# Automatentheorie und formale Sprachen

Vorlesungsmitschrift
von Achim Lücking

zur Vorlesung von
Prof. Dr. Klaus Indermark

Sommersemester 1999[1]

[1]Stand: 23. Oktober 2000

# Vorwort

Werte Freunde der Automatentheorie und der formalen Sprachen!

Nach nun einiger Zeit ist endlich das Skript fertiggestellt. Das Skript basiert auf der Vorlesung von Prof. Dr. Indermark aus dem Sommersemester 1999 an der RWTH Aachen. Dabei weise ich ausdrücklich daraufhin, daß es sich hierbei um **kein offizielles Skript** handelt und **kein Anspruch auf Vollständigleit und Richtigkeit** erhoben wird. Dieses Skript stützt sich ausschließlich auf meine Vorlesungsaufzeichnungen und -mitschriften.
Sollten sich noch Fehler eingeschlichen haben, sei es inhaltlich oder einfach nur satztechnisch, so möchte ich Euch bitten, mir dieses per e-Mail zu melden, damit ich das Skript entsprechend verbessern und optimieren kann. e-Mails bitte an folgende Adresse: `atfs-skript@achim-luecking.de`. Die jeweils neueste Version des Skriptes gibt es im PDF-Format auf meiner Homepage unter `http://www.achim-luecking.de` zum Download. Da das Skript aufgrund von Fehlermeldungen laufend aktualisiert wird. empfiehlt es sich, in regelmäßigen Abständen vorbeischauen.

Hinweisen möchte ich ferner auf die Notation im Skript: Kästchen, der Form $\boxed{\mathbf{1.2}}$ bezeichnen einen Verweis auf eine in der Vorlesung an dieser Stelle gezeigte Folie mit der angegebenen Nummer. Die Folien beinhalten noch zusätzliche Abbildungen, Grafiken und Übersichten, die im Skript so nicht berücksichtigt sind. Die **Folien sind nicht Bestandteil des Skriptes** und müssen, sofern noch vorhanden, getrennt vom Server des Lehrstuhls heruntergeladen werden oder von jemandem, der sie noch hat, kopiert werden.

Mein Dank bei der Erstellung dieses Skriptes geht an Olaf Chitil (betreuender Assistent der Vorlesung) für die "fachspezifische TEX-Nachhilfe", Tanja Schroedel für die "mentale" Unterstützung, Markus Roeffen für die ergänzenden Unterlagen (wenn mir mal was fehlte) und Markus Klinke dafür, daß er relativ schnell aufgehört hat zu nerven, wann denn das Skript fertig sei (*g*).

Viel Spaß mit dem Skript !

Aachen, den 23. Oktober 2000

Achim Lücking

# Inhaltsverzeichnis

# Kapitel 1

# Alphabete, Wörter, Sprachen

Dieses Kapitel behandelt Zeichen als Grundobjekte der Informatik. Um die Automatentheorie und die formalen Sprachen zu begreifen, sind aber zunächst ein paar Definitionen notwenig.

**Definition 1.1** *$\Sigma$ ist ein Alphabet. Ein Alphabet ist eine nicht-leere endliche Menge.*

**Definition 1.2** *$a \in \Sigma$ ist ein Buchstabe des Alphabets $\Sigma$. $a$ wird in der Literatur auch als Character oder Symbol bezeichnet.*

$\boxed{\textbf{0.1}}$

**Definition 1.3** *$\Sigma^*$ ist Menge der Wörter über $\Sigma$, d.h. die Menge aller aus $\Sigma$ bildbaren Wörter.*

$$\Sigma^* = \{a_1 a_2 ... a_n \mid n \in \mathbb{N}, a_i \in \Sigma\}$$

*Wörter werden zuweilen auch als Zeichenreihe, Zeichenkette oder String bezeichnet. Das Wort der Länge 0 ($n = 0$) wird als das leere Wort bezeichnet und mit $\varepsilon$ dargestellt.*

## 1.1 Operationen auf $\Sigma^*$

Auf der Menge $\Sigma^*$ lassen sich nun verschiedene Operationen durchführen, die nun näher betrachtet werden sollen.

- **Verkettung (Konkatenation):**

$$\cdot : \ \Sigma^* \times \Sigma^* \longrightarrow \Sigma^*$$

$$(w, v) \longmapsto w \cdot v := wv$$

Es gilt:

1. $\cdot$ ist assoziativ: $(u \cdot v) \cdot w = u \cdot (v \cdot w)$
2. $\varepsilon$ ist $\cdot$-neutral: $\varepsilon \cdot w = w \cdot \varepsilon = w$

Sprechweise: $(\Sigma^*; \cdot, \varepsilon)$ ist ein **Monoid** (Halbgruppe mit 1).
Es liegt eine freie Erzeugung vor.

- **Länge eines Wortes $w = a_1 a_2 ... a_n \in \Sigma^*$:**

$$|a_1 a_2 ... a_n| := n,$$

falls alle $a_i \in \Sigma$, also:

$$\varepsilon = 0$$

und

$$|wv| = |w| + |v|$$

- **Potenzen eines Wortes $w \in \Sigma^*$:**
Die Potenz eines Wortes wird indirekt definiert:

$$w^0 := \varepsilon$$

$$w^{n+1} := w \cdot w^n$$

- **Spiegelbild eines Wortes:**

$$\varepsilon^R := \varepsilon$$

$$(wa)^R := a(w)^R$$

## 1.2 Formale Sprachen

$\wp(\Sigma^*)$ ist die Menge der formalen Sprachen über $\Sigma$. Es gilt:

$$\wp(\Sigma^*) := \{L \mid L \subseteq \Sigma^*\}$$

Als Beispiel seien $\emptyset, \{\varepsilon\}, \{w_1, w_2, ..., w_n\}, \Sigma^*$ genannt. Auch auf der Menge der formalen Sprachen lassen sich Operationen durchführen, die im folgenden näher betrachtet werden.

### 1.2.1 Operationen auf $\wp(\Sigma^*)$:

- **boolesche Operationen:**

$$L_1 \cup L_2,\ L_1 \cap L_2,\ \bar{L} := \Sigma^* \backslash L$$

- **Komplexprodukt:**

$$L_1 L_2 := \{w_1 w_2 \mid w_i \in L_i\}$$

Hier gilt das Prinzip "jeder mit jedem": jedes Wort der ersten Sprache wird mit jedem Wort der zweiten Sprache verknüpft.

$$LL := \{w_1 w_2 \mid w_i \in L\}$$

- **Potenzen einer Sprache:**

$$L^0 := \{\varepsilon\}$$

$$L^{n+1} := LL^n$$

- **Stern einer Sprache (Iteration, Repetition):**

$$L^* := \bigcup_{n \in \mathbb{N}} L^n$$

($\varepsilon \in L^*$ gilt immer, da $L^0 = \{\varepsilon\}$)

$$L^+ := LL^* = \bigcup_{n=1}^{\infty} L^n$$

- **Spiegelbild einer Sprache:**

$$L^R : 0\{w^R \mid w \in L\}$$

- **reguläre Operationen:**

$$L_1 \cup L_2, L_1 L_2, L^*$$

$\boxed{\textbf{1.2}}$ $\boxed{\textbf{1.3}}$ $\boxed{\textbf{1.4}}$

# Kapitel 2

# Reguläre Ausdrücke und endliche Automaten

Dieses Kapitel behandelt primär die Eigenschaften von Automaten.

**2.1**

## 2.1 Reguläre Ausdrücke

Reguläre Ausdrücke sind endliche Beschreibungen unendlicher Sprachen. Ein wesentliches Hilfsmittel ist der Stern *.

**Definition 2.1** *Sei $\Sigma$ ein Alphabet. Die Menge $RA(\Sigma)$ der regulären Ausdrücke über $\Sigma$ ist induktiv definiert durch:*

1. $\Lambda \in RA(\Sigma)$
2. $a \in RA(\Sigma)$ *für jedes* $a \in \Sigma$
3. $(\alpha \vee \beta) \in RA(\Sigma)$ *falls* $\alpha, \beta \in RA(\Sigma)$
4. $(\alpha \cdot \beta) \in RA(\Sigma)$ *falls* $\alpha, \beta \in RA(\Sigma)$
5. $(\alpha^*) \in RA(\Sigma)$ *falls* $\alpha \in RA(\Sigma)$

*Die oben aufgeführten Punkte werden auch als Syntax regulärer Ausdrücke bezeichnet.*

**Bemerkung 2.2** *Zur Vereinfachung werden folgende Regeln bei der Schreibweise vorausgesetzt:*

- ***Präzedenzregel**, um Klammern zu sparen*<br>$*$ *bindet stärker als* $\cdot$<br>$\cdot$ *bindet stärker als* $\vee$
- *Das* $\cdot$ *wird unterdrückt*

Bei der obigen Definition ist zu beachten, daß $RA(\Sigma)$ selbst eine formale Sprache ist. Ferner gilt:

$$RA(\Sigma) \subseteq (\Sigma \cup \{\Lambda, (, ), \cdot, \vee, *\})^*$$

**Definition 2.3** *Ein regulärer Ausdruck beschreibt eine formale Sprache über $\Sigma$:*

$$[\![\_]\!] : RA(\Sigma) \longrightarrow \wp(\Sigma^*)$$

*Die wie folgt beschriebenen Sprachen heißen regulär:*

- $[\![\Lambda]\!] := \emptyset$
- $[\![a]\!] := \{a\}$
- $[\![(\alpha \vee \beta)]\!] := [\![\alpha]\!] \cup [\![\beta]\!]$
- $[\![(\alpha \cdot \beta)]\!] := [\![\alpha]\!] \cdot [\![\beta]\!]$
- $[\![(\alpha^*)]\!] := [\![\alpha]\!]^*$

*Man spricht auch von der* ***Semantik*** *regulärer Ausdrücke.*

**Definition 2.4** *Die Klasse $REG(\Sigma)$ der regulären Ausdrücke über $\Sigma$ ist induktiv definiert durch*

- $\emptyset, \{a\} \in REG(\Sigma)$ *für alle* $a \in \Sigma$
- $L, L' \in REG(\Sigma) \Rightarrow L \cup L', LL', L^* \in REG(\Sigma)$

$\boxed{\textbf{2.3}}$

Die Menge der **WHILE-Programme**[1] ist induktiv aufgebaut über den Mengen

$$A := \{X_i := X_i + 1, X_i := X_i - 1 \mid i \in \mathbb{N}\}$$

und

$$A := \{X_i > 0 \mid i \in \mathbb{N}\}$$

durch

1. $A \subseteq WP$
2. $P, Q \in WP \Longrightarrow \underline{\text{begin}}\ P; Q\ \underline{\text{end}} \in WP$
3. $b \in B; P, Q \in WP \Longrightarrow \underline{\text{if}}\ b\ \underline{\text{then}}\ P\ \underline{\text{else}}\ Q \in WP$
4. $b \in B; P \in WP \Longrightarrow \underline{\text{while}}\ b\ \underline{\text{do}}\ P \in WP$

Sei $\Sigma_{PATH} := A \cup (B \times \{true, false\})$ und $b_{true} = (b, true)$. Für $P \in WP$ definieren wir die formale Pfadsprache $L_P \subseteq \Sigma^*_{PATH}$ wie folgt:

1. $L_a := \{a\}$ für alle $a \in A$
2. $L_{\underline{begin}\ P;\ Q\ \underline{end}} := L_P L_Q$
3. $L_{\underline{if}\ b\ \underline{then}\ P\ \underline{else}\ Q} := \{b_{true}\}L_P \cup \{b_{false}\}L_Q$
4. $L_{\underline{while}\ b\ \underline{do}\ P} := (\{b_{true}\}L_P)^*\{b_{false}\}$

[1]vgl. Vorlesung "Berechenbarkeit und Komplexität" von Prof. Dr. Thomas aus dem Wintersemester 1998/99 an der RWTH Aachen

Es folgt die Beschreibung von $L_P$ durch $\alpha_P \in RA(\Sigma_{PATH})$, nämlich:

1. $\alpha_a := \{a\}$ für alle $a \in A$
2. $\alpha_{\underline{begin}\ P;\ Q\ \underline{end}} := \alpha_P \alpha_Q$
3. $\alpha_{\underline{if}\ b\ \underline{then}\ P\ \underline{else}\ Q} := b_{true}\alpha_P \vee b_{false}\alpha_Q$
4. $\alpha_{\underline{while}\ b\ \underline{do}\ P} := (b_{true}L_P)^* b_{false}$

Es gilt dabei die Analogie zwischen WHILE-Programmen, $RA$ und $\Sigma_{PATH}$ zu beachten. Jedoch existiert eine Abstraktion zwischen Anweisungs- und Bedingungssemantik.

**Definition 2.5** *$P$ ist pfadäquivalent $Q$* $:\Longleftrightarrow L_P = L_Q$

$$P \sim_{PATH} Q :\Longleftrightarrow L_P = L_Q$$

**Folgerung 2.6**

$$P \sim_{PATH} Q \Longrightarrow P \sim Q$$

*$P \sim_{PATH} Q$ ist durch endliche Automaten entscheidbar, $P \sim Q$ ist unentscheidbar, d.h. $(f_P = f_Q)$.*

Es gilt für $RA(\Sigma)$:

$$\alpha \longmapsto [\![\alpha]\!] \subseteq \Sigma^*$$

Außerdem gelten die Beziehungen:

$$L(\alpha) := [\![\alpha]\!]$$

$$\mathcal{L}(\Sigma, RA) = REG(\Sigma)$$

### 2.1.1 Standardprobleme für RA($\Sigma$)

Für die regulären Ausdrücke gibt es die folgenden 3 Standardprobleme:

1. **Wortproblem (matching problem)**
   Gilt $w \in [\![\alpha]\!]$ für $w \in \Sigma^*$ und $\alpha \in RA(\Sigma)$ ?
2. **Leerheitsproblem**
   Gilt $[\![\alpha]\!] = \emptyset$ für $\alpha \in RA(\Sigma)$ ?
3. **Äquivalenzproblem**
   Gilt $[\![\alpha]\!] = [\![\beta]\!]$ für beliebige Ausdrücke $\alpha, \beta \in RA(\Sigma)$ ?
   Schreibweise: $\alpha \sim \beta :\Leftrightarrow [\![\alpha]\!] = [\![\beta]\!]$

Alle Probleme sind mit endlichen Automaten entscheidbar.

## 2.2 Deterministische endliche Automaten

**Definition 2.7** *Seien* $Q, \Sigma$ *nicht-leere endliche Mengen.* $q_0 \in Q, F \subseteq Q$ *und* $\delta : Q \times \Sigma \longrightarrow Q$*. Dann heißt*

$$\mathfrak{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$$

*ein deterministischer Automat über* $\Sigma$ *mit der Zustandsmenge* $Q$*, dem Eingabealphabet* $\Sigma$*, der Transitionsfunktion* $\delta$*, dem Anfangszustand* $q_0$ *und der Endzustandsmenge* $F$*.*
*Schreibweise:* $\mathfrak{A} \in DFA(\Sigma)$

$\boxed{\textbf{2.2}}$ $\boxed{\textbf{2.3}}$

**Definition 2.8** *Der Automat* $\mathfrak{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ *bestimmt die erweiterte Transitionsfunktion*

$$\bar{\delta} : Q \times \Sigma \longrightarrow Q$$

*mit*

$$\bar{\delta}(q, \varepsilon) := q$$

$$\bar{\delta}(q, wa) := \bar{\delta}(\delta(q_0, a), w)$$

$$\left[\bar{\delta}(q, aw) := \bar{\delta}(\delta(q_0, a), w)\right]$$

*und damit die vom Automat* $\mathfrak{A}$ *erkannte Sprache*

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

Das Ziel: $\mathcal{L}(\Sigma, DFA) = REG(\Sigma)$. Als Hilfsmittel soll der nicht-deterministische Automat dienen.

## 2.3 Nicht-deterministische endliche Automaten

**Definition 2.9** *Seien* $Q, \Sigma, q_0$ *und* $F$ *wie bei einem* $\mathfrak{A} \in DFA$ *gegeben. Sei ferner* $\Sigma_\varepsilon := \Sigma \cup \{\varepsilon\}$ *und* $\delta : Q \times \Sigma_\varepsilon \to \wp(Q)$*. Dann heißt*

$$\mathfrak{A} = \langle Q, \Sigma_\varepsilon, \delta, q_0, F\rangle$$

*ein nicht-deterministischer endlicher Automat über* $\Sigma$*. Er wird auch als* $NFA(\Sigma)$ *oder* $NFA$ *bezeichnet.*

Es folgt: $DFA(\Sigma) \subseteq NFA(\Sigma)$. Werttransitionen sind dabei durch Zwischentransitionen realisierbar.
Die Semantik eines $\mathfrak{A} \in NFA(\Sigma)$ ist definiert als die Menge der Konfigurationen von $\mathfrak{A} : Q \times \Sigma^*$. Diese werden als **Transitionen** bezeichnet.
Für alle $q, q' \in Q, w \in \Sigma^*, a \in \Sigma$ definieren wir:

- $\Sigma$-Transitionen
$$(q, aw) \vdash (q', w) \Longleftrightarrow q' \in \delta(q, a)$$
- $\varepsilon$-Transitionen
$$(q, w) \vdash (q', w) \Longleftrightarrow q' \in \delta(q, \varepsilon)$$

Dann ist die durch $\mathfrak{A}$ erkannte Sprache definiert als:

$$L(\mathfrak{A}) := \{w \in \Sigma^* \mid (q_0, w) \overset{*}{\vdash} (q, \varepsilon), q \in F\}$$

Das **Ziel** soll sein:

$$\begin{array}{rcl} \alpha \in RA(\Sigma) & \longmapsto & \mathfrak{A}(\alpha) \in NFA(\Sigma) \\ & \longmapsto & \mathfrak{A}(\alpha)_P \in DFA(\Sigma) \end{array}$$

mit $[\![\alpha]\!] = L(\mathfrak{A}(\alpha)) = L(\mathfrak{A}(\alpha)_P)$.

**Definition 2.10** *Sei $\mathfrak{A} \in NFS(\Sigma)$. Dann ist der Potenzmengenautomat $\mathfrak{A}_P = \langle Q_P, \Sigma, \delta_P, q_{0P}, F_P\rangle$ definiert durch*

- $Q_P := \{T \subseteq Q \mid \exists w \in \Sigma^* : \forall q \in T : (q_0, w) \overset{*}{\vdash} (q, \varepsilon)\}$
- $q_{0P} := \{q \in Q \mid (q_0, \varepsilon) \overset{*}{\vdash} (q, \varepsilon)\}$
- $F_P := \{T \in Q_P \mid T \cap F \neq \emptyset\}$
- $\delta_P : Q_P \times \Sigma \longrightarrow Q_P$
  $\delta_P(T, a) := \{q' \in Q \mid (q, a) \overset{*}{\vdash} (q', \varepsilon) \text{ für ein } q \in T\}$

Dabei ist folgendes zu beachten:

$$\begin{array}{rcl} T \in Q_P & \Longrightarrow & \exists w \in \Sigma^* : T = \{q' \in Q \mid (q_0, w) \overset{*}{\vdash} (q', \varepsilon)\} \\ & \Longrightarrow & \delta_P(T, a) = \{q' \in Q \mid (q_0, wa) \overset{*}{\vdash} (q', \varepsilon)\} \in Q_P \end{array}$$

**Lemma 2.11** *Sei $\mathfrak{A} \in NFS(\Sigma)$. Dann gilt: $L(\mathfrak{A}) = L(\mathfrak{A}_P)$*

**Beweis 2.12** *Zunächst zeigen wir, daß für alle $w \in \Sigma^*$ und $q \in Q$ gilt:*

$$q \in \overline{\delta_P}(q_{0P}, w) \Longleftrightarrow (q_0, w) \overset{*}{\vdash} (q, \varepsilon)$$

*(i) $w = \varepsilon$:*

$$q \in \overline{\delta_P}(q_{0P}, \varepsilon) \Longleftrightarrow q \in q_{0P} \Longleftrightarrow (q_0, \varepsilon) \overset{*}{\vdash} (q, \varepsilon)$$

*(ii) $w = w'a$:*

$$\begin{array}{rc} & q \in \overline{\delta_P}(q_{0P}, w'a) = \delta(\overline{\delta_P}(q_{0P}, w'), a) \\ \Longleftrightarrow & \exists q' \in \overline{\delta_P}(q_{0P}, w') : (q', a) \overset{*}{\vdash} (q, \varepsilon) \\ \Longleftrightarrow & \exists q' \in Q : (q_0, w'a) \overset{*}{\vdash} (q', a) \overset{*}{\vdash} (q, \varepsilon) \\ \Longleftrightarrow & (q_0, w'a) \overset{*}{\vdash} (q, \varepsilon) \end{array}$$

*Daraus folgt die Behauptung:*

$$\begin{array}{rcl} w \in L(\mathfrak{A}) & \Longleftrightarrow & \exists q \in F : (q_0, w) \overset{*}{\vdash} (q, \varepsilon) \\ & \Longleftrightarrow & q \in \overline{\delta_P}(q_{0P}, w) \text{ und } q \in F \\ & \Longleftrightarrow & \overline{\delta_P}(q_{0P}, w) \in F \\ & \Longleftrightarrow & w \in L(\mathfrak{A}) \end{array}$$

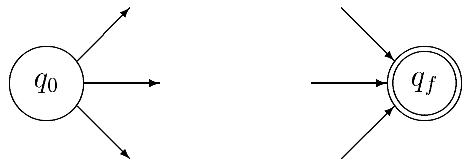
□

## 2.4 Synthese und Analyse endlicher Automaten

Die Synthese eines Automaten bedeutet, für $\alpha \in RA(\Sigma)$ einen äquivalenten $\mathfrak{A}(\alpha) \in NFA$ zu konstruieren, d.h. $[\![\alpha]\!] = L(\mathfrak{A}(\alpha))$.

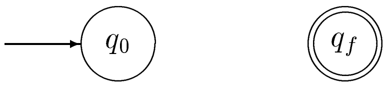
### 2.4.1 Algorithmus von Thomson

Die Idee ist, daß ein $\mathfrak{A}(\alpha)$ genau einen Endzustand $q_f$ hat. $q_0$ ist die Quelle und $q_f$ ist die Senke im Zustandsgraphen.

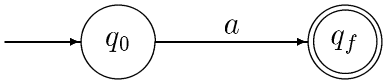



Dann läßt sich der Automat nach folgenden Regeln konstruieren:

- $\mathfrak{A}(\Lambda)$:

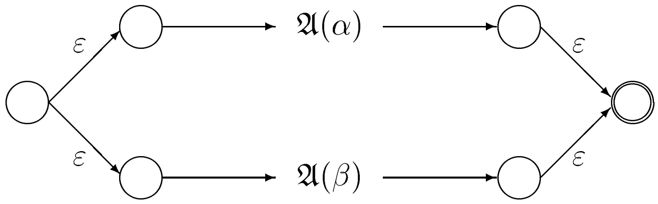



- $\mathfrak{A}(a)$:

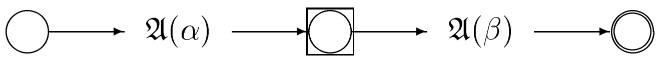



$a \in \Sigma$

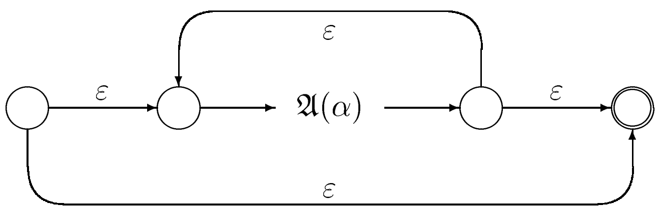
- $\mathfrak{A}(\alpha \vee \beta)$:




- $\mathfrak{A}(\alpha \cdot \beta)$:




- $\mathfrak{A}(\alpha^*)$:




Offensichtlich gilt für jedes $\alpha \in RA(\Sigma)$: $[\![\alpha]\!] = L(\mathfrak{A}(\alpha))$

**Korollar 2.13** *Es gilt:* $REG(\Sigma) \subseteq \mathcal{L}(\Sigma, DFA)$.

### 2.4.2 Analyse

Konstruiere für einen $\mathfrak{A} \in DFA(\Sigma)$ einen $\alpha(\mathfrak{A}) \in RA(\Sigma)$ mit $[\![\alpha(\mathfrak{A})]\!] = L(\mathfrak{A})$. Sei nun $\mathfrak{A} = \langle Q, \Sigma, \delta, q_1, F\rangle \in DFA(\Sigma)$ und $Q = \{q_1, \dots, q_n\}$. Für $i, j \in \{1, \dots, n\}$ und $k \in \{0, 1, \dots, n\}$ definieren wir

$$W_{ij}^k := \{w \in \Sigma \mid w \text{ überführt } q_i \text{ in } q_j \text{ o. Benutz. von } q_{k+1}, \dots, q_n \text{ als Zwischenzust.}\}$$

Dann gilt: $L(\mathfrak{A}) = \bigcup_{q_j \in F} W_{1j}^n$. Somit genügt der Nachweis der Regularität der Sprachen $W_{ij}^k$ per Induktion über $k$:

(1) $k = 0$:
$$W_{ij}^0 \subseteq \Sigma_\varepsilon \text{ ist regulär, } \{\varepsilon\} = [\![\Lambda^*]\!]$$

(2) $k - 1 \to k$:
$$W_{ij}^k = W_{ij}^{k-1} \cup W_{ik}^{k-1} \cup \left(W_{kk}^{k-1}\right)^* \cup W_{kj}^{k-1}$$

$\boxed{\textbf{2.6}}$ $\boxed{\textbf{2.11}}$ $\boxed{\textbf{2.12}}$

**Satz 2.14** *(Satz von Kleene) Es gilt:* $\mathcal{L}(\Sigma, DFA) = REG(\Sigma)$

**Korollar 2.15** $REG(\Sigma)$ *ist auch unter Schnitt und Komplement abgeschlossen.*

**Beweis 2.16** *(Idee)*

*(a) Komplement:* $F' := Q \setminus F$

*(b)* $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

□

## 2.5 Das Pumping-Lemma

Das sogenannte **Pumping-Lemma** ist ein Hilfsmittel zum Nachweis nicht-regulärer Sprachen.

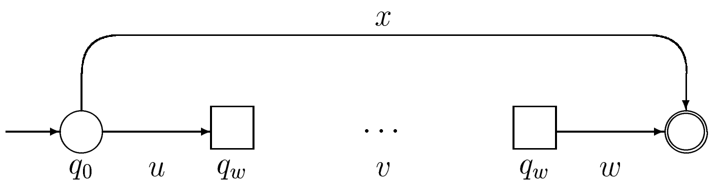
**Satz 2.17** *Sei* $L \in REG(\Sigma)$. *Dann existiert ein* $k \in \mathbb{N}$, *so daß für jedes* $x \in L$ *mit* $|x| \geq k$ *eine Zerlegung* $x = uvw$ *mit folgenden Eigenschaften existiert:*

*(i)* $|v| \geq 1$

*(ii)* $|uv| \leq k$

*(iii)* $uv^iw \in L$ *für alle* $i \in \mathbb{N}$, *insbesondere für* $i = 0$

**Beweis 2.18** *Sei* $\mathfrak{A} \in DFA(\Sigma)$ *mit* $L(\mathfrak{A}) = L$ *und* $k = |Q|$. *Sei ferner* $x \in L$ *mit* $|x| \geq k$. *Dann sieht die Erkennung von* $x$ *in* $\mathfrak{A}$ *wie folgt aus:*



*Seien die beiden $q_w$ das erste Wiederholungspaar von Zuständen, so folgen die Eigenschaften (i),(ii) und (iii).*

□

Beim Pumping-Lemma ist unbedingt zu beachten, daß es sich lediglich um eine **notwendige Bedingung** handelt, jedoch **keine hinreichende Eigenschaft** regulärer Sprachen beschreibt. Es ist daher zum Nachweis nicht-regulärer Sprachen über einen Widerspruchsbeweis geeignet.
Als Beispiel wählen wir nun die folgende Sprache:

$$L = \{a^n b^n \mid n \geq 1\} \notin REG(\{a, b\})$$

Der Beweis sieht dann wie folgt aus: Angenommen, $L \in REG(\{a, b\})$. Dann existiert ein $k \in \mathbb{N}$ mit den Eigenschaften des Pumping-Lemmas (auch "Pumping-Index" genannt). Für $x = a^k b^k$ muß dann eine Zerlegung existieren ($a^k b^k = uvw$) mit $v \neq \varepsilon$ und $|uv| \leq k$, also $v \in \{a^i \mid i > 0\}$ und $uw = a^{k-|v|} b^k \in L$. Dies führt zum Widerspruch, womit der Beweis getätigt wäre.

□

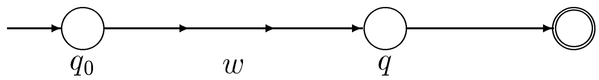
## 2.6 Zustandsreduktion endlicher Automaten

Das Ziel ist, einen endlichen Automaten mit minimaler Zustandsmenge zu konstruieren. Dazu werden zunächst einige Definitionen benötigt:

**Definition 2.19** *Für $L \subseteq \Sigma^*$ und $w \in \Sigma^*$ heißt $d_w(L) := \{v \in \Sigma^* \mid wv \in L\}$ die Ableitung von L nach w.*

**Lemma 2.20** *Sei $L = L(\mathfrak{A})$ für $\mathfrak{A} = \langle Q, \Sigma, \delta, q_0, F\rangle \in DFA(\Sigma)$. Dann gilt für $D(L) := \{d_w(L) \mid w \in \Sigma^*\}$ die Beziehung $|D(L)| \leq |Q|$.*

**Beweis 2.21** *Für $q \in Q$ bezeichne $L(q) := L(\langle Q, \Sigma, \delta, q, F\rangle)$. Dann gilt: $d_w(L) = d_w(L(q_0)) = L(\overline{\delta}(q_0, w))$*




□

**Definition 2.22** *Der Ableitungsautomat $\mathfrak{A}_L = \langle D(L), \Sigma, \delta, q_0, F\rangle \in DFA(\Sigma)$ ist für $L \in REG$ definiert durch:*

- $q_0 := d_\varepsilon(L) = L$
- $F := \{d_w(L) \mid w \in L\}$, *d.h.* $\varepsilon \in d_w(L)$
- $\delta(d_w(L), a) := d_{wa}(L)$

Dabei ist zu beachten, daß gilt:

$$d_w(L) = d_v(L) \Longrightarrow d_{wa}(L) = d_{va}(L)$$

**Lemma 2.23** *Es gilt:* $L(\mathfrak{A}_L) = L$

**Beweis 2.24**

$$\begin{aligned} w \in L(\mathfrak{A}_L) &\iff \overline{\delta}(q_0, w) \in F \\ &\iff d_w(L) \in F \\ &\iff w \in L \end{aligned}$$

□

**Korollar 2.25** *Es folgt:*

*(i) der Ableitungsautomat ist zustandsminimal*

*(ii) für* $L \subseteq \Sigma^*$ *gilt:* $L \in REG \iff |D(L)| < \infty$

Eine Zustandsrduktion besteht aus der Konstruktion eines zustandsminimalen Automaten aus einem gegebenen Automaten durch

- Weglassen nicht erreichbarer Zustände.
- Verschmelzen äquivalenter Zustände.

Dann heißt

- $q \in Q$ **erreichbar** $:\iff \exists w \in \Sigma^* : \overline{\delta}(q_0, w) = q$
- $q_1 \sim q_2$ **äquivalent** $:\iff L(q_1) = L(q_2)$

**Definition 2.26** *Für* $\mathfrak{A} = \langle Q, \Sigma, \delta, q_0, F\rangle \in DFA(\Sigma)$ *ist der Faktorautomat* $\mathfrak{A}_\sim = \left\langle \widetilde{Q}, \Sigma, \widetilde{\delta}, \widetilde{q_0}, \widetilde{F} \right\rangle \imath n DFA($
*wie folgt definiert:*

- $\widetilde{Q} := \{[\overline{\delta}(q_0, w)] \mid w \in \Sigma^*\}$
- $\widetilde{q_0} := [q_0]$
- $\widetilde{F} := \{[q] \mid q \in F\}$
- $\widetilde{\delta}\ ([q], a) := [\delta(q, a)]$

Dabei ist zu beachten, daß gilt:

$$q \sim q' \implies L(q) = L(q') \implies \delta(q, a) \sim \delta(q', a)$$

**Lemma 2.27** *Für* $\mathfrak{A} \in DFA(\Sigma)$ *gilt:* $\mathfrak{A}_\sim = \mathfrak{A}_{L(\mathfrak{A})}$ *(mit Ausnahme der Zustandsnamen) Dabei bezeichne* $\mathfrak{A}_{L(\mathfrak{A})}$ *den Ableitungsautomat über* $L(\mathfrak{A})$*. Insbesondere ist* $\mathfrak{A}_\sim$ *äquivalent zu* $\mathfrak{A}$*, und so folgt: Zustandsminimale Automaten sind bis auf Isomorphie eindeutig bestimmt.*

**Beweis 2.28** *Sei* $\beta : \widetilde{Q} \longrightarrow D(L(\mathfrak{A}))$ *definiert durch:* $\beta([\overline{\delta}(q_0, w)]) := d_w(L(\mathfrak{A}))$*. Dann ist* $\beta$

- *unabhängig von Repräsentanten $w \in \Sigma^*$:*

$$\begin{aligned} & \overline{\delta}(q_0, v) \sim \overline{\delta}(q_0, w) \\ \Longrightarrow \quad & L(\overline{\delta}(q_0, v)) = L(\overline{\delta}(q_0, w)) \\ \Longrightarrow \quad & d_w(L(\mathfrak{A})) = d_v(L\mathfrak{A}) \end{aligned}$$

- *bijektiv, weil obige Folgerung umkehrbar ist.*

- *strukturerhaltend:*

  – $\beta([q_0]) = \beta([\overline{\delta}(q_0, \varepsilon)]) = d_\varepsilon(L(\mathfrak{A}))$ *ist Anfangszustand des Ableitungsautomaten.*

  – $\overline{\delta}(q_0, w) = q \in F : \beta([\overline{\delta}(q_0, w)]) = d_w(L(\mathfrak{A}))$ *ist Endzustand des Ableitungsautomaten.*

  – *Es gilt:*

$$\begin{aligned} & \overline{\delta}([q], a) = [\delta(q, a)] \\ \Longrightarrow \quad & \tilde{\delta}([\overline{\delta}(q_0, w)], a) = [\overline{\delta}(q_0, wa)] \\ \Longrightarrow \quad & \delta(d_w(L(\mathfrak{A})), a) = d_w(L(\mathfrak{A})) \end{aligned}$$

□

**Definition 2.29** *Sei $\mathfrak{A} = \langle Q, \Sigma, \delta, q_0, F \rangle \in DFA(\Sigma)$ und zustandstandserreichbar. Sei ferner $k \in \mathbb{N}, q_1, q_2 \in Q$. Dann sagt man:*

- *$q_1$ ist k-äquivalent zu $q_2$ ($q_1 \overset{k}{\sim} q_2$) $:\Longleftrightarrow \forall w \in \Sigma^*, |w| \leq k : w \in L(q_1) \Longleftrightarrow w \in L(q_2)$*

- *$\mathfrak{A}$ induziert die Abbildung $r^k : Q^2 \longrightarrow \{0, 1\}$ mit der Eigenschaft $r^k(q_1, q_2) = 1 :\Longleftrightarrow q_1 \overset{k}{\sim} q_2$. Diese können als Matrizen $R^k = (r^k(q_i, q_j))_{i,j=1}^n$ mit $n$ als Anzahl der Zustände dargestellt werden.*

*Es gilt: $r^k(q_i, q_j) = r^k(q_j, q_i)$*

Die Berechnung der k-Äquivalenzmatrizen erfolgt nach folgendem Schema:

- $q_1 \overset{0}{\sim} q_2 \Longleftrightarrow (q_1 \in F \Longleftrightarrow q_2 \in F)$

- $q1 \overset{k+1}{\sim} q_2 \Longleftrightarrow q1 \overset{0}{\sim} q_2 \wedge \delta(q_1, a) \overset{k}{\sim} \delta(q_2, a)\ \forall a \in \Sigma$

**Lemma 2.30** *Es gibt kein $K \in \mathbb{N}$, so daß für alle $n \in \mathbb{N}$ gilt: $R^k = R^{k+n}$*

**Beweis 2.31** *Da $r^k(q_i, q_j) = 0 \Longrightarrow r^{k+1}(q_i, q_j) = 0$, muß es ein $k$ geben mit $r^k = r^{k+1}$. Dann muß auch $r^k = r^{k+1}$ für alle $n \in \mathbb{N}$ gelten. Sei $r^k(q_1, q_2) = r^{k+1}(q_1, q_2) = 1$, also $q_1 \overset{k}{\sim} q_2$ und $q_1 \overset{k+1}{\sim} q_2$. Sei ferner $a_1 \ldots a_{k+2} \in L(q1)$. Dann ist $\delta(q_1, a_1) \overset{k}{\sim} \delta(q_2, a_1)$, also auch $\delta(q_1, a_1) \overset{k+1}{\sim} \delta(q_2, a_1)$, sonst $a_2 \ldots a_{k+2} \in L(\delta(q_2, a_1))$ und $a_1 \ldots a_{k+2} \in L(q_2)$. Wegen der Symmetrie folgt $q_1 \overset{k+2}{\sim} q_2$.*

□

### 2.6.1 Markierungsalgorithmus

- **Eingabe:** $\mathfrak{A} = \langle Q, \Sigma, \delta, q_0, F\rangle \in DFA(\Sigma)$, jedes $q \in Q$ ist erreichbar.
- **Ausgabe:** äquivalente Zustände
- **Verfahren:**
  - Tabelle aller Zustandspaare $\{q, q'\}$ mit $q \neq q'$
  - Markiere alle Paare $\{q, q'\}$ mit $q \in F$ und $q' \notin F$ und umgekehrt.
  - Für jedes unmarkierte Paar $\{q, q'\}$ und $a \in \Sigma$ teste, ob $\{\delta(q, a), \delta(q', a)\}$ markiert ist. Wenn ja, dann markiere auch $\{q, q'\}$.
  - Wiederhole den letzten Schritt, bis keine Änderung mehr erfolgt.
  - Unmarkierte Paare repräsentieren nun äquivalente Zustände.

Die Implementierung des Markierungsalgorithmus ist mit $O(|Q|^2)$-Zeitkomplexität möglich.

## 2.7 Entscheidbare Eigenschaften

### 2.7.1 Deterministische endliche Automaten

- **Wortproblem:**
  Gilt $w \in L(\mathfrak{A})$ ? Das Problem ist für $w \in \Sigma^*$ und $\mathfrak{A} \in DFA(\mathfrak{A})$ durch Eingabe in $|w| + 1$ Schritten entscheidbar.

- **Leerheitsproblem:**
  Gilt $L(\mathfrak{A}) = \emptyset$ ? Das Problem ist für $\mathfrak{A} \in DFA(\mathfrak{A})$ durch Eingabe aller Worte $w$ mit $|w| < |Q|$ entscheidbar. Dabei ist folgendes zu beachten:

  $$w \in L(\mathfrak{A}), |w| \geq |Q| \Longrightarrow \exists v \in L(\mathfrak{A}), |v| < |w|$$

  Das Verfahren prüft, ob $F$ vom $q_0$ erreichbar ist. Die Durchführung erfolgt in $O(|Q|^2)$-Zeit, bei geschickter Implementierung ggf. auch in besserer Zeit.

- **Äquivalenzproblem:**
  Gilt $L(\mathfrak{A}) = L(\mathfrak{A}')$ ? Die Lösung des Problems für $\mathfrak{A}, \mathfrak{A}' \in DFA(\Sigma)$ erfolgt durch Reduktion auf das Leerheitsproblem:

  $$\begin{aligned} L(\mathfrak{A}) = L(\mathfrak{A}') &\iff L(\mathfrak{A}) \subseteq L(\mathfrak{A}') \wedge L(\mathfrak{A}') \subseteq L(\mathfrak{A}) \\ &\iff L(\mathfrak{A}) \cap \overline{L(\mathfrak{A}')} = \emptyset \wedge L(\mathfrak{A}') \cap \overline{L(\mathfrak{A})} = \emptyset \end{aligned}$$

  Nun sind zwei Produktautomaten mit $|Q| \cdot |Q'|$ Zuständen auf Leerheit zu testen. Damit ist das Äquivalenzproblem in $O(|Q|^2 \cdot |Q'|^2)$ lösbar.

### 2.7.2 Reguläre Ausdrücke, nichtdeterministische endliche Automaten

Durch Transformation in einen äquivalenten DFA sind alle 3 Probleme entscheidbar. Allerdings steigt der Aufwand.

- **Wortproblem:**
  Lösbar in $O(|\alpha| \cdot |w|)$-Zeit bzw. mit $O(|\alpha| \cdot |w|)$-Platz.

- **Leerheitsproblem:**
  Hier ist der NFA wie ein DFA zu behandeln. $RA \longmapsto NFA$ ist in $O(|\alpha|)$-Zeit mit $|Q| \leq 2 \cdot |\alpha|$ lösbar.

- **Äquivalenzproblem:**
  Das Problem ist NP-hart ($Pr \in NP \longrightarrow P \leq_P \sim$-Probe).

## 2.8 Endliche Automaten mit Ausgabe

Die Idee der endlichen Automaten mit Ausgabe ist sequentielle Ausgabe bei Zustandsübergängen.

**Definition 2.32** *Sei $\mathfrak{A} = \langle Q, \Sigma, \delta, q_0, F\rangle \in DFA(\mathfrak{A})$ und $\Delta$ ein weiteres Alphabet (Ausgabealphabet), also eine nicht-leere, endliche Menge. Sei ferner $\lambda : Q \times \Sigma \longrightarrow \Delta$ eine Ausgabefunktion. Dann ist $\mathfrak{A} = \langle Q, \Sigma, \Delta, \delta, \lambda, q_0, F\rangle$ ein Mealy-Automat. Der Automat $\mathfrak{A}$ berechnet die sequentielle Transformation $f_{\mathfrak{A}} : \Sigma^* \longrightarrow \Delta^*$ mit $f_{\mathfrak{A}}(\varepsilon) := \varepsilon$ und $f_{\mathfrak{A}}(wa) := f_{\mathfrak{A}}(w)\lambda(\overline{\delta}(q_0, w), a)$.*

$\boxed{\textbf{2.14}}$

## 2.9 Automaten als abstrakte Programme

### 2.9.1 GOTO-Programme

GOTO-Programme sind aufgebaut über den Mengen $A = \{X_i := X_i + 1, X_i := X_i - 1 \mid i \in \mathbb{N}\}$ (Anweisungen) und $B = \{X_i = 0 \mid i \in \mathbb{N}\}$ (Bedingungen), nämlich als Zeichenreihen der Form $1 : \alpha_1; 2 : \alpha_2; \ldots k : \alpha_k$ mit $\alpha_i \in A \cup \{\underline{if}\ b\ \underline{then}\ j_0\ \underline{else}\ j_1 \mid b \in B, 1 \leq j_0 \leq j_1 \leq k\}$ und $\alpha_k = STOP$. Sei $\Sigma_{PATH} := A \cup (B \times \{true, false\})$. Für ein GOTO-Programm $P = 1 : \alpha_1; \ldots k : STOP$ definieren wir einen Automaten $\mathfrak{A}_{Pr} = \langle Q, \Sigma_{PATH}, \delta, q_0, F\rangle \in DFA(\Sigma)$ mit

- $Q = \{1, \ldots, k, 0\}$
- $q_0 = 1$
- $F = \{k\}$
- $\delta(i, a) = i + 1$ falls $i : a;$ in $P$ für $a \in A$
  $\left.\begin{array}{l}\delta(i, b_{true}) = j_0 \\ \delta(i, b_{false}) = j_1\end{array}\right\}$ falls $i : \underline{if}\ b\ \underline{then}\ j_0\ \underline{else}\ j_1$
  $\delta(i, c) = 0$ sonst

**Definition 2.33** *$L_{Pr} = L(\mathfrak{A}_{Pr})$ heißt formale Pfadsprache von $Pr$. Es gilt:*

$$P \sim_{PATH} Q \Longleftrightarrow L_P = L_Q$$

**Lemma 2.34** *Es gilt:*

$$\begin{array}{ccc} P \sim_{PATH} Q & \Longrightarrow & P \sim Q \\ \downarrow & & \downarrow \\ \textit{entscheidbar} & & \textit{nicht entscheidbar} \end{array}$$

*Der Umkehrschluß gilt jedoch nicht !*

$\boxed{\textbf{2.14}}$

### 2.9.2 Parallele Programme, Verteilte Systeme

Hierbei erfolgt die Reduktion der Parallelität auf Nichtdeterminismus. Für Aktionen $a$ und $b$ setzen wir $a \parallel b := ab \vee ba$. Zur Modellierung verteilter Systeme werden NFA's benutzt. Als Beispiel ist hier der Deadlock als Senkzustand zu nennen.

# Kapitel 3

# Kontextfreie Grammatiken und Kellerautomaten

Im folgenden Kapitel sollen nun die Grammatiken näher betrachtet werden.

$\boxed{\textbf{3.1}}$

## 3.1 Kontextfreie Grammatiken

**Definition 3.1** *Seien $N$ und $\Sigma$ nicht-leere, endliche Mengen mit $M \cap \Sigma = \emptyset$. Sie $P \subseteq N \times (N \cup \Sigma)^*$ mit $|P| < \infty$ und $S \in N$. Dann heißt $\mathcal{G} = \langle N, \Sigma, P, S \rangle$ eine kontextfreie Grammatik. Sie wird auch als $\mathcal{G} \in CFG(\Sigma)$ oder $\mathcal{G} \in CFG$ bezeichnet.*

### 3.1.1 Bezeichnungen und Konventionen

Für kontextfreie Grammatiken gelten im weiteren Verlauf die folgenden Konventionen:

- $A, B, C, ... \in N$ Nichterminalsymbole
- $a, b, c, ... \in \Sigma$ Terminalsymbole
- $S \in N$ Startsymbol
- $X, Y, Z, ... \in \mathcal{X} := N \cup \Sigma$ Symbole
- $\alpha, \beta, \gamma, ... \in \mathcal{X}^*$ Satzformen
- $u, v, w, ... \in \Sigma^*$ Terminalwörter
- $A \to \alpha := (A, \alpha) \in P$ Regeln, Produktion

**Definition 3.2** *Sei $\mathcal{G} = \langle N, \Sigma, P, S \rangle \in CFG$ und $\pi = A \to \alpha \in P$. $\pi$ bestimmt eine Ableitungsrelation $\Longrightarrow_\pi \subseteq \mathcal{X}^* \times \mathcal{X}^*$ mit $\beta_1 \Longrightarrow_\pi \beta_2 :\Longleftrightarrow$ es existiert ein Kontext $\gamma_1; \gamma_2 \in \mathcal{X}^*$, so daß $\beta_1 = \gamma_1 A \gamma_2$ und $\beta_2 = \gamma_1 \alpha \gamma_2$. Dann heißt das Tripel $(\gamma_1, \pi, \gamma_2)$ auch Ableitungsschritt. $\mathcal{G}$ bestimmt die Ableitungsrelation $\Longrightarrow_\pi \subseteq \mathcal{X}^* \times \mathcal{X}^*$ durch*

$$\Longrightarrow_\pi := \bigcup_{\pi \in P} \Longrightarrow_\pi$$

*und damit die von $\mathcal{G}$ erzeugte Sprache*

$$L(\mathcal{G}) := \{w \in \Sigma^* \mid S \overset{*}{\Longrightarrow}_\pi w\}$$

$\overset{*}{\Longrightarrow}_\mathcal{G} := \bigcup_{n=0}^{\infty} \overset{n}{\Longrightarrow}_\mathcal{G}$ *sei die transitive und reflexive Hülle.*

Es folgt:

$$w \in L(\mathcal{G}) \Longleftrightarrow \exists \alpha_0, \alpha_1, ..., \alpha_n \in \mathcal{X}^*$$

und für $\pi_1, ..., \pi_n \in P$ :

$$S = \alpha_0 \Longrightarrow_{\pi_1} \alpha_1 \Longrightarrow_{\pi_2} \alpha_2 ... \Longrightarrow_{\pi_n} \alpha_n = w (n \geq 1)$$

Bezeichnung:

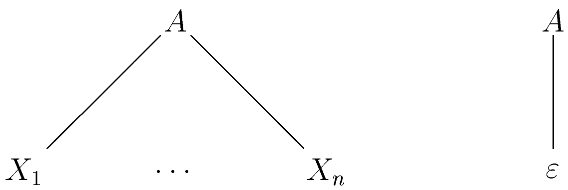
$$CFL(\Sigma) := \{L \subseteq \Sigma^* \mid \exists \mathcal{G} \in CFG(\Sigma) : L(\mathcal{G}) = L\}$$

**0.1**

### 3.1.2 Ableitungsbäume, Rechts- und Linksableitungen, Ein- und Mehrdeutigkeiten

Bei einer Grammatik $\mathcal{G} = \langle N, \Sigma, P, S \rangle \in CFG$ kann man die Ableitungen der Grammatik als Bäume darstellen.

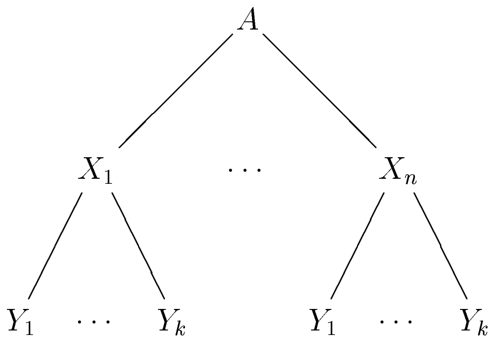
1. Eine Regel $\pi = A \longrightarrow X_1...X_n$ bestimmt den folgenden Regelbaum mit seinem Spezialfall für $n = 0$:



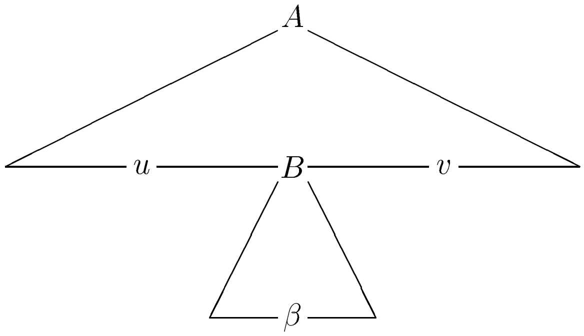

2. Eine Ableitung $A \Longrightarrow \alpha_1 \Longrightarrow \alpha_2 \Longrightarrow \ldots \Longrightarrow \alpha_n$ bestimmt den Ableitungsbaum durch entsprechendes "Verkleben" der Regelbäume.

**Definition 3.3** *Der Ableitungsbaum wird induktiv definiert über $n \in \mathbb{N}$:*

- $n = 1$*: Regelbaum*
- $n \to n + 1$*:* $A \Longrightarrow \alpha_1 \Longrightarrow \alpha_2 \ldots \Longrightarrow \alpha_n$ *habe den folgenden Ableitungsbaum:*



*Es sei* $\alpha_n \Longrightarrow \alpha_{n+1}$ *mit dem Ableitungsschritt* $(u, \pi, w)$ $\pi : B \longrightarrow \beta$*. Dann entsteht der Ableitungsbaum von* $A \Longrightarrow \alpha_1 \ldots \Longrightarrow \alpha_{n+1}$ *durch anhängen des Regelbaums von* $\pi$ *zwischen* $u$ *und* $v$*.*




**Folgerung 3.4** *Ein Ableitungsbaum repräsentiert eine Klasse von Ableitungen, die sich nur in der Reihenfolge von Ableitungsschritten unterscheiden. Der Ableitungsbaum repräsentiert die relevante syntaxtische Struktur.*

**Definition 3.5** *Eine Ableitung heißt Rechtsableitung (Linksableitung), wenn jeder Ableitungsschritt* $(\alpha, \pi, \beta)$ *ein Rechtsableitungsschritt (Linksableitingsschritt), d.h.* $\beta$ *besteht nur aus Terminalsymbolen (*$\alpha$ *besteht nur aus Terminalsymbolen), ist. Man schreibt* $\stackrel{r}{\Longrightarrow}$ *bzw.* $\stackrel{l}{\Longrightarrow}$*.*

**Folgerung 3.6** *Ein Ableitungsbaum hat genau eine Rechts- und genau eine Linksableitung.*

**Definition 3.7** $G \in CFG(\Sigma)$ *heißt eindeutig* $:\Longleftrightarrow$ *zu jedem* $w \in L(G)$ *gibt es genau eine Rechtsableitung* $S \stackrel{r}{\Longrightarrow} \alpha \stackrel{r}{\Longrightarrow} \ldots \stackrel{r}{\Longrightarrow} w$*. Ferner heißt* $G \in CFG(\Sigma)$ *mehrdeutig* $:\Longleftrightarrow$ $G$ *ist nicht eindeutig.*

Im folgenden ist ein Beispiel für Mehrdeutigkeit dargestellt.


## 3.2 Einseitig-lineare Grammatiken

**Definition 3.8** *Sei* $G = \langle N, \Sigma, P, S \rangle \in CFG(\Sigma)$*. Dann heißt diese Grammatik*

- *linkslinear, wenn für jedes* $\pi = A \longrightarrow x \in P$ *gilt:* $\alpha = Bw$ *oder* $\alpha = w$*, für* $B \in N$ *und* $w \in \Sigma^*$*.*
- *rechtslinear, wenn für jedes* $\pi = A \longrightarrow x \in P$ *gilt:* $\alpha = wB$ *oder* $\alpha = w$*, für* $B \in N$ *und* $w \in \Sigma^*$*.*
- *einseitig linear, wenn* $G$ *linkslinear oder* $G$ *rechtslinear ist.*

Das **Ziel** läßt sich wie folgt beschreiben:

$$\{L(G) \mid G \text{ linkslinear}\} = \{L(G) \mid G \text{ rechtslinear}\} = \{L(G) \mid G \text{ einseitig linear}\} = REG$$

**Satz 3.9** $\mathcal{L}(\Sigma, NFA) = \{L \subseteq \Sigma^* \mid L = L(G), G$ *rechtslinear*$\}$.

**Beweis 3.10** *Es sind zwei Richtungen zu zeigen:*

"$\supseteq$" *Sei* $G = \langle N, \Sigma, P, S\rangle$ *eine rechtslineare Grammatik. Fasse* $G$ *als Automat* $\mathfrak{A} = \langle Q, \Sigma, \delta, q_0, F\rangle \in NFA(\Sigma)$ *auf mit* $Q = N \cup \{q_f\}$, $F = \{q_f\}$, $q_0 = S$, $\delta(A, w) \ni B$ *falls* $A \to wB$ *in* $P$ *und* $\delta(A, w) \ni q_f$ *falls* $A \leftarrow w$ *in* $P$. *Ofenbar gilt dann* $L(G) = L(\mathfrak{A})$. *Wortübergänge werden durch Zwischenzustände beseitigt.*

"$\subseteq$" $\mathfrak{A} = \langle Q, \Sigma, \delta, q_0, F\rangle \in NFA(\Sigma)$ *kann ebenso als rechtslineare Grammatik aufgefaßt werden. Es gilt:*

$$q \xrightarrow{a} q' \qquad \Longleftrightarrow \qquad q \longrightarrow aq'$$

*Endzustände:*

$$q_f \quad \longmapsto \quad q_f \to \varepsilon$$

$\square$

**Korollar 3.11** $REG(\Sigma) \subseteq CFL(\Sigma)$. *Für* $|\Sigma| \geq 2$ *ist diese Inklusion echt, weil* $\{a^n b^n \mid n \in \mathbb{N}\} = L(G)$ *mit* $G = (S \to aSb \mid \varepsilon)$.

**Bemerkung 3.12** *Ist* $|\Sigma| = 1$, *dann folgt daraus* $REG(\Sigma) = CFL(\Sigma)$.

**Lemma 3.13** *Zu* $\mathfrak{A} \in DFA(\Sigma)$ *konstruieren wir* $\mathfrak{A}^R \in NFA(\Sigma)$ *mit* $L(\mathfrak{A}^R) = L(\mathfrak{A})^R$.

Die Idee dabei ist, die Richtungspfeile im Automaten umzudrehen (d.h. ersetze $q \xrightarrow{a} q'$ durch $q \xleftarrow{a} q'$), den Startzustand zum Endzustand zu machen und die Endzustände zu Startzuständen. Da nur ein Startzustand existieren darf, generiert man den Zustand $\widetilde{q_0}$ als neuen Startzustand und verweist von diesem aus auf die eigentlichen, neuen Startzustände mit $\varepsilon$-Transitionen. Der entstehende Automat hat genau einen End- und einen Startzustand und stellt exakt $L(\mathfrak{A}^R) = L(\mathfrak{A})^R$ dar.

**Korollar 3.14** *Es gilt:*

$$\{L \subseteq \Sigma^* \mid L = L(G), G \text{ rechtslinear}\} \quad = \quad \{L \subseteq \Sigma^* \mid L = L(G), G \text{ linkslinear}\}$$

**Beweis 3.15** *Es gilt folgende Äquivalenz:*

$$\begin{aligned} L \text{ rechtslinear erzeugbar} \quad &\Longleftrightarrow \quad L^R \text{ rechtslinear erzeugbar} \\ &\Longleftrightarrow \quad L = L^{RR} \text{ linkslinear erzeugbar} \end{aligned}$$

$\square$

## 3.3 Normalformen kontextfreier Grammatiken

Als Hilfsmittel dienen Vorgänger von Satzformen:

$$\begin{aligned} \alpha \Rightarrow \beta \quad &\Longrightarrow \quad \alpha \textbf{ direkter} \text{ Vorgänger von } \beta \\ \alpha \overset{*}{\Rightarrow} \beta \quad &\Longrightarrow \quad \alpha \text{ Vorgänger von } \beta \end{aligned}$$

**Definition 3.16** *Sei* $G = \langle N, \Sigma, P, S\rangle \in CFG(\Sigma)$ *und* $L \subseteq \mathcal{X}^*$. *Dann ist*

(i) *die direkte Vorgängermenge von $L$ definiert durch* $\underline{Pre}_G(L) := \{\alpha \in \mathcal{X}^* \mid \exists \beta \in L : \alpha \Rightarrow_G \beta\}$.

(ii) *der Vorgängerabschluß von $L$ definiert durch* $\underline{Pre}^*_G(L) := \{\alpha \in \mathcal{X}^* \mid \exists \beta \in L : \alpha \overset{*}{\Rightarrow}_G \beta\}$.

**Lemma 3.17** *Sei $G = \langle N, \Sigma, P, S\rangle \in CFG(\Sigma)$ und $L \subseteq \mathcal{X}^*$. Dann gilt:*

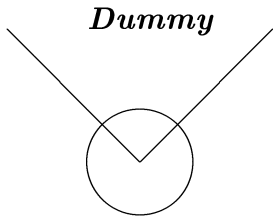
$$L \in REG(\mathcal{X}) \quad \Longrightarrow \quad \underline{Pre}^*_G(L) \in REG(\Sigma)$$

**Beweis 3.18** *Sei $\mathfrak{A} = \langle Q, \mathcal{X}, \delta, q_0, F\rangle \in NFA(\Sigma)$ mit $L(\mathfrak{A}) = L$. Konstruiere $\mathfrak{A}' = \langle Q, \mathcal{X}, \delta', q_0, F\rangle \in NFA(\Sigma)$ durch das Hinzufügen folgender Transitionen: wenn $A \to \alpha \in P$ und $q' \in \overline{\delta}(q, \alpha)$, so ist auch $q' \in \delta'(q, A)$. Ergänze $\delta$ nun um solche Transitionen, solange dies möglich ist. Dieser Erweiterungprozeß terminiert, weil $Q$ und $\mathcal{X}$ endlich sind; die Korrekthiet ist damit offensichtlich.*

□

**Beispiel 3.19** *Gegeben sei folgende Beispiel-Grammatik:*

$$G: \quad \begin{array}{lcl} S & \to & bSb \mid A \\ A & \to & aA \mid \varepsilon \end{array}$$




**Definition 3.20** *Sei $G \in CFG(\Sigma)$ und $A \in N$.*

1. *$A$ heißt **produktiv** $:\Longleftrightarrow \exists w \in \Sigma^* : A \overset{*}{\Rightarrow} w$*
2. *$A$ heißt **erreichbar** $:\Longleftrightarrow \exists \alpha, \beta \in \mathcal{X}^* : S \overset{*}{\Rightarrow} \alpha A \beta$*

**Folgerung 3.21** *Daraus können folgende Schlüße gezogen werden:*

1. *$A$ produktiv* $\quad \Longleftrightarrow \quad A \in \underline{Pre}^*_G(\Sigma^*)$
2. *$A$ erreichbar* $\quad \Longleftrightarrow \quad S \in \underline{Pre}^*_G(\mathcal{X}^*\{A\}\mathcal{X}^*)$

**Definition 3.22** *$G \in CFG(\Sigma)$ heißt **reduziert** genau dann, wenn entweder $P = \emptyset$ oder jedes $A \in N$ produktiv und erreichbar ist.*

**Lemma 3.23** *Jedes $G \in CFG(\Sigma)$ läßt sich in eine äquivalente reduzierte Grammatik $G' \in CFG(\Sigma)$ transformieren.*

**Beweis 3.24** *Man stellt für jedes $A \in N$ mit Hilfe von NFA's fest, ob $A$ erreichbar und produktiv ist. Es entstehen 2 Fälle:*

1. *$S$ nicht produktiv: wähle $P = \emptyset$*

2. *S produktiv: Weglassen aller $A \in N$, die nicht produktiv oder nicht erreichbar sind, sowie aller Regeln, in denen solche A's vorkommen.*

$\square$

**Korollar 3.25** *Das Leerheitsproblem von CFG's ist entscheidbar.*

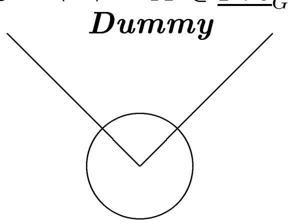
**Definition 3.26** *$G \in CFG(\Sigma)$ heißt $\varepsilon$-**frei** genau dann, wenn gilt:*

$$A \to \varepsilon \quad \Longrightarrow \quad A = S$$

*und außerdem S auf keiner rechten Regelseite steht.*

**Lemma 3.27** *Es gilt:*

$$X \overset{*}{\Rightarrow} \varepsilon \quad \Longleftrightarrow \quad X \in \underline{Pre}^*_G(\{\varepsilon\})$$




**Satz 3.28** *Jedes $G \in CFG(\Sigma)$ läßt sich in eine äquivalente $\varepsilon$-frei Grammatik $G' \in CFG(\Sigma)$ transformieren.*

**Beweis 3.29** *Konstruiere mit $\underline{Pre}^*(\{\varepsilon\})$ die Menge $N_\varepsilon = \{A \in N \mid A \overset{*}{\Rightarrow} \varepsilon\}$ und damit $G' = \langle N', \Sigma, P', S'\rangle \in CFG(\Sigma)$ mit:*

- $N' = N \mathbin{\dot\cup} \{S'\}$
- $P' = \{S' \to S\} \cup \{S' \to \varepsilon \mid S \in \underline{Pre}(\{\varepsilon\})\} \cup \{A \to X_1 \dots X_n \mid n \geq 1, X_i \in (N \cup \Sigma), \exists \alpha_0, \alpha_1, \dots, \alpha_n \in N_\varepsilon^* : A \to \alpha_0 X_1 \dots \alpha_n X_n \in P\}$

***Beachte dabei:*** *wegen $n \geq 1$ entfallen $\varepsilon$-Regeln und $G'$ ist $\varepsilon$-frei ! Damit bleibt nur noch zu zeigen, daß gilt: $L(G') = L(G)$.*

1. $w = \varepsilon$ :

$$\begin{aligned} \varepsilon \in L(G') &\Leftrightarrow S \in N_\varepsilon \\ &\Leftrightarrow (S \Rightarrow_G \varepsilon) \\ &\Leftrightarrow \varepsilon \in L(G) \end{aligned}$$

2. $w \neq \varepsilon$ :

   *(i)* $w \in L(G')$ :

$$\begin{aligned} w \in L(G') &\Longrightarrow (S \overset{*}{\Rightarrow} w) \\ &\Longrightarrow (S \overset{*}{\Rightarrow}_G w) \\ &\Longrightarrow w \in L(G) \end{aligned}$$

*weil Ableitung in $G'$ mit $\alpha_i \overset{*}{\Rightarrow}$ aufgefüllt werden kann zu einr Ableitung in G.*

*(ii)* $w \in L(G)$ :
*Wir zeigen durch Induktion über die Ableitungslänge $r$, daß gilt:*

$$A \overset{r}{\Rightarrow}_G w \in \Sigma^* \qquad \Longrightarrow \qquad A \overset{*}{\Rightarrow}_G w$$

$r = 1$ :

$$\begin{aligned} A \Rightarrow w, w \neq \varepsilon &\Longrightarrow (A \to w) \in P \\ &\Longrightarrow (A \to w) \in P' \\ &\Longrightarrow A \overset{*}{\Rightarrow}_G w \end{aligned}$$

$r \to r+1$ :

$$A \Rightarrow X_1 \ldots X_n \overset{r}{\Rightarrow}_G w$$

*Dann existieren Ableitungen $X_i \overset{r_i}{\Rightarrow}_G w_i$ mit $w = W_1 \ldots w_k$, $r_i \leq r$. Falls $w_i \neq \varepsilon$, so $X_i \overset{*}{\Rightarrow}_{G'} w_i$ nach Induktionsvoraussetzung. Falls $w_i = \varepsilon$, so $X_i \in N_\varepsilon$. Durch entsprechendes Löschen entsteht:*

$$A \Rightarrow_{G'} X'_1 \ldots X'_j \overset{*}{\Rightarrow} w'_1 \ldots w'_j = w$$

□

## 3.3.1 Elimination von Kettenregeln

**Definition 3.30** *Eine Regel der Form $A \to B$ heißt **Kettenregel**.*

**Satz 3.31** *Jedes $G = \langle N, \Sigma, P, S\rangle \in CFG(\Sigma)$ läßt sich in eine äquivalente, $\varepsilon$-freie Grammatik $G' = \langle N', \Sigma, P', S'\rangle \in CFG(\Sigma)$ **ohne Kettenregeln** umformen.*

**Beweis 3.32** *Ohne Bedingung der Allgemeinheit sei $G$ $\varepsilon$-frei. Wir definieren eine Grammtik $G' = \langle N', \Sigma, P', S'\rangle \in CFG(\Sigma)$ durch $A \to \alpha \in P' :\Leftrightarrow \alpha \notin N$ und es gibt ein $B \in N$ mit $A \in \underline{Pre}^*(B)$ und $B \to \alpha \in P$.*

□

## 3.3.2 Die Chomsky-Normalenform

**Definition 3.33** *$G = \langle N, \Sigma, P, S\rangle \in CFG(\Sigma)$ ist eine Chomsky-Normalenform ($G \in CNF(\Sigma)$), wenn gilt, daß jede Regel von $P$ die folgende Form hat:*

- *$A \to BC$ mit $B, C \in N$ oder*
- *$A \to a$ mit $a \in \Sigma$ oder*
- *$S \to \varepsilon$ und $S$ auf keiner rechten Regelseite.*

**Satz 3.34** *Jede Grammatik $G = \langle N, \Sigma, P, S\rangle \in CFG(\Sigma)$ läßt sich in eine äquivalente Grammatik $G' \in CNF(\Sigma)$ transformieren.*

**Beweis 3.35** *o.B.d.A. sei $G$ $\varepsilon$-frei und ohne Kettenregeln. Außerdem können wir annehmen, daß für $k \geq 2$ gilt:*

$$A \to X_1 \dots X_k \in P \quad \Longrightarrow \quad X_i \in N$$

*Denn $X_j = a$ kann ersetzt werden durch ein neues $C_j \in N$ durch Hinzufügen von $C_j \to a$. Bleibt die Elimination der Regeln der Form $A \to B_1 \dots B_k$ mit $k \geq 3$:*

$$\left.\begin{array}{c} A \to B_1 C_1 \\ C_1 \to B_2 C_2 \\ \vdots \\ C_{k-2} \to B_{k-1} B_k \end{array}\right\} \text{mit neuen } N\text{-Symbolen } C_1, \dots, C_{k-2}$$

□

### 3.3.3 Die Greibach-Normalenform, Linksrekursion

**Definition 3.36** *$G = \langle N, \Sigma, P, S \rangle \in CFG(\Sigma)$ ist in Greibach-Normalenform ($G \in GNF(\Sigma)$), wenn gilt, daß jede Regel von $P$ die folgende Form hat:*

- *$A \to aB_1 \dots B_n$ oder*
- *$A \to a$ oder*
- *$S \to \varepsilon$ und $S$ auf keiner rechten Regelseite.*

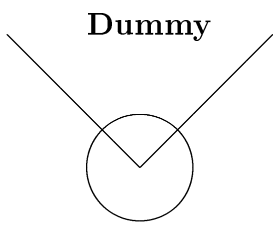
**Bemerkung 3.37** *Eine rechtslineare Grammatik $G$ ($A \to wB, A \to w$) läßt sich offensichtlich in GNF transformieren.*

**Satz 3.38** *Jedes $G \in CFG(\Sigma)$ läßt sich in ein äquivalentes $G' \in GNF$ transformieren.*

Statt des Beweises sei hier nur kurz die Beweisidee erwähnt: es geht um die Elimination linksrekursiver N-Symbole.

**Definition 3.39** *$A \in N$ heißt linksrekursiv, wenn gilt: $A \overset{*}{\Rightarrow} A\alpha$ für $\alpha \in \mathcal{X}^*$.*

Ein Spezialfall ist dabei die direkte Linksrekursion: Seien $A \to A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_r$ alle Regeln der Form $A \to A\alpha$ und $A \to \beta_1 \mid \dots \mid \beta_s$. Dann erfolgt die Transformation, wie im Bild dargestellt:




Die neuen Regelsätze lauten dann:

$$\begin{array}{rcl} A & \to & \beta_1 Z \mid \dots \beta_s Z \mid \beta_1 \dots \mid \beta_s \\ Z & \to & \alpha_1 Z \mid \dots \alpha_r Z \mid \alpha_1 \dots \mid \alpha_r \end{array}$$

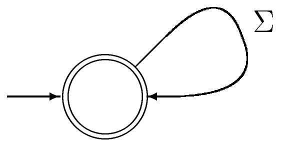
## 3.4 Abschlußeigenschaften von CFL, Pumping-Lemma

## 3.5 Entscheidbare Eigenschaften von CFG

**Satz 3.40** *Das Wortproblem und auch das Leerheitsproblem sind für CFG's entscheidbar.*

**Beweis 3.41** *Es gilt:*

$$\begin{aligned} w \in L(G) &\iff S \in \underline{Pre}^*(\{w\}) \\ L(G) \neq \emptyset &\iff S \notin \underline{Pre}^*(\Sigma^*) \end{aligned}$$




□

**Bemerkung 3.42** *Folgendes sei noch bemerkt:*

- *Das Wortproblem ist in $O(n^3)$ entscheidbar. (CYK-Algorithmus, demnächst)*
- *Das Äquivalenzproblem ist nicht entscheidbar.*

## 3.6 Kellerautomaten

### 3.6.1 Deterministische Kellerautomaten

## 3.7 Der Algorithmus von Cocke, Younger und Kasami

Der Algorithmus ist eine effiziente Lösung des Wortproblems für beliebige CFL. Er basiert auf dem Prinzip der dynamischen Programmierung und ist in $O(n^3)$ Zeit und $O(n^2)$ Platz zu realisieren. Sei nun o.B.d.A. $G \in CNF(\Sigma)$ (das Prinzip ist auf beliebige CFG's übertragbar).
Für $G = \langle N, \Sigma, P, S \rangle \in CNF(\Sigma)$ und $w = a_1 a_2 a_3 a_4 \dots a_n$ mit $n > 0$ definieren wir:

$$w_{ij} := a_i a_{i+1} \dots a_j \text{ für } 1 \leq i \leq j \leq n$$

und

$$N_{ij} := \left\{ A \in N \mid A \overset{*}{\Rightarrow} w_{ij} \right\}$$

so daß nun gilt:

$$w \in L(G) :\iff S \in N_{1n}$$

Die Bestimmung von $N_{1n}$ folgt nun folgendem Prinzip:

- Bestimme $N_{11}, N_{22}, \dots, N_{nn}$
- dann $N_{12}, N_{23}, \dots, N_{(n-1)n}$

- dann $N_{13}, N_{24}, \ldots, N_{(n-2)n}$
- ...
- dann $N_{1(n-1)}, \ldots, N_{2n}$
- bis schließlich $N_{1n)}$

Die jeweilige Bestimmung geschieht mit Hilfe der folgenden Regeln:

1. $A \in N_{ii} :\Leftrightarrow A \to a_i \in P$
2. $A \in N_{ij} :\Leftrightarrow \exists k : i \leq k \leq j \quad \exists A \to BC \in P$ mit $B \in N_{ik}$ und $C \in N_{(k+1)j}$ für alle $i < j$.

**3.9**

**Beispiel 3.43** *Sei $G$ gegeben mit $\{S \to AB \mid a, A \to BA \mid a, B \to AB \mid b\} \in P$. Dann ergibt sich die folgende Tabelle:*

| *i / j* | *1* | *2* | *3* | *4* |
|---|---|---|---|---|
| *1* | $\{B\}$ | $\{A\}$ | $\emptyset$ | $\{S,B\}$ |
| *2* | | $\{S,A\}$ | $\emptyset$ | $\{S,B\}$ |
| *3* | | | $\{S,A\}$ | $\{S,B\}$ |
| *4* | | | | $\{B\}$ |
| | $b$ | $a$ | $a$ | $b$ |

*Also gilt: $baab \in L(G)$, da $S \in N_{14}$ ist.*

Bei der Betrachtung der Komplexität sieht man sofort, daß die Platzkomplexität mit $O(n^2)$ durch die zwiedimensionale Matrix eindeutig gegeben ist. Nicht direkt einsichtig ist die Laufzeitkomplexität. Es gilt zunächst:

- äußeres "for i" hat $c_1 \cdot n$ Schritte
- "for k" hat $c_2$ Schritte
- inneres "for i" hat $c_2 \cdot (n-d) \cdot d$ Schritte
- "for d" hat $c_2 \cdot \sum_{d=1}^{n-1}(n-d) \cdot d$ Schritte

Also ergibt sich für die Laufzeit insgesamt:

$$\begin{aligned}
& c \cdot \sum_{d=1}^{n-1}(n-d) \cdot d \\
= \; & c \cdot \left(n \cdot \sum_{d=1}^{n} d - \sum_{d=1}^{n} d^2\right) \\
= \; & c \cdot \left(n^2 \cdot \frac{n+1}{2} - n \cdot \frac{(n+1)(n+2)}{6}\right) \\
= \; & c \cdot \frac{n^3 - n}{6} \\
= \; & O(n^3)
\end{aligned}$$

## 3.8 Erweiterte kontextfreie Grammatiken (EBNF)

Hier wird nun ein höherer Beschreibungskomfort durch reguläre Ausdrücke als rechte Regelseite erreicht. Es ist eine äuquivalente Erweiterung.

**Definition 3.44** *$G = \langle N, \Sigma, P, S\rangle$ ist eine* ***erweiterte*** *CFG , bezeichnet als $G \in ECFG$, wenn $\langle N, \Sigma, \emptyset, S\rangle \in CFG$ und $P : N \to RA(N \cup \Sigma)$ ist. Schreibweise: $A \to \alpha$, falls $P(A) = \alpha$.*

Die Semantik sei wie folgt gegeben: P repräsentiert die Regelmenge $\widetilde{P}$ mit $A \to \widetilde{\alpha} \in \widetilde{P}$ genau dann, wenn $\alpha \in [\![P(A)]\!]$ ist. Dabei ist zu beachten, daß $\widetilde{P}$ im allgemeinen unendlich ist. $L(G) = L(\langle N, \Sigma, \widetilde{P}, S\rangle)$

$\boxed{\textbf{3.11}}$

**Satz 3.45** *$CFL(\Sigma) = \mathcal{L}(\Sigma, ECFG)$*

Zu diesem Satz sei nur grob die Beweisidee skizziert:

"$\subseteq$" nach Definition

"$\supseteq$" Transformation von ECFG nach CFG durch Elimination regulärer Ausdrücke:

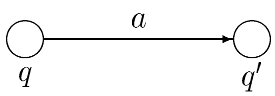
- Disjunktion: Regelalternativen
- Stern: neues N-Symbol mit $A \to \alpha A \mid \varepsilon$

## 3.9 Endliche rekursive Automaten

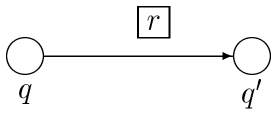
Syntaxdiagramme entsprechen endlichen Automaten, die sich rekursiv aufrufen können.

$\boxed{\textbf{3.12}}$

Neben dem folgendem Schema




ist dann auch das folgende Schema möglich:




**Definition 3.46** *$\mathfrak{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ heißt* ***rekursiver, endlicher Automat*** *über $\Sigma$, falls $\delta : Q \times (\Sigma_\varepsilon \cup Q) \to \wp(Q)$ und ansonsten wie ein endlicher Automat definiert ist. Er wird als mit $\mathfrak{A} \in RFA(\Sigma)$ bezeichnet.*

Die Semantik ist dann gegeben durch folgende Punkte:

- Konfigurationen: $Q \times \Sigma^*$

- Transitionsrelationen: $\vdash \subseteq (Q \times \Sigma^*)^2$ ist definiert durch

$$\frac{p \in \delta(q,a)}{(q,aw) \vdash (p,w)} \qquad \frac{p \in \delta(q,\varepsilon)}{(q,w) \vdash (q,w)}$$

(wie bei NFA's)

$$\frac{p \in \delta(q,r) \qquad (r,w) \overset{*}{\vdash} (s,v) \quad s \in F}{(q,w) \vdash (p,v)}$$

Es gilt:

$$L(\mathfrak{A}) := \{w \in \Sigma^* \mid (q_0,w) \overset{*}{\vdash} (p,\varepsilon), p \in F\}$$

**Satz 3.47** $\mathcal{L}(\Sigma, RFA) = \mathcal{L}(\Sigma, PDA)$

**Beweis 3.48** *Der Beweis ist in zwei Teile gegliedert:*

*"$\subseteq$" Sei $\mathfrak{A} = \langle Q, \Sigma, \delta, q_0, F\rangle \in RFA(\Sigma)$. Simulation von $\mathfrak{A}$ durch $\mathfrak{A}' = \langle Q, \Sigma, \Gamma, \delta', q_0, q_0, \emptyset\rangle \in PDA$. Die Idee ist, den Keller für "Rücksprungadressen" (Zustände) zu gebrauchen.*

$$\begin{aligned} \delta'(q,a,s) &= \{(p,s) \mid p \in \delta(q,a)\} \\ \delta'(q,\varepsilon,s) &= \{(p,s) \mid p \in \delta(q,\varepsilon)\} \cup \{(r,ps) \mid p \in \delta(q,r)\} \cup \{(s,\varepsilon) \mid q \in F\} \end{aligned}$$

*für alle $q,r,s,p \in Q$ und $a \in \Sigma$.*

*"$\supseteq$" Für $L \in \mathcal{L}(\Sigma, PDA)$ existiert $G \in CNF(\Sigma)$ mit $L = L(G)$. Konstruiere $\mathfrak{A} = \langle Q, \Sigma, \delta, q_0, F\rangle \in RFA(\Sigma)$ durch $Q = N \dot\cup \{q_s\}$, $q_0 = S$, $F = \{q_f\}$ und $C \in \delta(A,B)$ falls $A \to BC$, $q_f \in \delta(A,a)$ falls $A \to a$, $q_f \in \delta(S,\varepsilon)$ falls $S \to \varepsilon$.*

$\square$

# Kapitel 4

# Turingmaschinen

## 4.1 Chomsky-Grammatiken

An dieser Stelle sollen nun kontextfreie Grammatiken verallgemeinert werden, d.h. man betrachtet auch kontextabhängige Ersetzungen und Wortersetzungen.

**Definition 4.1** *Seinen $N, \Sigma, \mathcal{X}$ und $S$ wie bei CFG. Sei ferner $P \subseteq \mathcal{X}^* N \mathcal{X}^* \times \mathcal{X}^*$, $P$ sei endlich. Dann heißt $G = \langle N, \Sigma, P, S \rangle$ eine* ***Chomsky-Grammatik****.*

Die Semantik ist dabei wie folgt bestimmt: $\pi = \alpha_1 \to \alpha_2 \in P$ bestimmt die **Ableitungsrelation** $\Rightarrow_\pi$, für die gilt $\Rightarrow_\pi \subseteq \mathcal{X}^* \times \mathcal{X}^*$, durch $\beta_1 \Rightarrow_\pi \beta_2$ genau dann, wenn $\gamma, \delta \in \mathcal{X}^*$ existieren, mit $\beta_1 = \gamma\alpha_1\delta$ und $\beta_2 = \gamma\alpha_2\delta$. Dann ist die Ableitungsrelation von G definiert als:

$$\Rightarrow_G := \bigcup_{\pi \in P} \Rightarrow_\pi$$

G erzeugt $L(G) := \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow}_G w\}$.

$\boxed{\textbf{4.2}}$ $\boxed{\textbf{4.3}}$

Wir wollen nun die Abschlußeigenschaften von $\mathcal{L}_0$ und $\mathcal{L}_1$ untersuchen.

**Definition 4.2** *Sei $G = \langle N, \Sigma, P, S \rangle$ vom Typ0. G heißt normiert, wenn gilt:*

- *Für jedes $\pi = \alpha_1 \to \alpha_2 \in P$ gibt es $\gamma, \delta \in N^*, A \in N, \beta \in \mathcal{X}^*$ mit $\alpha_1 = \gamma A \delta$ und $\alpha_2 = \gamma\beta\delta$.*
- *$\Sigma$-Symbole nur in Regeln der Form $A \to a$*

Dabei ist zu beachten, daß $\beta = \varepsilon$, im Gegensatz zu Typ1, erlaubt ist.

**Satz 4.3** *ZU jedem $G = \langle N, \Sigma, P, S \rangle$ vom Typ0 läßt sich eine äquivalente Grammatik $G' = \langle N', \Sigma, P', S' \rangle$ konstruieren.*

**Beweis 4.4** *Der Beweis gliedert sich in 2 Teile:*

1. *Terminalsymbol-Bedingung:*
   *$a \in \Sigma \mapsto A$ sei neues N-Symbol. Ersetze ferner in P a durch $A_a$ und füge außerdem $A_a \to a$ hinzu.*

2. *Simulation von* $A_1 \dots A_n \to B_1 \dots B_m \quad (n \geq 1, m \in \mathbb{N})$ *durch folgende Regeln:*

$$\begin{array}{rcl} A_1 \dots A_n & \to & A'_1 A_2 \dots A_n \\ A'_1 \dots A_n & \to & A'_1 A'_2 A_3 \dots A_n \\ A'_1 \dots A'_{n-1} A_n & \to & A'_1 \dots A'_n \\ A'_1 \dots A'_n & \to & B_1 A'_2 \dots A'_n \\ & \vdots & \end{array}$$

*An dieser Stelle sind 2 Fälle zu unterscheiden:*

- *1. Fall:* $n \leq m$

$$B_1 \dots B_{n-1} A'_n \to B_1 \dots B_m$$

- *2. Fall:* $m < n$

$$\begin{array}{rcl} B_1 \dots B_m A'_{m+1} \dots A'_n & \to & B_1 \dots B_m A'_{m+2} \dots A'_n \\ B_1 \dots B_m A'_{m+2} \dots A'_n & \to & B_1 \dots B_m A'_{m+3} \dots A'_n \\ & \vdots & \\ B_1 \dots B_m A'_n & \to & B_1 \dots B_m \end{array}$$

$A'_i$ *sind neue Terminalsymbole, die Seiteneffekte verhindern (neue Satzformen, mehr Ableitungen).*

□

**Satz 4.5** $\mathcal{L}_0(\Sigma)$ *ist unter Substitution abgeschlossen, d.h. ist*

$$\varphi : \wp(\Sigma^*) \to \wp\left(\left(\bigcup_{a \in \Sigma} \Sigma_a\right)^*\right)$$

*eine Substitutionsabbildung, dann gilt für* $L \in \mathcal{L}_0(\Sigma)$ *und* $\varphi(a) \in \mathcal{L}_0(\Sigma)$*:*

$$\forall a \in \Sigma \Longrightarrow \varphi(L) \in \mathcal{L}_0\left(\bigcup_{a \in \Sigma} \Sigma_a\right)$$

**Beweis 4.6** *Eine Konstruktion von CFG ist nicht anwendbar wegen möglicher neuer Satzformen. Die Idee ist also die Sequentialisierung mit einem Kontrollsymbol.*
*Sei* $L = L(G)$ *mit* $G = \langle N, \Sigma, P, S \rangle$ *vom Typ0. Sei ferner* $\varphi(a) = L_a = L(G_a)$ *mit* $G_a = \langle N_a, \Sigma_a, P_a, S_a \rangle$ *vom Typ0. Es gilt o.B.d.A., daß* $G, G_a$ *normiert sind und daß* $N, N_a$ *disjunkt sind.*
*Konstruktion von* $\overline{G}$ *mit* $L(\bar{(G)}) = \varphi(L)$ *wie folgt:*

- *Nichtterminalsymbole:*

$$\overline{N} := N \,\dot{\cup} \left(\dot{\bigcup_{a \in \Sigma}} N_a\right) \dot{\cup} \{A_a \mid a \in \Sigma\} \,\dot{\cup}\, \{\overline{S}, C\}$$

- *Terminalsymbole:*

$$\overline{\Sigma} := \bigcup_{a \in \Sigma} \Sigma_a$$

- *Produktionen:*

$$\overline{P} := P_0 \cup P_1 \cup P_2 \cup \left( \bigcup_{a \in \Sigma} P_a \right) \cup \{\overline{S} \to CS\} \cup \{C \to \varepsilon\}$$

*$P_0$ entstehe aus $P$ durch Ersetzen von $a$ durch $A_a$ ($a \in \Sigma$). Ferner sei $P_1 := \{CA_a \to CS_a \mid a \in \Sigma\}$ und $P_2 := \{C_a \to aC \mid a \in \Sigma\}$.*

1. *$\varphi(L) \subseteq L(\overline{G})$:*

$$\begin{aligned} \overline{S} \Longrightarrow CS &\overset{*}{\Longrightarrow} CA_{a_1} \dots A_{a_r} \\ &\overset{*}{\Longrightarrow} CS_{a_1} A_{a_2} \dots A_{a_r} \\ &\overset{*}{\Longrightarrow} CB_{w_1} A_{a_2} \dots A_{a_r} \\ &\overset{*}{\Longrightarrow} w_1 C A_{a_2} \dots A_{a_r} \\ &\quad \vdots \\ &\overset{*}{\Longrightarrow} w_1 \dots w_r \end{aligned}$$

2. *$L(\overline{G}) \subseteq \varphi(L)$:*
*keine Ableitungen neuer Wörter, weil $G$ normiert und daher $A_a$ auf keiner linken Regelseite ist, weil die N-Alphabete disjunkt sind und weil $C$ die Teilableitungen der $G_a$ sequentialisiert.*

□

**Korollar 4.7** *$\mathcal{L}(\Sigma)$ ist unter den regulären Operationen abgeschlossen.*

$$L, L' \in \mathcal{L}_0(\Sigma) \Longrightarrow L \cup L', L \cdot L', L^* \in \mathcal{L}_0(\Sigma)$$

**Beweis 4.8** *Der Beweis läuft wie für kontextfreie Sprachen.*

□

**Satz 4.9** *$\mathcal{L}_0(\Sigma)$ ist unter Schnitt abgeschlossen.*

**Beweis 4.10** *Sei $L_i = L(G_i)$ und $G_i = \langle N_i, \Sigma, P_i, S_i \rangle$ vom Typ0 für $i = 1, 2$. Sei ferner o.B.d.A. $N_1 \cap N_2 = \emptyset$. Konstruiere dann $G = \langle N, \Sigma, P, S \rangle$ wie folgt:*

$$\begin{aligned} N &:= N_1 \cup N_2 \cup \{A_a \mid a \in \Sigma\} \,\dot{\cup}\, \{S, C_1, C_2\} \\ P &:= P_1 \cup P_2 \cup Q \\ Q &:= \{S \to C_1 S_1 C_2 S_2 C_1,\ C_2 a \to A_a C_2,\ b A_a \to A_a b,\ C_1 A_a \to a C_1,\ C_1 C_2 C_1 \to \varepsilon\} \end{aligned}$$

□

Für kontextsensitive Grammatiken gilt:

$$Typ1 \quad \sim \quad \text{Platzbedarf von Berechnungen}$$

**Definition 4.11** *$G = \langle N, \Sigma, P, S\rangle$ heißt* ***von wachsender Länge*** *genau dann, wenn für jede Regel $\alpha \to \beta \in P$ gilt: $|\alpha| \leq |\beta|$, es sei denn, daß $S \to \varepsilon \in P$ und $S$ auf keiner rechten Regelseite ist.*

**Korollar 4.12** *Eine Typ1-Grammatik ist von wachsender Länge.*

**Satz 4.13** *Zu jeder Grammatik von wachsender Länge läßt sich eine äquivalente Typ1-Grammatik konstruieren.*

**Definition 4.14** *Normierung: Sei $G = \langle N, \Sigma, P, S\rangle$ vom Typ0, $\delta = (A \Rightarrow \alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n)$ eine Abbildung von $G$ und $w \in \Sigma^*$. Dann heißt*

- $\underline{pl}_G(\delta) := max\{|\alpha_i| \mid 0 \leq i \leq n\}$ *der* ***Platzbedarf von*** *$\delta$ und*
- $\underline{pl}_G(w) := min\{\underline{pl}_G(\delta) \mid \delta = (S \Rightarrow \ldots \Rightarrow w)\}$ *der Platzbedarf von $w$.*

**Folgerung 4.15** *Daraus ergeben sich folgende 2 Forderungen:*

1. $\underline{pl}_G(w) \geq |w|$
2. *$G$ vom Typ1, $w \neq \varepsilon \Longrightarrow \underline{pl}_G(w) = |w|$*

**Satz 4.16** ***(Platzbedarfssatz)*** *Sei $G$ vom Typ0 und $p \in N, p > 0$, so daß für alle $w \in L(G)\backslash\{\varepsilon\}$ gilt: $\underline{pl}_G(w) \leq p|w|$. Dann ist $L(G) \in \mathcal{L}_1$.*

**Beweis 4.17** *Sei $p = 1$, also für $w \in L(G)\backslash\{\varepsilon\}$, $\underline{pl}_G(w) = |w|$. Elimination verkürzender Regeln durch Auffüllen. Konstruktion einer äquivalenten Grammatik von wachsender Länge: $G' = \langle N \cup \{\$\}, \Sigma, P', S\rangle$*

1. $\alpha \to \beta \in P$ *mit* $|\alpha| = |\beta| + i \quad (i > 0) \quad \Longrightarrow \quad \alpha \to \$^i\beta \in P'$
2. $\alpha \to \beta \in P$ *mit* $|\alpha| \leq |\beta| \quad \Longrightarrow \quad \$^j\alpha \to \beta \in P' \quad \forall j = 0, 1, \ldots, |\beta| - |\alpha|$
3. $\$X \to X\$, X\$ \to \$X \in P' \quad \forall x \in \mathcal{X}$

*Wegen $\underline{pl}_G(w) = |w|$ lassen sich die eingeschobenen \$ löschen. Für $p > 1$: Induktion, Idee: $N' \supseteq N^P$.*

□

### 4.1.1 Abschlußeigenschaften von $\mathcal{L}_1(\Sigma)$ mit Hilfe des Platzbedarfssatzes

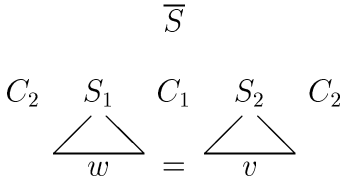
**Satz 4.18** *$\mathcal{L}_1(\Sigma)$ ist unter $\varepsilon$-freier Substitution abgeschlossen. ($\varepsilon$-freie Substitution bedeutet: $\varepsilon \in \varphi(a)$)*

**Beweis 4.19** *Platzbedarf der Ausgangsgrammatik mit $p = 1 \Longrightarrow$ Platzbedarf der Substitutionsgrammatik $|w|+1 \leq 2|w|$ wegen zusätzlichem Kontrollsymbol und fehlender $\varepsilon$-Regeln.*

□

**Korollar 4.20** *$\mathcal{L}_1(\Sigma)$ ist unter Durchschnitt abgeschlossen.*

**Beweis 4.21** *$L_1, L_2 \in \mathcal{L}_1(\Sigma)$ sind mit linearem Platzbedarf ($p = 1$) erzeugbar. Die $\cap$-Konstruktion für Typ0*




*hat einen Platzbedarf von $2|w| + 3 \leq 5|w|$.*

$\square$

**Korollar 4.22** $\mathcal{L}_2(\Sigma) \subsetneqq \mathcal{L}_1(\Sigma)$

**Beweis 4.23** *$\mathcal{L}_2(\Sigma)$ ist nicht unter $\cap$ abgeschlossen.*

$\square$

**Korollar 4.24** *$\mathcal{L}_1(\Sigma)$ ist unter regulären Operationen ($\cup, \cdot, *$) abgeschlossen.*

**Beweis 4.25** *Reduktion auf $\varepsilon$-freie Typ1-Sprachen.*

$$L \in \mathcal{L}_1(\Sigma) \quad \Longrightarrow \quad L \setminus \{\varepsilon\} \in \mathcal{L}_1(\Sigma)$$

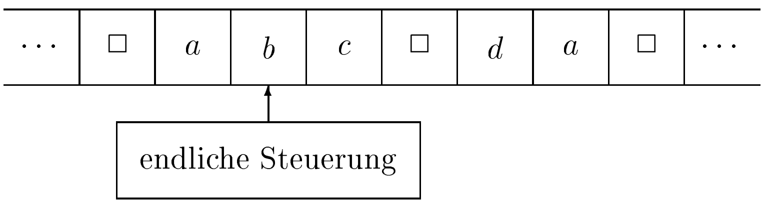
*Für $\varepsilon$-freie $L, L' \in \mathcal{L}_1(\Sigma)$ folgt: $L \cup L', L \cdot L', L^* \in \mathcal{L}_1(\Sigma)$ wie für Typ0-Sprachen durch $\varepsilon$-freie Substitution. Falls $\varepsilon \in L$ und/oder $\varepsilon \in L'$, so folgert man aus dem $\varepsilon$-freien Teilsprachen, z.B. $(L \dot{\cup} \{\varepsilon\})(L' \dot{\cup} \{\varepsilon\}) = L \cdot L'$: $L \cup L' \cup \{\varepsilon\} \in \mathcal{L}_1(\Sigma)$.*

$\square$

Ein lange offenes Problem war der Abschluß von $\mathcal{L}_1$ unter Komplement. Erst 1987 wurde bewiesen, daß $\mathcal{L}_1(\Sigma)$ unter Komplement abgeschlossen ist.

## 4.2 Turingmaschinen, linear-beschränkte Automaten

Ziel ist es, zu zeigen, daß $\mathcal{L}_0(\Sigma) = \mathcal{L}(\Sigma, TM)$ und $\mathcal{L}_1(\Sigma) = \mathcal{L}(\Sigma, LBA)$ gilt.




**Definition 4.26** *(nicht-deterministisch erkennende Turingmaschine) Seien folgende nicht-leere, endliche Mengen gegeben:*

*$Q$ als Zustandsmenge*

*$\Sigma$ als Eingabealphabet*

*$\Gamma$ als Arbeitsalphabet mit $\Sigma \subseteq \Gamma$*

*Sei ferner gegeben:*

- $q_0 \in Q$ *ein Anfangszustand*
- $\Box \in \Gamma \setminus \Sigma$ *das sogenannte "Blank"*
- $F \subseteq Q$ *eine Endzustandsmenge*

*und außerdem die Transistionsfunktion*

$$\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R, N\})$$

*Dann heißt* $\mathfrak{A} = \langle Q, \Sigma, \Gamma, q_0, \Box, F, \delta \rangle$ *eine (nicht-deterministisch)* ***erkennende Turingmaschine****, kurz* $\mathfrak{A} \in TM(\Sigma)$*.*

Die Semantik ist wie folgt beschrieben: $\underline{Conf}(\mathfrak{A}) := Q \times \Gamma^* \times \Gamma \times \Gamma^*$ ist die Konfigurationsmenge. Dann ist eine Einzelschrittrelation $\vdash_{\mathfrak{A}} \subseteq \underline{Conf}(\mathfrak{A})^2$ definiert durch:

- $(q, \alpha, X, \beta) \vdash (q', \alpha, X', \beta)$, falls $(q', X', N) \in \delta(q, X)$
- $(q, \alpha Y, X, \beta) \vdash (q', \alpha, Y, X'\beta)$, falls $(q', X', L) \in \delta(q, X)$
- $(q, \alpha, X, Y\beta) \vdash (q', \alpha X', Y, \beta)$, falls $(q', X', R) \in \delta(q, X)$
- $(q, \varepsilon, X, \beta) \vdash (q', \varepsilon, \Box, X'\beta)$, falls $(q', X', L) \in \delta(q, X)$
- $(q, \alpha, X, \varepsilon) \vdash (q', \alpha X', \Box, \varepsilon)$, falls $(q', X', R) \in \delta(q, X)$

**Bemerkung 4.27** *Es sei ausdrücklich betont, daß das es sich bei der Turingmaschine um ein beiderseits unendliches Band handelt, welches überwiegend mit* $\Box$ *beschrieben ist.*

Die Anfangskonfiguration von $w \in \Sigma^*$ ist beschrieben als

$$\mathcal{K}(w) := \begin{cases} (q_0, \varepsilon, a, v) & \text{falls } w = av \\ (q_0, \varepsilon, \Box, \varepsilon) & \text{falls } w = \varepsilon \end{cases}$$

Die Endkonfiguration ist dann $(q, \alpha, X', \beta)$ falls $q \in F$. Die von $\mathfrak{A}$ erkannte Sprache ist dann $L(\mathfrak{A}) := \{w \in \Sigma^* \mid \mathcal{K}(w) \vdash^* (q, \alpha, X, \beta), q \in F\}$. Dabei ist zu beachten, daß es genügt, ein $q \in F$ zu erreichen. Es ist kein Anhalten gefordert. Die Klasse der TM-erkennbaren Sprachen ist $\mathcal{L}(\Sigma, TM)$.

**Satz 4.28** $\mathcal{L}(\Sigma, TM) = \mathcal{L}_0(\Sigma)$

**Beweis 4.29** *Der Beweis ist erneut zweigeteilt.*

- $\mathcal{L}(\Sigma, TM) \subseteq \mathcal{L}_0(\Sigma)$
  *Sei* $\mathfrak{A} = \langle Q, \Sigma, \Gamma, q_0, \Box, F, \delta \rangle \in TM(\Sigma)$*. Dann kann* $\mathfrak{A}$ *durch eine Typ0-Grammatik simuliert werden. Die Idee ist, eine beliebige Ausgangskonfiguration mit hinreichend vielen Blanks an den Rändern zu erzeugen (bei Ableitungen sind Ränder nicht erkennbar). Dazu benutzt man 2 Spuren: die erste Spur hält die Eingabe für die Erzeugung, die zweite Spur simuliert die Turingmaschine. Für* $a_1 \dots a_r \in \Sigma^*$ *mit* $m, n \in \mathbb{N}$ *erzeugt man also:*

  $$(\varepsilon, \Box)^m q_0 (a_1, a_1)(a_2, a_2) \dots (a_r, a_r)(\varepsilon, \Box)^n$$

  *Dazu konstruiere die Grammatik G wie folgt:*

$G = \langle N, \Sigma, P, S \rangle$ *und* $a \in \Sigma$*mit*

$$\begin{array}{rcl} N & := & Q \dot{\cup} (\Sigma_\varepsilon \times \Gamma) \dot{\cup} \{S, C_1, C_2\} \\ P & := & \{S \to (\varepsilon, \Box)S \mid q_0 C_1, \quad C_1 \to (a,a)C1 \mid C_2, \quad C_2 \to (\varepsilon, \Box)C2 \mid \varepsilon\} \end{array}$$

*Es gilt:*

$$\begin{array}{rcl} q(a,X) & \to & q'(a,X'),\ \textit{falls}\ (q', X', N) \in \delta(q,X) \\ q(a,X) & \to & (a,X')q',\ \textit{falls}\ (q', X', R) \in \delta(q,X) \\ (b,Y)q(a,X) & \to & q'(b,Y')(a,X'),\ \textit{falls}\ (q', X', L) \in \delta(q,X) \\ q & \to & \varepsilon,\ \textit{falls}\ q \in F \\ (a,X) & \to & a \quad (a \in \Sigma_\varepsilon) \end{array}$$

*Dabei bewirken die beiden letzten Punkte das Löschen von q in der 2. Spur, womit dann das ursrüngliche Wort übrigbleibt. Es folgt:* $L(G) = L(\mathfrak{A})$.

- $\mathcal{L}(\Sigma, TM) \supseteq \mathcal{L}_0(\Sigma)$
  *Hier erfolgt die Simulation einer Typ0-Grammatik durch eine Turingmaschine. Das Vorgehen ist, neben dem Eingabewort Ableitungen von G zu simulieren und das erzeugte Wort mit der Eingabe zu vergleichen. Dabei sei bemerkt, daß es technisch sehr aufwendig ist, Wörter auf einem Turingband einzuschieben und zu löschen.*

$\Box$

### 4.2.1 Automatencharakterisierung von $\mathcal{L}_1(\Sigma)$ durch Platzbedarf

**Definition 4.30** *(Platzbedarf für Turingmaschinen) Sei* $\mathfrak{A} \in TM(\Sigma)$ *und* $\gamma = (\mathcal{K}_0 \vdash \mathcal{K}_1 \vdash \ldots \mathcal{K}_n)$ *eine Berechnung und* $w \in L(\mathfrak{A})$. *Dann ist*

- $|\mathcal{K}_i| = |\alpha \times \beta|$ *falls* $\mathcal{K}_i = (q, \alpha, X, \beta)$
- $\underline{bv}_{\mathfrak{A}}(\gamma) = max\{|\mathcal{K}_i| \mid 0 \le i \le n\}$
- $\underline{bv}_{\mathfrak{A}}(w) = min\{\underline{bv}_{\mathfrak{A}}(\gamma) \mid \gamma\ \textit{erkennt}\ w\}$

*der* ***Bandverbrauch von*** $\mathfrak{A}$ *bezüglich* $\gamma$ *bzw. w.*

**Definition 4.31** *Eine Turnigmaschine heißt* ***linear beschränkt*** *, wenn* $p \ge 1$ *existiert, so daß für alle* $w \in L(\mathfrak{A}) \setminus \{\varepsilon\}$ *gilt:*

$$\underline{bv}_{\mathfrak{A}}(w) \le p \cdot |w|$$

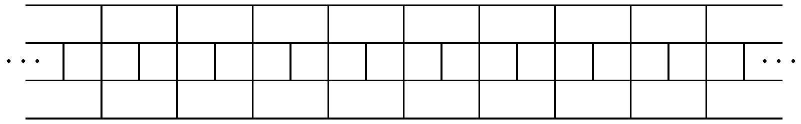
*Man schreibt kurz lbTM.*

**Korollar 4.32** *Es gilt* $\mathcal{L}_1(\Sigma) = \mathcal{L}(\Sigma, lbTM)$, *weil die Simulationen einen linearen Platzbedarf/Bandverbrauch erhalten.*

**Definition 4.33** $\mathfrak{A} \in LBA(\Sigma) :\Leftrightarrow \mathfrak{A} = lbTM$ *mit* $p = 1$, *d.h. nur Eingabefelder werden benutzt.*

**Satz 4.34** $\mathcal{L}_1(\Sigma) = \mathcal{L}(\Sigma, LBA)$

**Beweis 4.35** *Die Idee des Beweises ist die Kompression des benutzten Bandbereichs durch Vergrößerung des Arbeitsalphabets:* $\Gamma' := \Gamma^P$



$\square$

### 4.2.2 Deterministische Turingmaschinen, k-Band-Turingmaschinen

Die Reduktion nicht-deterministischer Turingmaschinen auf deterministische Turingmaschinen erfolgt in 2 Schritten:

1. Simulation einer nicht-deterministischen Turingmaschine durch 3-Band-Turingmaschine
2. Reduktion von k-Band-Turingmaschinen auf 1-Band-dTM

**Definition 4.36** *(deterministische k-Band-Turingmaschine)* $\mathfrak{A} = \langle Q, \Sigma, \Gamma, q, \Box, F, \delta \rangle \in$ *k-Band-dTM, wenn* $\delta : Q \times \Gamma^k - \rightarrow Q \times (\Gamma \times \{L, R, N\})^k$ *und sonst wie eine normale Turingmaschine.*

Dann ist $Q \times (\Gamma^* \times \Gamma \times \Gamma^*)^k$ eine Konfiguration. Die Einzelschrittrelationen werden durch simultane Arbeit auf k-Bändern gemäß $\delta$ beschrieben. Eine Anfangskonfiguration für $w \in \Sigma^*$ ist dann

$$\mathcal{K}_w := \begin{cases} (q_o, (\varepsilon, a, v), (\varepsilon, B, \varepsilon)^{k-1}) \text{ falls } w = av \\ (q_0, (\varepsilon, \Box, \varepsilon)^k) \text{ falls } w = \varepsilon \end{cases}$$

Außerdem ist

$$L(\mathfrak{A}) := \{w \in \Sigma^* \mid \mathcal{K}_w \overset{*}{\vdash} (q, \dots) \text{ mit } q \in F\}$$

die von $\mathfrak{A}$ erkannte Sprache.

**Satz 4.37** *Zu jeder* $\mathfrak{A} \in TM$ *läßt sich eine äquivalente* $\mathfrak{A}' \in 3 - dTM$ *konstruieren.*

**Beweis 4.38** *Für* $\delta_{\mathfrak{A}} : Q \times \Gamma \longrightarrow \wp_f(Q \times \Gamma \times \{L, R, N\})$ *gelte* $r := max\{|\delta(q, X)| \mid (q, X) \in (Q \times \Gamma)\}$*. Die Bezeichnunf der Alternativen von* $\mathfrak{A}$ *durch Elemente von* $\mathfrak{A}$ *sei gegeben durch Elemente von* $[r] := \{1, \dots, r\}$*. Die Berechnung von* $\mathfrak{A}$ *bestimmt* $\xi \in [r]^*$*. Umgekehrt erfolgt die Auswahl einer Berechnung durch ein* $\xi$ *als Steuerwert.*

- *Band 1: Eingabe, wiederholtes Lesen*
- *Band 2: Erzeugung der Zahlenfolgen* $\xi \in [r]^*$
- *Band 3: Simulation von* $\mathfrak{A}$ *gemäß Band 2*

*Auf diese Art und Weise erfoilgt die sukzessive Berechnung aller* $\xi \in [r]^*$*. Für jedes* $\xi$ *erfolgt eine Kopie der Eingabe auf Band 3. Man beachte dabei, daß nicht jedes* $\xi$ *einer Berechnung entspeicht.*

$\square$

**Satz 4.39** *Zu jeder* $\mathfrak{A} \in k - dTM$ *läßt sich eine äquivalente* $\mathfrak{A}' \in 1 - dTM$ *konstruieren.*

**Korollar 4.40** $\mathcal{L}(\Sigma, TM) = \mathcal{L}(\Sigma, dTM)$

**Bemerkung 4.41** *Für LBA's ist diese Frage noch offen (das LBA-Problem) !*

## 4.3 Aufzählbare und entscheidbare Sprachen

Intuitiv könnte man sagen:

- $L \subseteq \Sigma^*$ ist **aufzählbar**, wenn es ein effektives Verfahren gibt, welches genau die Elemente von $L$ erzeugt.
- $L \subseteq \Sigma^*$ ist **entscheidbar**, wenn es ein effektives Verfahren gibt, welches für jedes $w \in \Sigma^*$ feststellt, ob$w \in L$ oder ob $w \notin L$ gilt.

**Folgerung 4.42** *Für $L \subseteq \Sigma^*$ gilt:*

$$L \text{ entscheidbar} \quad \Longleftrightarrow \quad L \text{ und } \Sigma^* \setminus L \text{ aufzählbar}$$

Die Formulierung erfolgt über eine Turingmaschine mit Ausgabe.

**Satz 4.43** $\mathcal{L}_0(\Sigma) \subsetneqq \wp(\Sigma^*)$

**Beweis 4.44** *Die Menge der Typ0-Grammatiken über $\Sigma$ läßt sich abzählen (nach ihrer Größe). Also ist $\mathcal{L}_0(\Sigma)$ abzählbar. $\wp(\Sigma^*)$ ist jedoch überabzählbar.*

***Diagonalverfahren nach Cantor***

$$|\Sigma| = 1, \quad \Sigma^* = N, \quad L \subseteq \Sigma^* \mapsto \underline{char}_L : \mathbb{N} \to \{0,1\}$$

*Angenommen, $\wp(\mathbb{N})$ wäre abzählbar. Dann ist*

$$f : \mathbb{N}^2 \longrightarrow \{0,1\}$$

$$f(i,j) = 1 \quad \Longleftrightarrow \quad \underline{char}_{L_{diag}}(j) = 1$$

$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & \dots \\ \hline 0 & 0 & 1 & 1 & 0 & \\ 1 & 1 & 0 & 1 & 0 & \\ 2 & & & 0 & & \\ \vdots & & & & \ddots & \end{array}$$

*Definiere $L_{diag} \subseteq N$ durch Änderung der Diagonalen wie folgt:*

$$\underline{char}_{L_{diag}}(j) = 1 - f(i,j)$$

*Dann kann $L_{diag}$ nicht in der Abzählung vorkommen.*

□

**Satz 4.45** $DEC(\Sigma) := \{L \subseteq \Sigma^* \mid L \text{ entscheidbar}\} \subsetneqq \mathcal{L}_0(\Sigma)$

**Beweis 4.46** *Die Sprache UNIV ist nicht entscheidbar laut Diagonalisierungsschluß (Selbstanwendungsproblem)*[1]*.*

□

[1] vgl. Vorlesung "Berechenbarkeit und Komplexität"

**Satz 4.47** *Jede Typ1-Sprache ist entscheidbar.*

**Beweis 4.48** *Sei $G$ eine Typ1-Grammatik über $\Sigma$ und $w \in \Sigma^*$.*

- *Fall 1:*
$$w = \varepsilon \in L \iff S \to \varepsilon \in P$$

- *Fall 2:*
$$w \neq \varepsilon \in L \iff S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n = w$$
  *mit* $|\alpha_i| \leq |w|$.

*Es existieren nur endlich viele solcher Ableitungen.*

□

**Satz 4.49** *Es gibt entscheidbare, nicht-kontextsensitive Sprachen.*

**Beweis 4.50** *Hier sei auf die Literatur verwiesen.*

□

$\boxed{\textbf{4.4}}$

# Kapitel 5

# Untentscheidbare Probleme

Im letzten Kapitel sollen zum Abschluß noch unentscheidbare Probleme behandelt werden. Ziel ist es, zu zeigen, daß die folgenden Probleme unetnscheidbar sind:

- das Wortproblem für Typ0-Beschreibungen
- das Leerheitsproblem für Typ1-Beschreibungen
- das Äquivalenzproblem für Typ2-Beschreibungen

Dabei sei bemerkt, daß das Äquivalenzproblem für DPDA'S erst kürzlich als entscheidbar nachgewiesen wurde. Als Hilfsmittel für die folgenden Betrachtungen dient das **Postsche Korrespondenzproblem (PCP)**.

**Definition 5.1** *Seien $w = (w_1, w_2, \dots, w_n)$ und $v = (v_1, v_2, \dots, v_n)$ mit $w_i, v_i \in \Sigma^*$ und $1 \leq i \leq n$. Dann heißt $(i_1, \dots, i_k) \in \{1, \dots, n\}^k$ eine Lösung von PCP(w,v), falls $w_{i_1} \dots w_{i_k} = v_{i_1} \dots v_{i_k}$.*

Zur Verdeutlichung sei das folgende Beispiel:
Seien $(1, 10, 011)$ und $(101, 00, 11)$ Eingaben und $(1, 3, 2, 3)$ die Indizes. Dann muß die Zahlenfolge (1 011 10 011) identisch sein mit (101 11 00 11). Wie man unschwer erkennen kann, ist das in diesem Beispiel gegeben.

**Satz 5.2** *Das PCP ist unentscheidbar, d.h. es gibt kein Verfahren, welches für $n \in \mathbb{N}$ und $w, v \in (\Sigma^*)^n$ feststellt, obes eine Lösung gibt oder nicht für PCP(w,v).*

**Beweis 5.3** *Reduktion des Halteproblems für Turing-Maschinen auf das PCP.*

$\square$

**Satz 5.4** *Das Leerheitsproblem von Typ1-Grammatiken ist unentscheidbar.*

**Beweis 5.5** *Reduktion des PCP auf das Leerheitsproblem (Typ1). Seien $w = (w_1, \dots, w_n), v = (v_1, \dots, v_n) \in (\Sigma^*)^n$. Konstruktion von $\mathfrak{A}(w, v) \in lbTM$ mit PCP(w,v) lösbar. Das ist genau dann der Fall, wenn $L(\mathfrak{A}(w, v)) \neq \emptyset$. $\mathfrak{A}(w, v)$ erzeugt bei Eingabe von $u \in \Sigma^+$ alle Folgen $(i_1, \dots, i_k) \in \{1, \dots, n\}^k$ mit $k \leq |n|$ und prüft jeweils, ob $w_{i_1} \dots w_{i_k} = v_{i_1} \dots v_{i_k}$ ist. $\mathfrak{A}$ erkennt n, falls keine solche Indexfolge auftritt. Der Platzbedarf liegt linear in $|n|$. Wäre also das Leerheitsproblem entscheidbar, so wäre auch das PCP entscheidbar. Also liegt ein Widerspruch vor.*

□

**Satz 5.6** *Schnittproblem für kontextfreie Sprachen: es ist nicht entscheidbar, ob für $G_1, G_2 \in CFG$ gilt: $L(G_1) \cap L(G_2) = \emptyset$*

**Beweis 5.7** *Reduktion des PCP-Problems auf das Schnittproblem von CFL. Sei $w = (w_1 \dots w_n), v = (v_1 \dots v_n) \in (\Sigma^*)^n$. Konstruiere $G_w, G_v \in CFG$, so daß PCP(w,v) lösbar ist, genau dann wenn $L(G_w) \cap L(G_v) = \emptyset$. Es gilt:*

$$\tilde{\Sigma} = \Sigma \mathbin{\dot{\cup}} \{b_1, \dots, b_n\}$$

*Außerdem konstruiere $G_w := \left\langle \{S_w\}, \tilde{\Sigma}, P_w, S_w \right\rangle$ mit allen Regeln $P_w := \{S_w \to w_i S_w b_i\ ,\ S_w \to w_i b_i \mid 1 \leq i \leq n\}$. Die Konstruktion von $G_v$ läuft analog. Es folgt:*

$$L_w := L(G_w) = \{w_{i_1} \dots w_{i_k} b_{i_k} b_{i_{k-1}} \dots b_{i_1} \mid k \geq 1,\ 1 \leq i_j \leq n\}$$

*Für $L_v$ gilt die Folgerung analog. Außerdem folgt*

$$L_w \cap L_v \neq \emptyset \Longleftrightarrow \exists i_1 \dots i_k \in \{1, \dots, n\}$$

*also*

$$w_{i_1} \dots w_{i_k} b_{i_k} b_{i_{k-1}} \dots b_{i_1} = v_{i_1} \dots v_{i_k} b_{i_k} b_{i_{k-1}} \dots b_{i_1}$$

*Damit ist aber auch $w_{i_1} \dots w_{i_k} = v_{i_1} \dots v_{i_k}$, und somit das PCP(w,v) lösbar. Das ist ein Widerspruch, womit der Satz gezeigt ist.*

□

**Korollar 5.8** *Das Schnittproblem für DPDA'S ist nicht entscheidbar.*

**Beweis 5.9** *$G_w$ und $G_v$ sind durch deterministische Kellerautomaten simulierbar.*

□

**Satz 5.10** *Es ist nicht entscheidbar, ob fr $G_1, G_2 \in CFG$ gilt: $L(G_1) = L(G_2)$ (Äquivalenzproblem)*

**Beweis 5.11** *Reduktion des Schnittproblems für DPDA's auf das Äquivalenzproblem von CFG. Dabei ist zu beachten:*

1. *Zu $\mathfrak{A} \in DPDA(\Sigma)$ läßt sich $\overline{\mathfrak{A}} \in DPDA(\Sigma)$ konstruieren mit $L(\overline{\mathfrak{A}}) = \overline{L(\mathfrak{A})}$.*
2. *ZU $\mathfrak{A} \in PDA(\Sigma)$ läßt sich $G_{\mathfrak{A}} \in CFG(\Sigma)$ konstruieren mit $L(G_{\mathfrak{A}}) = L(\mathfrak{A})$.*
3. *Zu $G_1, G_2 \in CFG(\Sigma)$ läßt sich $G_v \in CFG(\Sigma)$ konstruieren mit $L(G_v) = L(G_1) \cup L(G_2)$.*

*Seien $\mathfrak{A}_1, \mathfrak{A}_2 \in DPDA$. Dann gilt:*

$$\begin{aligned} & L(\mathfrak{A}_1) \cup L(\mathfrak{A}_2) = \emptyset \\ \Longleftrightarrow\ & L(\mathfrak{A}_1) \subseteq L(\overline{\mathfrak{A}_2}) \\ \Longleftrightarrow\ & L(G_{\mathfrak{A}_1}) \subseteq L(G_{\overline{\mathfrak{A}_2}}) \\ \Longleftrightarrow\ & L(G_{\mathfrak{A}_1}) \cup L(G_{\overline{\mathfrak{A}_2}}) = L(G_{\overline{\mathfrak{A}_2}}) \\ \Longleftrightarrow\ & L(G_v) = L(G_{\overline{\mathfrak{A}_2}}) \end{aligned}$$

*Die Entscheidbarkeit des Äquivalenzproblems für CFG würde daher die Entscheidbarkeit des Schnittproblems von DPDA's nach sich ziehen.*

□

**- E N D E -**

# Literaturverzeichnis


[1] John E. Hopcroft, Jeffrey D. Ullman. *Einführung ind die Automatentheorie, formale Sprachen und Komplexitätstheorie.* Addison-Wesley (Deutschland), dritte Auflage, 1994.

[2] Thomas Schöning. *Theoretische Informatik - kurz gefaßt.* Spektrum - Akademischer Verlag, dritte Auflage, 1997.

# Index