
- $L(r + s) := L(r) \cup L(s), \quad L(r \cdot s) := L(r) \cdot L(s), \quad L(r^*) := L(r)^*$

Eine Sprache $L \subseteq \Sigma^*$ heißt *regulär*, falls es ein $r \in Reg_\Sigma$ gibt mit $L = L(r)$.

Beispiel:

- $(a+b)^*ab(a+b)^*$ 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 im folgenden häufig einfach r.
Dies ermöglicht es zu schreiben:

- $abb \in ab^* + b$ (eigentlich $abb \in L(ab^* + b)$)
- $(ab)^*a = a(ba)^*$ (eigentlich $L((ab)^*a) = L(a(ba)^*)$)

Wir zeigen nun, dass man mit regulären Ausdrücken genau die erkennbaren Sprachen definieren kann.

Satz 5.4 (Kleene)
Für eine Sprache $L \subseteq \Sigma^*$ sind äquivalent:

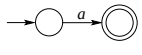
1) L ist regulär.

2) L ist erkennbar.

Beweis.

„$1 \to 2$“: Induktion über den Aufbau regulärer Ausdrücke

Verankerung:

- $L(\emptyset) = \emptyset$ erkennbar:  ist NEA für $\emptyset$ (kein Endzustand).
- $L(\varepsilon) = \{\varepsilon\}$ erkennbar:  ist NEA für $\{\varepsilon\}$.
- $L(a) = \{a\}$ erkennbar:  ist NEA für $\{a\}$.

Schritt: Weiß man bereits, dass $L(r)$ und $L(s)$ erkennbar sind, so folgt mit Satz 4.1 (Abschlusseigenschaften), dass auch

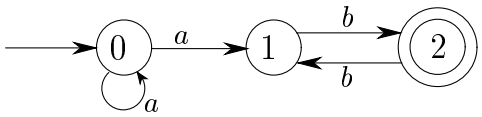
- $L(r + s) = L(r) \cup L(s)$
- $L(r \cdot s) = L(r) \cdot L(s)$ und
- $L(r^*) = L(r)^*$

erkennbar sind.

„$2 \to 1$“: Sei $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ ein NEA mit $L = L(\mathcal{A})$. Wie in Definition 2.8 sei für $q \in Q$ der NEA $\mathcal{A}_q := (Q, \Sigma, q, \Delta, F)$ und $L_q = L(\mathcal{A}_q)$ definiert, d.h. insbesondere $L = L_{q_0}$.
Der Zusammenhang zwischen den L_qs kann nun als *Gleichungssystem* beschrieben werden, dessen Lösungen eindeutig bestimmte *reguläre* Sprachen sind.

Diesen Zusammenhang werden wir im Folgenden näher beleuchten.

Beispiel 5.5
Gegeben sei der folgende NEA $\mathcal{A}$.



Die Sprachen L_q für $q \in Q = \{0, 1, 2\}$ kann man bestimmen als

$$L_0 = \{a\} \cdot L_0 \cup \{a\} \cdot L_1$$
$$L_1 = \{b\} \cdot L_2$$
$$L_2 = \{b\} \cdot L_1 \cup \{\varepsilon\}$$

Allgemein:
Seien $p, q \in Q$. Wir definieren:

- $A_{p,q} = \{a \in \Sigma \mid (p, a, q,) \in \Delta\}$ und
- $B_p = \begin{cases} \{\varepsilon\} & \text{falls} \quad p \in F \\ \emptyset & \text{falls} \quad p \notin F \end{cases}$

Damit erfüllen die Sprachen L_p die folgenden Gleichungen. Für alle $p \in Q$ gilt:

$$L_p = (\bigcup_{q \in Q} A_{p,q} \cdot L_q) \cup B_p$$

Behauptung:
Das Gleichungssystem

$$(\star) \qquad X_p = (\bigcup_{q \in Q} A_{p,q} \cdot X_q) \cup B_p \qquad (p \in Q)$$

hat *genau eine* Lösung und diese besteht aus *regulären* Sprachen.

Aus der Behauptung folgt, dass die Sprachen $X_p = L_p$ regulär sind, da sie ja das System $(\star)$ lösen. Damit erhalten wir den Abschluss des Beweises von Satz 5.4, Richtung „$2 \to 1$“.

Wir zeigen zunächst, wie man eine einzige Gleichung der Form

$$(\star\star) \qquad X = A \cdot X \cup B$$

lösen kann. Daraus ergibt sich dann die Lösung von Gleichungssystemen der Form $(\star)$ durch Induktion über die Anzahl der Variablen.

Lemma 5.6 (Arden)
Es seien $A, B \subseteq \Sigma^*$ und $\varepsilon \notin A$. Die Gleichung $(\star\star)$ $X = A \cdot X \cup B$ hat als eindeutige *Lösung $X = A^* \cdot B$.*

Beachte:
Sind A, B regulär, so auch A^*B.

Aus dem Lemma folgt dann für die Gleichung ($\star$), die wir als Gleichung der Form ($\star\star$) für ein fixiertes $p \in Q$ auffassen:

$$X_p = A_{p,p} \cdot X_p \cup \left(\left(\bigcup_{p \neq q} A_{p,q} \cdot X_q \right) \cup B_p \right)$$

hat als eindeutige Lösung

$$X_p = A_{p,p}^* \cdot \left(\left(\bigcup_{q \neq p} A_{p,q} \cdot X_q \right) \cup B_p \right).$$

Setzt man diese Lösung in ($\star$) ein, so erhält man ein System mit einer Variablen weniger. Nach Induktion hat dieses eine eindeutige Lösung, die aus regulären Sprachen besteht. Da in obiger Lösung für X_p nur reguläre Operationen $(\cup, \cdot, ^*)$ verwendet werden, ist auch diese Lösung regulär (und eindeutig nach Arden).

Beispiel 5.5 (Fortsetzung)

$$\begin{aligned} X_0 &= \{a\} \cdot X_0 \cup \{a\} \cdot X_1 \\ X_1 &= \{b\} \cdot X_2 \\ X_2 &= \{b\} \cdot X_1 \cup \{\varepsilon\} \end{aligned}$$

Auflösen nach X_0:

$X_0 = \{a\}^* \cdot \{a\} \cdot X_1$

Einsetzen ändert die anderen Gleichungen nicht.

Auflösen nach X_1:

$X_1 = \emptyset^* \cdot \{b\} \cdot X_2 = \{b\} \cdot X_2$

Einsetzen liefert:

$X_2 = \{b\} \cdot \{b\} \cdot X_2 \cup \{\varepsilon\}$

Auflösen nach X_2:

$X_2 = (\{b\} \cdot \{b\})^*$

Damit ist

$X_1 = \{b\} \cdot (\{b\} \cdot \{b\})^*$

und

$X_0 = \{a\}^* \cdot \{a\} \cdot \{b\} \cdot (\{b\} \cdot \{b\})^* = L(\mathcal{A})$.

Als regulären Ausdruck für $L(\mathcal{A})$ liefert dieses Verfahren also: $L(\mathcal{A}) = L(a^*ab(bb)^*)$.

Beweis von Lemma 5.6.

1) A^*B ist Lösung:
   $A \cdot A^* \cdot B \cup B = (A \cdot A^* \cup \{\varepsilon\}) \cdot B = A^* \cdot B$
2) Eindeutigkeit:
   Es sei L eine Lösung, d.h. $L = A \cdot L \cup B$.
   2.1) $A^*B \subseteq L$.
   Wir zeigen durch Induktion über n: $A^nB \subseteq L$.
   - $A^0B = B \subseteq (A \cdot L \cup B) = L$
   - Gelte $A^nB \subseteq L$.
     Es folgt: $A^{n+1}B = A \cdot A^nB \subseteq A \cdot L \subseteq A \cdot L \cup B = L$

   Wegen $A^*B = \bigcup_{n\geq 0} A^nB$ folgt damit $A^*B \subseteq L$.
   2.2) $L \subseteq A^*B$.
   Angenommen, dies gilt nicht. Es sei $w \in L$ ein Wort minimaler Länge mit $w \notin A^*B$.
   $w \in L = A \cdot L \cup B$, d.h. $w \in A \cdot L$ oder $w \in B$. Für $w \in B$ folgt $w \in A^*B$ (Widerspruch).
   Aus $w \in A \cdot L$ folgt, dass es $u \in A$ und $v \in L$ gibt mit $w = uv$. Wegen $\varepsilon \notin A$ ist $|v| < |w|$. Wegen Minimalität von w folgt $v \in A^*B$ (Widerspruch, da dann $w = uv \in A^*B$).

□

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

Endliche Automaten mit Ausgabe:

Übergänge $p \xrightarrow{a/v}_{\mathcal{A}} q$, wobei $v \in \Gamma^*$ ein Wort über einem Ausgabealphabet ist. Solche Automaten beschreiben spezielle Funktionen $\Sigma^* \to \Gamma^*$.

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 ist endlich.

Automaten auf unendlichen Wörtern:

Büchi-Automaten sind endliche Automaten, bei denen man unendliche Wörter (unendliche Folgen von Buchstaben) eingibt.

Baumautomaten:

Sind Automaten, die Bäume statt Wörter als Eingaben haben.

II Grammatiken, kontextfreie Sprachen und Kellerautomaten

Einführung

Wir werden hier Klassen formaler Sprachen untersuchen, die allgemeiner sind als die der regulären Sprachen. Insbesondere werden wir die Klasse der kontextfreien Sprachen genauer betrachten, da durch sie z.B. die Syntax von Programmiersprachen (zumindest in großen Teilen) beschreibbar ist.

Zunächst führen wir allgemein den Begriff der Grammatik ein. Klassen formaler Sprachen erhält man durch einschränkende Bedingungen an die Form der Grammatik.

6 Die Chomsky-Hierarchie

Grammatiken dienen dazu, Wörter zu erzeugen. Man hat dazu *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* erzeugt werden können durch wiederholtes Ersetzen.

Beispiel 6.1

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

Eine mögliche Ableitung eines Wortes wäre:

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

Das Symbol S ist hier ein Hilfssymbol (*nichtterminales Symbol*) und man ist nur an den erzeugten Wörtern interessiert, die das Hilfssymbol nicht enthalten (*Terminalwörter*). Man sieht leicht, dass dies hier alle Wörter $a^n b^n$ mit $n \geq 0$ sind.

Definition 6.2 (Grammatik)

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

- N und Σ endliche, disjunkte Alphabete sind. (Man bezeichnet die Symbole aus N als *Nichtterminalsymbole*, die Symbole aus Σ als *Terminalsymbole*),
- $S \in N$ das *Startsymbol* ist,
- $P \subseteq (N \cup \Sigma)^+ \times (N \cup \Sigma)^*$ eine endliche Menge von Ersetzungsregeln (*Produktionen*) ist.

Produktionen $(u, v) \in P$ schreibt man gewöhnlich als $u \longrightarrow v$.

Beispiel 6.3

$G = (N, \Sigma, P, S)$ 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 man ein Wort durch Anwenden der Regeln aus einem anderen ableiten kann.

Definition 6.4 (durch eine Grammatik erzeugte Sprache)
Es sei $G = (N, \Sigma, P, S)$ eine Grammatik und x, y seien Wörter aus $(N \cup \Sigma)^*$.

1) *y aus x direkt ableitbar:*
   $x \vdash_G y$ gdw. $\exists x_1, x_2 \in (N \cup \Sigma)^* : \exists u \longrightarrow v \in P : x = x_1 u x_2 \wedge y = x_1 v x_2$
2) *y aus x in n Schritten ableitbar:*
   $x \vdash_G^n y$ gdw. $\exists x_0, x_1, \ldots, x_n \in (N \cup \Sigma)^* : x_0 = x \wedge x_n = y \wedge x_i \vdash_G x_{i+1}$ für $0 \leq i < n$
3) *y aus x ableitbar:*
   $x \vdash_G^* y$ gdw. $\exists n \geq 0 : x \vdash_G^n y$
4) Die durch G *erzeugte Sprache* ist
   $L(G) := \{w \in \Sigma^* \mid S \vdash_G^* w\}$.

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

Beispiel 6.3 (Fortsetzung)

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

Es gilt: $L(G) = \{a^n b^n c^n \mid n \geq 1\}$

Beweis.

„$\supseteq$“: Für $n = 1$ ist $abc \in L(G)$ klar. Für $n > 1$ sieht man leicht:

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

„$\subseteq$“: Gelte $S \vdash_G^* w$ mit $w \in \Sigma^*$. Wird sofort die Produktion $S \longrightarrow abc$ verwendet, so ist $w = abc \in \{a^n b^n c^n \mid n \geq 1\}$.
Sonst betrachten wir die Stelle in der Ableitung, an der S das letzte mal auftritt:

$$S \vdash_G^* a^{n-1} S u \text{ mit } u \in \{c, B\}^* \text{ und } |u|_B = |u|_c = n - 1$$

(nur $S \longrightarrow aSBc$, $cB \longrightarrow Bc$ angewendet, da noch kein b vorhanden).

$$a^{n-1} S u \vdash_G a^n bcu \vdash_G^* w.$$

Um von $a^n bcu$ aus nun alle nichtterminalen Symbole B zu entfernen, muss man $bB \longrightarrow bb$ anwenden. Damit B aber auf b stößt, muss es links von allen cs stehen.

□

Beispiel 6.5

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

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

$L(G) = \Sigma^* \cdot \{a\} \cdot \{b\} \cdot \Sigma^*$

Die Grammatiken aus Beispiel 6.5, 6.3 und 6.1 gehören zu unterschiedlichen Stufen der **Chomsky-Hierarchie**:

Definition 6.6 (Typen von Chomsky-Grammatiken)

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

- Jede Grammatik G heißt Grammatik vom ***Typ 0***.
- G heißt Grammatik vom ***Typ 1 (kontextsensitiv)***, falls jede Produktion von G die Form
  - $u_1 A u_2 \longrightarrow u_1 w u_2$ mit $A \in N$, $u_1, u_2, w \in (\Sigma \cup N)^*$ und $|w| \geq 1$ oder
  - $S \longrightarrow \varepsilon$ hat.

  Ist $S \longrightarrow \varepsilon \in P$, so kommt S nicht auf der rechten Seite einer Produktion vor.
- G heißt Grammatik vom ***Typ 2 (kontextfrei)***, falls jede Regel von G die Form $A \longrightarrow w$ hat mit $A \in N, w \in (\Sigma \cup N)^*$.
- G heißt Grammatik vom ***Typ 3 (rechtslinear)***, falls jede Regel von G die Form $A \longrightarrow uB$ oder $A \longrightarrow u$ hat mit $A, B \in N$, $u \in \Sigma^*$.

kontextfrei: Die linke Seite jeder Produktion besteht nur aus einem Nichtterminalsymbol A, das daher stets unabhängig vom Kontext im Wort ersetzt werden kann.

kontextsensitiv: $u_1 A u_2 \longrightarrow u_1 w u_2$. Hier ist die Ersetzung von A durch w abhängig davon, dass der richtige Kontext (u_1 links und u_2 rechts) vorhanden ist. $|w| \geq 1$ sorgt dafür, dass die Produktionen die Wörter verlängern (Ausnahme $S \longrightarrow \varepsilon$, die aber nur zur Erzeugung von ε dient).

Definition 6.7 (Klasse der Typ-i-Sprachen)

Für $i = 0, 1, 2, 3$ ist die *Klasse der Typ-i-Sprachen definiert* als
$\mathcal{L}_i := \{L(G) \mid G \text{ ist Grammatik vom Typ i}\}$.

Wir werden sehen: $\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0$.

Nach Definition der Grammatiktypen gilt offenbar $\mathcal{L}_3 \subseteq \mathcal{L}_2$ und $\mathcal{L}_1 \subseteq \mathcal{L}_0$.

7 Rechtslineare Grammatiken und reguläre Sprachen

Satz 7.1
Die Typ-3-Sprachen sind genau die regulären/erkennbaren Sprachen, d.h.
$\mathcal{L}_3 = \{L \mid L \text{ ist regulär}\}$.

Beweis. Der Beweis wird in zwei Richtungen durchgeführt:

1. Jede Typ-3-Sprache ist erkennbar

   Es sei $L \in \mathcal{L}_3$, d.h. $L = L(G)$ für eine Typ-3-Grammatik $G = (N, \Sigma, P, S)$. Es gilt $w \in L(G)$ gdw.
   Es gibt eine Ableitung

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

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

   Wir konstruieren nun einen *NEA mit Worttransitionen*, der die Nichtterminalsymbole von G als Zustände hat:

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

   - $\Omega \notin N$ neuer Endzustand 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 $(\star)$ entsprechen nun genau Pfaden in $\mathcal{A}$:

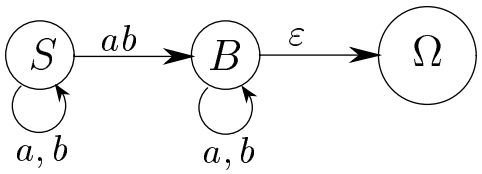
   $$(\star\star) \quad (S, w_1, B_1)(B_1, w_2, B_2) \ldots (B_{n-2}, w_{n-1}, B_{n-1})(B_{n-1}, w_n, \Omega)$$

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

   Beispiel 6.5 (Fortsetzung)

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

   liefert den folgenden NEA mit Wortübergängen:

2. Jede erkennbare Sprache ist eine Typ-3-Sprache

   Es sei $L = L(\mathcal{A})$ für einen NEA $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$. Wir definieren daraus eine Typ-3-Grammatik $G = (N, \Sigma, P, S)$ wie folgt:

$$\begin{aligned} N &:= Q \\ S &:= q_0 \\ P &:= \{p \longrightarrow aq \mid (p, a, q) \in \Delta\} \cup \{p \longrightarrow \varepsilon \mid p \in F\} \end{aligned}$$

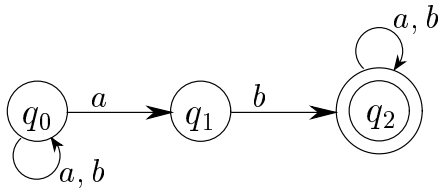
   Ein Pfad in $\mathcal{A}$ der Form

$$(q_0, a_1, q_1)(q_1, a_2, q_2) \dots (q_{n-1}, a_n, q_n)$$

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

$$q_0 \vdash_G a_1 q_1 \vdash_G a_1 a_2 q_2 \vdash_G \dots \vdash_G a_1 \dots a_n q_n \vdash_G a_1 \dots a_n.$$

   Beispiel:
   Der folgende NEA



   liefert die Grammatik mit den rechtslinearen Produktionen

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

□

Korollar 7.2
$\mathcal{L}_3 \subset \mathcal{L}_2$.

Beweis. Wir wissen bereits, dass $\mathcal{L}_3 \subseteq \mathcal{L}_2$ gilt. Außerdem haben wir mit Beispiel 6.1 eine Sprache $L := \{a^n b^n \mid n \geq 0\} \in \mathcal{L}_2$. Im Teil I haben wir gezeigt, dass L nicht erkennbar/regulär ist, d.h. mit Satz 7.1 folgt $L \notin \mathcal{L}_3$. □

Beispiel 7.3

Als ein weiteres Beispiel für eine kontextfreie Sprache, die nicht regulär ist, betrachten wir $L = \{a^n b^m \mid n \neq m\}$. (Vgl. Beispiele 3.3, 4.2) Man kann diese Sprache mit folgender kontextfreier Grammatik erzeugen:

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

- $N = \{S, S_{\geq}, S_{\leq}\}$
- $\Sigma = \{a, b\}$
- $P = \{S \longrightarrow aS_{\geq}, \quad S \longrightarrow S_{\leq}b,$
  $S_{\geq} \longrightarrow aS_{\geq}b, \quad S_{\leq} \longrightarrow aS_{\leq}b,$
  $S_{\geq} \longrightarrow aS_{\geq}, \quad S_{\leq} \longrightarrow S_{\leq}b,$
  $S_{\geq} \longrightarrow \varepsilon, \quad S_{\leq} \longrightarrow \varepsilon\}$

Es gilt nun:

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

woraus sich ergibt:

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

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

8 Normalformen kontextfreier Grammatiken

Wir werden zeigen, dass man die Klasse aller kontextfreien Sprachen bereits mit kontextfreien Grammatiken einer *eingeschränkten* Form erzeugen kann.

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

Zunächst zeigen wir, dass man „überflüssige“ Symbole aus kontextfreien Grammatiken eliminieren kann.

Definition 8.1 (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.

Lemma 8.2
Zu einer kontextfreien Grammatik $G = (N, \Sigma, P, S)$ kann man effektiv die Menge der erreichbaren Nichtterminalsymbole bestimmen.

Beweis. Wir definieren dazu

$$\begin{aligned} E_0 &:= \{S\} \\ E_{i+1} &:= E_i \cup \{A \mid \exists B \in E_i \text{ mit Regel } B \longrightarrow u_1 A u_2 \in P\} \end{aligned}$$

Es gilt

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

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:

„$\subseteq$“: Offenbar enthalten alle E_i nur erreichbare Symbole

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

□

Beispiel:

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

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

Es ist also A das einzige nichterreichbare Symbol.

Lemma 8.3
Zu einer kontextfreien Grammatik $G = (N, \Sigma, P, S)$ kann man effektiv die Menge der terminierenden Symbole bestimmen.

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

$$T_1 \subseteq T_2 \subseteq \ldots \subseteq N.$$

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

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

„$\subseteq$“: Offenbar sind alle Elemente von T_i $(i \geq 1)$ terminierend:

- $i = 1$: $\quad A \vdash_G w \in \Sigma^*$.
- $i \to i+1$: $A \vdash_G u_1 B_1 u_2 B_2 \ldots u_n B_n u_{n+1},\ B_i \vdash_G^* w_i \in \Sigma^*$

„$\supseteq$“: Zeige durch Induktion über i:
$A \vdash_G^{\leq i} w \in \Sigma^* \Rightarrow A \in T_i$.

- $i = 1$: $\quad A \vdash_G^{\leq 1} w \in \Sigma^* \Rightarrow A \longrightarrow w \in P \Rightarrow A \in T_1$
- $i \to i+1$: $A \vdash_G u_1 B_1 \ldots u_n B_n u_{n+1}$
  $\vdash_G^{\leq i} w = u_1 w_1 \ldots u_n w_n u_{n+1},\ u_j w_j \in \Sigma^*,\ B_j \in N$
  $\Rightarrow$ Für alle j gilt: $B_j \vdash_G^{\leq i} w_j \in \Sigma^*$
  $\Rightarrow B_j \in T_i$ (Induktion) $\Rightarrow A \in T_{i+1}$

□

Aus Lemma 8.3 folgt unmittelbar:

Satz 8.4
Das Leerheitsproblem ist für kontextfreie Sprachen entscheidbar.

Beweis. Offenbar gilt $L(G) \neq \emptyset$ gdw. $\exists w \in \Sigma^* : S \vdash_G^* w$ gdw. S ist terminierend. □

Lemma 8.2 und 8.3 zusammen zeigen, wie man unerreichbare und nichtterminierende Symbole eliminieren kann.

Satz 8.5
Zu jeder kontextfreien Grammatik G mit $L(G) \neq \emptyset$ kann man effektiv eine äquivalente reduzierte kontextfreie Grammatik erzeugen.

Beweis.

Erster Schritt: Eliminieren nicht terminierender Symbole.

Zu $G = (N, \Sigma, P, S)$ definieren wir $G' := (N', \Sigma, P', S)$, wobei

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

Beachte: Aus $L(G) \neq \emptyset$ folgt $S \in N'$!

Zweiter Schritt: Eliminieren unerreichbarer Symbole.

Wir definieren $G'' := (N'', \Sigma, P'', S)$, wobei

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

Beachte: Ist $A \in N''$ und $A \longrightarrow u_1 B u_2 \in P'$, so ist $B \in N''$.

Man sieht leicht, dass $L(G) = L(G') = L(G'')$ und G'' reduziert ist. □

Vorsicht:
Die Reihenfolge der beiden Schritte ist wichtig! Symbole, die in G noch erreichbar waren, müssen es in G' nicht mehr sein.
Z.B.: $S \longrightarrow AB$ mit A terminierend, B nicht.

Als nächstes eliminieren wir Regeln der Form $A \longrightarrow \varepsilon$.

Lemma 8.6
Es sei G eine kontextfreie Grammatik. Dann lässt sich eine Grammatik G' ohne Regeln der Form $A \longrightarrow \varepsilon$ konstruieren mit $L(G') = L(G) \setminus \{\varepsilon\}$.

Beweis.

1) Finde alle $A \in N$ mit $A \vdash_G^* \varepsilon$:

$$N_1 := \{A \in N \mid A \longrightarrow \varepsilon \in P\}$$
$$N_{i+1} := N_i \cup \{A \in N \mid A \longrightarrow B_1 \dots B_n \in P \text{ für } B_j \in N_i\}$$

Es gibt ein k mit $N_k = N_{k+1} = \bigcup_{i \geq 1} N_i$. Für dieses k gilt: $A \in N_k$ gdw. $A \vdash_G^* \varepsilon$.

2) Eliminiere in G alle Regeln $A \longrightarrow \varepsilon$. Um dies auszugleichen, nimmt man für alle Regeln

$$A \longrightarrow u_1 B_1 \dots u_n B_n u_{n+1} \text{ mit } B_i \in N_k \text{ und } u_i \in (\Sigma \cup N \setminus N_k)^*$$

die Regeln

$$A \longrightarrow u_1 \beta_1 u_2 \dots u_n \beta_n u_{n+1} \text{ mit } \beta_i \in \{B_i, \varepsilon\} \text{ und } u_1 \beta_1 u_2 \dots u_n \beta_n u_{n+1} \neq \varepsilon$$

hinzu. □

Beispiel:

$$
\begin{aligned}
P = \quad & \{S \longrightarrow aS,\ S \longrightarrow SS,\ S \longrightarrow bA,\\
& A \longrightarrow BB,\\
& B \longrightarrow CC,\ B \longrightarrow aAbC,\\
& C \longrightarrow \varepsilon\}\\
N_0 = \quad & \{C\},\\
N_1 = \quad & \{C, B\},\\
N_2 = \quad & \{C, B, A\} = N_3\\
P' = \quad & \{S \longrightarrow aS,\ S \longrightarrow SS,\ S \longrightarrow bA,\ S \longrightarrow b,\\
& A \longrightarrow BB,\ A \longrightarrow B,\\
& B \longrightarrow CC,\ B \longrightarrow C,\\
& B \longrightarrow aAbC,\ B \longrightarrow abC,\ B \longrightarrow aAb, B \longrightarrow ab\}
\end{aligned}
$$

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

Definition 8.7 (ε-freie kontextfreie Grammatik)
Eine kontextfreie Grammatik heißt *ε-frei*, falls gilt:

1) Sie enthält keine Regeln $A \longrightarrow \varepsilon$ für $A \neq S$.
2) Ist $S \longrightarrow \varepsilon$ enthalten, so kommt S nicht auf der rechten Seite einer Regel vor.

Satz 8.8
Zu jeder kontextfreien Grammatik G kann effektiv eine äquivalente ε-freie Grammatik konstruiert werden.

Beweis. Konstruiere G' wie im Beweis von Lemma 8.6 beschrieben. Ist $\varepsilon \notin L(G)$ (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 $S_0 \longrightarrow \varepsilon$. □

Korollar 8.9
$\mathcal{L}_2 \subseteq \mathcal{L}_1$.

Beweis. Offenbar ist jede ε-freie kontextfreie Grammatik eine Typ-1-Grammatik:

- Produktionen: $u_1 A u_2 \longrightarrow u_1 w u_2$ mit $|w| \geq 1$
- kontextfrei: $u_i = \varepsilon$; $S \longrightarrow \varepsilon$ und S nicht auf rechter Seite). □

Wir zeigen nun, dass man auch auf Regeln der Form $A \longrightarrow B$ verzichten kann.

Satz 8.10
Zu jeder kontextfreien Grammatik kann man effektiv eine äquivalente kontextfreie Grammatik konstruieren, die keine Regeln der Form $A \longrightarrow B$ $(A, B \in N)$ enthält.

Beweisskizze.

1) Bestimme zu jedem $A \in N$ die Menge $N(A) := \{B \in N \mid A \vdash_G^* B\}$ (effektiv machbar).
2) $P' = \{A \longrightarrow w \mid B \longrightarrow w \in P,\ B \in N(A) \text{ und } w \notin N\}$ □

Beispiel:

$$\begin{aligned} P &= \{S \longrightarrow A,\ A \longrightarrow B,\ B \longrightarrow aA,\ B \longrightarrow b\} \\ N(S) &= \{S, A, B\}, \\ N(A) &= \{A, B\}, \\ N(B) &= \{B\} \\ P' &= \{B \longrightarrow aA,\ A \longrightarrow aA,\ S \longrightarrow aA, \\ &\qquad B \longrightarrow b, \quad A \longrightarrow b, \quad S \longrightarrow b\} \end{aligned}$$

Wir zeigen nun, dass man sich auf Regeln sehr einfacher Form beschränken kann, nämlich $A \longrightarrow a$ und $A \longrightarrow BC$.

Satz 8.11 (Chomsky-Normalform)
Jede kontextfreie Grammatik lässt sich umformen in eine äquivalente Grammatik, die nur Regeln der Form

- $A \longrightarrow a,\ A \longrightarrow BC$ *mit* $A, B, C \in N,\ a \in \Sigma$
- *und eventuell* $S \longrightarrow \varepsilon$*, wobei* S *nicht rechts vorkommt*

enthält. Eine derartige Grammatik heißt dann Grammatik in Chomsky-Normalform.

Beweis.

1) Konstruiere zu der gegebenen Grammatik eine äquivalente ε-freie ohne Regeln der Form $A \longrightarrow B$. (Dabei ist die Reihenfolge wichtig!)
2) Führe für jedes $a \in \Sigma$ ein neues Nichtterminalsymbol X_a und die Produktion $X_a \longrightarrow a$ ein.
3) Ersetze in jeder Produktion $A \longrightarrow w$ mit $w \notin \Sigma$ alle Terminalsymbole a durch die zugehörigen X_a.
4) Produktionen $A \longrightarrow B_1 \ldots B_n$ für $n > 2$ werden ersetzt 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.

□

Beachte:

Für einen NEA $\mathcal{A}$ (eine rechtslineare Grammatik G) kann man das Wortproblem, also die Frage „$w \in L(\mathcal{A})$?“ („$w \in L(G)$?“) z.B. deshalb entscheiden, da ein Pfad in $\mathcal{A}$, der w erkennt (eine Ableitung in G, die w erzeugt) *genau* die Länge $|w|$ haben muss. Es gibt aber nur endlich viele Pfade (Ableitungen) dieser festen Länge.

Bei allgemeinen kontextfreien Grammatiken kann man keine Schranke für die Länge einer Ableitung von w aus S angeben, wohl aber bei Grammatiken in Chomsky-Normalform:

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

Es folgt: $w \in L(G) \setminus \{\varepsilon\}$ wird durch eine Ableitung der Länge $2|w| - 1$ erzeugt.

Da es aber i.a. $\geq 2^n$ Ableitungen der Länge n geben kann, liefert dies ein *exponentielles Verfahren* zur Entscheidung des Wortproblems.

Ein besseres Verfahren (*kubisch*) liefert die folgende Überlegung:

Definition 8.12

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

- $w_{ij} := a_i \dots a_j \quad$ (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.	$A \vdash_G^* a_i \dots a_j$
		gdw.	$\exists A \longrightarrow BC \in P$ und ein k mit $i \leq k < j$ mit $B \vdash_G^* a_i \dots a_k$ und $C \vdash_G^* a_{k+1} \dots a_j$
		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 das folgende iterative Verfahren zur Bestimmung von N_{1k}:

Algorithmus 8.13 (CYK-Algorithmus von Cocke, Younger, Kasami)

For $i := 1$ **To** n **Do**
  $N_{ii} := \{A \mid A \longrightarrow a_i \in P\}$
For $d := 1$ **To** $n-1$ **Do** (wachsende Differenz $d = j - i$)
  For $i := 1$ **To** $n-d$ **Do**
    $j := i + d$
    $N_{ij} := \emptyset$
    For $k := i$ **To** $j-1$ **Do**
      $N_{ij} := N_{ij} \cup \{A \mid \exists A \longrightarrow BC \in P \text{ mit } B \in N_{ik} \text{ und } C \in N_{(k+1)j}\}$

Beachte:

In der innersten Schleife sind die Differenzen $k - i$ und $j - k + 1$ kleiner als das aktuelle d, also sind N_{ik} und $N_{(k+1)j}$ bereits berechnet.

Satz 8.14

Für eine gegebene Grammatik in Chomsky-Normalform entscheidet Algorithmus 8.13 die Frage „$w \in L(G)$?“ in der Zeit $O(|w|^3)$.

Beweis. Drei geschachtelte Schleifen, die jeweils $\leq |w| = n$ Schritte machen, daraus folgt: $|w|^3$ Schritte in der innersten Schleife. □

Beachte:

Die Größe von G ist hier als konstant angenommen (fest vorgegebenes G). Daher ist die Suche nach den Produktionen $A \longrightarrow BC$ und $A \longrightarrow a_i$ auch konstant.

Beispiel:

$P = \{S \longrightarrow SA,\ S \longrightarrow a,$
$A \longrightarrow BS,$
$B \longrightarrow BB,\ B \longrightarrow BS,\ B \longrightarrow b,\ B \longrightarrow c\}$

und $w = abacba$:

$i\backslash j$	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

$S \in N_{1,6} = \{S\}$
$\Rightarrow w \in L(G)$

Eine weitere interessante Normalform für kontextfreie Grammatiken ist die *Greibach-Normalform*, bei der jede Produktion mindestens ein Terminalsymbol erzeugt.

Satz 8.15

Jede kontextfreie Grammatik lässt sich effektiv umformen in eine äquivalente Grammatik, die nur Regeln der Form

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

enthält (ohne Beweis!).
Eine derartige Grammatik heißt Grammatik in Greibach-Normalform.

9 Abschlusseigenschaften kontextfreier Sprachen

Satz 9.1

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

Beweis. Es seien $L_1 = L(G_1)$ und $L_2 = L(G_2)$ die Sprachen für kontextfreie Grammatiken $G_i = (N_i, \Sigma, P_i, S_i) \quad (i = 1, 2)$. O.B.d.A. nehmen wir 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 $L(G) = L_1 \cup L_2$.
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 $L(G') = L_1 \cdot L_2$.
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_1^*

□

Wir werden zeigen, dass Abschluss unter *Durchschnitt* 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. Um dieses zu zeigen, stellt man Ableitungen als Bäume, sogenannte *Ableitungsbäume* dar.

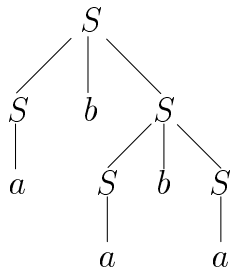
Beispiel:

$P = \{S \longrightarrow SbS, S \longrightarrow a\}$
Drei *Ableitungen* des Wortes *ababa*:

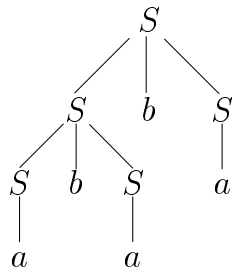
1) $S \vdash SbS \vdash abS \vdash abSbS \vdash ababS \vdash ababa$
2) $S \vdash SbS \vdash abS \vdash abSbS \vdash abSba \vdash ababa$
3) $S \vdash SbS \vdash Sba \vdash SbSba \vdash Sbaba \vdash ababa$

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.

Allgemein:

Die Knoten des Ableitungsbaumes sind mit Elementen aus $\Sigma \cup N$ beschriftet. Ein mit A beschrifteter Knoten kann mit $\alpha_1, \ldots, \alpha_n$ beschriftete Nachfolgerknoten haben, wenn $A \longrightarrow \alpha_1 \ldots \alpha_n \in P$ ist.

Ein Ableitungsbaum, dessen Wurzel mit A beschriftet ist und dessen Blätter (von links nach rechts) mit $\alpha_1, \ldots, \alpha_n \in \Sigma \cup N$ beschriftet sind, beschreibt eine Ableitung $A \vdash_G^* \alpha_1 \ldots \alpha_n$.

Lemma 9.2 (Pumping-Lemma für kontextfreie Sprachen)

Es sei G eine ε-freie kontextfreie Grammatik, die

- *keine Regeln der Form $A \longrightarrow B$ enthält,*
- *m nichtterminale Symbole enthält*
- *nur rechten Regelseiten der Länge $\leq k$ hat*

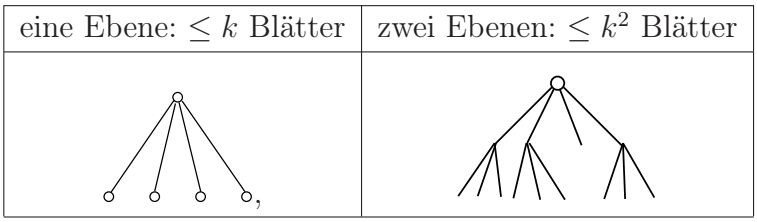
Es sei $n = k^{m+1}$.

Dann gibt es für jedes $z \in L(G)$ mit $|z| > n$ eine Zerlegung $z = uvwxy$ mit:

- *$vx \neq \varepsilon$ und $|vwx| \leq n$*
- *$uv^iwx^iy \in L(G)$ für alle $i \geq 0$.*

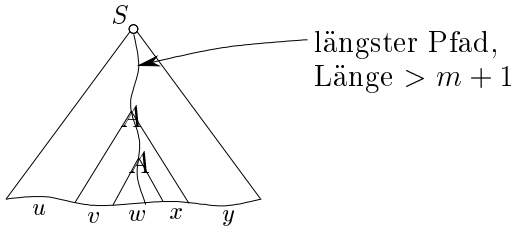
Beweis.

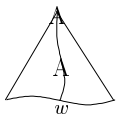
1) Ein Baum der Tiefe $\leq t$ und der Verzweigungszahl $\leq k$ hat maximal k^t viele Blätter:



Da der Ableitungsbaum für z als Blattanzahl $|z| > k^{m+1}$ hat, ist die maximale Tiefe (längster Pfad von Wurzel bis Blatt) $> m + 1$.

2) Da es nur m verschiedene Elemente in N gibt, kommt auf diesem längsten Pfad ein nichtterminales Symbol A zweimal vor.



Wir wählen hier o.B.d.A. A so, dass es in dem Baum 

keine andere Variablenwiederholung gibt. Daher hat dieser Baum eine Tiefe $\leq m + 1$, was $|vwx| \leq k^{m+1} = n$ 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) $vx \neq \varepsilon$: Da G ε-frei ist, wäre sonst $A \vdash_G^* vAx$ nur bei Anwesenheit von Regeln der Form $A \longrightarrow B$ möglich. □

Satz 9.3
$\mathcal{L}_2 \subset \mathcal{L}_1$.

Beweis. Wir haben bereits gezeigt, dass $\mathcal{L}_2 \subseteq \mathcal{L}_1$ gilt (Korollar 8.9). Es bleibt zu zeigen, dass die Inklusion echt ist.
Dafür betrachten wir die Sprache $L = \{a^nb^nc^n \mid n \geq 1\}$, und zeigen: $L \in \mathcal{L}_1 \setminus \mathcal{L}_2$

1) Wir zeigen zunächst $L \notin \mathcal{L}_2$. Angenommen, $L \in \mathcal{L}_2$. Dann gibt es eine ε-freie kontextfreie Grammatik G ohne Regeln der Form $A \longrightarrow B$ für L. Es sei $n_0 = k^{m+1}$ die zugehörige Zahl aus Lemma 9.2. Wir betrachten $z = a^{n_0}b^{n_0}c^{n_0} \in L = L(G)$. Mit Satz 9.2 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 Buchstaben. Man sieht leicht, dass dann

$$uv^2wx^2y \notin a^*b^*c^* \supseteq L.$$

   2. Fall: x enthält verschiedene Buchstaben. Dies führt zu entsprechendem Widerspruch.

   3. Fall: v enthält lauter gleiche Buchstaben und x enthält lauter gleiche Buchstaben. Dann gibt es einen Buchstaben aus $\{a, b, c\}$, der in xv nicht vorkommt. Daher kommt dieser in $uv^0wx^0y = uwy$ weiterhin n_0-mal vor. Aber es gilt $|uwy| < 3n_0$, was $uwy \notin L$ zeigt.

2) In Beispiel 6.3 haben wir eine Grammatik G mit $L(G) = \{a^nb^nc^n \mid n \geq 1\}$ angegeben. Leider enthält G eine Produktion, die nicht die Bedingungen für kontextsensitive Produktionen ($uAv \longrightarrow uwv, \ |w| \geq 1$) erfüllt: $cB \longrightarrow Bc$. Wir modifizieren G wie folgt:

   - Ersetze in allen Produktionen c durch ein neues nichtterminales Symbol C und nimm die Produktion $C \longrightarrow c$ hinzu:

$$\{S \longrightarrow aSBC, \ S \longrightarrow abC, \ CB \longrightarrow BC, \ bB \longrightarrow bb, \ C \longrightarrow c\}.$$

   - Ersetze nun $CB \longrightarrow BC$ durch die kontextsensitiven Produktionen

$$CB \longrightarrow A_1B, \ A_1B \longrightarrow A_1A_2, \ A_1A_2 \longrightarrow BA_2, \ BA_2 \longrightarrow BC.$$

     Diese Produktionen können nur dazu verwendet werden, $CB \longrightarrow BC$ zu simulieren. □

Beachte:

Auf diese Art kann man leicht zeigen, dass *jede nichtkürzende* Produktion $u \longrightarrow v$ (mit $|u| \leq |v|$) durch *kontextsensitive* Produktionen ersetzt werden kann, ohne die Sprache zu ändern.

Korollar 9.4

Die Klasse $\mathcal{L}_2$ der kontextfreien Sprachen ist nicht *unter Durchschnitt 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.
   Widerspruch zu Teil 1) des Beweises von Satz 9.3.
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. Wiederspruch zu 2).

□

Beachte:

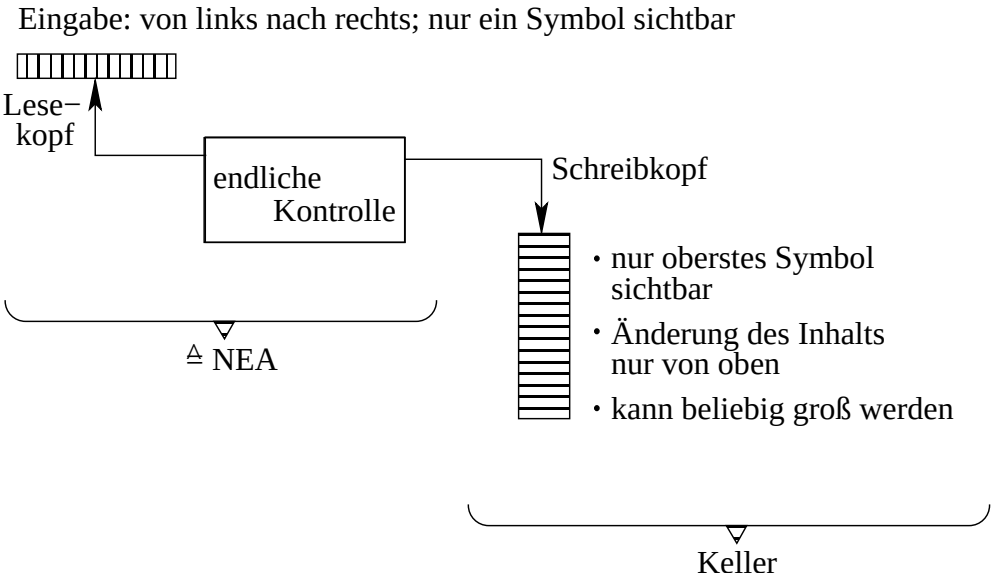
Man kann daher das Äquivalenzproblem für kontextfreie Sprachen nicht einfach auf das Leerheitsproblem reduzieren. Wir werden später sehen, dass das Äquivalenzproblem für kontextfreie Sprachen sogar *unentscheidbar* ist.

10 Kellerautomaten

Bisher hatten 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, muss man diese um eine (potentiell unendliche) *Speicherkomponente*, einen sogenannten *Keller*, erweitern.

Die folgende Abbildung zeigt eine schematische Darstellung eines *Kellerautomaten*:



Definition 10.1 (Kellerautomat)
Ein *Kellerautomat (pushdown automaton, kurz PDA)* hat die Form $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, Z_0, \Delta)$, wobei

- Q eine endliche Menge von *Zuständen* ist,
- Σ das *Eingabealphabet* ist,
- Γ das *Kelleralphabet* ist,
- $q_0 \in Q$ der *Anfangszustand* ist,
- $Z_0 \in \Gamma$ das *Kellerstartsymbol* ist und
- $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times \Gamma^* \times Q$ eine endliche *Übergangsrelation* 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 durch γ ersetzen und in den Zustand q' und zum nächsten Eingabesymbol übergehen.

$(q, \varepsilon, Z, \gamma, q')$: wie oben, nur dass das aktuelle Eingabesymbol nicht relevant ist und man nicht zum nächsten Eingabesymbol übergeht (der Lesekopf ändert seine Position nicht).

Den aktuellen *Zustand* einer Kellerautomatenberechnung kann man beschreiben durch

- den *noch zu lesenden Rest* $w \in \Sigma^*$ der Eingabe (Lesekopf steht auf dem ersten Symbol von w)
- den *Zustand* $q \in Q$
- den *Kellerinhalt* $\gamma \in \Gamma^*$ (Schreiblesekopf steht auf dem ersten Symbol von γ)

Definition 10.2

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*:

- $(q, aw, Z\gamma) \vdash (q', w, \beta\gamma)$ falls $(q, a, Z, \beta, q') \in \Delta$
- $(q, w, Z\gamma) \vdash (q', w, \beta\gamma)$ falls $(q, \varepsilon, Z, \beta, q') \in \Delta$
- $\mathcal{K} \vdash^* \mathcal{K}'$ gdw. $\exists n \geq 0\ \exists \mathcal{K}_0, \ldots, \mathcal{K}_n$ mit

$$\mathcal{K}_0 = \mathcal{K},\ \mathcal{K}_n = \mathcal{K}' \text{ und } \mathcal{K}_i \vdash \mathcal{K}_{i+1} \text{ für } 0 \leq i < n.$$

Der Automat $\mathcal{A}$ *akzeptiert* das Wort $w \in \Sigma^*$ gdw.

$$(q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon) \quad \text{(Eingabewort ganz gelesen und Keller leer).}$$

Die von $\mathcal{A}$ *akzeptierte Sprache* ist

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

Beispiel 10.3

Ein PDA für $\{a^n b^n \mid n \geq 1\}$.

- $Q = \{q_0, q_1, f\}$,
- $\Gamma = \{Z, Z_0\}$,
- $\Sigma = \{a, b\}$ und
- $\Delta = \{(q_0, a, Z_0, ZZ_0, q_0)$, (erstes a, speichere Z)
  (q_0, a, Z, ZZ, q_0), (weitere a's, speichere Z)
  $(q_0, b, Z, \varepsilon, q_1)$, (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)

Betrachten wir uns einige Konfigurationsübergänge:

1) $(q_0, aabb, Z_0) \vdash (q_0, abb, ZZ_0) \vdash (q_0, bb, ZZZ_0) \vdash (q_1, b, ZZ_0) \vdash (q_1, \varepsilon, Z_0) \vdash (f, \varepsilon, \varepsilon)$
   – akzeptiert
2) $(q_0, aab, Z_0) \vdash^* (q_0, b, ZZZ_0) \vdash (q_1, \varepsilon, ZZ_0)$
   – kein Übergang mehr möglich
3) $(q_0, abb, Z_0) \vdash (q_0, bb, ZZ_0) \vdash (q_1, b, Z_0) \vdash (f, b, \varepsilon)$
   – nicht akzeptiert

Der Kellerautomat aus Beispiel 10.3 ist deterministisch, da es zu jeder Konfiguration höchstens eine Folgekonfiguration gibt.

Definition 10.4

Der PDA $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, Z_0, \Delta)$ heißt *deterministisch*, falls die folgenden beiden Bedingungen erfüllt sind:

1) $\forall q \in Q\ \forall a \in \Sigma \cup \{\varepsilon\}\ \forall Z \in \Gamma$ existiert höchstens ein Übergang der Form $(q, a, Z, \ldots, \ldots) \in \Delta$.
2) Existiert ein Tupel $(q, \varepsilon, Z, \ldots, \ldots) \in \Delta$, so existiert kein Tupel der Form $(q, a, Z, \ldots, \ldots) \in \Delta$ mit $a \in \Sigma$.

Beispiel:

Die Sprache $L = \{w\ \overleftarrow{w}\ \mid w \in \{a, b\}^*\}$ (wobei für $w = a_1 \ldots a_n$ gilt $\overleftarrow{w} = a_n \ldots a_1$) wird von einem nichtdeterministischen PDA akzeptiert.

Idee: Der PDA legt die erste Worthälfte im Keller ab und greift darauf *in umgekehrter Reihenfolge* beim Lesen der zweiten Worthälfte zurück. So kann Nichtübereinstimmung festgestellt werden (kein Übergang bei Nichtübereinstimmung).

Nichtdeterminismus ist intuitiv nötig, da man „raten“ muss, wann die Wortmitte erreicht ist (Man kann beweisen, dass kein deterministischer PDA diese Sprache akzeptiert).

$Q = \{q_0, q_1, q_2, f\},\ \Gamma = \{a, b, Z_0\},\ \Sigma = \{a, b\},\ \Delta =$

$\{(q_0,$	$\varepsilon,$	$Z_0,$	$\varepsilon,$	$f),$	akzeptiert ε
$(q_0,$	$a/b,$	$Z_0,$	$a/bZ_0,$	$q_1),$	Lesen
$(q_1,$	$a,$	$b,$	$ab,$	$q_1),$	und
$(q_1,$	$b,$	$a,$	$ba,$	$q_1),$	Speichern
$(q_1,$	$a,$	$a,$	$aa,$	$q_1),$	der ersten
$(q_1,$	$b,$	$b,$	$bb,$	$q_1),$	Worthälfte
$(q_1,$	$b,$	$b,$	$\varepsilon,$	$q_2),$	Lesen der
$(q_1,$	$a,$	$a,$	$\varepsilon,$	$q_2),$	zweiten Hälfte
$(q_2,$	$b,$	$b,$	$\varepsilon,$	$q_2),$	und Vergleichen
$(q_2,$	$a,$	$a,$	$\varepsilon,$	$q_2),$	mit erster
$(q_2,$	$\varepsilon,$	$Z_0,$	$\varepsilon,$	$f)\}$	Entfernen Kellerstartsymbol

Es gibt auch einen anderen Akzeptanzbegriff für PDAs, bei dem man nicht fordert, dass der Keller leer ist, sondern dass ein Endzustand (aus einer Menge $F \subseteq Q$) erreicht ist. Dieser Akzeptanzbegriff liefert aber dieselbe Sprachklasse.

Wir werden nun zeigen, dass man mit Kellerautomaten genau die kontextfreien Sprachen akzeptieren kann.

Satz 10.5

Für eine formale Sprache L sind äquivalent:

1) $L = L(G)$ *für eine kontextfreie Grammatik* G.

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

Beweis.

„1 $\longrightarrow$ 2“: Es sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik. Der zugehörige PDA simuliert Linksableitungen von G, d.h. Ableitungen, bei denen stets das am weitesten links stehende Nichtterminalsymbol zuerst ersetzt wird.

Beachte:

Jedes Wort in $L(G)$ kann durch eine Linksableitung erzeugt werden (da G kontextfrei ist).

Wir definieren dazu $\mathcal{A} = (\{q\}, \Sigma, \Sigma \cup N, q, S, \Delta)$ mit $\Delta :=$

$\{(q, \varepsilon, A, \gamma, q) \mid A \longrightarrow \gamma \in P\} \cup$	(Anwenden einer Produktion auf oberstes Kellersymbol) $(\star)$
$\{(q, a, a, \varepsilon, q) \mid a \in \Sigma\}$	(Entfernen bereits erzeugter Terminalsymbole von der Kellerspitze, wenn sie in der Eingabe vorhanden sind) $(\star\star)$

Beispiel:

$P = \{S \longrightarrow \varepsilon,\ S \longrightarrow aSa,\ S \longrightarrow bSb\}$ liefert die Übergänge:

$(q,$	$\varepsilon,$	$S,$	$\varepsilon,$	$q),$	$(S \longrightarrow \varepsilon)$
$(q,$	$\varepsilon,$	$S,$	$aSa,$	$q),$	$(S \longrightarrow aSa)$
$(q,$	$\varepsilon,$	$S,$	$bSb,$	$q),$	$(S \longrightarrow bSb)$
$(q,$	$a,$	$a,$	$\varepsilon,$	$q),$	(a entfernen)
$(q,$	$b,$	$b,$	$\varepsilon,$	$q)$	(b entfernen)

Die Ableitung $S \vdash aSa \vdash abSba \vdash abba$ entspricht der *Konfigurationsfolge*

$$(q, abba, S) \vdash (q, abba, aSa) \vdash (q, bba, Sa) \vdash (q, bba, bSba) \vdash$$
$$(q, ba, Sba) \vdash (q, ba, ba) \vdash (q, a, a) \vdash (q, \varepsilon, \varepsilon)$$

Behauptung:

Für $u, v \in \Sigma^*$ und $\alpha \in \{\varepsilon\} \cup N \cdot (\Sigma \cup N)^*$ gilt:

$$S \vdash_G^* u\alpha \text{ mit Linksableitung} \quad \underline{\text{gdw.}} \quad (q, uv, S) \vdash^* (q, v, \alpha).$$

Beachte:

Für $\alpha = \varepsilon = v$ folgt:

$$S \vdash_G^* u \quad \underline{\text{gdw}}. \quad (q, u, S) \vdash^* (q, \varepsilon, \varepsilon)$$

d.h. $L(G) = L(\mathcal{A})$.

Beweis der Behauptung.

„⇐“: Beweis durch Induktion über die Anzahl k der Übergänge mit Transitionen der Form $(\star) : (q, \varepsilon, A, \gamma, q)$.

$k = 0$:

Da im Keller S steht, sind auch keine Transitionen der Form $(\star\star) : (q, a, a, \varepsilon, q)$ möglich, d.h. $u = \varepsilon$ und $\alpha = S$. Offenbar gilt $S \vdash^* S$.

$k \to k + 1$:

$$(q, \underbrace{u_1 u_2}_{u} v, S) \vdash^* (q, u_2 v, A\alpha') \overset{\text{letzte Trans. } (\star)}{\vdash} (q, u_2 v, \underbrace{\gamma\alpha'}_{u_2\alpha}) \overset{\text{Trans. } (\star\star)}{\vdash^*} (q, v, \alpha)$$

Induktion liefert: $S \vdash^* u_1 A\alpha'$ und außerdem wegen $A \longrightarrow \gamma \in P$:

$$u_1 A\alpha' \vdash u_1\gamma\alpha' = u_1 u_2 \alpha = u\alpha.$$

„⇒“: Beweis durch Induktion über die Länge der Linksableitung

$k = 0$:

Dann ist $u = \varepsilon,\ \alpha = S$ und $(q, v, S) \vdash^0 (q, v, S)$

$k \to k + 1$:

$S \vdash_G^{k+1} u\alpha$, d.h.

$S \vdash_G^k u' A\beta \vdash \underbrace{u'\gamma\beta}_{=u\alpha}$ mit $u' \in \Sigma^*$ (da Linksableitung) und $A \longrightarrow \gamma \in P$.

Es sei u'' das längste Anfangsstück von $\gamma\beta$, das in Σ^* liegt. Dann ist $u'u'' = u$ und α ist der Rest von $\gamma\beta$, d.h. $\gamma\beta = u''\alpha$.

Induktion liefert:

$$(q, u' \underbrace{u''v}_{v'}, S) \vdash^* (q, u''v, A\beta) \vdash (q, u''v, \gamma\beta) = (q, u''v, u''\alpha).$$

Außerdem gibt es Übergänge $(q, u''v, u''\alpha) \vdash^* (q, v, \alpha)$. □

„$2 \longrightarrow 1$“: Es sei $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, Z_0, \Delta)$ ein PDA. Die Nichtterminalsymbole der zugehörigen Grammatik G sind alle Tripel $[p, Z, q] \in Q \times \Gamma \times Q$.

Idee:

Es soll gelten: $[p, Z, q] \vdash_G^* u \in \Sigma^*$ $\underline{\text{gdw}}$.

- $\mathcal{A}$ kommt *vom Zustand p in den Zustand q*
- durch *Lesen von u* auf dem Eingabeband und
- *Löschen von Z* aus dem Keller

(ohne dabei die Symbole unter Z anzutasten).

Wie realisiert man dies durch geeignete Produktionen?

Betrachte den PDA-Übergang $(p, a, Z, X_1 \dots X_n, p') \in \Delta$. Hier wird a auf dem Eingabeband verbraucht (falls nicht $a = \varepsilon$). Z wird durch $X_1 \dots X_n$ ersetzt.
Um den Kellerinhalt zu erreichen, den man durch einfaches Wegstreichen von Z erhalten würde, muss man also nun noch $X_1 \dots X_n$ löschen. Löschen von X_i kann durch das Nichtterminalsymbol $[p_{i-1}, X_i, p_i]$ (für geeignete Zwischenzustände p_{i-1}, p_i) beschrieben werden.

Formale Definition:

$$
\begin{aligned}
G &:= (N, \Sigma, P, S) \text{ mit} \\
N &:= \{S\} \cup \{[p, Z, q] \mid p, q \in Q, Z \in \Gamma\} \\
P &:= \{S \longrightarrow [q_0, Z_0, q] \mid q \in Q\} \cup \\
&\quad \{[p, Z, q] \longrightarrow a[p_0, X_1, p_1][p_1, X_2, p_2] \dots [p_{n-1}, X_n, q] \mid \\
&\qquad p, q, p_0, p_1, \dots, p_{n-1} \in Q, \\
&\qquad a \in \Sigma \cup \{\varepsilon\} \text{ und} \\
&\qquad (p, a, Z, X_1 \dots X_n, p_0) \in \Delta \text{ und} \\
&\qquad q = p_0 \text{ falls } n = 0\}
\end{aligned}
$$

Beachte:

Für $n = 0$ hat man die Produktion $[p, Z, q] \longrightarrow a$, welche dem Übergang $(p, a, Z, \varepsilon, q) \in \Delta$ entspricht.

Behauptung:

Für alle $p, q \in Q, u \in \Sigma^*, Z \in \Gamma$ gilt:

$$(\star) \qquad (p, u, Z) \vdash_{\mathcal{A}}^* (q, \varepsilon, \varepsilon) \;\; \underline{\text{gdw}}. \;\; [p, Z, q] \vdash_G^* u$$

Für $p = p_0$ und $Z = Z_0$ folgt daraus:

$$(q_0, u, Z_0) \vdash^* (q, \varepsilon, \varepsilon) \;\; \underline{\text{gdw}}. \;\; S \vdash_G [q_0, Z_0, q] \vdash_G^* u$$

d.h. $u \in L(\mathcal{A}) \;\; \underline{\text{gdw}}. \;\; u \in L(G)$.

Der Beweis dieser Behauptung kann durch Induktion über die Länge der Konfigurationsfolge („$\Rightarrow$“) bzw. über die Länge der Ableitung („$\Leftarrow$“) geführt werden.

□

Beispiel: (vgl. Beispiel 10.3)
Gegeben ist der PDA für $\{a^n b^n \mid n \geq 1\}$.

$$(q_0, ab, Z_0) \overset{(q_0, a, Z_0, ZZ_0, q_0)}{\vdash} (q_0, b, ZZ_0) \overset{(q_0, b, Z, \varepsilon, q_1)}{\vdash} (q_1, \varepsilon, Z_0) \overset{(q_1, \varepsilon, Z_0, \varepsilon, f)}{\vdash} (f, \varepsilon, \varepsilon)$$

entspricht der Ableitung

$$S \vdash_G [q_0, Z_0, f] \vdash_G a[q_0, Z, q_1][q_1, Z_0, f] \vdash_G ab[q_1, Z_0, f] \vdash_G ab.$$

III Turingmaschinen und Grammatiken

Einführung

Aus der Sicht der Theorie der *formalen Sprachen* geht es in diesem Teil darum, Automatenmodelle für die Typ-0- und die Typ-1-Sprachen zu finden. Allgemeiner geht es aber darum, die intuitiven Begriffe „*berechenbare* Funktion“ und „*entscheidbares* Problem“ zu formalisieren.

Um für eine (eventuell partielle) Funktion

$$\mathrm{f} : \mathbb{N}^k \to \mathbb{N} \quad (\text{bzw. } (\Sigma^*)^k \to \Sigma^*)$$

intuitiv klarzumachen, dass sie berechenbar ist, genügt es, ein Berechnungsverfahren (einen *Algorithmus*) für die Funktion anzugeben (z.B. in Form eines Modula-, Pascal- oder C-*Programmes* oder einer *abstrakten Beschreibung* der Vorgehensweise bei der Berechnung). Entsprechend hatten wir in den Teilen I und II die Entscheidbarkeit von Problemen (Wortproblem, Leerheitsproblem, Äquivalenzproblem, ...) dadurch begründet, dass wir *intuitiv* beschrieben haben, wie man die Probleme mit Hilfe eines Rechenverfahrens (d.h. *effektiv*) entscheiden kann. Aus dieser Beschreibung hätte man jeweils ein Modula-, Pascal-, etc. -Programm zur Entscheidung des Problems gewinnen können.

Beim Nachweis der Nichtentscheidbarkeit bzw. Nichtberechenbarkeit ist eine solche intuitive Vorgehensweise nicht mehr möglich, da man hier formal nachweisen muss, dass es *kein Berechnungsverfahren geben kann*. Damit ein solcher Beweis durchführbar ist, benötigt man eine formale Definition dessen, was man unter einem Berechnungsverfahren versteht. Man will also ein *Berechnungsmodell*, das

1) **einfach** ist, damit formale Beweise erleichtert werden (z.B. nicht Programmiersprache ADA),
2) **berechnungsuniversell** ist, d.h. alle intuitiv berechenbaren Funktionen damit berechnet werden können (z.B. nicht nur endliche Automaten).

Wir werden in diesem Teil ein derartiges Modell betrachten:

- Turingmaschinen

Es gibt noch eine Vielzahl anderer Modelle:

- μ-rekursive Funktionen
- WHILE-Programme
- Minskymaschinen
- GOTO-Programme
- URMs (unbeschränkte Registermaschinen)
- Pascal-Programme
- ...

Es hat sich herausgestellt, dass all diese Modelle *äquivalent* sind, d.h. die gleiche Klasse von Funktionen berechnen. Außerdem ist es bisher nicht gelungen, ein formales Berechnungsmodell zu finden, so dass

- die dadurch berechneten Funktionen noch intuitiv berechenbar erscheinen,
- dadurch Funktionen berechnet werden können, die nicht in den oben genannten Modellen ebenfalls berechenbar sind.

Aus diesen beiden Gründen geht man davon aus, dass die genannten Modelle genau den intuitiven Berechenbarkeitsbegriff formalisieren. Diese Überzeugung nennt man die:

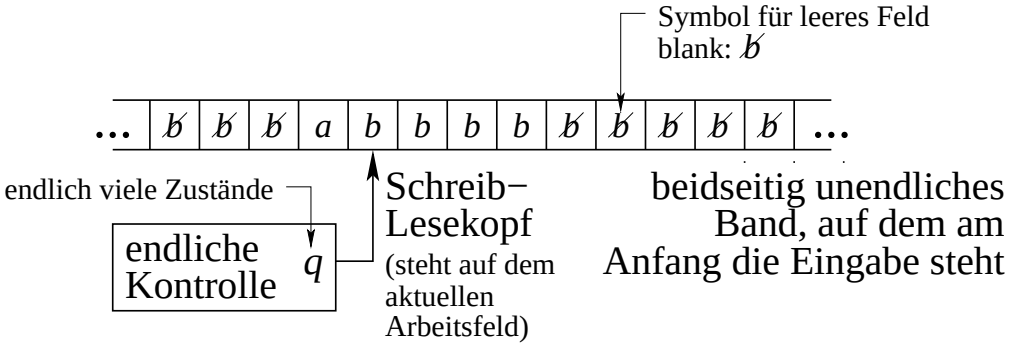
Churchsche These:
Die (intuitiv) berechenbaren Funktionen sind genau die mit Turingmaschinen (und damit mit GOTO-, WHILE-, Pascal-Programmen, URMs, ...) berechenbaren Funktionen.

Man spricht hier von einer *These* und nicht von einem *Satz*, da es nicht möglich ist, diese Aussage formal zu beweisen. Dies liegt daran, dass der intuitive Berechenbarkeitsbegriff ja nicht formal definierbar ist. Es gibt aber sehr viele Indizien, die für die Richtigkeit der These sprechen (Vielzahl äquivalenter Berechnungsmodelle).

Im Folgenden betrachten wir:

- Turingmaschinen
- Zusammenhang zwischen Turingmaschinen und Grammatiken

11 Turingmaschinen



Zu jedem Zeitpunkt sind nur *endlich viele* Symbole auf dem Band verschieden von $\not b$.

Definition 11.1 (Turingmaschine)
Eine *Turingmaschine über dem Eingabealphabet* Σ hat die Form $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \Delta, F)$, wobei

- Q endliche *Zustandsmenge* ist,
- Σ das *Eingabealphabet* ist,
- Γ das *Arbeitsalphabet* ist mit $\Sigma \subseteq \Gamma$, $\not b \in \Gamma \setminus \Sigma$,
- $q_0 \in Q$ der *Anfangszustand* ist,
- $F \subseteq Q$ die *Endzustandsmenge* ist und
- $\Delta \subseteq Q \times \Gamma \times \Gamma \times \{r, l, n\} \times Q$ die *Übergangsrelation* ist.

Dabei bedeutet $(q, a, a', \begin{matrix} r \\ l \\ n \end{matrix}, q')$:

- Im Zustand q
- mit a auf dem gerade gelesenen Feld (Arbeitsfeld)

kann die Turingmaschine $\mathcal{A}$

- das Symbol a durch a' ersetzen,
- in den Zustand q' gehen und
- den Schreib-Lesekopf entweder um ein Feld nach rechts (r), links (l) oder nicht (n) bewegen.

Die Maschine $\mathcal{A}$ heißt *deterministisch*, falls es für jedes Tupel $(q, a) \in Q \times \Gamma$ höchstens ein Tupel der Form $(q, a, \ldots, \ldots) \in \Delta$ gibt.

NTM steht im folgenden für (möglicherweise nichtdeterministische) Turingmaschinen und *DTM* für deterministische.

Einen Berechnungszustand (*Konfiguration*) einer Turingmaschine kann man beschreiben durch ein Wort $\alpha q \beta$ mit $\alpha, \beta \in \Gamma^+,\ q \in Q$:

- q ist der momentane Zustand
- α ist die Beschriftung des Bandes links vom Arbeitsfeld
- β ist die Beschriftung des Bandes beginnend beim Arbeitsfeld nach rechts

Dabei werden (um endliche Wörter $\alpha,\ \beta$ zu erhalten) unendlich viele Blanks weggelassen, d.h. α undβ umfassen mindestens den Bandabschnitt, auf dem Symbole $\neq \not b$ stehen.

Beispiel:

Der Zustand der Maschine zu Beginn des Abschnitts wird durch die Konfiguration *aqbbbb*, aber auch durch $\not b \not b aqbbbb \not b$ beschrieben.

Die Übergangsrelation Δ ermöglicht die folgenden *Konfigurationsübergänge*:

Es seien $\alpha, \beta \in \Gamma^+,\ \beta' \in \Gamma^*,\ a, b, a' \in \Gamma,\ q, q' \in Q$.

- $\left.\begin{array}{rcl} \alpha q a \beta & \vdash_{\mathcal{A}} & \alpha a' q' \beta \\ \alpha q a & \vdash_{\mathcal{A}} & \alpha a' q' \not b \end{array}\right\}$ falls $(q, a, a', r, q') \in \Delta$
- $\left.\begin{array}{rcl} \alpha b q a \beta & \vdash_{\mathcal{A}} & \alpha q' b a' \beta \\ b q a \beta & \vdash_{\mathcal{A}} & \not b\, q' b a' \beta \end{array}\right\}$ falls $(q, a, a', l, q') \in \Delta$
- $\alpha q a \beta \ \vdash_{\mathcal{A}}\ \alpha q' a' \beta$ falls $(q, a, a', n, q') \in \Delta$

Weitere Bezeichnungen:

- Gilt $k \vdash_{\mathcal{A}} k'$, so heißt k' *Folgekonfiguration* von k.
- Die Konfiguration $\alpha q \beta$ heißt *akzeptierend*, falls $q \in F$.
- Die Konfiguration $\alpha q \beta$ heißt *Stoppkonfiguration*, falls sie keine Folgekonfiguration hat.
- Die von $\mathcal{A}$ *akzeptierte Sprache* ist
  $L(\mathcal{A}) = \{w \in \Sigma^* \mid \not b\, q_0 w \not b \vdash^*_{\mathcal{A}} k,$ wobei k *akzeptierende Stoppkonfiguration* ist$\}$.

Definition 11.2 (Turing-akzeptierbar, Turing-berechenbar)

1) Die Sprache $L \subseteq \Sigma^*$ heißt *Turing-akzeptierbar*, falls es eine NTM $\mathcal{A}$ gibt mit $L = L(\mathcal{A})$.
2) Die (partielle) Funktion $\mathrm{f} : (\Sigma^*)^n \to \Sigma^*$ heißt *Turing-berechenbar*, falls es eine DTM $\mathcal{A}$ gibt mit
   - $\forall x_1, \ldots, x_n \in \Sigma^* : \not b\, q_0 x_1 \not b \ldots \not b\, x_n \not b \vdash^*_{\mathcal{A}} k$ mit k Stoppkonfiguration gdw. $(x_1, \ldots, x_n) \in \mathrm{dom(f)}$ (Definitionsbereich von f).
   - Im Fall $(x_1, \ldots, x_n) \in \mathrm{dom(f)}$ muss die Beschriftung des Bandes in der Stoppkonfiguration k rechts vom Schreib-Lesekopf bis zum ersten Symbol $\notin \Sigma$ der Wert $y = \mathrm{f}(x_1, \ldots, x_n)$ der Funktion sein, d.h. k muss die Form $uqyv$ haben mit
     - $v \in (\Gamma \setminus \Sigma) \cdot \Gamma^* \cup \{\varepsilon\}$
     - $y = \mathrm{f}(x_1, \ldots, x_n)$

Beachte:

1) *Undefiniertheit* des Funktionswertes von f entspricht der Tatsache, dass die Maschine bei dieser Eingabe nicht terminiert.
2) Bei berechenbaren Funktionen betrachten wir nur *deterministische* Maschinen, da sonst der Funktionswert nicht eindeutig sein müsste.
3) Bei $|\Sigma| = 1$ kann man Funktionen von $(\Sigma^*)^n \to \Sigma^*$ als Funktionen von $\mathbb{N}^k \to \mathbb{N}$ auffassen (a^k entspricht k).

Beispiel:
Die Funktion

$$\mathrm{f} : \mathbb{N} \to \mathbb{N}, \quad n \mapsto 2n$$

ist Turing-berechenbar. Wie kann eine Turingmaschine die Anzahl der a's auf dem Band verdoppeln?

Idee:

- Ersetze das erste a durch b,
- laufe nach rechts bis zum ersten blank, ersetze dieses durch c,
- laufe zurück bis zum zweiten a (unmittelbar rechts vom b), ersetze dieses durch b,
- laufe nach rechts bis zum ersten blank etc.
- Sind alle a's aufgebraucht, so ersetze noch die b's und c's wieder durch a's.

Dies wird durch die folgende *Übergangstafel* realisiert:

$(q_0, \not b, \not b, n, \text{stop})$,	$2 \cdot 0 = 0$
(q_0, a, b, r, q_1),	ersetze a durch b $\quad(\star)$
(q_1, a, a, r, q_1), (q_1, c, c, r, q_1),	laufe nach rechts über a's und bereits geschriebene c's
$(q_1, \not b, c, n, q_2)$,	schreibe weiteres c
(q_2, c, c, l, q_2), (q_2, a, a, l, q_2),	laufe zurück über c's und a's
(q_2, b, b, r, q_0), (q_0, c, c, r, q_3),	bei erstem b eins nach rechts und weiter wie $(\star)$ oder alle a's bereits ersetzt
(q_3, c, c, r, q_3), $(q_3, \not b, \not b, l, q_4)$,	laufe nach rechts bis Ende der c's letztes c erreicht
(q_4, c, a, l, q_4), (q_4, b, a, l, q_4),	ersetze c's und b's durch a's
$(q_4, \not b, \not b, r, \text{stop})$	bleibe am Anfang der erzeugten $2n$ a's stehen

Man sieht hier, dass das Programmieren von Turingmaschinen sehr umständlich ist. Wie bereits erwähnt, betrachtet man solche einfachen (und unpraktischen) Modelle, um das Führen von Beweisen zu erleichtern. Wir werden im folgenden häufig nur die Arbeitsweise einer Turingmaschine beschreiben, ohne die Übergangstafel voll anzugeben.

Beispiel:
Die Sprache $L = \{a^n b^n c^n \mid n \geq 0\}$ ist Turing-akzeptierbar. Die Turingmaschine, welche L akzeptiert, geht wie folgt vor:

- Sie ersetzt das erste a durch a', das erste b durch b' und das erste c durch c';
- läuft zurück zum zweiten a, ersetzt es durch a', das zweite b durch b' und das zweite c durch c' etc.
- Dies wird solange gemacht, bis nach erzeugtem c' ein $\not b$ steht.
- Danach wird von rechts nach links geprüft, ob – in dieser Reihenfolge – nur noch ein c'-Block, dann ein b'-Block und dann ein a'-Block (abgeschlossen durch $\not b$) vorhanden ist.
- Die Maschine blockiert, falls dies nicht so ist. Die kann auch bereits vorher geschehen, wenn erwartetes a, b oder c nicht gefunden wird.

Eine entsprechende Turingmaschine $\mathcal{A}$ kann z.B. wie folgt definiert werden:
$\mathcal{A} = (\{q_0, q_{akz}, finde\_b, finde\_c, zu\_Ende\_?, zu\_Ende\_!, zurück\}$,
$\{a, b, c\}$,
$\{a, a', b, b', c, c', \not b\}$,
$q_0, \Delta, \{q_{akz}\})$ mit $\Delta =$

$\{(q_0,$	$\not b,$	$\not b,$	$N,$	$q_{akz}),$
$(q_0,$	$a,$	$a',$	$R,$	$finde\_b),$
$(finde\_b,$	$a,$	$a,$	$R,$	$finde\_b),$
$(finde\_b,$	$b',$	$b',$	$R,$	$finde\_b),$
$(finde\_b,$	$b,$	$b',$	$R,$	$finde\_c),$
$(finde\_c,$	$b,$	$b,$	$R,$	$finde\_c),$
$(finde\_c,$	$c',$	$c',$	$R,$	$finde\_c),$
$(finde\_c,$	$c,$	$c',$	$R,$	$zu\_Ende\_?),$
$(zu\_Ende\_?,$	$c,$	$c,$	$L,$	$zurück),$
$(zurück,$	$c',$	$c',$	$L,$	$zurück),$
$(zurück,$	$b,$	$b,$	$L,$	$zurück),$
$(zurück,$	$b',$	$b',$	$L,$	$zurück),$
$(zurück,$	$a,$	$a,$	$L,$	$zurück),$
$(zurück,$	$a',$	$a',$	$R,$	$q_0),$
$(zu\_Ende\_?,$	$\not b,$	$\not b,$	L	$zu\_Ende\_!),$
$(zu\_Ende\_!,$	$c',$	$c',$	$L,$	$zu\_Ende\_!),$
$(zu\_Ende\_!,$	$b',$	$b',$	$L,$	$zu\_Ende\_!),$
$(zu\_Ende\_!,$	$a',$	$a',$	$L,$	$zu\_Ende\_!),$
$(zu\_Ende\_!,$	$\not b,$	$\not b,$	$N,$	$q_{akz})\}$

Varianten von Turingmaschinen:
In der Literatur werden verschiedene Versionen der Definition der Turingmaschine angegeben, die aber alle äquivalent zueinander sind, d.h. dieselben Sprachen akzeptieren und dieselben Funktionen berechnen. Hier zwei Beispiele:

- Turingmaschinen mit nach links begrenztem und nur nach rechts unendlichem Arbeitsband
- Turingmaschinen mit mehreren Bändern und Schreib-Leseköpfen

Wir betrachten das zweite Beispiel genauer und zeigen Äquivalenz zur in Definition 11.1 eingeführten 1-Band-TM.

Definition 11.3 (k-Band-TM)
Eine *k-Band-NTM* hat die Form $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \Delta, F)$ mit

- $Q, \Sigma, \Gamma, q_0, F$ wie in Definition 11.1 und
- $\Delta \subseteq Q \times \Gamma^k \times \Gamma^k \times \{r, l, n\}^k \times Q$.

Dabei bedeutet $(q, (a_1, \ldots, a_k), (b_1, \ldots, b_k), (d_1, \ldots, d_k), q') \in \Delta$:

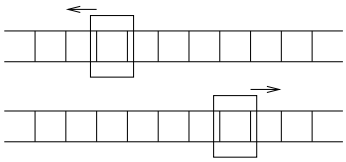
- Vom Zustand q aus
- mit $a_1, \ldots, a_k$ auf den Arbeitsfeldern der k Bänder

kann $\mathcal{A}$

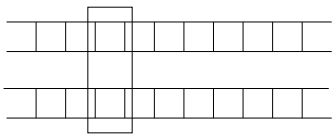
- das Symbol a_i auf dem i-ten Band durch b_i ersetzen,
- in den Zustand q' gehen und
- die Schreib-Leseköpfe der Bänder entsprechend d_i bewegen.

Das erste Band wird (o.B.d.A.) als Ein- und Ausgabeband verwendet.

Beachte:
Wichtig ist hier, dass sich die Köpfe der verschiedenen Bänder auch verschieden bewegen können:



Wären die Köpfe gekoppelt, so hätte man im Prinzip nicht mehrere Bänder, sondern ein Band mit mehreren Spuren:



k Spuren erhält man einfach, indem man eine normale NTM (nach Definition 11.1) verwendet, die als Bandalphabet Γ^k statt Γ hat.

Offenbar kann man jede 1-Band-NTM (nach Definition 11.1) durch eine k-Band-NTM ($k > 1$) simulieren, indem man nur das erste Band wirklich verwendet. Der nächste Satz zeigt, dass auch die Umkehrung gilt:

Satz 11.4
Wird die Sprache L durch eine k-Band-NTM akzeptiert, so auch durch eine 1-Band-NTM.

Beweis. Es sei $\mathcal{A}$ eine k-Band-NTM. Gesucht ist

- eine 1-Band-NTM $\mathcal{A}'$ und
- eine Kodierung $k \mapsto k'$ der Konfigurationen k von $\mathcal{A}$ durch Konfigurationen k' von $\mathcal{A}'$,

so dass gilt:

$$k_1 \vdash_{\mathcal{A}} k_2 \quad \underline{\text{gdw}}. \quad k_1' \vdash_{\mathcal{A}'}^* k_2'$$

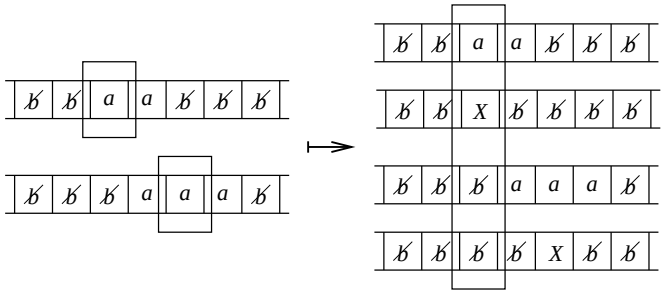
(Zur Simulation eines Schrittes von $\mathcal{A}$ sind mehr Schritte nötig).

Arbeitsalphabet von $\mathcal{A}'$: $\Gamma^{2k} \cup \Sigma \cup \{\not b\}$;

- $\Sigma \cup \{\not b\}$ wird für Eingabe benötigt;
- Γ^{2k} sorgt dafür, dass sich $\mathcal{A}'$ wie eine 1-Band-NTM mit $2k$ Spuren verhält.

Kodierung: Jeweils 2 Spuren kodieren ein Band von $\mathcal{A}$.

- Die erste Spur enthält die Bandbeschriftung.
- Die zweite Spur enthält eine Markierung X (und sonst Blanks), die zeigt, wo das Arbeitsfeld des Bandes ist, z.B.



Hierbei wird angenommen, dass das Arbeitsfeld von $\mathcal{A}'$ stets bei dem am weitesten links stehenden X liegt.

Initialisierung: Zunächst wird die Anfangskonfiguration

$$\not{b} q_0 a_1 \dots a_m \not{b}$$

von $\mathcal{A}'$ in die *Kodierung* der entsprechenden Anfangskonfiguration von $\mathcal{A}$ umgewandelt (durch entsprechende Übergänge):

$\not{b}$	$\not{b}$	a_1	a_2	$\dots$	a_m	$\not{b}$	Spur 1 und 2 kodieren
$\not{b}$	$\not{b}$	X	$\not{b}$	$\dots$	$\not{b}$	$\not{b}$	Band 1
$\not{b}$	$\not{b}$	$\not{b}$	$\not{b}$	$\dots$	$\not{b}$	$\not{b}$	Spur 3 und 4 kodieren
$\not{b}$	$\not{b}$	X	$\not{b}$	$\dots$	$\not{b}$	$\not{b}$	Band 2
		$\vdots$					

Simulation der Übergänge: Betrachte $(q, (a_1, \dots, a_k), (b_1, \dots, b_k), (d_1, \dots, d_k), q') \in \Delta$.

- Von links nach rechts suche die mit X markierten Felder. Dabei merke man sich (in dem Zustand der TM) die Symbole, welche jeweils über dem X stehen. Außerdem zählt man (durch Zustand der TM) mit, wie viele X man schon gelesen hat, um festzustellen, wann das k-te erreicht ist. Man bestimmt so, ob das aktuelle Tupel tatsächlich $(a_1, \dots, a_k)$ ist.
- Von rechts nach links gehend überdrucke man die a_i bei den X-Marken jeweils durch das entsprechende b_i und verschiebt die X-Marken gemäß d_i.
- Bleibe bei der am weitesten links stehenden X-Markierung, lasse den Kopf dort stehen und gehe in Zustand q'.

□

Bemerkung 11.5.

1) War $\mathcal{A}$ deterministisch, so liefert obige Konstruktion auch eine deterministische 1-Band-Turingmaschine.
2) Diese Konstruktion kann auch verwendet werden, wenn man sich für die berechnete Funktion interessiert. Dazu muss man am Schluss (wenn $\mathcal{A}$ in Stoppkonfiguration ist) in der Maschine $\mathcal{A}'$ noch die Ausgabe geeignet aufbereiten.

Bei der Definition von Turing-berechenbar haben wir uns von vornherein auf *deterministische* Turingmaschinen beschränkt. Der folgende Satz zeigt, dass man dies auch bei Turing-akzeptierbaren Sprachen machen kann, ohne an Ausdrucksstärke zu verlieren.

Satz 11.6
Zu jeder NTM gibt es eine DTM, die dieselbe Sprache akzeptiert.

Beweis. Wegen Satz 11.4 und Bemerkung 11.5 genügt es, eine deterministische 3-Band-Turingmaschine zu konstruieren. Es sei $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \Delta, F)$ eine NTM.

Die Maschine $\mathcal{A}'$ soll für wachsendes n auf dem dritten Band jeweils alle Konfigurationsfolgen

$$k_0 \vdash_{\mathcal{A}} k_1 \vdash_{\mathcal{A}} k_2 \vdash_{\mathcal{A}} \ldots \vdash_{\mathcal{A}} k_n$$

beginnend mit der Startkonfiguration $k_0 = \not{b}\, q_0 w \not{b}$ erzeugen.

Die Kontrolle, dass tatsächlich alle solchen Folgen erzeugt werden, wird auf dem zweiten Band vorgenommen. Das erste Band speichert das Eingabewort w (damit man stets weiß, was k_0 sein muss).

Genauer: Es sei

$$r = \text{ maximale Anzahl von Transitionen in } \Delta \text{ pro festem Paar } (q, a) \in Q \times \Gamma$$

(entspricht dem maximalen Verzweigungsgrad der nichtdeterministischen Berechnung).

Eine Indexfolge $i_1, \ldots, i_n$ mit $i_j \in \{1, \ldots, r\}$ bestimmt dann von k_0 aus für n Schritte die Auswahl der jeweiligen Transition, und somit von k_0 aus eine feste Konfigurationsfolge

$$k_0 \vdash_{\mathcal{A}} k_1 \vdash_{\mathcal{A}} \ldots \vdash_{\mathcal{A}} k_n$$

Zählt man daher alle endlichen Wörter über $\{1, \ldots, r\}$ auf und zu jedem Wort $i_1 \ldots i_n$ die zugehörige Konfigurationsfolge, so erhält man eine Aufzählung aller endlichen Konfigurationsfolgen.

$\mathcal{A}'$ realisiert dies auf den drei Bändern wie folgt:

- Auf Band 1 bleibt die Eingabe gespeichert.
- Auf dem zweiten Band werden sukzessive alle Wörter $i_1 \ldots i_n \in \{1, \ldots, r\}^*$ erzeugt.
- Für jedes dieser Wörter wird auf dem dritten Band die zugehörige Konfigurationsfolge realisiert. Erreicht man hierbei eine akzeptierende Konfiguration von $\mathcal{A}$, so geht auch $\mathcal{A}'$ in eine akzeptierende Konfiguration.

□

12 Zusammenhang zwischen Turingmaschinen und Grammatiken

Wir zeigen zunächst den Zusammenhang zwischen den *Typ-0-Sprachen* und *Turing-akzeptierbaren Sprachen*. Dieser beruht darauf, dass es eine Entsprechung von Ableitungen einer Typ-0-Grammatik einerseits und Konfigurationsfolgen von Turingmaschinen andererseits gibt. Beim Übergang von der Turingmaschine zur Grammatik dreht sich allerdings die Richtung um:

- eine akzeptierende Konfigurationsfolge beginnt mit dem zu akzeptierenden Wort
- eine Ableitung endet mit dem erzeugten Wort

Satz 12.1

Eine Sprache L gehört zu $\mathcal{L}_0$ gdw. sie Turing-akzeptierbar ist.

Beweis.

„⇒“: Es sei $L = L(G)$ für eine Typ-0-Grammatik $G = (N, \Sigma, P, S)$ mit $|P| = k$ Regeln. Wir geben eine 2-Band-NTM an, die $L(G)$ akzeptiert (die nach Satz 11.4 äquivalent zu einer 1-Band-NTM ist).

1. Band: speichert Eingabe w

2. Band: ausgehend von S werden gemäß den Regeln von G ableitbare Wörter gebildet

Es wird verglichen, ob auf Band 2 irgendwann w (d.h. der Inhalt von Band 1) entsteht. Wenn ja, so geht man in akzeptierende Stoppkonfiguration, sonst sucht die Turingmaschine weiter.

Die Maschine geht dabei wie folgt vor:

1) Schreibe S auf Band 2, gehe nach links auf das $\not b$ vor S.

2) Gehe auf Band 2 nach rechts und wähle (nichtdeterministisch) eine Stelle aus, an der die linke Seite der anzuwendenden Produktion beginnen soll.

3) Wechsle nun (nichtdeterministisch) in einen der Zustände $\text{Regel}_1, \ldots, \text{Regel}_k$ (entspricht der Entscheidung, dass man diese Regel anwenden will).

4) Es sei Regel_i der gewählte Zustand und $\alpha \longrightarrow \beta$ die entsprechende Produktion.

   Überprüfe, ob α tatsächlich die Beschriftung des Bandstücks der Länge $|\alpha|$ ab der gewählten Bandstelle ist.

5) Falls der Test erfolgreich war, so ersetze α durch β.

   Vorher müssen bei $|\alpha| < |\beta|$ die Symbole $\neq \not b$ rechts von α um $|\beta| - |\alpha|$ Positionen nach rechts

   bzw. bei $|\alpha| > |\beta|$ um $|\alpha| - |\beta|$ Positionen nach links geschoben werden.

6) Gehe nach links bis zum ersten $\not b$ und vergleiche, ob die Beschriftung auf dem Band 1 mit der auf Band 2 übereinstimmt.

7) Wenn ja, so gehe in akzeptierenden Stoppzustand. Sonst fahre fort bei 2).

„⇐“: Es sei $L = L(\mathcal{A})$ für eine NTM $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \Delta, F)$.

Wir konstruieren eine Grammatik G, die jedes Wort $w \in L(\mathcal{A})$ wie folgt erzeugt:

1. Phase: Erst wird w mit „genügend vielen“ $\not b$-Symbolen links und rechts davon erzeugt (dies passiert für jedes w, auch für $w \notin L(\mathcal{A})$).
„Genügend viele“ bedeutet dabei: so viele, wie $\mathcal{A}$ beim Akzeptieren von w vom Arbeitsband benötigt.

2. Phase: Auf dem so erzeugten Arbeitsband simuliert G die Berechnung von $\mathcal{A}$ bei Eingabe w.

3. Phase: War die Berechnung akzeptierend, so erzeuge nochmals das w.

Damit man in der zweiten Phase das in der dritten Phase benötigte w nicht vergisst, verwendet man als Nichtterminalsymbole Tupel aus

$$(\Sigma \cup \{\not b\}) \times \Gamma$$

wobei man

- sich in der ersten Komponente w merkt und
- in der zweiten Komponente die Berechnung simuliert.

Formale Definition:

$$N = \{S, A, B, E\} \cup Q \cup (\Sigma \cup \{\not b\}) \times \Gamma,$$

wobei

- S, A, B zum Aufbau des Rechenbandes am Anfang,
- E zum Löschen am Schluss,
- Q zur Darstellung des aktuellen Zustandes,
- $\Sigma \cup \{\not b\}$ zum Speichern von w und
- Γ zur $\mathcal{A}$-Berechnung

dienen.

Regeln:

1. Phase: Erzeuge w und ein genügend großes Arbeitsband.

$$\begin{aligned} S &\longrightarrow Bq_0A \\ A &\longrightarrow [a, a]A \text{ für alle } a \in \Sigma \\ A &\longrightarrow B \\ B &\longrightarrow [\not b, \not b]B \\ B &\longrightarrow \varepsilon \end{aligned}$$

Man erhält also somit für alle $a_1 \dots a_n \in \Sigma^*, k, l, \geq 0$:

$$S \vdash_G^* [\not b, \not b]^k q_0 [a_1, a_1] \dots [a_n, a_n] [\not b, \not b]^l$$

2. Phase: simuliert TM-Berechnung in der „zweiten Spur“:

- $p[a,b] \longrightarrow [a,b']q$
  falls $(p,b,b',r,q) \in \Delta,\ a \in \Sigma \cup \{\not b\}$
- $[a,c]p[a',b] \longrightarrow q[a,c][a',b']$
  falls $(p,b,b',l,q) \in \Delta,\ a,a' \in \Sigma \cup \{\not b\},\ c \in \Gamma$
- $p[a,b] \longrightarrow q[a,b']$
  falls $(p,b,b',n,q) \in \Delta,\ a \in \Sigma \cup \{\not b\}$

Beachte:

Da wir in der ersten Phase genügend viele Blanks links und rechts von $a_1 \dots a_n$ erzeugen können, muss in der zweiten Phase das „Nachschieben“ von Blanks am Rand nicht mehr behandelt werden.

3. Phase: Aufräumen und erzeugen von $a_1 \dots a_n$, wenn die TM akzeptiert hat

- $q[a,b] \longrightarrow EaE$ für $a \in \Sigma,\ b \in \Gamma$
  $q[\not b,b] \longrightarrow E$ für $b \in \Gamma$
  falls $q \in F$ und es keine Transition der Form $(q,b,\dots,\dots) \in \Delta$ gibt (d.h. akzeptierende Stoppkonfiguration erreicht)
- $E[a,b] \longrightarrow aE$ für $a \in \Sigma, b \in \Gamma$
  (Aufräumen nach rechts)
- $[a,b]E \longrightarrow Ea$ für $a \in \Sigma, b \in \Gamma$
  (Aufräumen nach links)
- $E[\not b,b] \longrightarrow E$ für $b \in \Gamma$
  (Entfernen des zusätzlich benötigten Arbeitsbandes nach rechts)
- $[\not b,b]E \longrightarrow E$ für $b \in \Gamma$
  (Entfernen des zusätzlich benötigten Arbeitsbandes nach links)
- $E \longrightarrow \varepsilon$

Man sieht nun leicht, dass für alle $w \in \Sigma^*$ gilt:

$$w \in L(G) \ \ \underline{\text{gdw}}.\ \ \mathcal{A} \text{ akzeptiert } w. \qquad \square$$

Für Typ-0-Sprachen gelten die folgenden Abschlusseigenschaften:

Satz 12.2

1) $\mathcal{L}_0$ *ist abgeschlossen unter* $\cup, \cdot, ^*$ *und* $\cap$.

2) $\mathcal{L}_0$ *ist nicht abgeschlossen unter Komplement.*

Beweis.

1) Für die regulären Operationen $\cup, \cdot, ^*$ zeigt man dies im Prinzip wie für $\mathcal{L}_2$ durch Konstruktion einer entsprechenden Grammatik.
   Damit sich die Produktionen der verschiedenen Grammatiken nicht gegenseitig beeinflussen, genügt es allerdings nicht mehr, nur die Nichtterminalsymbole der Grammatiken disjunkt zu machen.

Zusätzlich muss man die Grammatiken in die folgende Form bringen:
Die Produktionen sind von der Form

$$\begin{aligned} u &\longrightarrow v && \text{mit } u \in N_i^+ \text{ und } v \in N_i^* \\ X_a &\longrightarrow a && \text{mit } X_a \in N_i \text{ und } a \in \Sigma \end{aligned}$$

Für den Durchschnitt verwendet man Turingmaschinen:

Die NTM für $L_1 \cap L_2$ verwendet zwei Bänder und simuliert zunächst auf dem ersten die Berechnung der NTM für L_1 und dann auf dem anderen die der NTM für L_2.

Wenn beide zu akzeptierenden Stoppkonfigurationen der jeweiligen Turingmaschinen führen, so geht die NTM für $L_1 \cap L_2$ in eine akzeptierende Stoppkonfiguration.

Beachte:

Es kann sein, dass die NTM für L_1 auf einer Eingabe w nicht terminiert, die NTM für $L_1 \cap L_2$ also gar nicht dazu kommt, die Berechnung der NTM für L_2 zu simulieren. Aber dann ist ja w auch nicht in $L_1 \cap L_2$.

2) Wir werden später sehen, dass Turing-akzeptierbare Sprachen nicht unter Komplement abgeschlossen sind. □

Wir werden später außerdem zeigen, dass für Turing-akzeptierbare Sprachen (und damit für $\mathcal{L}_0$) alle bisher betrachteten Entscheidungsprobleme unentscheidbar sind:

Satz 12.3

Für $\mathcal{L}_0$ sind das Leerheitsproblem, das Wortproblem und das Äquivalenzproblem unentscheidbar.

Von den Sprachklassen aus der Chomsky-Hierarchie sind nun alle bis auf $\mathcal{L}_1$ (kontextsensitiv) durch geeignete Automaten/Maschinenmodelle charakterisiert. Man kann mit der Beweisidee von Satz 12.1 auch eine Charakterisierung kontextsensitiver Sprachen durch spezielle Turingmaschinen erhalten.

Diese Maschinen dürfen nur auf dem Bandabschnitt arbeiten, auf dem anfangs die Eingabe stand. Um ein Überschreiten der dadurch gegebenen Bandgrenzen zu verhindern, verwendet man Randmarker $\not\in$, \$.

Definition 12.4 (linear beschränkter Automat)

Ein *linear beschränkter Automat (LBA)* ist eine NTM $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \Delta, F)$, so dass

- $\$, \not\in \in \Gamma \setminus \Sigma$
- Übergänge $(q, \not\in, \ldots)$ sind nur in der Form $(q, \not\in, \not\in, r, q')$ erlaubt (linker Rand darf nicht überschritten werden).
- Übergänge $(q, \$, \ldots)$ sind nur in der Form $(q, \$, \$, l, q')$ erlaubt.
- $\not\in$ und \$ dürfen nicht geschrieben werden.

Ein gegebener LBA $\mathcal{A}$ akzeptiert die Sprache

$$L(\mathcal{A}) := \{w \in \Sigma^* \mid \not\in q_0 w\$ \vdash^* k, \text{ wobei } k \text{ akzeptierende Stoppkonfiguration ist}\}.$$

Korollar 12.5

Eine Sprache L gehört zu $\mathcal{L}_1$ gdw. sie von einem LBA akzeptiert wird.

Beweis.

„⇒“: Verwende die Konstruktion aus dem Beweis zu Satz 12.1.

Da alle Produktionen von kontextsensitiven Grammatiken nichtkürzend sind (mit Ausnahme $S \longrightarrow \varepsilon$), muss man auf dem zweiten Band nur ableitbare Wörter bis zur Länge $|w|$ erzeugen (längere können nie mehr zu w abgeleitet werden). Daher kommt man mit $|w|$ vielen Feldern aus.

Beachte:

Zwei Bänder liefern nicht ein doppelt so langes Band, sondern ein größeres Alphabet.

„⇐“: Durch Modifikation der Beweisidee von Satz 12.1 gelingt es, zu einem LBA eine Grammatik zu konstruieren, die nur nichtkürzende Regeln hat

$$\alpha \longrightarrow \beta \text{ mit } |\alpha| \leq |\beta|$$

Wie im Beweis von Satz 9.3 illustriert, kann man diese in eine kontextsensitive Grammatik transformieren.

Idee:

Da man mit $|w|$ Arbeitsfeldern auskommt, muss man keine $[\not b, \not b]$ links und rechts von w erzeugen. Dadurch fallen dann auch die folgenden kürzenden Regeln weg:

$$E[\not b, \not b] \longrightarrow E$$

$$[\not b, \not b]E \longrightarrow E$$

Es gibt allerdings noch einige technische Probleme:

- Man muss die Randmarker $\not\in$ und \$ einführen und am Schluss löschen.
- Man muss das Hilfssymbol E und den Zustand q löschen.

Lösung:

Führe die Symbole $\not\in$, \$ und den Zustand q nicht als zusätzliche Symbole ein, sondern kodiere sie in die anderen Symbole hinein.

z.B. statt $[a, b]q[a', b'][a'', b'']$ verwende $[a, b][q, a', b'][a'', b'']$.

□

Satz 12.6

$\mathcal{L}_1$ ist abgeschlossen unter $\cup, \cdot, ^*, \cap$ und Komplement.

Beweis. Für $\cup, \cdot, ^*$ und $\cap$ geht dies wie bei $\mathcal{L}_0$.

Komplement: schwierig, war lange offen und wurde dann in den 1980ern unabhängig von zwei Forschern gezeigt (Immerman und Szelepcsenyi). □

Für LBAs ist bisher nicht bekannt, ob deterministische LBAs genauso stark wie nichtdeterministische LBAs sind.

Satz 12.7

Für $\mathcal{L}_1$ ist das Wortproblem entscheidbar.

Beweis. Da kontextsensitive Produktionen (bis auf Spezialfall $S \longrightarrow \varepsilon$) nichtkürzend sind, muss man zur Entscheidung „$w \in L(G)$?“ nur alle aus S ableitbaren Wörter aus $(N \cup \Sigma)^*$ der Länge $\leq |w|$ erzeugen.

Dies sind *endlich viele.* □

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
geg. ..... gegeben
i.a. ..... im allgemeinen
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 Allgemeingültigkeit
PDA ..... pushdown automaton (Kellerautomat)
PKP ..... Postsches Korrespondenzproblem
PL1 ..... Prädikatenlogik erster Stufe
SAT ..... satisfiability problem (Erfüllbarkeitstest der Aussagenlogik)
TM ..... Turingmaschine (allgemein)
u.a. ..... unter anderem
URM ..... unbeschränkte Registermaschine
vgl. ..... vergleiche
z.B. ..... zum Beispiel
□ ..... was zu beweisen war (q.e.d)

Literaturverzeichnis

[Schö_97] U. Schöning. *Theoretische Informatik – kurzgefaßt.* Spektrum Akademischer Verlag Berlin, Heidelberg, 1997

[Thom_89] W. Thomas. *Grundzüge der Theoretischen Informatik.* Skript zur Vorlesung an der RWTH Aachen, 1989

[Wege_93] I. Wegener. *Theoretische Informatik.* Teubner-Verlag, 1993