# Theoretische Informatik HS2009 [1]

Stefan Heule

9. Dezember 2009

# Inhaltsverzeichnis

# 1 Grundlagen

## 1.1 Formale Sprachen, Wörter, Alphabete

**Definition 1.1.** *Eine endliche, nichtleere Menge $\Sigma$ heisst **Alphabet**. Die Elemente nennt man **Buchstaben**, **Symbole** oder **Zeichen**.*

**Definition 1.2.** *Sei $\Sigma$ ein Alphabet. Ein **Wort** $w$ über $\Sigma$ ist eine endliche, möglicherweise leere Folge von Buchstaben. Das **leere Wort** $\lambda$ (manchmal auch $\varepsilon$) ist die leere Buchstabenfolge.*
*Die **Länge** $|w|$ eines Wortes bezeichnet die Anzahl Buchstaben, aus denen sich $w$ zusammensetzt.*
*$\Sigma^*$ bezeichnet die Menge aller Wörter über $\Sigma$, und $\Sigma^+ = \Sigma \cdot \Sigma^* = \Sigma^* \setminus \{\lambda\}$.*

Nun kann man weitere nützliche Notationen einführen, etwa die Anzahl Vorkommen eines Buchstabens $a \in \Sigma$ in einem Wort $w \in \Sigma^*$: $|w|_a$.

Weiter ist ein **Teilwort** eine zusammenhängende Teilmenge der Zeichen eines Wortes. Liegt ein Teilwort am Anfang des Wortes, nennt man es **Präfix**, und am Ende heisst es **Suffix**.

Mithilfe der **kanonischen Ordnung** lassen sich die Wörter einer beliebigen Sprache aufzählen. In dieser Ordnung kommen kurze Wörter vor langen, während Wörter gleicher Länge gemäss einer beliebigen lexikographischen Sortierung geordnet werden.

**Definition 1.3.** *Die **Konkatenation (Verkettung)** ist eine Abbildung $\mathcal{K} : \Sigma^* \times \Sigma^* \to \Sigma^*$ mit folgender Definition:*

$$\mathcal{K}(x, y) = x \cdot y = xy$$

*Offensichtlich gilt $|xy| = |x| + |y|$. Weiter bildet $(\Sigma^*, \cdot, \lambda)$ eine algebraische Struktur, genauer ein Monoid.*
*Um etwas Schreibarbeit zu sparen, ist es nützlich, $x^i$ folgendermassen zu definieren:*

$$x^0 = \lambda \quad x^i = xx^{i-1}$$

**Definition 1.4.** *Eine **Sprache** $L$ über einem Alphabet $\Sigma$ ist eine beliebige Teilmenge von $\Sigma^*$. Deren Komplement $L^c$ bezüglich $\Sigma$ ist definiert als $\Sigma^* - L$.*
*$L_\emptyset = \emptyset$ ist die leere Sprache, während $L_\lambda = \{\lambda\}$ die Sprache ist, welche nur das leere Wort enthält.*
*Die **Konkatenation zweier Sprachen** $L_1$ und $L_2$ ist definiert als*

$$L_1 \cdot L_2 = \{xy \mid x \in L_1 \text{ und } y \in L_2\}$$

*Weiter definiert man als nützliche notationelle Konventionen*

$$L^0 = L_\lambda \quad \text{und} \quad L^i = LL^{i-1}$$

$$L^* = \bigcup_{i \in \mathbb{N}} L^i \quad \text{und} \quad L^+ = LL^*$$

*wobei man $L^*$ den **kleene'schen Stern** nennt.*

**Lemma 1.1.** *Seien $L_1, L_2$ und $L_3$ Sprachen über einem Alphabet $\Sigma$. Dann gilt*

$$L_1L_2 \cup L_1L_3 = L_1(L_2 \cup L_3)$$

*Weiter gilt für den Schnitt*

$$L_1(L_2 \cap L_3) \subseteq L_1L_2 \cap L_1L_3$$

Wichtig ist dabei zu bemerken, dass für den Schnitt wirklich nur diese eine Richtung gilt, nicht aber die umgekehrte Richtung.

**Definition 1.5.** *Seien $\Sigma_1$ und $\Sigma_2$ zwei beliebige Alphabete. Ein **Homomorphismus** von $\Sigma_1$ nach $\Sigma_2$ ist eine Funktion $h : \Sigma_1^* \to \Sigma_2^*$ mit folgenden Eigenschaften:*

*i)* $h(\lambda) = \lambda$

*ii)* $h(uv) = h(u) \cdot h(v)$ *für alle* $u, v \in \Sigma_1^*$.

Es genügt natürlich, für alle Buchstaben $a \in \Sigma_1$ $h(a)$ anzugeben.

## 1.2 Algorithmen

**Definition 1.6.** *Ein **Algorithmus** ist ein Programm, welches für jede zulässige Eingabe hält und eine Ausgabe liefert. Typischerweise realisiert ein Algorithmus also eine Funktion $A : \Sigma_1^* \to \Sigma_2^*$, wobei $\Sigma_1$ das Eingabe- und $\Sigma_2$ das Ausgabealphabet darstellen.*
*Zwei Algorithmen $A$ und $B$ heissen nun **äquivalent**, wenn sie auf demselben Eingabealphabet $\Sigma$ arbeiten und für alle $x \in \Sigma$ gilt: $A(x) = B(x)$.*

**Definition 1.7.** *Ein **Entscheidungsproblem** $(\Sigma, L)$ ist für eine gegebene Sprache $L \subseteq \Sigma^*$ über einem Alphabet $\Sigma$ für jedes $x \in \Sigma^*$ zu entscheiden, ob*

$$x \in L \quad \text{oder} \quad x \notin L$$

*Ein Algorithmus der dieses Problem löst, **erkennt** die Sprache $L$.*
*Ein Algorithmus ist ein **Aufzählungsalgorithmus** für eine Sprache $L$, wenn er als Eingabe eine natürliche Zahl $n$ nimmt, und damit $x_1, x_2, \ldots, x_n$ erzeugt, wobei dies die ersten $n$ Worte bezüglich der kanonischen Ordnung von $L$ sind.*

## 1.3 Kolmogorov Komplexität

**Definition 1.8.** *Für jedes Wort $x \in \Sigma^*$ ist die **Kolmogorov-Komlexität** $K(x)$ des Wortes $x$ die binäre Länge des kürzesten Pascal-Programmes, welches $x$ erzeugt.*
*Es existiert eine Konstante $d$, sodass für jedes Wort aus $(\Sigma_{bool})^*$ gilt:*

$$K(x) \leq |x| + d$$

*Die Kolmogorov-Komplexität einer natürlichen Zahl $n$ ist definiert als $K(Bin(n))$.*

Bei Zahlen benötigt man oftmals die binäre Kodierung $Bin(n)$ einer natürlichen Zahl $n$. Die Kosten für diese binäre Darstellung von $n$ sind $\lceil \log_2(n+1) \rceil$.

**Lemma 1.2.** *Für jede natürlich Zahl $n > 0$ existiert ein Wort $w_n$ aus $(\Sigma_{bool})^*$ mit*

$$K(w_n) \geq |w_n| = n$$

*d.h. für jede Zahl $n$ existiert ein nichtkomprimierbares Wort der Länge $n$.*

**Definition 1.9.** *Ein Wort $w$ heisst **zufällig**, wenn $K(w) \geq |w|$. Eine Zahl $n$ heisst zufällig, wenn $K(n) = K(Bin(n)) \geq \lceil \log_2(n+1) \rceil - 1$.*

Die "-1" in der Aussage für zufällige Zahlen lässt sich folgendermassen begründen: $\lceil \log_2(n+1) \rceil$ bezeichnet die genaue Länge von $Bin(n)$. Da das erste Bit jedoch immer eine 1 ist, muss diese nicht explizit gespeichert werden.

**Satz 1.1.** *Sei $L$ eine Sprache über $\Sigma_{bool}$. Sei, für jedes $n \in \mathbb{N} \setminus \{0\}$, $z_n$ das $n$-te Wort in $L$ bezüglich der kanonischen Ordnung. Wenn ein Programm $A_L$ existiert, welches das Entscheidungsproblem $(\Sigma_{bool}, L)$ löst, so gilt für alle $n \in \mathbb{N} \setminus \{0\}$*

$$K(z_n) \leq \lceil \log_2(n+1) \rceil + c$$

*wobei $c$ eine von $n$ unabhängige Konstante ist.*

**Satz 1.2.** ***Primzahlsatz.*** *Sei $Prim(n)$ die Anzahl Primzahlen kleiner gleich $n$.*

$$\lim_{n \to \infty} \frac{Prim(n)}{n / \ln n} = 1$$

**Lemma 1.3.** *Sei $n_1, n_2, n_3, \ldots$ eine steigende unendliche Folge von natürlichen Zahlen mit $K(n_i) \geq \lceil \log_2 n_i \rceil / 2$. Sei für $i = 1, 2, \ldots$ die Zahl $q_i$ die grösste Primzahl, die die Zahl $n_i$ teilt. Dann ist die Menge $Q = \{q_i | i \in \mathbb{N} \setminus \{0\}\}$ unendlich.*

*Beweis.* Der Beweis läuft indirekt. Angenommen, $Q$ sei endlich. Sei $p_m$ die grösste Primzahl in $Q$. Dann kann man jede Zahl $n_i$ eindeutig als

$$n_i = p_1^{r_{i,1}} \cdot p_2^{r_{i,2}} \cdot \cdots \cdot p_m^{r_{i,m}}$$

für irgendwelche $r_{i,1}, r_{i,2}, \ldots, r_{i,m} \in \mathbb{N}$ schreiben. Somit kann man ein einfaches Programm $A$ erstellen, welches für gegebene $r_{i,j}$ die binäre Darstellung von $n_i$ erzeugt. Sei $c$ die binäre Länge des Programmes $A$ ausser der Darstellung der Parameter (also die Darstellung des Teils von $A$, welcher für alle $i \in \mathbb{N}$ gleich ist). Dann gilt

$$K(n_i) \leq c + 2 \cdot (\lceil \log_2(r_{i,1}+1) \rceil + \lceil \log_2(r_{i,2}+1) \rceil + \cdots + \lceil \log_2(r_{i,m}+1) \rceil)$$

Die multiplikative Konstante 2 kommt, um die Kodierung der Parameter so zu gestalten, dass ihre Darstellung klar vom Rest des Programmes trennbar ist. Weil $r_{i,j} \leq \log_2 n_i$ für alle $j \in \{1, 2, \ldots, m\}$ erhalten wir

$$K(n_i) \leq c + 2 \cdot m \cdot \lceil \log_2(\log_2 n_i + 1) \rceil$$

für alle $i \in \mathbb{N} \setminus \{0\}$. Weil $m$ und $c$ Konstanten bezüglich $i$ sind, kann

$$\lceil \log_2 n_i \rceil / 2 \leq c + 2 \cdot m \cdot \lceil \log_2(\log_2 n_i + 1) \rceil$$

nur für endlich viele Zahlen $i$ gelten. Dies ist aber ein Widerspruch zu unserer Voraussetzung $K(n_i) \geq \lceil \log_2 n_i \rceil / 2$ für alle $i \in \mathbb{N}$. □

# 2 Endliche Automaten

## 2.1 Darstellung und Definition von endlichen Automaten

**Definition 2.1.** *Ein (deterministischer)* ***endlicher Automat (EA)*** *ist ein Quintupel* $M = (Q, \Sigma, \delta, q_0, F)$*, wobei*

*i)* $Q$ *eine endliche Menge der* ***Zustände*** *ist*

*ii)* $\Sigma$ *ein Alphabet, das sogenannte* ***Eingabealphabet*** *ist*

*iii)* $q_0 \in Q$ *der* ***Anfangszustand*** *ist*

*iv)* $F \subseteq Q$ *die* ***Menge der akzeptierenden Zustände*** *ist, und*

*v)* $\delta$ *eine Funktion von* $Q \times \Sigma$ *nach* $Q$ *ist, die* ***Übergangsfunktion***

*Eine* ***Konfiguration*** *von* $M$ *ist eine Element aus* $Q \times \Sigma^*$*. Ein Automat der sich in der Konfiguration* $(q, w)$ *befindet, ist zur Zeit im Zustand* $q$ *und muss noch den Suffix* $w$ *des Eingabewortes lesen.*

*Eine Konfiguration* $(q_0, x) \in \{q_0\} \times \Sigma^*$ *heisst* ***Startkonfiguration*** *von* $M$ *auf* $x$*. Jede Berechnung von* $M$ *muss offensichtlich in einer Startkonfiguration beginnen und endet in einer* ***Endkonfiguration****, wobei diese ein Element aus* $Q \times \{\lambda\}$ *ist.*

*Ein* ***Schritt*** *von* $M$ *ist eine Relation (auf Konfigurationen)* $\vdash_M \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$*, definiert durch*

$$(q, w) \vdash_M (p, x) \quad \iff \quad w = ax, a \in \Sigma \text{ und } \delta(q, a) = p$$

*Eine* ***Berechnung*** $C$ *von* $M$ *ist eine endliche Folge* $C = C_0, C_1, \ldots, C_n$ *von Konfigurationen, so dass* $C_i \vdash_M C_{i+1}$ *für alle* $0 \leq i \leq n-1$*.* $C$ *ist die Berechnugn von* $M$ *auf einer Eingabe* $x \in \Sigma^*$*, falls* $C_0 = (q_0, x)$ *und* $C_n \in Q \times \{\lambda\}$ *eine Endkonfiguration ist. Falls* $C_n \in F \times \{\lambda\}$*, sagen wir, dass* $C$ *eine* ***akzeptierende Berechnung*** *von* $M$ *auf* $x$ *ist, und dass* $M$ *das Wort* $x$ *akzeptiert. Andernfalls ist* $C$ *eine verwerfende Berechnung, und* $M$ *verwirft das Wort* $x$*.*

*Die von* $M$ ***akzeptierte Sprache*** $L(M)$ *ist definiert als*

$$L(M) := \{w \in \Sigma^* \mid M \text{ akzeptiert } w\}$$

$\mathcal{L}(EA) = \{L(M) \mid L \text{ ist ein EA}\}$ *ist die Klasse der Sprachen, die von endlichen Automaten erkannt wird. Diese wird auch als die Klasse der* ***regulären Sprachen*** *bezeichnet.*

**Definition 2.2.** *Sei* $M = (Q, \Sigma, \delta, q_0, F)$ *ein endlicher Automat. Wir definieren* $\vdash_M^*$ *als die reflexive und transitive Hülle der Schrittrelation* $\vdash_M$ *von* $M$*.*

*Weiter definieren wir* $\hat{\delta} : Q \times \Sigma^* \to Q$ *durch*

*i)* $\hat{\delta}(q, \lambda) = q$*, für alle* $q \in Q$ *und*

*ii)* $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$ *für alle* $a \in \Sigma, w \in \Sigma^*, q \in Q$*.*

*Mit dieser Relation lässt sich* $\Sigma^*$ *nun einfach in* $|Q|$ *Klassen unterteilen:*

$$\mathrm{Kl}[p] := \{w \in \Sigma^* \mid \hat{\delta}(q_0, w)\} = \{w \in \Sigma^* \mid (q_0, w) \vdash_M^* (p, \lambda)\}$$

## 2.2 Simulation und Beweise der Nichtexistenz

**Lemma 2.1.** *Sei* $\Sigma$ *ein Alphabet und seien* $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ *und* $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ *zwei EA. Für jede Mengenoperation* $\star \in \{\cup, \cap, \setminus\}$ *existiert ein EA* $M$ *mit*

$$L(M) = L(M_1) \star L(M_2)$$

**Lemma 2.2.** *Sei* $A = (Q, \Sigma, \delta, q_0, F)$ *ein EA. Seien* $x, y \in \Sigma^*$*,* $x \neq y$*, so dass*

$$(q_0, x) \vdash_A^* (p, \lambda) \quad \text{und} \quad (q_0, y) \vdash_A^* (p, \lambda)$$

*für ein* $p \in Q$*. Dann existiert für jedes* $z \in \Sigma^*$ *ein* $r \in Q$*, so dass* $xz$ *und* $yz \in \mathrm{Kl}[r]$*, also*

$$xz \in L(A) \quad \iff \quad yz \in L(A)$$

**Lemma 2.3.** ***Das Pumping-Lemma für reguläre Sprachen***
*Sei* $L$ *regulär. Dann existiert eine Konstante* $n_0 \in \mathbb{N}$*, so dass sich jedes Wort* $w \in \Sigma^*$ *mit* $|w| \geq n_0$ *in drei Teile* $y, x$ *und* $z$ *zerlegen lässt mit* $w = yxz$*, wobei*

*i)* $|yx| \leq n_0$*,*

*ii)* $|x| \geq 1$*, und*

*iii) entweder* $\{yx^k z \mid k \in \mathbb{N}\} \subseteq L$ *oder* $\{yx^k z \mid k \in \mathbb{N}\} \cap L = \emptyset$

**Satz 2.1.** *Sei* $L \subseteq (\Sigma_{bool})^*$ *eine reguläre Sprache. Sei* $L_x = \{y \in \Sigma^* \mid xy \in L\}$ *für jedes* $x \in \Sigma^*$*. Dann existiert eine Konstante* $c$*, so dass für alle* $x, y \in \Sigma^*$

$$K(y) \leq \lceil \log_2(n+1) \rceil + c$$

*falls* $y$ *das* $n$*-te Wort in der Sprache* $L_x$ *ist.*

## 2.3 Nichtdeterminismus

**Definition 2.3.** *Ein **nichtdeterministischer endlicher Automat (NEA)** ist ein Quintupel $M = (Q, \Sigma, \delta, q_0, F)$. Dabei ist der einzige Unterschide zum deterministischen Automaten die Übergangsfunktion $\delta : Q \times \Sigma \to \mathcal{P}(Q)$. Sie lässt nun mehrere Zustände zu, die von einem Ausgangszustand mit dem Lesen eines Symboles erreichbar sind.*

Um entscheiden zu können, ob ein NEA $M$ ein Wort $x$ akzeptiert, muss man alle Berechnungen von $M$ auf $x$ verfolgen. Die anschaulichste Darstellunge dieser Berechnungen ist durch einen sogenannten **Berechnungsbaum** $\mathcal{B}_M(x)$ von $M$ auf $x$. Die Knoten des Baumes sind Konfigurationen mit der Anfangskonfiguration als Wurzel. Die Söhne jedes Knotens sind Konfigurationen, die in einem Schritt erreicht werden können.

**Satz 2.2.** *Zu jedem NEA $M$ existiert ein EA $A$ mit $L(M) = L(A)$.*

*Beweis.* Sei $M = (Q, \Sigma, \delta_M, q_0, F)$ ein NEA. Wir konstruieren einen endlichen Automaten $A = (Q_A, \Sigma, \delta_A, q_{0A}, F_A)$ mit

i) $Q_A = \{\langle P\rangle \mid P \subseteq Q\}$

ii) $q_{0A} = \langle\{q_0\}\rangle$

iii) $F_A = \{\langle P\rangle \mid P \subseteq Q \text{ und } P \cap F \neq \emptyset\}$

iv) $\delta_A$ ist eine Funktion von $Q_A \times \Sigma$ nach $Q_A$, die wie folgt definiert ist. Für jedes $\langle P\rangle \in Q_A$ und jedes $a \in \Sigma$

$$\begin{aligned}\delta_A(\langle P\rangle, a) &= \left\langle \bigcup_{p\in P} \delta_M(p,a) \right\rangle \\ &= \langle\{q \in Q \mid \exists p \in P, \text{ so dass } q \in \delta_M(p,a)\}\rangle\end{aligned}$$

Um $L(M) = L(A)$ zu zeigen, genügt es, folgende Behauptung zu beweisen:

$$\forall x \in \Sigma^* : \hat{\delta}_M(q_0, x) = P \quad \Longleftrightarrow \quad \hat{\delta}_A(q_{0A}, x) = \langle P\rangle$$

Die ist jedoch einfach mittels Induktion über die Wortlänge $|x|$. Sei $w = xa$, dann gilt

$$\begin{aligned}\hat{\delta}_A(q_{0A}, xa) &= \delta_A(\hat{\delta}_A(q_{0A}, x), a) \\ &= \delta_A(\left\langle \hat{\delta}_M(q_0, x)\right\rangle, a) \\ &= \left\langle \bigcup_{p\in\hat{\delta}_M(q_0,x)} \delta_M(p,a) \right\rangle \qquad = \left\langle \hat{\delta}_M(q_0, xa)\right\rangle\end{aligned}$$

□

Um für ein Sprache $L$ zu beweisen, dass jeder EA $A$, der $L$ akzeptiert, mindestens $k$ Zustände haben muss, kann folgendermassen vorgegangen werden: Man sucht sich eine Menge $S$ von Wörtern, sodass für zwei Wörter $x, y \in S$ gelten muss:

$$x \neq y \quad \Longrightarrow \quad \hat{\delta}(q_0, x) \neq \hat{\delta}(q_0, y)$$

Dies kann man zeigen, indem man man für jedes Paar $x, y \in S$ ein $z \in \Sigma^*$ findet, sodass

$$xz \in L \quad \neq \quad yz \in L$$

Gelingt dies, so hat jeder EA $A$, der $L$ akzeptiert, mindestens $|S|$ Zustände.

# 3 Grammatiken

## 3.1 Das Konzept der Grammatiken

**Definition 3.1.** *Eine **Grammatik** $G$ ist ein 4-Tupel $G = (\Sigma_N, \Sigma_T, P, S)$, wobei*

*i)* $\Sigma_N$ *das Nichtterminalalphabet ist,*

*ii)* $\Sigma_T$ *das Terminalalphabet ist,*

*iii)* $S \in \Sigma_N$ *das Startnichtterminal oder Startsymbol darstellt, und*

*iv)* $P \subseteq \Sigma^*\Sigma_N\Sigma^* \times \Sigma^*$ *mit* $\Sigma = \Sigma_N \cup \Sigma_T$ *eine endliche Menge von Ableitungsreglen ist. Die Elemente von $P$ heissen Regeln oder auch Produktionen. Statt $(\alpha, \beta) \in P$ schreiben wir auch $\alpha \to_G \beta$.*

*Seien $\gamma, \delta \in \Sigma^*$. Wir sagen, dass $\delta$ aus $\gamma$ in einem Ableitungsschritt in $G$ ableitbar ist (oder dass $\gamma$ in $\delta$ übergeht),*

$$\gamma \Rightarrow_G \delta$$

*genau dann, wenn $w_1$ und $w_2$ aus $\Sigma^*$ und eine Regel $(\alpha, \beta) \in P$ existieren, so dass*

$$\gamma = w_1, \alpha, w_2 \quad \textit{und} \quad \delta = w_1\beta, w_2$$

*Eine Folgen von Ableitungsschritten*

$$w_0 \Rightarrow_G w_1 \Rightarrow_G \cdots \Rightarrow_G w_n \quad \textit{oder} \quad w_0 \Rightarrow_G^* w_n$$

*heisst Ableitung in $G$.*

*Falls $S \Rightarrow_G^* w$ für ein Wort $w \in \Sigma_T^*$, dann sagen wir, dass $w$ von $G$ erzeugt wird. Die von $G$ erzeugte Sprache ist*

$$L(G) = \{w \in \Sigma_T^* \mid S \Rightarrow_G^* w\}$$

**Definition 3.2.** *Sei $G = (\Sigma_N, \Sigma_T, P, S)$ eine Grammatik.*

*i) $G$ heisst **Typ-0-Grammatik**. Typ-0-Grammatiken stellen die Klasse der allgemeinen uneingeschränkten Grammatiken dar.*

*ii) $G$ heisst **kontextsensitive** oder **Typ-1-Grammatik**, falls für alle $(\alpha, \beta) \in P$ gilt, dass*

$$|\alpha| \leq |\beta|$$

*Man kann also nie ein Teilwort $\alpha$ durch ein kürzeres $\beta$ ersetzen.*

*iii) $G$ heisst **kontextfrei** oder **Typ-2-Grammatik** falls für alle Regeln $(\alpha, \beta) \in P$ gilt dass*

$$\alpha \in \Sigma_N \textit{ und } \beta \in (\Sigma_T \cup \Sigma_N)^*$$

*iv) $G$ heisst **regulär** oder **Typ-3-Grammatik**, falls für alle Regeln $(\alpha, \beta) \in P$ gilt, dass*

$$\alpha \in \Sigma_N \textit{ und } \beta \in \Sigma_T^*\Sigma_N \cup \Sigma_T^*$$

## 3.2 Reguläre Grammatiken und endliche Automaten

**Lemma 3.1.** *$\mathcal{L}_3$ enthält alle endlichen Sprachen.*

**Lemma 3.2.** *$\mathcal{L}_3$ ist abgeschlossen bezüglich*

*i) Vereinigung*

*ii) Konkatenation*

**Satz 3.1.** *Zu jedem endlichen Automaten $A$ existiert eine reguläre Grammatik $G$ mit $L(A) = L(G)$.*

**Definition 3.3.** *Eine reguläre Grammatik $G = (\Sigma_N, \Sigma_T, P, S)$ heisst **normiert**, wenn alle Regeln nur eine der folgenden drei Formen haben*

*i) $S \to \lambda$, wobei $S$ das Startsymbol ist,*

*ii) $A \to a$ für $A \in \Sigma_N$ und $a \in \Sigma_T$, oder*

*iii) $B \to bC$ für $B, C \in \Sigma_N$ und $b \in \Sigma_T$.*

**Satz 3.2.** *Zu jeder regulären Grammatik $G$ exisitert eine äquivalente reguläre Grammatik $G'$, welche normiert ist.*

Um eine Grammatik zu normieren, kann folgendermassen vorgegangen werden:

i) Entferne alle Kettenregeln, also Regeln der Form $A \to B$ für $A, B \in \Sigma_N$.

ii) Entferne alle Lambdaregeln, also Regeln der Form $A \to \lambda$ für $A \in \Sigma_N$, ausser wenn $A$ das Startsymbol ist.

iii) Füge Hilfsnichtterminale ein, um Regeln der Form $A \to w$ für $A \in \Sigma_N$ und $w \in \Sigma_T^*$ mit $|w| > 1$ in mehrere Regeln zu zerlegen, sodass $G$ normiert ist.

**Satz 3.3.** $\mathcal{L}_3 = L(EA)$

*Beweisskizze.* Gemäss einem Satz weiter oben gibt es zu jedem EA eine äquivalente reguläre Grammatik, es bleibt also nur noch die Gegenrichtung zu zeigen.
Sei $G = (\Sigma_N, \Sigma_T, P, S)$ die normierte reguläre Grammatik. Wir zeigen, dass ein äquivalenter NEA $M = (\Sigma_N \cup \{q_F\}, \Sigma_T, \delta, S, F)$ existiert. Dazu sei

$$F = \begin{cases} \{q_F\} & \text{falls } S \to \lambda \notin P \\ \{q_F, S\} & \text{falls } S \to \lambda \in P \end{cases}$$

$$\begin{aligned} \delta(q_F, a) &= \emptyset && \text{für alle } a \in \Sigma_T \\ Y &\in \delta(X, a) && \text{falls } X \to aY \in P,\ X, Y \in \Sigma_N,\ a \in \Sigma_T \\ q_F &\in \delta(X, a) && \text{falls } X \to a \in P,\ X \in \Sigma_N,\ a \in \Sigma_T \end{aligned}$$

□

## 3.3 Kontextfreie Grammatiken und Kellerautomaten

**Definition 3.4** (zur Erinnerung)**.** *Eine Grammatik $G = (\Sigma_N, \Sigma_T, P, S)$ heisst **kontextfrei**, wenn für alle Regeln $(\alpha, \beta) \in P$*

$$\alpha \in \Sigma_N \quad \textit{und} \quad \beta \in (\Sigma_N \cup \Sigma_T)^*$$

*gilt.*

**Definition 3.5.** *Sei $G = (\Sigma_N, \Sigma_T, P, S)$ eine kontextfreie Grammatik und $x$ ein Wort aus $L(G)$. Sei $\mathfrak{a}$ eine Ableitung von $x$ in $G$. Der **Syntaxbaum (Ableitungsbaum)** $T_\mathfrak{a}$ der Ableitung $\mathfrak{a}$ ist ein markierter Baum mit folgenden Eigenschaften:*

*i) $T_\mathfrak{a}$ hat eine Wurzel, die mit $S$ markiert ist.*

*ii) Jeder Knoten von $T_\mathfrak{a}$ ist mit einem Symbol aus $\Sigma_N \cup \Sigma_T \cup \{\lambda\}$ markiert.*

*iii) Innere Knoten sind mit Symbolen aus $\Sigma_N$ markiert.*

*iv) Alle Blätter sind mit Symbolen aus $\Sigma_T \cup \{\lambda\}$ makiert, und von links nach rechts gelesen ergeben sie genau das Wort $x$.*

*v) Für jede Anwendung einer Regel $A \to \alpha_1\alpha_2\cdots\alpha_k$ in $\mathfrak{a}$ existiert genau ein innerer Knoten $v$ mit der Markierung $A$ und $k$ Söhnen $v_1, v_2, \ldots, v_k$ von links nach rechts, die mit den Symbolen $\alpha_1, \alpha_2, \ldots, \alpha_k$ markiert sind ($\alpha_i \in \Sigma_N \cup \Sigma_T$ für $i = 1, 2, \ldots, k$).*

Es bleibt zu bemerken, dass jede Ableitung eindeutig einen Syntaxbaum bestimmt. Umgekehrt kann es allerdings mehrere Ableitungen geben, die zu demselben Syntaxbaum gehören. Daher verwendet man oftmals sogenannte **Linksableitungen**, wobei stets das am weitesten links stehende Nichtterminal ersetzt wird.

Wie auch bei regulären Grammatiken kann man kontextfreie Grammatiken so modifizieren, dass keine Regeln der Form $A \to \lambda$ (ausser $S \to \lambda$) und keine Kettenregeln mehr vorkommen.

Zudem lassen sich auch gewisse "unnötigen" Nichtterminale charakterisieren:

i) Ein Nichtterminal $X \in \Sigma_N$ heisst **nutzlos**, wenn es in keiner aus $S$ startenden Ableitung erreicht werden kann.

ii) Ein Nichtterminal $X \in \Sigma_N$ heisst **unproduktiv**, wenn aus $X$ kein Terminalwort abgeleitet werden kann.

**Definition 3.6.** *Sei $G = (\Sigma_N, \Sigma_T, P, S)$ eine kontextfreie Grammatik. Wir sagen, dass $G$ in **Chomsky-Normalform** ist, falls alle Regeln von der Form*

$$\begin{aligned} A \to BC &\quad \textit{für } A, B, C \in \Sigma_N \textit{ oder} \\ A \to a &\quad \textit{für } A \in \Sigma_N \textit{ und } a \in \Sigma_T \end{aligned}$$

*sind. Wir sagen, dass $G$ in **Greibach-Normalform** ist, wenn alle Produktionen von der Form*

$$A \to a\alpha \quad \textit{für } A \in \Sigma_N, a \in \Sigma_T \textit{ und } \alpha \in \Sigma_N^*$$

*sind.*

**Satz 3.4.** *Für jede kontextfreie Grammatik $G$ mit $\lambda \notin L(G)$ existieren zu $G$ äquivalente Grammtiken in Chomsky-Normalform und in Greibach-Normalform.*

**Lemma 3.3.** ***Das Pumping-Lemma für kontextfreie Sprachen***
*Für jede kontextfreie Sprache $L$ gibt es eine nur von $L$ abhängige Konstante $n_L$, so dass für alle Wörter $z \in L$ mit $|z| \geq n_L$ eine Zerlegung*

$$z = uvwxy$$

*existiert, so dass*

*i) $|vx| \geq 1$*

*ii) $|vwx| \leq n_L$, und*

*iii) $\{uv^iwx^iy \mid i \in \mathbb{N}\} \subseteq L$.*

Wie schon das Pumping-Lemma für reguläre Sprachen (siehe Lemma 2.3) bildet das Lemma keine vollständige Charakterisierung der kontextfreien Sprachen. Das heisst, es gibt nicht-kontextfreie Sprachen, welche das Lemma erfüllen. Das Lemma kann nur verwendet werden, um für eine gegebene Sprache $L$ zu zeigen, dass $L \notin \mathcal{L}_2$, also dass $L$ keine kontextfreie Sprache sein kann.

**Definition 3.7.** *Ein **nichtdeterministischer Kellerautomat (NPdA)** ist ein 6-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$, wobei die Elemente des Tupels folgende Bedeutung haben:*

*i)* $Q$ *ist eine endliche **Zustandsmenge***

*ii)* $\Sigma$ *ist das endliche **Eingabealphabet***

*iii)* $\Gamma$ *ist das endliche **Kelleralphabet***

*iv)* $q_0 \in Q$ *ist der **Anfangszustand***

*v)* $Z_0 \in \Gamma$ *ist das **Initialisierungssymbol** des Kellers*

*vi)* $\delta$ *ist eine Abbildung von* $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ *in eine endliche Teilmenge von* $Q \times \Gamma^*$.

*Eine **Konfiguration** von $M$ ist ein Tupel*

$$(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*,$$

*wobei $q \in Q$ der aktuelle Zustand, $w \in \Sigma^*$ der bisher nicht gelesene Teil der Eingabe und $\alpha \in \Gamma^*$ der aktuelle Kellerinhalt ist.*

*Ein **Schritt** $\vdash_M$ von $M$ ist eine Relation auf Konfigurationen, definiert durch*

*i)* $(q, aw, \alpha X) \vdash_M (p, w, \alpha\beta)$
*wenn $(p, \beta) \in \delta(q, a, X)$ für $p, q \in Q, a \in \Sigma, X \in \Gamma$ und $\alpha, \beta \in \Gamma^*$.*
*$M$ liest im Zustand $q$ das Eingabesymbol $a$ und das Kellersymbol $X$. Danach geht $M$ in den Zustand $p$, bewegt den Lesekopf ein Feld nach rechts und ersetzt das Top-Symbol $X$ des Kellers durch $\beta$.*

*ii)* $(q, w, \alpha X) \vdash_M (p, w, \alpha\beta)$
*wenn $(p, \beta) \in \delta(q, \lambda, X)$ für $p, q \in Q, X \in \Gamma$ und $\alpha, \beta \in \Gamma^*$.*
*$M$ liest im Zustand $q$ kein Eingabesymbol, sondern nur das Kellersymbol $X$. Deshalb ändert $M$ nur den Zustand und ersetzt $X$ durch $\beta$ im Keller.*

*Eine **endliche Berechnung von** $M$ ist eine Folge von Konfigurationen $C_1, C_2, \ldots, C_m$, so dass $C_i \vdash_M C_{i+1}$ für $i = 1, 2, \ldots, m-1$ gilt. Für jedes Wort $x \in \Sigma^*$ ist*

$$(q_0, x, Z_0)$$

*die **Anfangskonfiguration** von $M$ auf $x$.*

*Eine Berechnung $C_0, C_1, \ldots, C_m$ von $M$ heisst eine **akzeptierende Berechnung** von $M$ auf $x$, falls*

$$C_0 = (q_0, x, Z_0) \quad \textit{und} \quad C_m = (p, \lambda, \lambda)$$

*für irgendeinen Zustand $p \in Q$.*

*Die von einem **Kellerautomaten akzeptierte Sprache** $L(M)$ ist definiert durch*

$$L(M) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash_M^* (p, \lambda, \lambda) \ \textit{für ein } p \in Q\}$$

Wir bemerken, dass ein NPdA nicht mit leerem Keller arbeiten kann. Wenn daher in einer Berechnung der Keller geleert wurde und das Eingabewort noch nicht zu Ende gelesen wurde, wird in dieser Berechnung die Eingae nicht akzeptiert.

**Lemma 3.4.** *Sei $L$ eine kontextfreie Sprache, die $\lambda$ nicht enthält. Dann existiert ein NPdA $M$ mit $L(M) = L$.*

*Beweisidee.* Wir überführen die Grammatik erst in Greibach-Normalform. Dann hat jede Linksableitung von $S$ aus die Form $S \Rightarrow_G^* w\alpha$ für ein Wort $w \in \Sigma_T^*$ und $\alpha \in \Sigma_N^*$. Der konstruierte Kellerautomat $M$ hat nun nur einen Zustand. Für jede Ableitung $S \Rightarrow_G^* w\alpha$ existiert nun eine Berechnung, welche den Teil $w$ der Eingabe liest und $\alpha^R$ im Keller stehen hat. □

**Lemma 3.5.** *Für jeden nichtdeterministischen Kellerautomaten $M$ existiert ein äquivalenter nichtdeterministischer Kellerautomat $M_1$, der nur einen Zustand hat.*

*Beweisskizze.* Wir speichern den aktuellen und den vorherigen Zustand jeweils auf dem Keller, indem wir das Kelleralphabet folgendermassen festlegen: $\Gamma_1 = Q \times \Gamma \times Q$. Dabei hat $A' = (q, A, p) \in \Gamma_1$ die Bedeutung, dass der Kellerautomat $M$ das Symbol $A$ nur bei einem Übergang vom Zustand $q$ in den Zustand $p$ löschen darf. Wenn $A'$ das oberste Symbol des Kellers von $M_1$ ist, so ist $q$ der aktuelle Zustand von $M$. Wir wollen dabei immer darauf achten, dass der Keller wie abgebildet aussieht.

Abbildung 1: Keller unseres Automaten mit nur einem Zustand

| | | |
|---|---|---|
| $q$ | $A_1$ | $p_1$ |
| $p_1$ | $A_2$ | $p_2$ |
| $p_2$ | $A_3$ | $p_3$ |
| $p_3$ | $A_4$ | $p_4$ |
| | $\vdots$ | |
| $p_i$ | $A_{i+1}$ | $p_{i+1}$ |
| | $\vdots$ | |

Wenn immer nun $M$ Symbole $A_1 A_2 \cdots A_m$ auf den Keller schreibt, so schreibt $M_1$ für alle möglichen $q_1, \ldots, q_m, p_1, \ldots, p_m \in Q$ folgendes auf den Keller:

$$(q_1, A_1, q_2)(q_2, A_2, q_3) \cdots (q_m, A_m, q_{m+1})$$

□

**Satz 3.5.** *Die nichtdeterministischen Kellerautomaten erkennen genau die Klasse von kontextfreien Sprachen. Das heisst die Formalismen von nichtdeterministischen Kellerautomaten und kontextfreier Grammatiken sind bezüglich ihrer Beschreibungsstärke äquivalent.*

# 4 Turingmaschinen

## 4.1 Das Modell der Turingmaschine

**Definition 4.1.** *Eine **Turingmaschine**, oft abgekürzt als TM, ist ein 7-Tupel* $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ *mit*

*i)* $Q$ *ist die endliche **Zustandsmenge***

*ii)* $\Sigma$ *ist das **Eingabealphabet**, wobei* $¢$ *und* $␣$ *nicht dazu gehören*

*iii)* $\Gamma$ *ist das **Arbeitsalphabet**, mit* $¢, ␣ \in \Gamma$ *und* $\Sigma \subset \Gamma$

*iv)* $\delta : (Q - \{q_{accept}, q_{reject}\}) \times \Gamma \to Q \times \Gamma \times \{L, N, R\}$ *eine Abbildung, die **Übergangsfunktion** von M, ist mit der Eigenschaft*

$$\delta(q, ¢) \in Q \times \{¢\} \times \{N, R\}$$

*für alle* $q \in Q$.

*δ beschreibt die elementare Operation von M. M kann eine Aktion* $(q, X, Z) \in Q \times \Gamma \times \{L, N, R\}$ *aus dem aktuellen Zustand p beim Lesen eines Symboles* $Y \in \Gamma$ *durchführen, falls* $\delta(p, Y) = (q, X, Z)$. *Dies bedeutet den Übergang von p nach q, das Ersetzen von Y durch X und die Bewegung des Kopfes entsprechend Z.* $Z = L$ *bedeutet die Bewegung nach links,* $Z = R$ *nach rechts und bei* $Z = N$ *bewegt sich der Kopf gar nicht. Die spezielle Eigenschaft von δ verhindert das ersetzen des speziellen Symboles* $¢$ *oder dass sich der Lesekopf vom Arbeitsband wegbewegt.*

*v)* $q_0 \in Q$ *ist der **Anfangszustand***

*vi)* $q_{accept} \in Q$ *ist der **akzeptierende Zustand**. Es gibt stets nur einen akzeptierenden Zustand, und von dort kann M keine Arbeit mehr verrichten, unabhängig von der Position des Lesekopfes.*

*vii)* $q_{reject} \in Q \setminus \{q_{accept}\}$ *ist der **verwerfende Zustand**. Wie oben gibt es nur einen solchen, und M kann jederzeit die Eingabe verwerfen oder akzeptieren (insbesondere ohne die gesamte Eingabe gelesen zu haben).*

*Eine **Konfiguration** C von M ist ein Element aus*

$$\mathrm{Konf}(M) = (\{¢\} \cdot \Gamma^* \cdot Q \cdot \Gamma^+) \cup (Q \cdot \{¢\} \cdot \Gamma^*)$$

*und eine **Startkonfiguration** für ein Eingabewort x ist* $q_0 ¢ x$.

*Ein **Schritt** von M ist eine Relation* $\vdash_M$ *auf der Menge der Konfigurationen.*

*Weiter ist eine **Berechnung** von M eine (potenziell unendliche) Folge von Konfigurationen* $C_0, C_1, C_2, \ldots$, *so dass* $C_i \vdash_M C_{i+1}$ *für alle* $i = 1, 2, \ldots$.

*Die **Berechnung von** M **auf einer Eingabe** x ist eine Berechnung, die mit der Startkonfiguration* $C_0 = q_0 ¢ x$ *beginnt und entweder unendlich ist, oder in einer Konfiguration* $w_1 q w_2$ *endet, wobei* $q \in \{q_{accept}, q_{reject}\}$.

*Eine Berechnung von M aus x heisst **akzeptierend**, falls sie in einer akzeptierenden Konfiguration* $w_1 q_{accept} w_2$ *endet. Die Berechnung heisst **verwerfend**, wenn sie im Zustand* $q_{reject}$ *endet. Eine **nicht-akzeptierende** Berechnung kann entweder verwerfend oder unendlich sein.*

*Die von der **Turingmaschine** M **akzeptierte Sprache** ist damit*

$$L(M) = \{w \in \Sigma^* \mid q_0 ¢ w \vdash_M^* y q_{accept} z, y, z \in \Gamma^*\}$$

*Wir sagen, dass M eine Funktion* $F : \Sigma^* \to \Gamma^*$ ***berechnet**, falls*

$$\forall x \in \Sigma^* : q_0 ¢ x \vdash_M^* q_{accept} ¢ F(x)$$

*Eine Sprache heisst **rekursiv aufzählbar**, falls eine TM M existiert, mit* $L(M) = L$. *Weiter ist*

$$\mathcal{L}_{RE} = \{L(M) \mid M \text{ ist eine TM}\}$$

*die **Menge der rekursiv aufzählbaren Sprachen**.*

*Eine Sprache heisst **rekursiv (entscheidbar)**, falls* $L = L(M)$ *für eine TM M, welche keine unendlichen Berechnungen hat. M hält also immer.*

*Zuletzt ist*

$$\mathcal{L}_R = \{L(M) \mid M \text{ ist eine TM, die immer hält}\}$$

*die **Menge der rekursiven (algorithmisch erkennbaren) Sprachen**.*

## 4.2 Mehrband-Turingmaschinen und Church'sche These

Um sich die Argumentation mit Turingmaschinen zu vereinfachen, führt man für jede positive ganze Zahl $k$ die sogenannte $k$-Band-Turingmaschine ein, mit folgenden Komponenten:

- eine endliche Kontrolle (Programm)
- ein endliches Eingabeband mit einem Lesekopf, begrenzt durch ¢ und \$
- $k$ Arbeitsbänder, jedes mit eigenem Lese-/Schreibkopf

Sämtliche $k+1$ Köpfe können sich in jedem Schritt unabhängig voneinander um höchstens eine Position bewegen, wobei sie die Markierung ¢ nie überschreiten dürfen. Weiter kann der erste Lesekopf des Eingabebandes den Inhalt nicht verändern, und darf zudem die Begrenzung durch \$ auf der rechten Seite nicht überschreiten.

**Satz 4.1.** *Die Maschinenmodelle von Turingmaschinen und Mehrband-Turingmaschinen sind äquivalent.*

#### Church'sche These

*Die Turingmaschinen sind die Formalisierung des Begriffes "Algorithmus", d.h. die Klasse der rekursiven Sprachen (der entscheidbaren Entscheidungsprobleme) stimmt mit der Klasse der algorithmisch (automatisch) erkennbaren Sprachen überein.*

Offensichtlich kann die Church'sche These nicht bewiesen, sehr wohl aber wiederlegt werden. Dies liegt an der Verwendung des intuitiven Begriffes Algorithmus.

## 4.3 Nichtdeterminischtische Turingmaschinen

Analog zu nichtdeterministischen endlichen Automaten oder Kellerautomaten kann man natürlich auch für Turingmaschine den Nichtdeterminismus einführen. Dies funktioniert genauso wie früher, über leichte Modifizierungen an der Definition der (deterministischen) Turingmaschinen, insbesondere der Transitionsfunktion.

Ebenfalls kann man ganz analog Berechnungsbäume für Turingmaschinen einführen, und so über die Beitensuche die Arbeit von nichtdeterministischen Turingmaschinen simulieren.

**Satz 4.2.** *Sein $M$ eine NTM. Dann existiert eine TM $A$, so dass*

*i) $L(M) = L(A)$, und*

*ii) falls $M$ keine unendlichen Berechnungen auf Wörtern aus $(L(M))^{\complement}$ hat, dann hält $A$ immer.*

## 4.4 Kodierung von Turingmaschinen

Es erscheint trival, dass man eine (binäre) Kodierung für Turingmaschinen definieren kann. Ebenfalls ist es natürlich möglich, algorithmisch zu überprüfen, ob ein gegebenes Wort $x \in \{0,1\}^*$ eine Kodierung einer Turingmaschine darstellt.
Diese beiden Beobachtungen implizieren nun aber, dass KodTM, die Menge der Kodierungen aller Turingmaschinen, abzählbar ist. Man kann also für eine gegebene Turingmaschine deren Ordnung $i$ bestimmen. Die Ordnung ist dabei so definiert, dass es genau $i-1$ Wörter $y$ gibt, welche eine Turingmaschine kodieren, und in kanonischer Reihenfolge vor $x$ kommen. Ebenso gilt die Umkehrung; für gegebenes $i$ lässt sich also die Turingmaschine $M_i$ mit Ordnung $i$ erzeugen.

# 5 Berechenbarkeit

## 5.1 Die Methode der Diagonalisierung

**Definition 5.1.** *Seien $A$ und $B$ zwei Mengen. Wir sagen, dass*

$$|A| \leq |B|$$

*falls eine injektive Funktion $f$ von $A$ nach $B$ existiert. Wir sagen, dass*

$$|A| = |B|$$

*falls $|A| \leq |B|$ und $|B| \leq |A|$ (d.h. es existiert eine Bijektion zwischen $A$ und $B$). Weiter sagen wir*

$$|A| < |B|$$

*falls $|A| \leq |B|$ und es keine injektive Abbildung von $B$ nach $A$ gibt.*

**Definition 5.2.** *Eine Menge $A$ heisst **abzählbar**, falls $A$ endlich ist, oder wenn $|A| \leq |\mathbb{N}|$ gilt.*

**Lemma 5.1.** *Sei $\Sigma$ ein beliebiges Alphabet. Dann ist $\Sigma^*$ abzählbar, wobei die kanonische Ordnung eine mögliche Aufzählung liefert.*

**Satz 5.1.** *Die Menge* KodTM *der Turingmaschinen-Kodierungen ist abzählbar.*

**Lemma 5.2.** *$(\mathbb{N} \setminus \{0\}) \times (\mathbb{N} \setminus \{0\})$ und $\mathbb{Q}^+$ sind abzählbar.*

*Beweis.* Um die Aussage zu beweisen, genügt es eine Nummerierung der Mengen anzugeben. Dazu schreibt man sich die Elemente aus $(\mathbb{N} \setminus \{0\}) \times (\mathbb{N} \setminus \{0\})$ in eine Tabelle, wobei in der $i$-ten Zeile in der $j$-ten Spalte das Element $(i, j)$ steht. Nun beginnt man bei $(1, 1)$, und läuft dann den Diagonalen dieser Tabelle entlang. Dies liefert direkt eine Nummerierung. □

**Satz 5.2.** *Das Intervall $[0, 1]$ ist nicht abzählbar.*

*Beweis.* Der Beweis läuft über die sogennante **Diagonalisierungmethode**. Dazu überlegt man sich, dass sich alle Elemente aus $[0, 1]$ als (potenziell unendliche) Binärstrings darstellen lassen. Wir nehmen an, dass $[0, 1]$ abzählbar ist, und führen dies auf einen Widerspruch. Wegen der Abzählbarkeit gibt es eine Nummerierung aller dieser Binärfolgen, also können wir alle der Reihe nach in eine (unendliche) Tabelle schreiben. Wir zeigen nun, wie man daraus ein Element konstruieren kann, welches nicht in dieser Tabelle vorkommt, aber für eine Zahl aus $[0, 1]$ steht.
Dazu wählt man an der $i$-ten Stelle unseres Binärstings genau dann eine Eins, wenn der $i$-te String (gemäss der Aufzählung) an dieser Stelle eine Null hat. Das so entstehende Element unterscheidet sich mindestens in einem Bit von jedem Eintrag unserer Tabelle, was offensichtlich ein Widerspruch ist. Daher ist $[0, 1]$ nicht abzählbar. □

**Satz 5.3.** *$\mathcal{P}((\Sigma_{bool})^*)$ ist nicht abzählbar.*

**Korollar 5.1.** *$|\,\text{KodTM}\,| < |\mathcal{P}((\Sigma_{bool})^*)|$ und somit existieren unendlich viele nicht rekursiv abzählbare Sprachen über $\Sigma_{bool}$.*

Mithilfe der Diagonalisierungsmethode lässt sich nun direkt eine erste, nicht rekursiv aufzählbare Sprache finden. Dazu sei $w_i$ das $i$-te Wort und $M_i$ die $i$-te Turingmaschine bezüglich kanonischer Ordnung. Nun definieren wir uns eine unendliche Boole'sche Matrix $A = (d_{ij})_{i,j=1,\ldots,\infty}$ mit

$$d_{ij} = 1 \quad \Longleftrightarrow \quad M_i \text{ akzeptiert } w_j$$

Damit bestimmt die $i$-te Zeile $d_{i1}d_{i2}d_{i3}\ldots$ die Sprache der Turingmaschine $M_i$, nämlich

$$L(M_i) = \{w_j \mid d_{ij} = 1 \text{ für alle } j \in \mathbb{N} \setminus \{0\}\}$$

Nun können wir direkt die **Diagonalsprache** $L_{diag}$ angeben, die von keiner Turingmaschine akzeptiert wird.

$$L_{diag} = \{w \in (\Sigma_{bool})^* \mid w = w_i \text{ für ein } i \in \mathbb{N} \setminus \{0\} \text{ und } M_i \text{ akzeptiert } w_i \text{ nicht}\}$$

**Satz 5.4.** *$L_{diag} \notin \mathcal{L}_{RE}$*

## 5.2 Die Methode der Reduktion

**Definition 5.3.** *Seien $L_1 \subseteq \Sigma_1^*$ und $L_2 \subseteq \Sigma_2^*$ zwei Sprachen. Wir sagen, dass $L_1$ auf $L_2$ **rekursiv reduzierbar** ist, $L_1 \leq_R L_2$, falls*

$$L_2 \in \mathcal{L}_R \quad \Longrightarrow \quad L_1 \in \mathcal{L}_R$$

**Definition 5.4.** *Seien $L_1 \subseteq \Sigma_1^*$ und $L_2 \subseteq \Sigma_2^*$ zwei Sprachen. Wir sagen, dass $L_1$ auf $L_2$ **EE-reduzierbar** ist, $L_1 \leq_{EE} L_2$, wenn eine TM $M$ existiert, die eine Abbildung $f_M : \Sigma_1^* \to \Sigma_2^*$ mit der Eigenschaft*

$$x \in L_1 \quad \Longleftrightarrow \quad f_M(x) \in L_2$$

*für alle $x \in \Sigma_1^*$ berechnet. Wir sagen auch, dass die TM $M$ die Sprache $L_1$ auf die Sprache $L_2$ reduziert.*

**Lemma 5.3.** *Seien $L_1 \subseteq \Sigma_1^*$ und $L_2 \subseteq \Sigma_2^*$ zwei Sprachen. Falls $L_1 \leq_{EE} L_2$, dann gilt auch $L_1 \leq_R L_2$.*

**Lemma 5.4.** *Sei $\Sigma$ ein Alphabet. Für jede Sprache $L \subseteq \Sigma^*$ gilt*

$$L \leq_R L^{\complement} \quad \textit{und} \quad L^{\complement} \leq_R L$$

**Lemma 5.5.** $L, L^{\complement} \in \mathcal{L}_{RE} \quad \Longrightarrow \quad L, L^{\complement} \in \mathcal{L}_R$

*Beweis.* Wenn sowohl $L$ als auch das Komplement $L^{\complement}$ rekursiv aufzählbar sind, existieren zwei Programme $A$ und $B$ mit $L(A) = L$ und $L(B) = L^{\complement}$. Damit lässt sich nun ein Algorithmus $C$ konstruieren, mit $L(C) = L$, und zwar wie folgt: $C$ simuliert für jede Eingabe $x$ parallel die Arbeit von $A$ auf $x$ und die Arbeit von $B$ auf $x$, indem jeweils abwechselnd ein Schritt von $A$, und dann ein Schritt von $B$ simuliert wird. Da $x$ entweder in $L$ oder $L^{\complement}$ liegen muss, terminiert eine dieser Simulationen in endlicher Zeit, und anhand des Ergebnisses dieser Simulation kann $C$ dann entscheiden, ob $x \in L$ oder $x \notin L$. □

**Lemma 5.6.** $(L_{diag})^{\complement} \notin \mathcal{L}_R$ *und* $(L_{diag})^{\complement} \in \mathcal{L}_{RE}$

*Beweis.* Der erste Teil folgt direkt aus dem letzen Lemma: Es gilt $L_{diag} \leq_R (L_{diag})^{\complement}$, und da $L_{diag} \notin \mathcal{L}_{RE}$ ist $L_{diag}$ auch nicht rekursiv, und somit kann auch $(L_{diag})^{\complement}$ nicht rekursiv sein.
Für den zweiten Teil geben wir eine TM $M$ an, mit

$$L(M) = (L_{diag})^{\complement} = \{w \mid w = w_i \text{ und } M_i \text{ akzeptiert } w_i\}$$

Dazu bestimmt $M$ die Ordnung $i$ von der Eingabe $x$, konstruiert dann die $i$-te Turingmaschine $M_i$ und simuliert dann deren Arbeit auf $x$. □

**Korollar 5.2.** $\mathcal{L}_R \subsetneq \mathcal{L}_{RE}$

**Definition 5.5.** *Die **universelle Sprache** ist die Sprache*

$$L_U = \{\text{Kod}(M)\#w \mid w \in (\Sigma_{bool})^* \textit{ und } M \textit{ akzeptiert } x\}$$

**Lemma 5.7.** *Es gibt eine TM $U$, **universelle TM** genannt, so dass $L(U) = L_U$. Daher gilt $L_U \in \mathcal{L}_{RE}$*

**Satz 5.5.** $L_U \notin \mathcal{L}_R$

**Definition 5.6.** *Das **Halteproblem** ist das Entscheidungsproblem $(\{0,1,\#\}^*, L_H)$, mit*

$$L_H = \{\text{Kod}(M)\#x \mid x \in \{0,1\}^* \textit{ und } M \textit{hält auf } x\}$$

**Satz 5.6.** $L_H \notin \mathcal{L}_R$

**Lemma 5.8.** $(L_{empty})^{\complement} \in \mathcal{L}_{RE} \setminus \mathcal{L}_R$ *und* $L_{empty} \notin \mathcal{L}_R$, *wobei*

$$L_{empty} = \{\text{Kod}(M) \mid L(M) = \emptyset\}$$

Als Übersicht lässt sich nun folgende Tabelle erstellen, welche die vorgestellten Sprachen vollständig klassifiziert:

| | |
|---|---|
| $\in \mathcal{L}_R$ | endliche Sprachen, "einfache" Sprachen, ... |
| $\in \mathcal{L}_{RE} \setminus \mathcal{L}_R$ | $L_U, L_H, (L_{diag})^{\complement}, (L_{empty})^{\complement}, L_{H,\lambda}, \ldots$ |
| $\notin \mathcal{L}_{RE}$ | $L_{diag}, L_{empty}, (L_U)^{\complement}, (L_H)^{\complement}, L_{all}, L_{notall}, \ldots$ |

## 5.3 Satz von Rice

**Definition 5.7.** *Eine Sprache $L \subseteq \{\text{Kod}(M) \mid M \textit{ ist eine TM}\}$ heisst **semantisch nichttriviales Entscheidungsproblem über Turingmaschinen**, falls folgende Bedingungen gelten:*

*i) Es gibt eine TM $M_1$, so dass $\text{Kod}(M_1) \in L$ (daher $L \neq \emptyset$)*

*ii) Es gibt eine TM $M_2$, so dass $\text{Kod}(M_2) \notin L$ (daher enthält L nicht die Kodierungen aller Turingmaschinen)*

*iii) für zwei Turingmaschinen $A$ und $B$ impliziert $L(A) = L(B)$*

$$\text{Kod}(A) \in L \quad \Longleftrightarrow \quad \text{Kod}(B) \in L$$

**Lemma 5.9.** $L_{H,\lambda} \notin \mathcal{L}_R$, *wobei* $L_{H,\lambda} = \{\text{Kod}(M) \mid M \textit{ hält auf } \lambda\}$

**Satz 5.7.** ***Satz von Rice***
*Jedes semantisch nichttriviale Entscheidungsproblem über Turingmaschinen ist unentscheidbar.*

## 5.4 Die Methode der Kolmogorov-Komplexität

**Satz 5.8.** *Das Problem, für jedes $x \in (\Sigma_{bool})^*$ die Kolmogorov-Komplexität $K(x)$ von $x$ zu berechnen, ist algorithmisch unlösbar.*

*Beweis.* Der beweis läuft indirekt. Sei $A$ ein Algorithmus, der $K(x)$ berechnet, und sei $x_n$ das erste Wort (bezüglich kanonischer Ordnung), für welches $K(x) \geq n$ gilt. Dann lässt sich einfach ein Algorithmus $B_n$ angeben, der mittels $A$ das Wort $x_n$ generiert:

i) Beginne mit $x = \lambda$, dem ersten Wort der kanonischen Ordnung.

ii) Berechne nun, $K(x)$, und falls $K(x) < n$, generiere für $x$ das nächste Wort und beginne erneut bei ii).

iii) Gib $x = x_n$ aus.

Da nun aber der einzig variable Teil $n$ ist, gilt für $K(x_n)$ folgends:

$$K(x_n) \leq \lceil \log_2 n \rceil + c$$

Dies ist aber ein Widerspruch zur Annahme, dass $x_n$ das erste Wort mit $K(x_n) \geq n$ sei. □

**Lemma 5.10.** *Falls $L_H \in \mathcal{L}_R$, dann existiert ein Algorithmus zur Berechnung der Kolmogorov-Komplexität $K(x)$ für jedes $x \in (\Sigma_{bool})^*$.*

*Beweis.* Wenn man für gegebene TM entscheiden kann, ob diese auf einer Eingabe hält, so kann man einfach alle Programme (gemäss kanonischer Ordnung) durchgehen, jeweils prüfen ob es hält, und dann deren Arbeit simulieren. Falls die Ausgabe das gewünschte Wort ist, bestimmt man die Länge der momentan betrachteten TM, und hat so die Kolmogorov-Komplexität gefunden. □