



**TECHNISCHE  
UNIVERSITÄT  
DRESDEN**

---

Fakultät Informatik, Institut für Theoretische Informatik, Lehrstuhl Automatentheorie

Skript

# FORMALE SYSTEME

MODUL INF-B-270

TEIL 1 - AUTOMATEN UND FORMALE SPRACHEN

Prof. Dr.-Ing. Franz Baader

2016

Letzte L<sup>A</sup>T<sub>E</sub>X-Bearbeitung: Dr.-Ing. Monika Sturm (Version 2016-09-26)

# Inhaltsverzeichnis

<b>Organisation der Lehrveranstaltung</b>	<b>3</b>
<b>Einführung</b>	<b>4</b>
<b>I. Endliche Automaten</b>	<b>7</b>
1. Nichtdeterministische endliche Automaten . . . . .	10
2. Deterministische endliche Automaten . . . . .	15
3. Nachweis der Nichterkennbarkeit . . . . .	25
4. Abschlusseigenschaften und Entscheidungsprobleme . . . . .	28
5. Reguläre Ausdrücke und Sprachen . . . . .	33
<b>II. Grammatiken, kontextfreie Sprachen und Kellerautomaten</b>	<b>38</b>
6. Die Chomsky-Hierarchie . . . . .	39
7. Rechtslineare Grammatiken und reguläre Sprachen . . . . .	42
8. Normalformen kontextfreier Grammatiken . . . . .	45
9. Abschlusseigenschaften kontextfreier Sprachen . . . . .	53
10. Kellerautomaten . . . . .	57
<b>III. Turingmaschinen und Grammatiken</b>	<b>68</b>
11. Turingmaschinen . . . . .	70
12. Zusammenhang zwischen Turingmaschinen und Grammatiken . . . . .	77
<b>Abkürzungsverzeichnis</b>	<b>83</b>
<b>Literatur</b>	<b>84</b>

# Organisation der Lehrveranstaltung

Die Lehrveranstaltung

**„Formale Systeme“ (Modul INF-B-270)**

gliedert sich in

- **Teil 1:** Automaten 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/ws2016-2017/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:

## Automaten und formale Sprachen (Teil 1)

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

- Wie beschreibt man solche Mengen?  
→ *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?  
→ *Syntax und Semantik der Aussagenlogik*
- Wie kann man aus einer Menge von logischen Formeln weitere Aussagen schlussfolgern?  
→ *Kalküle des logischen Schließens*

### Hinweis

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

# Automaten 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*: Gegeben sind 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 gegebenes 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*: Gegeben ist 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

## 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:

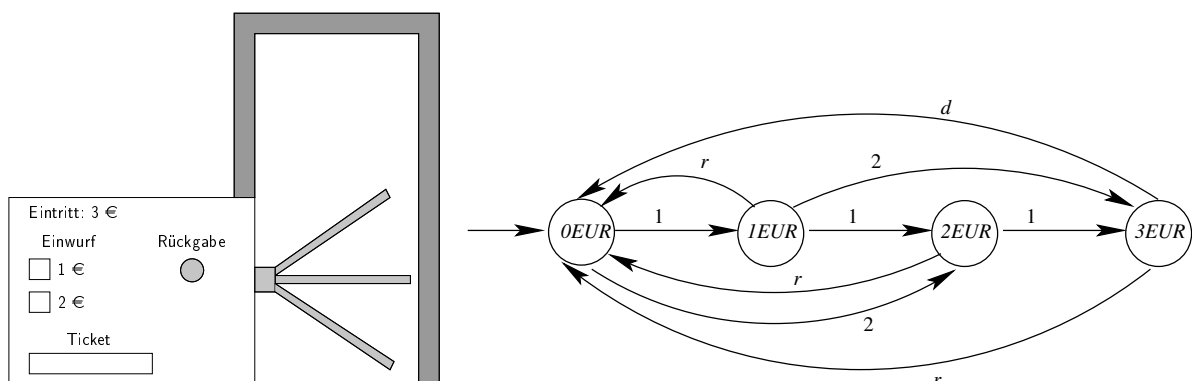
Wegen der extrem großen Anzahl von Zuständen sind endliche Automaten nicht die geeignete Beschreibungsebene für das Verhalten von Rechenmaschinen. Man abstrahiert daher von der Speicherplatzbeschränkung und nimmt potentiell unendlich großen Speicher an (z. B. unendliches Band bei Turingmaschine).

### 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.  $a, b, c, \dots$ ) heißt Alphabet und wird meist mit  $\Sigma$  bezeichnet.

**Wort:**

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

**Länge des Wortes:**

$|w| = |a_1 \dots a_n| = n$ , 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:**

$\varepsilon$  hat Länge 0

**Menge aller Wörter über  $\Sigma$ :**

$\Sigma^*$

**Menge aller nichtleeren Wörter über  $\Sigma$ :**

$\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$

**Formale Sprache:**

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

Beispiele:

- Menge der Wörter über  $\{a, \dots, z\}$ , 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:**

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

**Konkatenation:**

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

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

Beachte:

Der Konkatenationspunkt wird häufig weggelassen ( $L_1L_2$  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.  $v = w_1uw_2$  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!),
- $\Sigma$  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

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

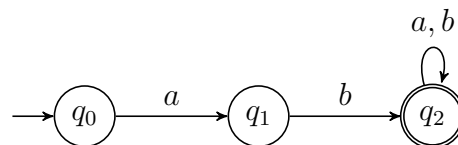
Anstelle von  $(Q, \Sigma, \{q_0\}, \Delta, F)$  schreiben wir  $(Q, \Sigma, q_0, \Delta, F)$ .

## 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  $i = 0, \dots, n - 1$ .

$\pi$  ist ein Pfad von  $p$  nach  $q$ , wenn  $p = p_0$  und  $q = p_n$ .

Die *Beschriftung* des Pfades  $\pi$  ist das Wort  $\beta(\pi) := a_1 \dots a_n$ .

Die *Länge* des Pfades  $\pi$  ist  $n$ .

Für  $n = 0$  sprechen wir vom *leeren Pfad*, welcher die Beschriftung  $\varepsilon$  hat.

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

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

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

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

**Definition 1.5 (akzeptierte Sprache)**

Das Transitionssystem  $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$  akzeptiert das Wort  $w \in \Sigma^*$  gdw.  $I \xrightarrow{\mathcal{A}}^w F$  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.

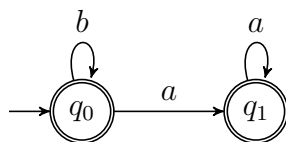
**Definition 1.6 (Erkennbarkeit einer Sprache)**

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. Allg. 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,  $\varepsilon$ -NEA)**

Ein NEA mit Wortübergängen hat die Form  $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ , wobei  $Q, \Sigma, q_0, F$  wie beim NEA definiert sind und  $\Delta \subseteq Q \times \Sigma^* \times Q$  eine endliche Menge von Wortübergängen ist.

Ein  $\varepsilon$ -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 (NEA mit Wortübergängen nicht stärker als NEA)**

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

Man zeigt dies mit Umweg über  $\varepsilon$ -NEAs.

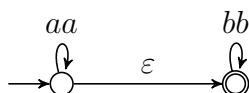
**Lemma 1.10 (NEA mit Wortübergängen nicht stärker als  $\varepsilon$ -NEA)**

Zu jedem NEA mit Wortübergängen kann man effektiv einen äquivalenten  $\varepsilon$ -NEA konstruieren.

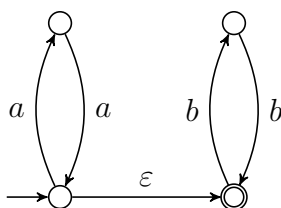
*Beweis.* Man ersetzt jeden Wortübergang  $(q, a_1 \dots a_n, q')$  mit  $n > 1$  durch Buchstabenübergänge  $(q, a_1, p_1), (p_1, a_2, p_2), \dots, (p_{n-1}, a_n, q')$ , wobei  $p_1, \dots, p_{n-1}$  jeweils neue Hilfszustände sind (die nicht zur Endzustandsmenge dazugenommen werden). Man sieht leicht,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

**Beispiel 1.11**

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



wird überführt in einen äquivalenten  $\varepsilon$ -NEA:



**Lemma 1.12 ( $\varepsilon$ -NEA nicht stärker als NEA)**

Zu jedem  $\varepsilon$ -NEA kann man effektiv einen äquivalenten NEA konstruieren.

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

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

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

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

1.  $\Delta'$  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  $\Delta$  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  $\varepsilon$  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  $\Delta'$  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  $\Delta'$  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  $w$ .

□

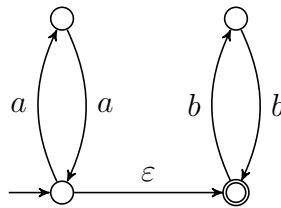
**Lemma 1.12** kann man auch verwenden, wenn man zeigen will, dass es zu jedem *endlichen* Transitionssystem  $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$  einen äquivalenten NEA  $\mathcal{A}'$  gibt. Man definiert für einen neuen Zustand  $q_0 \notin Q$  einen  $\varepsilon$ -NEA  $\mathcal{A}'' = (Q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

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

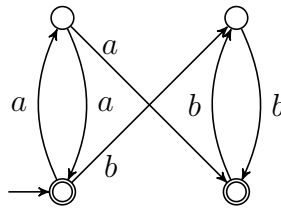
$\mathcal{A}''$  ist äquivalent zu  $\mathcal{A}$ . Man erhält nun  $\mathcal{A}'$  aus  $\mathcal{A}''$  wie im Beweis des Lemmas beschrieben.

**Beispiel:** (zu Lemma 1.12)

Der  $\varepsilon$ -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 Übergangrelation  $\Delta$  verwenden wir dann die *Übergangsfunktion*

$$\delta : Q \times \Sigma \rightarrow Q$$

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

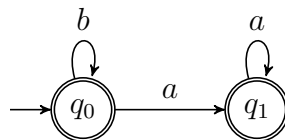
#### Beachte:

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

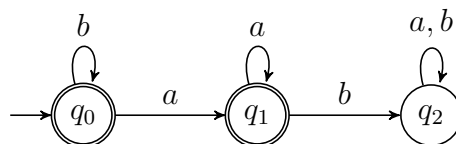
In der Literatur wird für deterministische Automaten manchmal nur die Eindeutigkeit des Übergangs gefordert, d.h. für  $(q, a)$  darf es höchstens ein  $q'$  mit  $(q, a, q') \in \Delta$  geben. Hingegen spricht man bei Existenz aller Übergänge von *vollständigen* Automaten, d.h. wenn es für  $(q, a)$  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 der Übergangsfunktion  $\delta$ )**

Die *kanonische Fortsetzung* von  $\delta : Q \times \Sigma \rightarrow Q$  auf eine Funktion  $\delta^* : Q \times \Sigma^* \rightarrow Q$  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 mindestens einen Endzustand enthält.

Wir definieren einen DEA mit Zustandsmenge  $2^Q$  und Übergangsfunktion  $\delta$  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.  $q \xrightarrow{w}_{\mathcal{A}} q'$  (★)

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



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

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

$|w| = 0$ :

$\delta(\{q\}, \varepsilon) = \{q\}$ .  $q$  ist der einzige Zustand, der mit  $\varepsilon$  von  $q$  aus erreichbar ist (keine  $\varepsilon$ -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  $q' \in \delta(\{q\}, u)$  gdw.  $q \xrightarrow{u}_{\mathcal{A}} q'$ .

Aus  $q \xrightarrow{u}_{\mathcal{A}} q'$  und  $(q', a, q'') \in \Delta$  ergibt sich  $q \xrightarrow{ua}_{\mathcal{A}} q''$ .

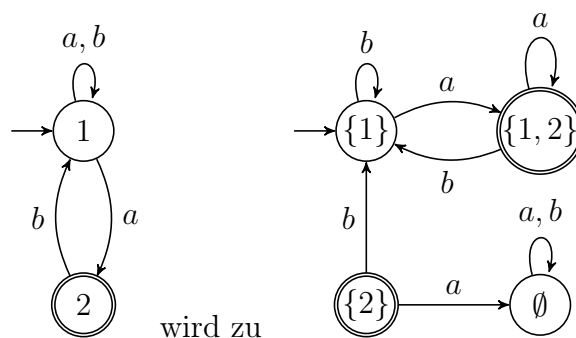
Umgekehrt gilt  $q \xrightarrow{ua}_{\mathcal{A}} q''$  gdw. es gibt ein  $q'$  mit  $q \xrightarrow{u}_{\mathcal{A}} q'$  und  $(q', a, q'') \in \Delta$ .

Es folgt daher  $q'' \in \delta(\{q\}, ua)$  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):



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)**

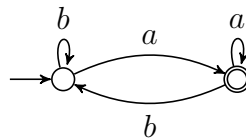
Ein Zustand  $q$  des DEA  $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$  heißt *erreichbar*, falls es ein Wort  $w \in \Sigma^*$  gibt mit  $\delta(q_0, w) = q$ . Sonst heißt  $q$  *unerreichbar*.

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

- $Q_0 = \{q \in Q \mid q \text{ ist erreichbar}\}$
- $\delta_0 = \delta \upharpoonright_{Q_0 \times \Sigma}$       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  $\mathcal{A}'_0$ :



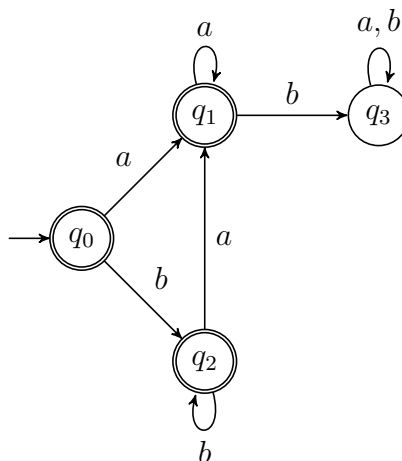
Bei der Potenzmengenkonstruktion kann man die Betrachtung unerreichbarer Elemente von  $2^Q$  vermeiden, indem man mit  $\{q_0\}$  beginnt und nur die davon erreichbaren Mengen sukzessive konstruiert.

**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 (Eigenschaften von  $\sim_{\mathcal{A}}$ )**

- 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:

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

- 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  $Q \times Q \supseteq \sim_0 \supseteq \sim_1 \supseteq \sim_2 \supseteq \dots \supseteq \sim_{\mathcal{A}}$ . 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  $\tilde{q} := \{q' \in Q \mid q \sim_{\mathcal{A}} q'\}$ .

**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*  $\tilde{\mathcal{A}} = (\tilde{Q}, \Sigma, \tilde{q}_0, \tilde{\delta}, \tilde{F})$  zu  $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$  ist definiert durch:

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

**Lemma 2.11**

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

*Beweis.*

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

**Definition 2.12 (reduzierter Automat zu einem DEA)**

Für einen DEA  $\mathcal{A}$  bezeichnet  $\mathcal{A}_{red} := \tilde{\mathcal{A}}_0$  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:  
 $u \simeq_L v$  gdw.  $\forall w \in \Sigma^* : uw \in L \Leftrightarrow vw \in L$ .

**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:
 

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

**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:
 

( $\star$ )  $uww' \in L \Leftrightarrow vww' \in L$

 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:

$$\begin{array}{lll} \forall w : uw \in L & \text{gdw.} & \delta(q_0, uw) \in F \\ & \text{gdw.} & \delta(\delta(q_0, u), w) \in F \\ & \text{gdw.} & \delta(\delta(q_0, v), w) \in F \\ & \text{gdw.} & \delta(q_0, vw) \in F \\ & \text{gdw.} & vw \in L \end{array}$$

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:

$$\begin{aligned} L(\mathcal{A}_L) &= \{u \mid \delta'(q'_0, u) \in F'\} \\ &= \{u \mid \delta'([\varepsilon], u) \in F'\} \\ &= \{u \mid [u] \in F'\} \quad (\text{wegen } \delta'([u], v) = [uv]) \\ &= \{u \mid u \in L\} \\ &= L \end{aligned} \quad \square$$

**Satz 2.18 (Satz von Nerode)**

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

*Beweis.*

„ $\Rightarrow$ “: Ergibt sich unmittelbar aus Lemma 2.15, 4).

„ $\Leftarrow$ “: 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:  $a^n \not\sim_L a^m$ . 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 (Isomorphie von Automaten)**

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  $f : Q \rightarrow Q'$  gibt mit:

- $f(q_0) = q'_0$
- $f(F) = F'$ , wobei  $f(F) := \{f(q) \mid q \in F\}$
- $f(\delta(q, a)) = \delta'(f(q), a)$  für alle  $q \in Q, a \in \Sigma$

**Lemma 2.21 (Isomorphie erhält Sprache)**

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

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

$$\begin{aligned} w \in L(\mathcal{A}) &\text{ gdw. } \delta(q_0, w) \in F \\ &\text{ gdw. } f(\delta(q_0, w)) \in F' \quad (\text{wegen } F' = f(F)) \\ &\text{ gdw. } \delta'(f(q_0), w) \in F' \\ &\text{ gdw. } \delta'(q'_0, w) \in F' \quad (\text{wegen } q'_0 = f(q_0)) \\ &\text{ gdw. } w \in L(\mathcal{A}') \end{aligned} \quad \square$$

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  $\mathcal{A}_{red} := \tilde{\mathcal{A}}_0$  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 so viele 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  $f : Q \rightarrow Q'$  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  $f(q) := [w_q]$ .

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  $w_p \not\sim_L w_q$ , d.h.  $[w_p] \neq [w_q]$ .

II)  $f$  ist surjektiv:

Folgt aus Injektivität und  $|Q| \geq |Q'|$  (Aussage 1) des Lemmas).

III)  $f(q_0) = q'_0$ :

Da  $w_{q_0} = \varepsilon$  und  $q'_0 = [\varepsilon]$ .

IV)  $f(F) = F'$ :

$q \in F$  gdw.  $\delta(q_0, w_q) = q \in F$  gdw.  $w_q \in L$  gdw.  $[w_q] \in F'$

V)  $f(\delta(q, a)) = \delta'(f(q), a)$ :

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

Außerdem ist  $\delta'(f(q), a) = \delta'([w_q], a) = [w_q a]$ .

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*

- $y \neq \varepsilon$
- $xy^kz \in L$  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  $x := a_1 \dots a_i$ ,  $y := a_{i+1} \dots a_j$ ,  $z := a_{j+1} \dots a_m$  gilt daher  $y \neq \varepsilon$  (da  $i < j$ ) und  $q_0 = p_0 \xrightarrow{x} p_i \xrightarrow{y} p_j \xrightarrow{z} p_m \in F$ .

Folglich gilt für alle  $k \geq 0$  auch  $p_i \xrightarrow{y^k} p_i$ , was  $xy^kz \in L$  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 3.2**

$L = \{a^n b^n \mid n \geq 0\}$  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  $xy^kz \in L$  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  $xz = xy^0z = a^{n_1+n_3} b^n \notin L$ , 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.  $\square$

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

**Satz 3.3 (Leerheitsproblem entscheidbar)**

*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:

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

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  $(\star)$ :

„ $\Leftarrow$ “: trivial

„ $\Rightarrow$ “: 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  $w$ .  $\square$

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

**Beispiel 3.4**

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  $k \geq 0$ :  $xy^k z = a^k a^{n-1} b^m \in L$ , 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)  $xy^0 z \in L$ , da  $n - 2 = m - 1 \neq m$

b)  $xy^k z \in L$  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 3.5 (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

- $y \neq \varepsilon$
- $uxy^kzw \in L$  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 3.4 im kürzeren Block).

**Beispiel 3.4 (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 3.5 geforderten Eigenschaften hat. Wir betrachten nun die Wörter

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

Offenbar ist  $uvw = a^{n_0} b^{n_0! + n_0} \in L$ . Mit Lemma 3.5 gibt es eine Zerlegung  $v = xyz$  mit  $y \neq \varepsilon$  und  $uxy^kzw \in L$  für alle  $k \geq 0$ . Es sei

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

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 (Abschlusseigenschaften 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 sei  $\mathcal{A}_i = (Q_i, \Sigma, q_{0i}, \Delta_i, F_i)$  ein NEA für  $L_i$  ( $i = 1, 2$ ).  
O.B.d.A. gelte  $Q_1 \cap Q_2 = \emptyset$ .

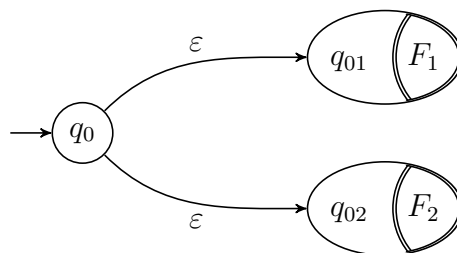
#### 1) Abschluss unter Vereinigung:

Der folgende  $\varepsilon$ -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:

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

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

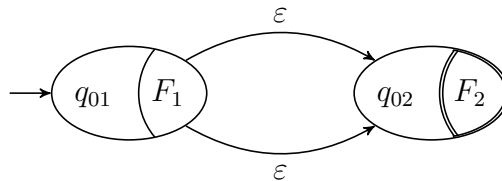
4) **Abschluss unter Differenz:**

Folgt aus 1) und 2), da  $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$ .

5) **Abschluss unter Konkatenation:**

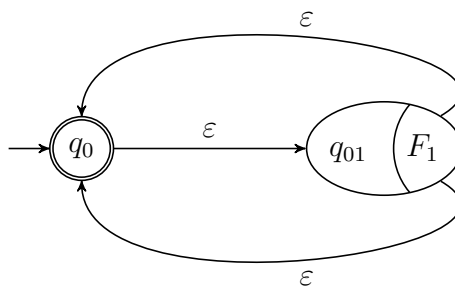
Der folgende  $\varepsilon$ -NEA akzeptiert  $L_1 \cdot L_2$ :

$\mathcal{A} := (Q_1 \cup Q_2, \Sigma, q_{01}, \Delta, F_2)$ , wobei  $\Delta := \Delta_1 \cup \Delta_2 \cup \{(f, \varepsilon, q_{02}) \mid f \in F_1\}$



6) **Abschluss unter Kleene-Stern:**

Der folgende  $\varepsilon$ -NEA akzeptiert  $L_1^*$ :  $\mathcal{A} := (Q_1 \cup \{q_0\}, \Sigma, q_0, \Delta, \{q_0\})$ , wobei  $q_0 \notin Q_1$  und  $\Delta := \Delta_1 \cup \{(f, \varepsilon, q_0) \mid f \in F_1\} \cup \{(q_0, \varepsilon, q_{01})\}$ .



□

**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 3.4). Anstatt dies direkt mit Lemma 3.5 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  $L' = \bar{L} \cap \{a\}^* \cdot \{b\}^*$ . 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  $\rightarrow$  DEA exponentiell sein kann.

**Leerheitsproblem:**

**Geg.:** erkennbare Sprache  $L$  (durch DEA oder NEA)

**Frage:** Ist  $L \neq \emptyset$ ?

Wir wissen bereits (Satz 3.3), 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.  $\square$

**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  $\delta$  berechnen, zu welchem Zustand in  $\mathcal{A}$  man mit  $w$  kommt und prüfen, ob dies ein Endzustand ist. Nimmt man  $\Sigma$  als konstant an, so benötigt eine Anwendung von  $\delta$  nur konstante Zeit. Dies liefert:

**Satz 4.4**

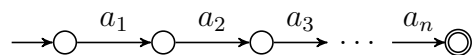
*Es sei  $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$  ein DEA und  $w \in \Sigma^*$ . Dann kann man „ $w \in L(\mathcal{A})$ ?“ in der Zeit  $O(|w|)$  entscheiden.*

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  $\bigcirc \xrightarrow{a_i} \bigcirc$

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  $L(\mathcal{A}) \cap L(\mathcal{A}_w) \neq \emptyset$ :

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

**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.  $\square$

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  $\varepsilon$ -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  $\Sigma$  ein endliches Alphabet. Die Menge  $Reg_\Sigma$  der *regulären Ausdrücke über  $\Sigma$*  ist induktiv definiert:

- $\emptyset, \varepsilon, a$  (für  $a \in \Sigma$ ) sind Elemente von  $Reg_\Sigma$ .
- 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  $\Sigma$  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 5.4** •  $(a + b)^*ab(a + b)^*$  definiert die Sprache aller Wörter über  $\{a, b\}$ , die Infix  $ab$  haben.

- $L(ab^* + b) = \{ab^i \mid i \geq 0\} \cup \{b\}$

**Bemerkung:**

Statt  $L(r)$  schreiben wir 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 5.5 (Kleene)**

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

- 1)  $L$  ist regulär.
- 2)  $L$  ist erkennbar.

*Beweis.*

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

**Verankerung:**

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

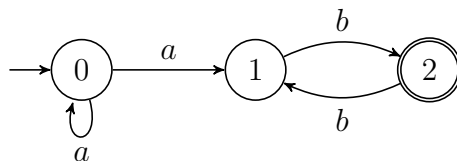
„2  $\rightarrow$  1“: Sei  $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$  ein NEA mit  $L = L(\mathcal{A})$ . Wie in Definition 2.8 sei für  $q \in Q$  der NEA  $\mathcal{A}_q := (Q, \Sigma, q, \Delta, F)$  und  $L_q = L(\mathcal{A}_q)$  definiert, d.h. insbesondere  $L = L_{q_0}$ .

Der Zusammenhang zwischen den  $L_q$ s kann nun als *Gleichungssystem* beschrieben werden, dessen Lösungen eindeutig bestimmte *reguläre Sprachen* sind.

Diesen Zusammenhang werden wir im Folgenden näher beleuchten.

**Beispiel 5.6**

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
- $B_p = \begin{cases} \{\varepsilon\} & \text{falls } p \in F \\ \emptyset & \text{falls } p \notin F \end{cases}$

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

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

**Behauptung:**

Das Gleichungssystem

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

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 5.5, Richtung „2  $\rightarrow$  1“.

Wir zeigen zunächst, wie man eine einzige Gleichung der Form

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

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

**Lemma 5.7 (Arden)**

Es seien  $A, B \subseteq \Sigma^*$  und  $\varepsilon \notin A$ . Die Gleichung  $(\star\star) \quad X = A \cdot X \cup B$  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 5.6** (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 5.7.*

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  $n$ :  $A^n B \subseteq L$ .

- $A^0 B = B \subseteq (A \cdot L \cup B) = L$
- Gelte  $A^n B \subseteq L$ .  
Es folgt:  $A^{n+1} B = A \cdot A^n B \subseteq A \cdot L \subseteq A \cdot L \cup B = L$

Wegen  $A^*B = \bigcup_{n \geq 0} A^n B$  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  $p \xrightarrow{a/v} q$ , wobei  $v \in \Gamma^*$  ein Wort über einem Ausgabealphabet ist. Solche Automaten beschreiben spezielle Funktionen  $\Sigma^* \rightarrow \Gamma^*$ .

**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

$$\begin{aligned} \text{Regeln:} \quad S &\longrightarrow aSb & (1) \\ S &\longrightarrow \varepsilon & (2) \end{aligned}$$

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  $\Sigma$  endliche, disjunkte Alphabete sind. (Man bezeichnet die Symbole aus  $N$  als *Nichtterminalsymbole*, die Symbole aus  $\Sigma$  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  $\Sigma$  mit Kleinbuchstaben.

Wir definieren nun, was es heißt, dass man ein Wort durch Anwenden der Regeln aus einem anderen ableiten kann.

**Definition 6.4 (Ableitbarkeit, 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.  $\exists x_1, x_2 \in (N \cup \Sigma)^* : \exists u \rightarrow v \in P : x = x_1 u x_2 \wedge y = x_1 v x_2$
- 2)  $y$  aus  $x$  in  $n$  Schritten ableitbar:  
 $x \vdash_G^n y$  gdw.  $\exists x_0, x_1, \dots, x_n \in (N \cup \Sigma)^* : x_0 = x \wedge x_n = y \wedge x_i \vdash_G x_{i+1}$  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  $abc \in L(G)$ . Für  $n > 1$  sieht man leicht:

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

„ $\subseteq$ “: Gelte  $S \vdash_G^* w$  mit  $w \in \Sigma^*$ . Wird sofort die Produktion  $S \rightarrow abc$  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  $S \rightarrow aSBc, cB \rightarrow Bc$  angewendet, da noch kein  $b$  vorhanden).

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

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

□



### Beispiel 6.5

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

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

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

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

#### Definition 6.6 (Chomsky-Hierarchie)

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
  - $u_1 A u_2 \rightarrow u_1 w u_2$  mit  $A \in N, u_1, u_2, w \in (\Sigma \cup N)^*$  und  $|w| \geq 1$  oder
  - $S \rightarrow \varepsilon$  hat.

Ist  $S \rightarrow \varepsilon \in P$ , 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  $A \rightarrow w$  hat mit  $A \in N, w \in (\Sigma \cup N)^*$ .
- $G$  heißt Grammatik vom **Typ 3 (rechtslinear)**, falls jede Regel von  $G$  die Form  $A \rightarrow uB$  oder  $A \rightarrow u$  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:**  $u_1 A u_2 \rightarrow u_1 w u_2$ . 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 nicht verkürzen (Ausnahme  $S \rightarrow \varepsilon$ , die aber nur zur Erzeugung von  $\varepsilon$  dient).

#### Definition 6.7 (Sprachklassen)

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 (Typ 3 = regulär)

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

Wir konstruieren einen NEA mit Worttransitionen  $\mathcal{A} = (N \cup \{\Omega\}, \Sigma, S, \Delta, \{\Omega\})$  für  $L$ , wobei

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

Es gilt  $w \in L(G)$

gdw. es gibt eine Ableitung

$$S = B_0 \vdash_G w_1 B_1 \vdash_G w_1 w_2 B_2 \vdash_G \dots \vdash_G w_1 \dots w_{n-1} B_{n-1} \vdash_G w_1 \dots w_{n-1} w_n = w$$

für Produktionen  $B_{i-1} \rightarrow w_i B_i \in P$  mit  $(i \leq n-1)$  und  $B_{n-1} \rightarrow w_n \in P$

gdw. es gibt einen Pfad

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

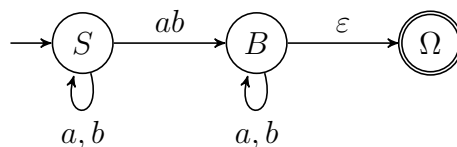
im Automaten  $\mathcal{A}$ .

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

### Beispiel 6.5 (Fortsetzung)

$$P = \{S \rightarrow aS, \\ S \rightarrow bS, \\ S \rightarrow abB, \\ B \rightarrow aB, \\ B \rightarrow bB, \\ B \rightarrow \varepsilon\}$$

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

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

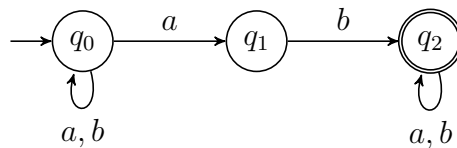
mit  $q_n \in F$  entspricht nun genau einer Ableitung

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

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

### Beispiel 7.2

Der folgende NEA



liefert die Grammatik mit den rechtslinearen Produktionen

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

□

### Korollar 7.3

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

*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 7.4

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 3.4, 4.2) Man kann diese Sprache mit folgender kontextfreier Grammatik erzeugen:

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

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

Es gilt nun:

- $S_{\geq} \vdash_G^* w \in \{a, b\}^* \Rightarrow w = a^n b^m$  mit  $n \geq m$ ,
  - $S_{\leq} \vdash_G^* w \in \{a, b\}^* \Rightarrow w = a^m b^n$  mit  $n \geq m$ ,
  - $S \vdash_G^* w \in \{a, b\}^* \Rightarrow w = aa^n b^m$  oder  $w = a^m b^n b$  mit  $n \geq m$ ,
- 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 (Berechnung der erreichbaren Symbole)

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

*Beweis.* Wir definieren dazu

$$E_0 := \{S\}$$

$$E_{i+1} := E_i \cup \{A \mid \exists B \in E_i \text{ mit Regel } B \rightarrow u_1 A u_2 \in P\}$$

Es gilt

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

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 8.3

$$P = \left\{ \begin{array}{ll} S \rightarrow aS, & A \rightarrow ASB, \\ S \rightarrow SB, & A \rightarrow C, \\ S \rightarrow SS, & B \rightarrow Cb, \\ S \rightarrow \varepsilon & \end{array} \right\}$$

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

Es ist also  $A$  das einzige nichterreichbare Symbol.

**Lemma 8.4 (Berechnung der terminierenden Symbole)**

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

$$T_1 \subseteq T_2 \subseteq \dots \subseteq N.$$

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:

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

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

$$A \vdash_G^{\leq i} w \in \Sigma^* \Rightarrow A \in T_i.$$

- $i = 1$ :  $A \vdash_G^{\leq 1} w \in \Sigma^* \Rightarrow A \longrightarrow w \in P \Rightarrow A \in T_1$
- $i \rightarrow i + 1$ :  $A \vdash_G u_1 B_1 \dots u_n B_n u_{n+1}$   
 $\vdash_G^{\leq i} w = u_1 w_1 \dots u_n w_n u_{n+1}$ ,  $u_j, w_j \in \Sigma^*$ ,  $B_j \in N$   
 $\Rightarrow$  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 8.4 folgt unmittelbar:

**Satz 8.5**

*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 8.4 zusammen zeigen, wie man unerreichbare und nichtterminierende Symbole eliminieren kann.

**Lemma 8.6 (Erzeugen einer reduzierten Grammatik)**

*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

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

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

**Zweiter Schritt:** Eliminieren unerreichbarer Symbole.

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

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

**Beachte:** Ist  $A \in N''$  und  $A \rightarrow u_1 B u_2 \in P'$ , 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 sein:

z. B.:  $S \rightarrow AB$  mit  $A$  terminierend,  $B$  nicht.

Als Nächstes eliminieren wir Regeln der Form  $A \rightarrow \varepsilon$ .

**Lemma 8.7 (Eliminieren von  $\varepsilon$ -Produktionen)**

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

*Beweis.*

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

$$N_1 := \{A \in N \mid A \rightarrow \varepsilon \in P\}$$

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

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

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

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

die Regeln

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

hinzu. □

**Beispiel:**

$$P = \{S \rightarrow aS, S \rightarrow SS, S \rightarrow bA, \\ A \rightarrow BB, \\ B \rightarrow CC, B \rightarrow aAbC, \\ C \rightarrow \varepsilon\}$$

$$N_1 = \{C\},$$

$$N_2 = \{C, B\},$$

$$N_3 = \{C, B, A\} = N_4$$

$$P' = \{S \rightarrow aS, S \rightarrow SS, S \rightarrow bA, S \rightarrow b, \\ A \rightarrow BB, A \rightarrow B, \\ B \rightarrow CC, B \rightarrow C, \\ B \rightarrow aAbC, B \rightarrow aAb, B \rightarrow abC, B \rightarrow ab\}$$

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

**Definition 8.8 ( $\varepsilon$ -freie kontextfreie Grammatik)**

Eine kontextfreie Grammatik heißt  $\varepsilon$ -frei, falls gilt:

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

**Satz 8.9**

*Zu jeder kontextfreien Grammatik  $G$  kann effektiv eine äquivalente  $\varepsilon$ -freie Grammatik konstruiert werden.*

*Beweis.* Konstruiere  $G'$  wie im Beweis von Lemma 8.7 beschrieben. Ist  $\varepsilon \notin L(G)$  (d.h.  $S \not\vdash_G^* \varepsilon$ , also  $S \notin N_k$ ), so ist  $G'$  die gesuchte  $\varepsilon$ -freie Grammatik. Sonst erweitere  $G'$  um ein neues Startsymbol  $S_0$  und die Produktionen  $S_0 \rightarrow S$  und  $S_0 \rightarrow \varepsilon$ . □

**Korollar 8.10**

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

*Beweis.* Offenbar ist jede  $\varepsilon$ -freie kontextfreie Grammatik eine Typ-1-Grammatik:

- Produktionen:  $u_1Au_2 \rightarrow u_1wu_2$  mit  $|w| \geq 1$
- kontextfrei:  $u_i = \varepsilon$ ;  $S \rightarrow \varepsilon$  und  $S$  nicht auf rechter Seite. □

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

**Satz 8.11 (Entferne Produktionen  $A \rightarrow B$ )**

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



*Beweisskizze.*

- 1) Bestimme zu jedem  $A \in N$  die Menge  $N(A) := \{B \in N \mid A \vdash_G^* B\}$ .

$$N_0(A) := \{A\},$$

$$N_{i+1}(A) := N_i(A) \cup \{B \in N \mid \exists B' \in N_i(A) : B' \longrightarrow B \in P\}$$

Es gibt ein  $k$  mit  $N_k(A) = N_{k+1}(A)$  und für dieses  $k$  ist  $N(A) = N_k(A)$ .

- 2) Definiere  $P' = \{A \longrightarrow w \mid B \longrightarrow w \in P, B \in N(A) \text{ und } w \notin N\}$ . □

**Beispiel:**

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

$$N(S) = \{S, A, B\},$$

$$N(A) = \{A, B\},$$

$$N(B) = \{B\}$$

$$P' = \{B \longrightarrow aA, A \longrightarrow aA, S \longrightarrow aA,$$

$$B \longrightarrow b, A \longrightarrow b, S \longrightarrow b\}$$

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

**Satz 8.12 (Chomsky-Normalform)**

*Jede kontextfreie Grammatik lässt sich umformen in eine äquivalente Grammatik in Chomsky-Normalform, d.h. eine Grammatik, die nur Regeln der folgenden Form hat:*

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

*Beweis.*

- 1) Konstruiere zu der gegebenen Grammatik eine äquivalente  $\varepsilon$ -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  $A \longrightarrow B_1 \dots B_n$  für  $n > 2$  werden ersetzt durch

$$A \longrightarrow B_1 C_1, C_1 \longrightarrow B_2 C_2, \dots, C_{n-2} \longrightarrow B_{n-1} B_n$$

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. Entsprechend kann eine Ableitung von  $w$  in einer rechtslinearen Grammatik  $G$  höchstens die Länge  $|w|$  haben. 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  $A \rightarrow BC$  verlängern um 1, d.h. sie können maximal  $|w| - 1$ -mal angewandt werden.
- Produktionen der Form  $A \rightarrow a$  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. Diese kann man alle erzeugen.

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

**Ziel:**

Angabe eines polynomiellen (kubischen) Verfahrens zur Entscheidung des Wortproblems.

**Definition 8.13**

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

- $w_{ij} := a_i \dots a_j$  (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. $A \rightarrow a_i \in P$  |
| 3) | $A \in N_{ij}$ für $i < j$ | gdw. $A \vdash_G^* a_i \dots a_j$   |
|    |                            | gdw. $\exists A \rightarrow BC \in P$ und ein $k$ mit $i \leq k < j$ mit<br>$B \vdash_G^* a_i \dots a_k$ und $C \vdash_G^* a_{k+1} \dots a_j$ |
|    |                            | gdw. $\exists A \rightarrow BC \in P$ und $k$ mit $i \leq k < j$ mit<br>$B \in N_{ik}$ und $C \in N_{(k+1)j}$                                 |

Diese Überlegungen liefern das folgende iterative Verfahren zur Bestimmung von  $N_{1k}$ :

**Algorithmus 8.14 (CYK-Algorithmus von Cocke, Younger, Kasami)**

```

for  $i := 1$  to  $n$  do
     $N_{ii} := \{A \mid A \rightarrow a_i \in P\}$ 
    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
                 $N_{ij} := N_{ij} \cup \{A \mid \exists A \rightarrow BC \in P \text{ mit } B \in N_{ik} \text{ und } C \in N_{(k+1)j}\}$ 

```

**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 8.15**

Für eine gegebene Grammatik in Chomsky-Normalform entscheidet Algorithmus 8.14 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  $A \rightarrow BC$  und  $A \rightarrow a_i$  auch konstant.

**Beispiel:**

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

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 8.16**

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

- $A \rightarrow aw$  ( $A \in N, a \in \Sigma, w \in N^*$ )
- und eventuell  $S \rightarrow \varepsilon$ , 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  $G_i = (N_i, \Sigma, P_i, S_i)$  ( $i = 1, 2$ ). O.B.d.A. nehmen wir an, dass  $N_1 \cap N_2 = \emptyset$ .

- 1)  $G := (N_1 \cup N_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}, S)$  mit  $S \notin N_1 \cup N_2$  ist eine kontextfreie Grammatik mit  $L(G) = L_1 \cup L_2$ .
- 2)  $G' := (N_1 \cup N_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\}, S)$  mit  $S \notin N_1 \cup N_2$  ist eine kontextfreie Grammatik mit  $L(G') = L_1 \cdot L_2$ .
- 3)  $G'' := (N_1 \cup \{S\}, \Sigma, P_1 \cup \{S \rightarrow \varepsilon, S \rightarrow S S_1\}, S)$  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.

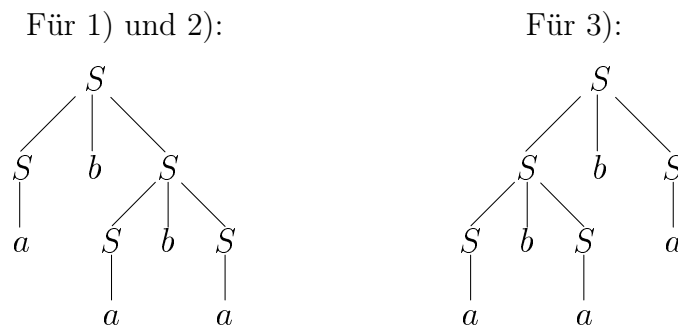
### Beispiel 9.2

$P = \{S \rightarrow SbS, S \rightarrow a\}$

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*:



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  $\alpha_1, \dots, \alpha_n$  beschriftete Nachfolgerknoten haben, wenn  $A \rightarrow \alpha_1 \dots \alpha_n \in P$  ist. Ein Ableitungsbaum, dessen Wurzel mit  $A$  beschriftet ist und dessen Blätter (von links nach rechts) mit  $\alpha_1, \dots, \alpha_n \in \Sigma \cup N$  beschriftet sind, beschreibt eine Ableitung

$$A \vdash_G^* \alpha_1 \dots \alpha_n.$$

**Lemma 9.3 (Pumping-Lemma für kontextfreie Sprachen)**

Es sei  $G$  eine kontextfreie Grammatik, die

- $\varepsilon$ -frei,
- keine Regeln der Form  $A \rightarrow B$  enthält,
- $m$  nichtterminale Symbole enthält,
- nur rechte Regelseiten der Länge  $\leq k$  hat.

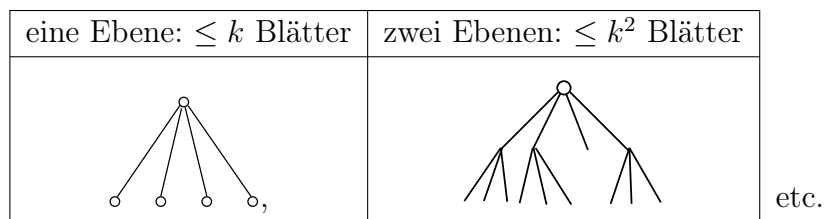
Es sei  $n = k^{m+1}$ .

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  $i \geq 0$ .

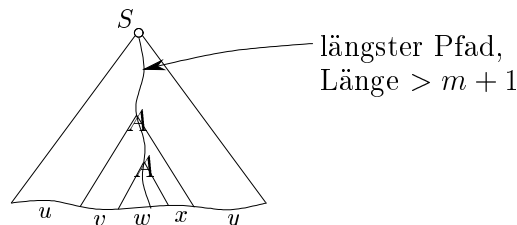
*Beweis.*

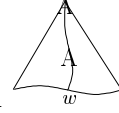
- 1) Ein Baum der Tiefe  $\leq t$  und der Verzweigungszahl  $\leq k$  hat maximal  $k^t$  viele Blätter:



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 Wiederholung von Nichtterminalsymbolen gibt. Daher hat dieser Baum eine Tiefe  $\leq m+1$ , was  $|vwx| \leq k^{m+1} = n$  zeigt.

3) Es gilt:  $S \vdash_G^* uAy$ ,  $A \vdash_G^* vAx$ ,  $A \vdash_G^* w$ , woraus folgt:

$$S \vdash_G^* uAy \vdash_G^* uv^iAx^i y \vdash_G^* uv^iwx^i y.$$

4)  $vx \neq \varepsilon$ : Da  $G$   $\varepsilon$ -frei ist, wäre sonst  $A \vdash_G^* vAx$  nur bei Anwesenheit von Regeln der Form  $A \rightarrow B$  möglich. □

**Satz 9.4**

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

*Beweis.* Wir haben bereits gezeigt, dass  $\mathcal{L}_2 \subseteq \mathcal{L}_1$  gilt (Korollar 8.10). Es bleibt zu zeigen, dass die Inklusion echt ist.

Dafür betrachten wir die Sprache  $L = \{a^n b^n c^n \mid n \geq 1\}$ , 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  $\varepsilon$ -freie kontextfreie Grammatik  $G$  ohne Regeln der Form  $A \rightarrow B$  für  $L$ . Es sei  $n_0 = k^{m+1}$  die zugehörige Zahl aus Lemma 9.3. Wir betrachten  $z = a^{n_0} b^{n_0} c^{n_0} \in L = L(G)$ . Mit Lemma 9.3 gibt es eine Zerlegung von  $z = a^{n_0} b^{n_0} c^{n_0}$  :

$$z = uvwxy, \quad vx \neq \varepsilon \quad \text{und} \quad uv^iwx^i y \in L \quad \text{für alle } i \geq 0.$$

**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  $L(G) = \{a^n b^n c^n \mid n \geq 1\}$  angegeben. Leider enthält  $G$  eine Produktion, die nicht die Bedingungen für kontext-sensitive Produktionen ( $uAv \rightarrow uvv$ ,  $|w| \geq 1$ ) erfüllt:  $cB \rightarrow Bc$ .

Wir modifizieren  $G$  wie folgt:

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

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

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

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

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

**Beachte:**

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

**Korollar 9.5**

Die Klasse  $\mathcal{L}_2$  der kontextfreien Sprachen ist nicht unter Durchschnitt und Komplement abgeschlossen.

*Beweis.*

- 1) Die Sprachen

$$\{a^n b^n c^m \mid n \geq 1, m \geq 1\} = \{a^n b^n \mid n \geq 1\} \cdot \{c\}^+ \text{ und}$$

$$\{a^m b^n c^n \mid n \geq 1, m \geq 1\} = \{a\}^+ \cdot \{b^n c^n \mid n \geq 1\}$$

sind in  $\mathcal{L}_2$ :

- $\{a^n b^n \mid n \geq 1\} \in \mathcal{L}_2$  und  $\{b^n c^n \mid n \geq 1\} \in \mathcal{L}_2$
- $\{a\}^+ \in \mathcal{L}_3 \subseteq \mathcal{L}_2$  und  $\{c\}^+ \in \mathcal{L}_3 \subseteq \mathcal{L}_2$
- $\mathcal{L}_2$  ist unter Konkatenation abgeschlossen.

- 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 9.4.

- 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. Widerspruch 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.

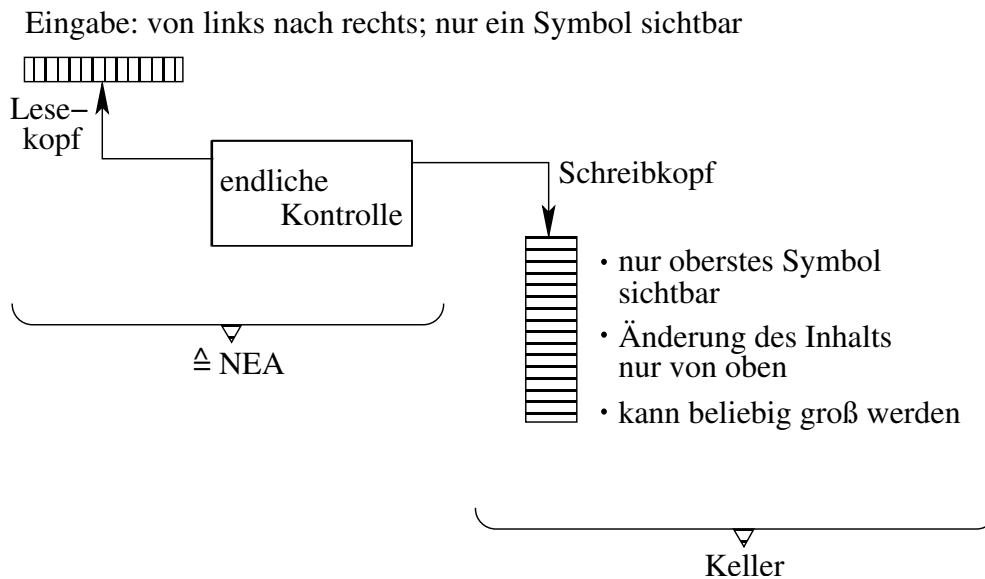


## 10. Kellerautomaten

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  $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, Z_0, \Delta, F)$ , wobei

- $Q$  eine endliche Menge von *Zuständen* ist,
- $\Sigma$  das *Eingabealphabet* ist,
- $\Gamma$  das *Kelleralphabet* ist,
- $q_0 \in Q$  der *Anfangszustand* ist,
- $Z_0 \in \Gamma$  das *Kellerstartsymbol* ist,
- $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times \Gamma^* \times Q$  eine endliche *Übergangsrelation* ist und
- $F \subseteq Q$  eine Menge von Endzuständen ist.

Anschaulich bedeutet die Übergangsrelation:

$(q, a, Z, \gamma, q')$ : Im Zustand  $q$  mit aktuellem Eingabesymbol  $a$  und oberstem Kellersymbol  $Z$  darf der Automat  $Z$  durch  $\gamma$  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 (die aktuelle Konfiguration) 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  $\gamma$ )

### 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.  $\exists n \geq 0 \exists \mathcal{K}_0, \dots, \mathcal{K}_n$  mit  $\mathcal{K} = \mathcal{K}_0 \vdash \mathcal{K}_1 \vdash \dots \mathcal{K}_n = \mathcal{K}'$ .

Der Automat  $\mathcal{A}$  *akzeptiert* das Wort  $w \in \Sigma^*$  gdw.  $(q_0, w, Z_0) \vdash^* (q, \varepsilon, \gamma)$  mit  $q \in F, \gamma \in \Gamma^*$  (Eingabewort ganz gelesen und Endzustand erreicht).

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
- $\Delta = \{(q_0, a, Z_0, ZZ_0, q_0), \text{ (erstes } a, \text{ speichere } Z)$   
 $(q_0, a, Z, ZZ, q_0), \text{ (weitere } a\text{'s, speichere } Z)$   
 $(q_0, b, Z, \varepsilon, q_1), \text{ (erstes } b, \text{ entnimm } Z)$   
 $(q_1, b, Z, \varepsilon, q_1), \text{ (weitere } b\text{'s, entnimm } Z)$   
 $(q_1, \varepsilon, Z_0, \varepsilon, f)\}$  (lösche das Kellerstartsymbol  
und gehe in Endzustand)

Betrachten wir 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 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  $(q, a, Z, \dots, \dots) \in \Delta$ .
- 2) Existiert ein Tupel  $(q, \varepsilon, Z, \dots, \dots) \in \Delta$ , so existiert kein Tupel der Form  $(q, a, Z, \dots, \dots) \in \Delta$  mit  $a \in \Sigma$ .

**Beispiel 10.5**

Die Sprache  $L = \{w \overleftarrow{w} \mid w \in \{a, b\}^*\}$  (wobei für  $w = a_1 \dots a_n$  gilt  $\overleftarrow{w} = a_n \dots a_1$ ) 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 (Wir werden später zeigen, dass kein deterministischer PDA diese Sprache akzeptiert).

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

$\{(q_0, \varepsilon, Z_0, \varepsilon, f),$	akzeptiert $\varepsilon$
$(q_0, a/b, Z_0, a/bZ_0, q_1),$	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
$(q_2, \varepsilon, Z_0, \varepsilon, f)\}$	Übergang zu Endzustand

Für Kellerautomaten gibt es noch einen anderen (äquivalenten) Akzeptanzbegriff:

**Definition 10.6 (Akzeptanz mit leerem Keller)**

Es sei  $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, Z_0, \Delta)$  ein PDA ohne ausgezeichnete Endzustandsmenge.

Wir sagen:

$\mathcal{A}$  akzeptiert  $w$  mit leerem Keller, falls

$$(q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon) \text{ für ein beliebiges } q \in Q.$$

Die durch  $\mathcal{A}$  mit leerem Keller akzeptierte Sprache ist

$$N(\mathcal{A}) := \{w \in \Sigma^* \mid \mathcal{A} \text{ akzeptiert } w \text{ mit leerem Keller}\}.$$

**Beispiel**

Die PDAs aus Beispiel 10.3 und 10.5 akzeptieren die entsprechende Sprache sowohl mit Endzustandsmenge  $\{f\}$  als auch mit leerem Keller, denn nur beim Übergang zu  $f$  wird das Kellerstartsymbol  $Z_0$  entfernt und damit der Keller leer.

**Satz 10.7**

Für PDAs ist Akzeptanz mit Endzuständen äquivalent zu Akzeptanz mit leerem Keller, d. h. für eine formale Sprache  $L$  sind äquivalent:

1.  $L = L(\mathcal{A})$  für einen PDA  $\mathcal{A}$ .
2.  $L = N(\mathcal{B})$  für einen PDA  $\mathcal{B}$ .

*Beweis.*

„1  $\rightarrow$  2“:  $\mathcal{B}$  arbeitet im Prinzip wie  $\mathcal{A}$ . Bei Erreichen eines Endzustands leert  $\mathcal{B}$  noch den Keller. Zusätzlich muss verhindert werden, dass der Keller leer wird, ohne dass ein Endzustand erreicht war:

$\mathcal{A}$  wird daher erweitert um

- einen neuen Anfangszustand  $q_0'$ ,
- einen neuen Zustand  $q_1$  zum Leeren des Kellers,
- ein neues Startsymbol  $X_0$  ( $Z_0$  ist nicht mehr Startsymbol),

sowie die Übergänge

$$\begin{array}{ll} (q_0', \varepsilon, X_0, Z_0X_0, q_0), & \\ (q, \varepsilon, Z, \varepsilon, q_1) & \text{für } q \in F, Z \in \Gamma \cup \{X_0\}, \\ (q_1, \varepsilon, Z, \varepsilon, q_1) & \text{für } Z \in \Gamma \cup \{X_0\}. \end{array}$$

**Beachte:**

Da  $X_0$  in den Übergängen von  $\mathcal{A}$  nicht vorkommt, kann es nur entfernt werden, wenn vorher ein Endzustand erreicht war.

„2  $\rightarrow$  1“:  $\mathcal{A}$  arbeitet im Prinzip wie  $\mathcal{B}$ . Zusätzlich muss  $\mathcal{A}$  feststellen, wenn der Keller geleert ist und dann in einen Endzustand übergehen. Erkennen des leeren Kellers (bei  $\mathcal{B}$ ): verwende neues Kellerstartsymbol.

$\mathcal{B}$  wird daher ergänzt um

- einen neuen Anfangszustand  $q_0'$ ,
- einen neuen Zustand  $f$ , der einziger Endzustand ist,
- ein neues Kellerstartsymbol  $X_0$  ( $Z_0$  ist nicht mehr Startsymbol),

sowie die Übergänge

$$\begin{aligned} & (q_0', \varepsilon, X_0, Z_0X_0, q_0), \\ & (q, \varepsilon, X_0, \varepsilon, f) \text{ für alle } q \in Q \\ & \text{(wenn } X_0 \text{ wieder auftaucht, hat } \mathcal{B} \text{ seinen Keller geleert).} \end{aligned}$$

□

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

**Satz 10.8**

Für eine formale Sprache  $L$  sind äquivalent:

- 1)  $L = L(G)$  für eine kontextfreie Grammatik  $G$ .
- 2)  $L = N(\mathcal{A})$  für einen PDA  $\mathcal{A}$ .

*Beweis.*

„1  $\rightarrow$  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 (kontextfrei!).

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

$$\begin{aligned} & \{(q, \varepsilon, A, \gamma, q) \mid A \rightarrow \gamma \in P\} \cup \text{(Anwenden einer Produktion auf oberstes} \\ & \text{Kellersymbol)} \quad (\star) \\ & \{(q, a, a, \varepsilon, q) \mid a \in \Sigma\} \quad \text{(Entfernen bereits erzeugter Terminalsym-} \\ & \text{bole von der Kellerspitze, wenn sie in der} \\ & \text{Eingabe vorhanden sind)} \quad (\star\star) \end{aligned}$$

**Beispiel:**

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

$$\begin{array}{ll}
 (q, \varepsilon, S, \varepsilon, q), & (S \longrightarrow \varepsilon) \\
 (q, \varepsilon, S, aSa, q), & (S \longrightarrow aSa) \\
 (q, \varepsilon, S, bSb, q), & (S \longrightarrow bSb) \\
 (q, a, a, \varepsilon, q), & (a \text{ entfernen}) \\
 (q, b, b, \varepsilon, q) & (b \text{ entfernen})
 \end{array}$$

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

$$\begin{array}{llll}
 (q, abba, S) \vdash (q, abba, aSa) & \vdash (q, bba, Sa) & \vdash (q, bba, bSba) \vdash \\
 (q, ba, Sba) \vdash (q, ba, ba) & \vdash (q, a, a) & \vdash (q, \varepsilon, \varepsilon) .
 \end{array}$$

**Behauptung:**

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

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

**Beachte:**

Für  $\alpha = \varepsilon = v$  folgt:

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

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

*Beweis der Behauptung.*

„ $\Leftarrow$ “: 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  $S \vdash^* S$ .

$k \rightarrow k + 1$ :

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

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

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

„ $\Rightarrow$ “: Beweis durch Induktion über die Länge der Linksableitung

$k = 0$ :

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

$k \rightarrow k + 1$ :

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

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

Es sei  $u''$  das längste Anfangsstück von  $\gamma\beta$ , das in  $\Sigma^*$  liegt. Dann ist  $u'u'' = u$  und  $\alpha$  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  $\rightarrow$  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:  $[p, Z, q] \vdash_G^* u \in \Sigma^*$  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  $a = \varepsilon$ ). Das Kellersymbol  $Z$  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:**

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

$N := \{S\} \cup \{[p, Z, q] \mid p, q \in Q, Z \in \Gamma\}$

$P := \{S \rightarrow [q_0, Z_0, q] \mid q \in Q\} \cup$

$\{[p, Z, q] \rightarrow a[p_0, X_1, p_1][p_1, X_2, p_2] \dots [p_{n-1}, X_n, q] \mid$

$(p, a, Z, X_1 \dots X_n, p_0) \in \Delta,$

$a \in \Sigma \cup \{\varepsilon\},$

$p, q, p_0, p_1, \dots, p_{n-1} \in Q,$

wobei  $q = p_0$ , falls  $n = 0$ }

**Beachte:**

Für  $n = 0$  hat man die Produktion  $[p, Z, q] \rightarrow a$ , 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:

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

Für  $p = q_0$  und  $Z = Z_0$  folgt daraus:

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

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

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

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

entspricht der Ableitung

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

Wegen der gerade gezeigten Äquivalenz zwischen kontextfreien Sprachen und PDA akzeptierbaren Sprachen, kann man Eigenschaften von kontextfreien Sprachen mit Hilfe von Eigenschaften von Kellerautomaten zeigen.

Als Beispiel betrachten wir den Durchschnitt von kontextfreien Sprachen mit regulären Sprachen. Wir wissen: der Durchschnitt zweier kontextfreier Sprachen muss nicht kontextfrei sein. Dagegen gilt:

**Korollar 10.9**

*Es sei  $L \subseteq \Sigma^*$  kontextfrei und  $R \subseteq \Sigma^*$  regulär. Dann ist  $L \cap R$  kontextfrei.*

*Beweis.*

Es sei

- $L = L(\mathcal{A})$  für einen PDA  $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, Z_0, \Delta, F)$  und
- $R = L(\mathcal{A}')$  für einen DEA  $\mathcal{A}' = (Q', \Sigma, q'_0, \delta', F')$ .



Wir konstruieren den „Produktautomaten“ von  $\mathcal{A}$  und  $\mathcal{A}'$ , der ein PDA für  $L \cap R$  ist:

$$\mathcal{B} := (Q \times Q', \Sigma, \Gamma, (q_0, q_0'), Z_0, \Delta', F \times F')$$

mit

$$\Delta' = \{((p, p'), a, Z, \gamma, (q, q')) \mid a \in \Sigma, (p, a, Z, \gamma, q) \in \Delta \text{ und } \delta(p', a) = q'\} \cup \{((p, p'), \varepsilon, Z, \gamma, (q, p')) \mid (p, \varepsilon, Z, \gamma, q) \in \Delta\}$$

Man zeigt nun leicht (durch Induktion über  $k$ ):

$$((p, p'), uv, \gamma) \vdash_{\mathcal{B}}^k ((q, q'), v, \beta) \quad \text{gdw.} \quad (p, uv, \gamma) \vdash_{\mathcal{A}}^k (q, v, \beta) \text{ und } p' \xrightarrow{u}_{\mathcal{A}'} q'.$$

□

*Bemerkung 10.10.*

Ist  $\mathcal{A}$  in dem Beweis ein deterministischer PDA, so ist der konstruierte Produktautomat  $\mathcal{B}$  auch deterministisch (da o. E.  $\mathcal{A}'$  deterministisch gewählt war). Man kann diese Abschlusseigenschaften von kontextfreien Sprachen u. a. dazu verwenden, für gewisse Sprachen nachzuweisen, dass sie nicht kontextfrei sind.

### Beispiel 10.11

Die Sprache  $L = \{ww \mid w \in \{a, b\}^*\}$  ist nicht kontextfrei.

*Beweis.*

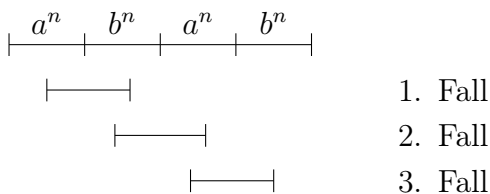
Angenommen,  $L$  ist kontextfrei. Dann ist mit Korollar 10.9 auch

$$L' := L \cap a^+b^+a^+b^+ = \{a^i b^j a^i b^j \mid i, j > 0\}$$

kontextfrei.

Es sei nun zu  $L'$  die Zahl  $n$  wie im Pumping Lemma. Wir betrachten  $z = a^n b^n a^n b^n \in L'$ . Es gibt eine Zerlegung  $z = uvwxy$  mit  $|vwx| \leq n$  und  $vx \neq \varepsilon$ , sodass  $uwy \in L'$ .

Wegen  $|vwx| \leq n$  enthält aber  $vwx$  höchstens 2 der aufeinanderfolgenden  $a^n/b^n$ -Blöcke:



In allen 3 Fällen kann  $uwy$  kein Element von  $L'$  sein. □

Für endliche Automaten hatten wir gesehen, dass bereits die deterministischen endlichen Automaten alle erkennbaren Sprachen akzeptieren. Dazu haben wir gezeigt, dass man zu jedem NEA einen äquivalenten DEA konstruieren kann. Für Kellerautomaten ist dies nicht möglich.

**Definition 10.12**

Eine Sprache heißt *deterministisch kontextfrei* (dkf), falls sie von einem deterministischen Kellerautomaten (DPDA) akzeptiert wird.

Wir werden zeigen, dass die Sprache

$$L = \{w\overleftarrow{w} \mid w \in \{a, b\}^*\}$$

nicht dkf ist. Wir haben bereits gesehen (Beispiel 10.5), dass  $L$  kontextfrei ist. Zunächst benötigen wir aber eine Definition und ein Lemma.

**Definition 10.13**

Zu einer Sprache  $L \subseteq \Sigma^*$  sei

$$\text{Min}(L) := \{w \in L \mid \text{kein echtes Präfix von } w \text{ liegt in } L\}.$$

**Lemma 10.14**

*Ist  $L$  dkf, so auch  $\text{Min}(L)$ .*

*Beweis.*

Es sei  $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, Z_0, \Delta, F)$  ein DPDA für  $L$ , d. h.  $L = L(\mathcal{A})$  (Akzeptanz mit Endzustand).

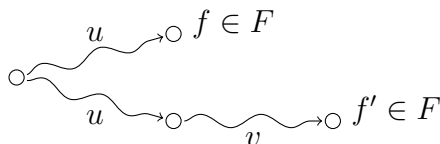
Wir ändern  $\mathcal{A}$  so ab, dass gilt: wurde das erste Mal ein Endzustand erreicht, so kann danach keiner mehr erreicht werden.

$$\begin{aligned} \mathcal{A}' &:= (Q', \Gamma, q_0, Z_0, \Delta', F) \quad \text{mit} \\ Q' &:= Q \cup \{\hat{q}\} \quad (\text{neuer Senkezustand}) \\ \Delta' &:= (\Delta \setminus (F \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times \Gamma^* \times Q) \\ &\quad \cup \{(q, a, Z, Z, \hat{q}) \mid q \in F, a \in \Sigma, Z \in \Gamma\} \\ &\quad \cup \{(\hat{q}, a, Z, Z, \hat{q}) \mid a \in \Sigma, Z \in \Gamma\} \end{aligned}$$

Man sieht leicht:  $L(\mathcal{A}') = \text{Min}(L)$  und  $\mathcal{A}'$  ist deterministisch. □

**Beachte:**

Bei einem nicht-det. PDA funktioniert die Konstruktion nicht, da es für Eingabe  $u$  verschiedene Konfigurationsfolgen geben kann.



**Satz 10.15**

*Die Sprache  $L := \{w\overleftarrow{w} \mid w \in \{a, b\}^*\}$  ist nicht deterministisch kontextfrei.*

*Beweis.*

Wäre  $L$  dkf, so mit Bemerkung 10.10 und Lemma 10.14 auch

$$L' := \text{Min}(L) \cap (ab)^+(ba)^+(ab)^+(ba)^+.$$

Es ist

$$L' = \{(ab)^i(ba)^j(ab)^j(ba)^i \mid i > j > 0\}.$$

Für  $i \leq j$  wäre  $(ab)^i(ba)^i \in L$  echtes Präfix.

Mit Hilfe des Pumping Lemmas zeigt man, dass  $L'$  nicht kf.

Angenommen doch. Dann sei  $n$  die Zahl aus dem Pumping Lemma für  $L'$ . Dann hat

$$z = (ab)^{n+1}(ba)^n(ab)^n(ba)^{n+1}$$

eine Zerlegung  $z = uvwxy$  mit  $|vwx| \leq n$ ,  $vx \neq \varepsilon$  und  $uv^iwx^iy \in L$  für alle  $i \geq 0$ .

1. **Fall:**  $vwx$  im Mittelteil  $(ba)^n(ab)^n$  Aufpumpen liefert ein Wort mit „ $i \leq j$ “, d. h.  $\notin \text{Min}(L)$ .
2. **Fall:**  $vwx$  im linken Teil  $(ab)^{n+1}(ba)^n$  oder im rechten Teil  $(ab)^n(ba)^{n+1}$  Aufpumpen liefert ein Wort, das nicht in  $L$  ist.

□

Deterministisch kontextfreie Sprachen sind für die *Syntaxanalyse* von Programmen interessant, da für sie das Wortproblem linear (im Gegensatz zu kubisch beim CYK-Algorithmus für allgemeine kontextfreie Sprachen) entscheidbar ist.

Dies ist wichtig im Compilerbau (siehe z. B. Ingo Wegner: Theoretische Informatik S. 199-216).

Im Gegensatz zu den kf Sprachen ist die Klasse der dkf Sprachen unter Komplement abgeschlossen.

**Satz 10.16**

*Ist  $L$  deterministisch kontextfrei, so auch  $\bar{L}$ .*

(ohne Beweis)

**Beachte:**

Für einen DPDA, der  $L$  mit Endzuständen aus  $F$  akzeptiert, genügt es nicht, als neue Endzustandsmenge einfach  $Q \setminus F$  zu nehmen, um einen Automaten für  $\bar{L}$  zu erhalten. Der Grund ist, dass es für den DPDA zwei verschiedene Gründe geben kann, warum er ein Wort  $w$  nicht akzeptiert:

- i) Wenn  $w$  ganz gelesen ist, so ist der Automat nicht in einem Endzustand.
- ii) Das Wort  $w$  wird nicht ganz gelesen.

Man muss den DPDA zunächst in einen DPDA umwandeln, bei dem der zweite Grund nie auftritt (siehe Wegener: S. 193-195, für Details).

# 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

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

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:

- $\mu$ -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.*

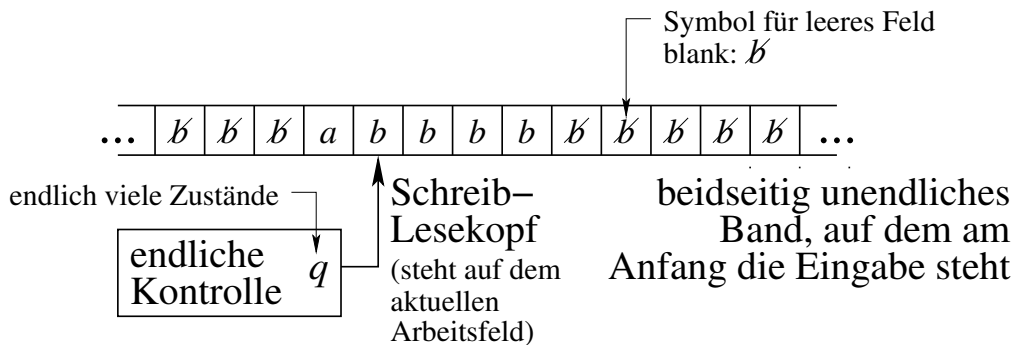
Man spricht hier von einer *These* und nicht von einem *Satz*, da es nicht möglich ist, diese Aussage formal zu beweisen. Dies liegt daran, dass der intuitive Berechenbarkeitsbegriff ja nicht formal definierbar ist. Es gibt aber sehr viele Indizien, die für die Richtigkeit der These sprechen (Vielzahl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

Im Folgenden betrachten wir:

- Turingmaschinen
- Zusammenhang zwischen Turingmaschinen und Grammatiken

## 11. Turingmaschinen

Turingmaschinen liefern Automatenmodelle für Typ-0- und Typ-1-Sprachen.



Zu jedem Zeitpunkt sind nur *endlich viele* Symbole auf dem Band verschieden von  $\text{blank}$ .

### Definition 11.1 (Turingmaschine)

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

- $Q$  endliche Zustandsmenge ist,
- $\Sigma$  das Eingabealphabet ist,
- $\Gamma$  das Arbeitsalphabet ist mit  $\Sigma \subseteq \Gamma$ ,  $\text{blank} \in \Gamma \setminus \Sigma$ ,
- $q_0 \in Q$  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  $(q, a, a', \overset{r}{l}, q')$ :

- 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  $(q, a, \dots, \dots) \in \Delta$  gibt. *NTM* steht im Folgenden für (möglicherweise nichtdeterministische) Turingmaschinen und *DTM* für deterministische. Bei einer DTM gibt es zu jedem Berechnungszustand höchstens einen Folgezustand, während es bei einer NTM mehrere geben kann.

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
- $\alpha$  ist die Beschriftung des Bandes links vom Arbeitsfeld
- $\beta$  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.  $\alpha$  und  $\beta$  umfassen mindestens den Bandabschnitt, auf dem Symbole  $\neq \flat$  stehen.

**Beispiel:**

Der Zustand der Maschine zu Beginn des Abschnitts wird durch die Konfiguration  $aqbbbb$ , aber auch durch  $\flat baqbbbb \flat$  beschrieben.

Die Übergangsrelation  $\Delta$  ermöglicht die folgenden *Konfigurationsübergänge*:

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

- $\left. \begin{array}{l} \alpha qa\beta \vdash_{\mathcal{A}} \alpha a' q' \beta \\ \alpha qa \vdash_{\mathcal{A}} \alpha a' q' \flat \end{array} \right\} \text{ falls } (q, a, a', r, q') \in \Delta$
- $\left. \begin{array}{l} \alpha b qa \beta' \vdash_{\mathcal{A}} \alpha q' b a' \beta' \\ b qa \beta' \vdash_{\mathcal{A}} \flat q' b a' \beta' \end{array} \right\} \text{ falls } (q, a, a', l, q') \in \Delta$
- $\alpha qa \beta' \vdash_{\mathcal{A}} \alpha q' a' \beta' \quad \text{falls } (q, a, a', n, q') \in \Delta$

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  $L(\mathcal{A}) = \{w \in \Sigma^* \mid \flat q_0 w \flat \vdash_{\mathcal{A}}^* k, \text{ wobei } k \text{ akzeptierende Stoppkonfiguration ist}\}$ .

**Definition 11.2 (Turing-akzeptierbar)**

Die Sprache  $L \subseteq \Sigma^*$  heißt *Turing-akzeptierbar*, falls es eine NTM  $\mathcal{A}$  gibt mit  $L = L(\mathcal{A})$ .

**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  $\flat$  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  $\beta$ ) vorhanden ist.
- Die Maschine blockiert, falls dies nicht so ist. Das kann auch bereits vorher geschehen, wenn ein erwartetes  $a$ ,  $b$  oder  $c$  nicht gefunden wird.

Eine entsprechende Turingmaschine  $\mathcal{A}$  kann z. B. wie folgt definiert werden:

$$\mathcal{A} = (\{q_0, q_{akz}, finde\_b, finde\_c, zu\_Ende\_?, zu\_Ende\_!, zurück\}, \\ \{a, b, c\}, \\ \{a, a', b, b', c, c', \beta\}, \\ q_0, \Delta, \{q_{akz}\}) \text{ mit } \Delta =$$

$(q_0,$	$\beta,$	$\beta,$	$n,$	$q_{akz}),$
$(q_0,$	$a,$	$a',$	$r,$	$finde\_b),$
$(finde\_b,$	$a,$	$a,$	$r,$	$finde\_b),$
$(finde\_b,$	$b',$	$b',$	$r,$	$finde\_b),$
$(finde\_b,$	$b,$	$b',$	$r,$	$finde\_c),$
$(finde\_c,$	$b,$	$b,$	$r,$	$finde\_c),$
$(finde\_c,$	$c',$	$c',$	$r,$	$finde\_c),$
$(finde\_c,$	$c,$	$c',$	$r,$	$zu\_Ende\_?),$
$(zu\_Ende\_?,$	$c,$	$c,$	$l,$	$zurück),$
$(zurück,$	$c',$	$c',$	$l,$	$zurück),$
$(zurück,$	$b,$	$b,$	$l,$	$zurück),$
$(zurück,$	$b',$	$b',$	$l,$	$zurück),$
$(zurück,$	$a,$	$a,$	$l,$	$zurück),$
$(zurück,$	$a',$	$a',$	$r,$	$q_0),$
$(zu\_Ende\_?,$	$\beta,$	$\beta,$	$l,$	$zu\_Ende\_!),$
$(zu\_Ende\_!,$	$c',$	$c',$	$l,$	$zu\_Ende\_!),$
$(zu\_Ende\_!,$	$b',$	$b',$	$l,$	$zu\_Ende\_!),$
$(zu\_Ende\_!,$	$a',$	$a',$	$l,$	$zu\_Ende\_!),$
$(zu\_Ende\_!,$	$\beta,$	$\beta,$	$n,$	$q_{akz})\}$

### 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  $(q, (a_1, \dots, a_k), (b_1, \dots, b_k), (d_1, \dots, d_k), q') \in \Delta$ :

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

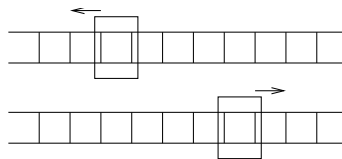
kann  $\mathcal{A}$  das Folgende tun:

- 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.

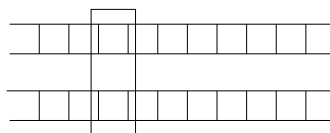
Das erste Band wird (o.B.d.A.) als Ein- und Ausgabeband verwendet.

**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  $\Gamma^k$  statt  $\Gamma$  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:

$$k_1 \vdash_{\mathcal{A}} k_2 \text{ gdw. } k'_1 \vdash_{\mathcal{A}'}^* k'_2$$

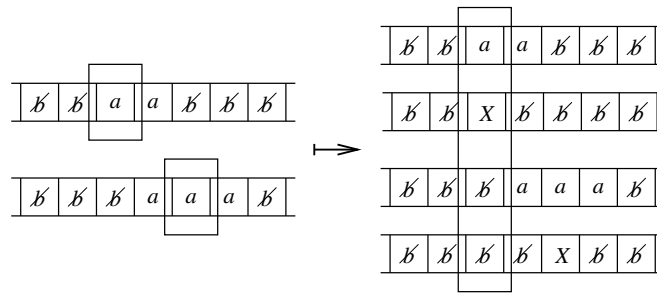
(Zur Simulation eines Schrittes von  $\mathcal{A}$  sind mehr Schritte nötig).

*Arbeitsalphabet von  $\mathcal{A}'$ :*  $\Gamma^{2k} \cup \Sigma \cup \{\flat\}$ ;

- $\Sigma \cup \{\flat\}$  wird für Eingabe benötigt;
- $\Gamma^{2k}$  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

$$\flat q_0 a_1 \dots a_m \flat$$

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

$\flat$	$\flat$	$a_1$	$a_2$	$\dots$	$a_m$	$\flat$	Spur 1 und 2 kodieren
$\flat$	$\flat$	$X$	$\flat$	$\dots$	$\flat$	$\flat$	Band 1
$\flat$	$\flat$	$\flat$	$\flat$	$\dots$	$\flat$	$\flat$	Spur 3 und 4 kodieren
$\flat$	$\flat$	$X$	$\flat$	$\dots$	$\flat$	$\flat$	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.*

War  $\mathcal{A}$  deterministisch, so liefert obige Konstruktion auch eine deterministische 1-Band-Turingmaschine.

Bei endlichen Automaten sind nichtdeterministische Automaten nicht stärker als deterministische, bei Kellerautomaten aber schon.

Bei Turingmaschinen: Sind DTM schwächer als NTM?

**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, zu jeder NTM 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

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

beginnend mit der Startkonfiguration  $k_0 = \#q_0w\#$  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  $i_1, \dots, i_n$  mit  $i_j \in \{1, \dots, r\}$  bestimmt dann von  $k_0$  aus für  $n$  Schritte die Auswahl der jeweiligen Transition, und somit von  $k_0$  aus eine feste Konfigurationsfolge

$$k_0 \vdash_{\mathcal{A}} k_1 \vdash_{\mathcal{A}} \dots \vdash_{\mathcal{A}} k_n .$$

Zählt man daher alle endlichen Wörter über  $\{1, \dots, r\}$  auf und zu jedem Wort  $i_1 \dots i_n$  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  $i_1 \dots i_n \in \{1, \dots, r\}^*$  erzeugt.
- Für jedes dieser Wörter wird auf dem dritten Band die zugehörige Konfigurationsfolge realisiert. Erreicht man hierbei eine akzeptierende Stoppkonfiguration von  $\mathcal{A}$ , so geht auch  $\mathcal{A}'$  in eine akzeptierende Stoppkonfiguration.

□

## 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.*

„ $\Rightarrow$ “: 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  $\$$  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  $\text{Regel}_1, \dots, \text{Regel}_k$  (entspricht der Entscheidung, dass man diese Regel anwenden will).
- 4) Es sei  $\text{Regel}_i$  der gewählte Zustand und  $\alpha \rightarrow \beta$  die entsprechende Produktion.  
Überprüfe, ob  $\alpha$  tatsächlich die Beschriftung des Bandstücks der Länge  $|\alpha|$  ab der gewählten Bandstelle ist.
- 5) Falls der Test erfolgreich war, so ersetze  $\alpha$  durch  $\beta$ .  
Vorher müssen bei  $|\alpha| < |\beta|$  die Symbole  $\neq \$$  rechts von  $\alpha$  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  $\#$  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).

„ $\Leftarrow$ “: 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“  $\#$ -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 Wort  $w$ .

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

$$(\Sigma \cup \{\#\}) \times \Gamma,$$

wobei man

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

**Formale Definition:**

$$N = \{S, A, B, E\} \cup Q \cup (\Sigma \cup \{\#\}) \times \Gamma,$$

wobei

- $S, A, B$  zum Aufbau des Rechenbandes am Anfang,
- $E$  zum Löschen am Schluss,
- $Q$  zur Darstellung des aktuellen Zustandes,
- $\Sigma \cup \{\#\}$  zum Speichern von  $w$  und
- $\Gamma$  zur  $\mathcal{A}$ -Berechnung

dienen.

**Regeln:**

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

$$S \rightarrow Bq_0A$$

$$A \rightarrow [a, a]A \text{ für alle } a \in \Sigma$$

$$A \rightarrow B$$

$$B \rightarrow [\#, \#]B$$

$$B \rightarrow \varepsilon$$

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

$$S \vdash_G^* [\mathcal{B}, \mathcal{B}]^k q_0 [a_1, a_1] \dots [a_n, a_n] [\mathcal{B}, \mathcal{B}]^l .$$

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

- $p[a, b] \longrightarrow [a, b']q$   
falls  $(p, b, b', r, q) \in \Delta$ ,  $a \in \Sigma \cup \{\mathcal{B}\}$
- $[a, c]p[a', b] \longrightarrow q[a, c][a', b']$   
falls  $(p, b, b', l, q) \in \Delta$ ,  $a, a' \in \Sigma \cup \{\mathcal{B}\}$ ,  $c \in \Gamma$
- $p[a, b] \longrightarrow q[a, b']$   
falls  $(p, b, b', n, q) \in \Delta$ ,  $a \in \Sigma \cup \{\mathcal{B}\}$

**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  $a \in \Sigma$ ,  $b \in \Gamma$   
 $q[\mathcal{B}, b] \longrightarrow E$  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)
- $E[\mathcal{B}, b] \longrightarrow E$  für  $b \in \Gamma$   
(Entfernen des zusätzlich benötigten Arbeitsbandes nach rechts)
- $[\mathcal{B}, b]E \longrightarrow E$  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:

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

Für Typ-0-Sprachen gelten die folgenden Abschlusseigenschaften:

**Satz 12.2**

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

*Beweis.* 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

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

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  $L_1 \cap L_2$ .

□

**Beachte:**

Man kann zeigen, dass die Klasse der Turing-akzeptierbaren Sprachen nicht unter Komplement abgeschlossen ist.

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  $\notin, \$$ .

**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



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

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

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

**Korollar 12.5**

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

*Beweis.*

„ $\Rightarrow$ “: Verwende die Konstruktion aus dem Beweis zu Satz 12.1.

Da alle Produktionen von kontextsensitiven Grammatiken nichtkürzend sind (mit Ausnahme  $S \rightarrow \varepsilon$ ), 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.

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

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

Wie im Beweis von Satz 9.4 illustriert, kann man diese in eine kontextsensitive Grammatik transformieren.

**Idee:**

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

$$E[\not, \not] \rightarrow E$$

$$[\not, \not]E \rightarrow E$$

Es gibt allerdings noch einige technische Probleme:

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

**Lösung:**

Führe die Symbole  $\ell$ ,  $\$$  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 Szelepcsényi). □

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

**Satz 12.7**

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

*Beweis.* Da kontextsensitive Produktionen (bis auf Spezialfall  $S \rightarrow \varepsilon$ ) 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. Allg.	im Allgemeinen
LBA	linear beschränkter Automat
NEA	nichtdeterministischer endlicher Automat
NP	nichtdeterministisch polynomiell
NTM	nichtdeterministische Turingmaschine
o.B.d.A.	ohne Beschränkung der Allgemeingültigkeit
PDA	pushdown automaton (Kellerautomat)
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